Computing and Informatics, Vol. 37, 2018, 1–22, doi: 10.4149/cai_2018_1_1

TWO-LAYER LOAD BALANCING FOR ONEDATA SYSTEM

Łukasz Opioła, Łukasz Dutka, Michał Wrzeszcz

AGH University of Science and Technology
ACC Cyfronet AGH
ul. Nawojki 11
30-950 Kraków, Poland
e-mail: {lopiola, dutka, wrzeszcz}@agh.edu.pl

Renata Słota, Jacek Kitowski

AGH University of Science and Technology
Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science
al. A. Mickiewicza 30
30-059 Kraków, Poland
&
ACC Cyfronet AGH
ul. Nawojki 11
30-950 Kraków, Poland
e-mail: {rena, kito}@agh.edu.pl

Abstract. The recent years have significantly changed the perception of web services and data storages, as clouds became a big part of IT market. New challenges appear in the field of scalable web systems, which become bigger and more complex. One of them is designing load balancing algorithms that could allow for optimal utilization of servers' resources in large, distributed systems. This paper presents an algorithm called Two-Level Load Balancing, which has been implemented and evaluated in Onedata – a global data access system. A study of Onedata architecture, request types and use cases has been performed to determine the requirements of load balancing set by similar, highly scalable distributed systems. The algorithm was designed to match these requirements, and it was achieved by using a synergy of

DNS and internal dispatcher load balancing. Test results show that the algorithm does not introduce considerable overheads and maintains the performance of the system on high level, even in cases when its servers are not equally loaded.

Keywords: Load balancing, geographically distributed systems, DNS, web clusters

Mathematics Subject Classification 2010: 68M11, 68M14, 68W99

1 INTRODUCTION

The number of Internet users grows every day, and the software and hardware vendors have to face demands for fast, convenient and massively scalable services. At the same time as web services are thriving, data centres around the globe are racing each other to create more and more powerful supercomputers. Many institutions form Grids and Clouds to provide platforms of immense computing power and storage space. Thanks to this, scientists of various disciplines can conduct research and cooperate to solve problems which have been beyond their reach for years. Such approach is called e-Science [9] and has created new trends in distributed computing. Again, there is a demand for scalable web services that will allow to harness the potential of distributed environments and ensure the convenience of using resources offered by data centres.

The answer to current challenges in the sector of web services are web clusters and distributed web systems, composed of tens or hundreds of servers. Recently, more and more systems try to use the advantages of highly distributed architectures. Popular web portals have to constantly extend their pools of servers to handle the growing amount of clients and data produced by them. Among them, there are cloud storage services such as Dropbox, Google Drive or OneDrive [1], which focus on any time and any place access to user data. Cloud solutions are also built for High Performance Computing (HPC) purposes, e.g. Amazon Web Services [2] and IBM HPC Cloud [3], which allow performing data-intensive computations on distributed architectures. Other massively scalable systems that handle millions of requests every minute and store enormous amounts of data in databases, include Facebook [4], Twitter [5] or Gmail [6]. As the complexity and distribution of those systems grow, it is harder to manage their numerous servers. Relevantly, the full, collective potential of multi-node architectures can be unlocked only by use of dedicated solutions that allow for cooperation of the servers. Among them, load balancing is arguably one of the most important. It determines the way that incoming requests are distributed to servers and strives to efficiently use their cumulative resources. The approaches used in modern systems range from simple static to complicated dynamic algorithms supported by server monitoring, dedicated hardware, advanced mathematical models and others. The choice of load balancing algorithm is dependent on many factors

including the system size, its purpose, type of clients and anticipated throughput of requests.

This paper describes an originally created load balancing algorithm called Two-Level Load Balancing. It is dedicated for highly scalable, distributed web systems that are meant to handle numerous requests that vary in size and processing time. It has been implemented and evaluated in `Onedata` [7, 8] to prove its viability in such systems.

`Onedata`, a global data access system, was created as an answer to the requirements of modern science. It virtualizes the storages of globally distributed storage providers and unifies them into one data space. From the user's point of view, it hides the growing complexity of storage systems. Moreover, it facilitates administration with advanced monitoring tools and automated data management. Beside being a new, powerful tool for research teams, it is also a promising choice for everyday users.

The rest of the paper is organized as follows. The second section includes gathered knowledge about existing load balancing strategies. In the third section, the `Onedata` system is described in detail to underline the requirements of load balancing algorithms that can be used in similar systems. In the next sections, the proposed solution is presented and evaluated.

2 STATE OF THE ART

For better understanding and to avoid ambiguities, several terms should be introduced that are used throughout this paper. Starting with architectural terms, a node is a single machine (computer). Nodes can be organized in groups and interconnected to form a cluster of nodes. Finally, a cluster of nodes can communicate with other clusters to form a distributed network of clusters. Another frequently used phrase is web system, which is a collection of hardware and software that constitutes a portal available on the Internet. A web system can be deployed on one of mentioned architectures, by installing server software on the nodes. A node that runs a server application is called a server (or a web server). An application that connects to a web system and sends requests is called a client, and it is most often a web browser. In cases where a web system runs on more than one node, a load balancing algorithm is required. It describes the decision process used to distribute incoming client requests among servers of a web system.

Load balancing is naturally an important aspect of every web system that can be deployed on a cluster of nodes. Hence, in recent years, numerous load balancing algorithms have been invented, tested and documented. The approaches differ depending on many factors, such as the distribution of the system, used hardware, the homogeneity and size of the cluster and more. The main focus of this section is to gather and systematize the knowledge about load balancing, in order to provide a clear background for issues addressed in this article.

2.1 Classifications of Load Balancing Algorithms

Load balancing algorithms can be categorized [10] based on four important features:

- physical location of cluster nodes,
- visibility in IP network,
- OSI model layer on which they operate,
- static or dynamic character.

Physical location of cluster nodes can be classified as local scale-out and global scale-out. Local scale-out is an installation where all the nodes reside in a separate area, close to each other and interconnected. If a web system is composed of geographically distributed servers, it is called a global scale-out setup. The distribution of servers is a key factor when designing a load balancing algorithm.

As far as visibility in IP network is concerned, two classes can be determined – web clusters and distributed web systems. If the whole system is visible under one virtual IP address, it is often called a web cluster or a cluster-based web system. This setup can be accomplished in several ways, usually by directing the incoming traffic to nodes via a front-end network device (switch or router), which often acts as a load balancer. In contrast, distributed web systems are structures where multiple nodes are visible to the clients under their distinct IP addresses. In these cases, mostly DNS servers with dedicated algorithms are employed to provide load balancing [15]. Intuitively, most web clusters are local scale-out setups, and many distributed web systems are built on global scale-out architectures, but this is not always the case.

The next categorization is based on the OSI model layer, on which the load balancing is executed. In case of web clusters, the network device which distributes load among cluster nodes commonly operates on layer 2, 3 or 4 [10]. It is not a rule, though, some algorithms include web switches that are aware of the application content (OSI layer 7) of incoming requests and consider it in decision process [11]. Layer 7 policies can likewise be based on a software dispatcher incorporated in a web server, which reroutes requests internally using the knowledge of request content and application logic.

Finally, load balancing algorithms can be categorized as static and dynamic. Static algorithms base on initial configuration that does not change in time, for example number of servers or their computing power. Round-Robin (RR) and Weighted Round-Robin (WRR) are good examples – they are widely used in web systems with satisfying results, thanks to being uncomplicated and not introducing considerable overheads. For these reasons they are also a good benchmark while testing more complex algorithms. Dynamic load balancing actively uses feedback from servers, for example current load or number of queued requests, to make request distribution decisions. Popular algorithms include shortest queue, least-connection and load-based approaches [11]. Although static algorithms are often sufficient for smaller, less complicated systems, they do not provide elasticity and are poorly fitted for systems with dynamic and database-driven workloads. It has been shown

in many publications that a well-designed dynamic algorithm can manifest better performance than RR or WRR in more complex systems. This is why dynamic algorithms are subject to research and there have been many attempts to create new, effective solutions.

2.2 Related Works

This section describes several examples of dynamic load balancing algorithms which can be found in literature. The first one is a load balancing algorithm presented in [12], intended for web clusters. It is based on an approximation model which helps estimate the utilization and capacity of web servers. In addition, feedback from servers is used to correct the estimation errors of the model. The knowledge gathered in this way is used to make load balancing decisions and the results are very appealing. However, the whole logic is enveloped in a complicated web switch and application servers have to be compatible with it.

Another algorithm for web clusters was presented in [13] and has also been proven to give better results than static algorithms. It utilizes a theoretical model based on Markov chain and a probability generating function. The analysis covers queue lengths and mean cyclic time for queries, and the authors show that the proposed model is realistic and applicable in load balancing. Nonetheless, it is not universal, as it assumes the environment to be a web cluster with a special hierarchical architecture.

Remarkable algorithms have also been developed in the field of distributed web systems. In such systems, load balancing is executed by DNS servers. Basic approach uses Round-Robin policy to distribute load among multiple servers, but often other algorithms are employed with better results than simple DNS RR. The authors of [14] proposed a solution where the DNS server (working in RR mode) does not require modifications, but it is dynamically updated with the list of available web servers. Based on the knowledge of current load of servers and a configurable threshold, the algorithm decides which servers are overloaded and should be temporarily removed from DNS server's list. Its advantage is compatibility with third party DNS server software, as the updates are part of DNS specification (see RFC 2136 [16]). However, the mechanism requires a load balancing module that gathers monitoring data from the servers and performs the updates. It might introduce substantial overheads and delays in reacting to current load fluctuations. What is more, this approach might be insufficient in systems with high incoming network traffic, because of DNS response caching in intermediate DNS servers and in client machines. Overloaded nodes would still receive requests until the expiry of a DNS reply.

Other algorithms include customizations of the DNS servers, for example in [15], where they are modified to direct clients to geographically closest web servers. The purpose is to reduce network impact on communication and it is achieved using a proximity algorithm. Moreover, if a web server becomes overloaded, it delegates some requests with a simple HTTP redirect. This should enhance the Quality of Service of the whole system, but it is not trivial to decide when and where such

redirects should be performed. If this issue is handled wrongly, the response times of the web system might increase. Moreover, if redirected requests reach a server that is overloaded too, it might cause another redirection, which leads to undesirable instability in periods of high network traffic.

To summarize, there are numerous load balancing algorithms documented, some are dedicated for certain systems and not each of them can be adapted to be used elsewhere. To the best of our knowledge, none of discussed policies could be employed with satisfying results in highly scalable, distributed systems with wide request type spectrum, such as `Onedata`. The reasons are usually connected with architectural assumptions of these solutions or the lack of features that are crucial in such systems.

3 LOAD BALANCING IN ONEDATA – REQUIREMENTS ANALYSIS

This section is intended to provide overview on the `Onedata` system, its use cases and main features. It outlines the challenges connected with designing an effective load balancing algorithm that would be compatible with its architecture and have desired qualities.

3.1 `Onedata` Overview

The main goal of `Onedata` is to provide unified and efficient access to data stored in globally distributed environments [8]. The need for such system was initially observed among users of big computing infrastructures [19], such as PL-Grid (The Polish Grid Infrastructure) [17] or EGI (The European Grid Infrastructure) [18]. Many scientists of various disciplines conduct data-intensive research, and sometimes they would like to simultaneously use infrastructures located in different data centres, or cooperate with research associates in other institutions. Currently, it is inconvenient as the users often have to manually manage and migrate their data between different data centres. What is more, various storage systems that they use have different interfaces and often require technical knowledge to use. `Onedata` removes these barriers by virtualizing globally distributed storage systems of different providers. In other words, it introduces a virtual file system that uses the collective resources of all storage systems to which a user has access and gives the user a unified view on his data. With data management and migration handled transparently to the users, it is an excellent tool for any time, any place data access. What distinguishes `Onedata` from other similar tools is its support for High Performance Computing (HPC), which is crucial from the perspective of computing infrastructures users. While `Onedata` was originally dedicated for big data centres and scientists, it might have potential in the commercial market too. It is possible because the today's everyday users are encouraged to store their data in clouds and would like to have a coherent and uniform view on their data.

3.1.1 System Architecture

From the global point of view, `Onedata` is composed of many cooperating deployments, one per every provider that decides to enter the `Onedata` system with its storage resources (see Figure 1). Such deployment is called `Oneprovider cluster` and is essentially a web system hosted on a cluster of nodes. The instances of `Oneprovider clusters` cooperate globally to unify user's view on his data. The significant feature of this cooperation is that the providers can retain their autonomy and do not have to trust each other. This is achieved with a mediator called *GlobalRegistry* and other innovative architectural solutions [8]. This paper describes a load balancing algorithm that is limited to the context of a single `Oneprovider cluster` instance.

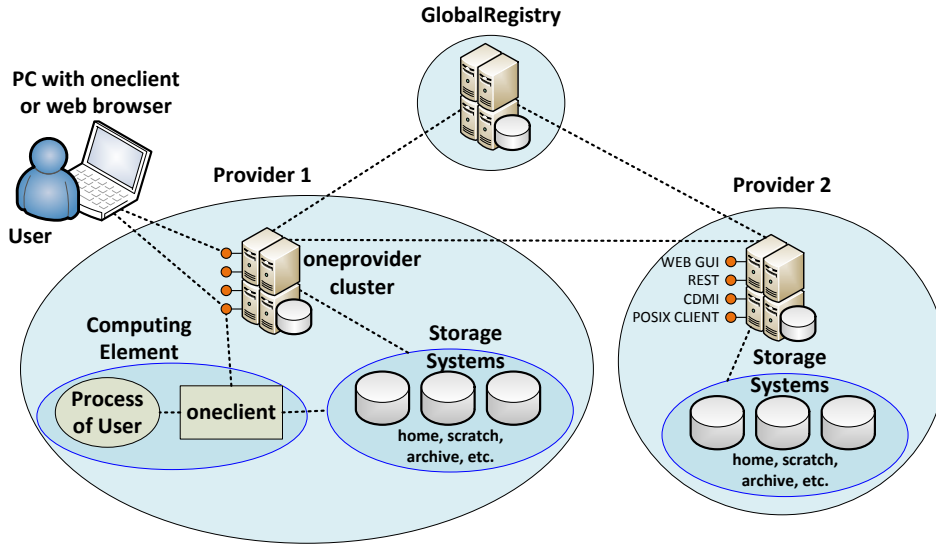


Figure 1. Exemplary environment with `Onedata`

`Oneprovider cluster` is a distributed application deployed on multiple nodes, each having its own, external IP address and each connected to underlying storage systems. The main responsibilities of `Oneprovider cluster` include file metadata management, data access coordination and, most importantly in the context of load balancing, processing of user requests. The application is written in Erlang, which is renowned for its powerful distribution mechanisms, as well as parallel processing capabilities. This choice of technology has proven advantageous in the light of scalability and high availability requirements of `Onedata`.

In reference to classifications presented before, the architecture of `Oneprovider cluster` falls in the category of distributed web systems, and is based on a local scale-out infrastructure. This is an unusual setup, and in connection with mech-

anisms implicitly offered by Erlang, it constitutes a unique case for load balancing.

3.1.2 Client and Request Types

There are several interfaces through which the `Onedata` system can be accessed. Firstly, a client application called `Oneclient`, which exposes a standard POSIX file system interface. Secondly, an intuitive web GUI that provides easy access from any place connected to the Internet. Finally, REST and CDMI (Cloud Data Management Interface) endpoints that are suited for third party service developers.

The `Oneclient` application is based on FUSE (Filesystem in Userspace). It allows for mounting a virtual file system in UNIX-based systems, so that a `Onedata` user can access his data exactly like on a local storage. `Oneclient` communicates with a `Oneprovider cluster` and handles low level file management transparently. This is a perfect solution for a typical user, but can also greatly facilitate computations and experiments performed by scientists in data centres. `Onedata` puts a great importance on support for HPC and the client application is fitted to handle data-intensive operations. If the host of `Oneclient` has direct access to storage systems used by `Oneprovider cluster`, their communication is limited to metadata only and `Oneclient` reads and writes the data directly from the storage systems. In other cases, e.g. when it is installed on a personal notebook, it works in a slower mode, as the `Oneprovider cluster` has to mediate in all file operations. Hence, the client applications can generate various types of requests. In both cases (direct and remote file access) it performs numerous, lightweight metadata requests, such as retrieving physical location of a file. However, a remote `Oneclient` (e.g. installed on a PC) has to send and receive a lot of data through the network connection during I/O operations, which consist of multiple read and write requests. It is also possible for a directly connected `Oneclient` to require data transfers on the wire, for instance when some files are located on the storage of another provider.

The web GUI was designed to provide access to data without installation of any software and ensure a user-friendly experience. It features an intuitive and responsive file manager with data sharing and publishing capabilities, among others. The communication with `Oneprovider cluster` is based on secure HTTPS and websocket protocols, thus some connections can be kept alive for a long time. Noticeably, all the operations must be performed via `Oneprovider cluster`, including those that cause high network interface usage by performing file uploads and downloads. On the other hand, some requests might be lightweight (e.g. renaming a file), or use a lot of computing resources (e.g. removing large directories).

Third party service developers can use the REST and CDMI APIs to integrate with `Onedata`. They both rely on stateless, secure HTTPS connections. The REST endpoints offer most file operations in a robust, concise and well documented API based on interoperable JSON. CDMI complaint interface, that allows both low and high level operations on user data, constitutes a more complicated yet usable and powerful alternative. The great advantage of CDMI is introduction of file opera-

tions that are not available in `Oneclient`, for example issuing copying of large files and directories, often used in pre-staging of content before data-intensive computations. As far as REST and CDMI are concerned, the processing time and volume of requests may vary dramatically depending on the type of operations performed. In addition, some requests do not carry much information on the wire, but might consume considerable amounts of server's processing power or memory (e.g. copying large files). Those interfaces might contribute considerably to `Oneprovider cluster` load, especially when it is deployed on a Grid infrastructure and integrated with middleware or job schedulers.

Noticeably, some of `Oneprovider cluster` load results from communication with *GlobalRegistry* and other `Oneprovider clusters`. It is based on REST interfaces and essentially the performed requests have the same characteristics as described above. Hence, this communication is not perceived as a separate class of requests.

	Oneclient direct i/o	Oneclient remote	web GUI	REST & CDMI
light	++	+	+	+
small transfer	+	++	+	+
large transfer	–	–	++	++
computationally requiring	–	–	+	++

Table 1. Request types in `Onedata`

Considering the impact of requests on resource usage, we classified them into four categories: *light*, *small transfer*, *large transfer* and *computationally requiring*. *Light* requests have high priority and should be processed with no delays, i.e. metadata requests. They do not carry much information on the wire nor use much computational resources. *Small transfer* requests carry minor amounts of data back and/or forth, for instance read/write operations generated by `Oneclient` working in remote mode. Requests classified as *large transfer* carry big data payloads that can cause high usage of network interfaces. Finally, *computationally requiring* requests consume significant amounts of computing power to be processed. Table 1 contains a summary of request categories in `Onedata`. The number of '+' signs indicates how often such requests appear for different interfaces.

The variety of requests is significant in the context of load balancing, as different requests have different impact on load of cluster nodes, both network and computational (CPU, memory). It is also important that in the `Oneclient` application we have full control over server preference when connecting to `Oneprovider cluster`. However, in case of other interfaces (all HTTP based), the only way to control the choice of servers by the clients is to use a DNS server. Even then, we still cannot influence the way the DNS resolvers and web browsers work. For those reasons, `Onedata` is a great example of a distributed web system that cannot fully use all of its design advantages without a well suited load balancing algorithm.

3.1.3 `Onedata` Use Cases

To better explain the influence of different users and requests on a `Oneprovider cluster` in the context of load balancing, some concrete use cases of different interfaces are presented below.

As far as web GUI is concerned, one of the key functionalities is upload and download of files. This relates both to private content and shared files that can be downloaded by anyone possessing a valid URL. Such operations cause a substantial load on network interfaces and generate I/O operations on the server.

Considering `Oneclient`, when a data center user is performing data-intensive computations on a directly connected storage, hundreds of light metadata requests are generated and they should be handled quickly so as not to limit the `Oneclient` performance.

Further on, a researcher might want to use a copy of a huge file with gigabytes of data as an input to his simulation. He would then use the CDMI interface, most probably indirectly by middleware that can pre-stage his data. Such request requires lots of CPU time, RAM and I/O operations to be processed.

3.2 Load Balancing Scenarios

Bearing in mind the above-mentioned characteristics of `Onedata`, we identified three representative scenarios that depict requirements of load balancing for `Oneprovider cluster`:

- sharp load fluctuations,
- unbalanced use of resources,
- node failure.

The first scenario includes a set of clients (of any type) connected to a node of `Oneprovider cluster` (*node A*). Assume that all other nodes maintain similar number of connections at the time, and the cluster load is balanced. However, it is possible that suddenly some of the clients of *node A* greatly increase the intensity of their requests, as, for instance, a large grid job has started. *Node A* becomes overloaded and there is no possibility of redirecting the clients to other nodes without breaking the connections. A mechanism is required that would allow for internal rerouting of requests to nodes which have more free resources.

Another example shows why the payloads and processing time of requests should be taken into account. Consider two nodes (*A* and *B*), which hold a comparable number of connections. However, *node A* has received a lot of file upload/download requests from web clients and its interfaces are practically exhausted. Nevertheless, it still has reserves of computing power. On the other hand, *node B* is busy with processing multiple, computationally requiring CDMI requests which causes full CPU utilization, while its network interfaces stand practically unused. The cluster should direct new connections to *node B*, but at the same time internally delegate

some requests from *node B* to *node A* to achieve optimal utilization of resources on both nodes.

The next case regards a cluster of nodes and a situation where one of the nodes (*node A*) becomes temporarily nonoperational. The reason might be either a network malfunction, software failure or overload. While it is not possible to amend existing connections of *node A*, the system should be able to stop new clients from connecting to it and direct them to healthy nodes.

All the aforementioned scenarios are possible, and while such cases might not occur often, the system must be able to react rationally in any circumstances. Beside the specific scenarios where a load balancing algorithm is indispensable, there are also general requirements that it must fulfil, as described below.

3.3 Load Balancing Challenges

Firstly, the main reason of introducing any load balancing algorithm is achieving scalability. Basically, it means ability to increase system's performance by extending the cluster with new nodes. Naturally, the bigger the cluster, the less improvement will be introduced with new extensions, as the overheads of managing such an infrastructure and maintaining communication between the nodes become too substantial. The desired load balancing algorithm should allow for building clusters big enough to handle expected number of clients.

Other crucial features for distributed systems are High Availability (HA) and Quality of Service (QoS). The former means that a service should remain fully operational despite failures of some nodes or components. The latter, from the user's point of view, is how efficient, responsive and failure-free the service is. Load balancing algorithm can have a great influence on both of these aspects if it can instantly react to undesired situations and minimize traffic on overloaded nodes.

An optimal load balancing solution should incorporate mechanisms that would allow for maintaining comparable resource utilization on all nodes of a cluster, which includes computational resources as well as network interfaces. It is especially important that none of the nodes becomes significantly more loaded than the others. Another feature that is somehow connected is avoiding bottlenecks. When this is neglected, it might ruin the system's scalability and the potential of parallel processing.

Manageability is also a relevant aspect. The system administrators should be able to easily modify the size of a cluster or migrate some applications between servers, while the load balancing algorithm adapts dynamically to the new circumstances.

Last, but not least, the distributed architecture of target systems must be considered when designing a load balancing algorithm. It must be well integrated, but should also exploit the benefits of distribution.

4 TWO-LEVEL LOAD BALANCING

Considering all the aforementioned requirements, an original load balancing algorithm, called Two-Level Load Balancing (TLLB), has been designed and evaluated. The two levels refer to DNS servers and internal, application layer (OSI 7) dispatchers. A sequence diagram for reference is presented in Figure 2. It presents the complete flow of requests, starting from domain resolving and ending with a HTTPS reply from a server, with symbols indicating where each level of load balancing is applied. The whole algorithm is discussed in detail later on.

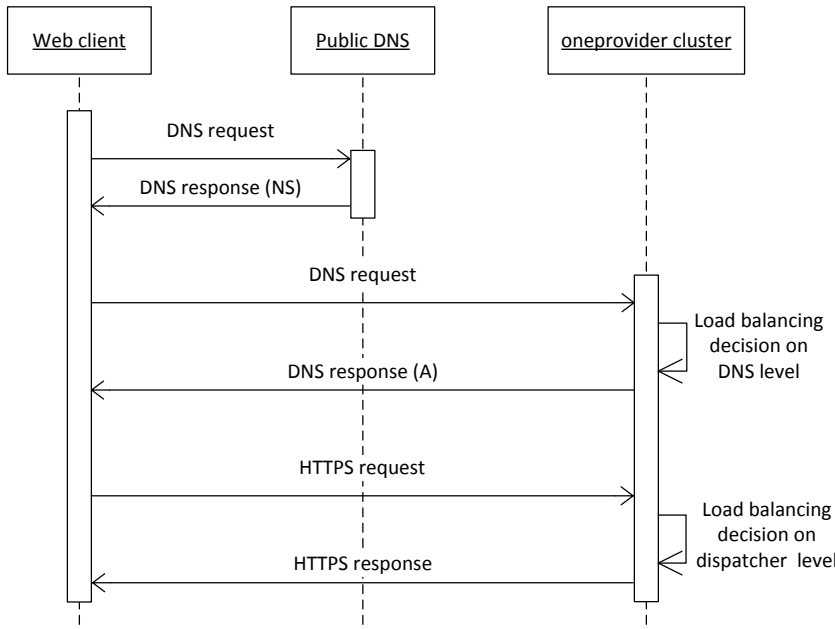


Figure 2. Two-level load balancing – sequence diagram

A significant aspect of the presented approach is that both levels are handled by software that is fully integrated in the cluster. Every node has its own external IP address and contains a DNS server and a dispatcher module. Both modules operate as part of the `Oneprovider cluster` application. It means that they can communicate with other components of the system, one of which is Central Cluster Manager (CCM). One of its responsibilities is collecting monitoring data from `Oneprovider cluster` nodes, which contains current load (CPU, memory) and network interfaces usage. This data can be used for load balancing decisions on both levels. Obviously, gathering monitoring data and processing it introduce overheads, and this fact is usually considered an important factor when comparing a dynamic algorithm to static algorithms like RR. It often happens that complicated supervision methods

and decision algorithms slow down the whole system so much that there is no point in using them. However, the monitoring in `Oneprovider cluster` was designed for advanced diagnostic and administrative tools. It is essential anyway, thus it can be used in load balancing with practically no additional cost. The algorithms are uncomplicated, and the data is processed by the CCM and served to DNS servers and dispatchers in a form of ready-to-use load balancing instructions. They are updated periodically in short intervals so that the cluster can quickly react to load fluctuations.

4.1 First Level – DNS Server

By default, all nodes IP addresses appear in the DNS response. High Availability (HA) is achieved by including multiple addresses in responses and the ability to temporarily exclude nodes that are nonoperational or unreachable. The nodes with lower load have proportionally greater probability to be placed at the top of the list (see Figure 3). While this cannot ensure that they will be preferred, most web browsers and operating system level resolvers will be more eager to choose addresses that come first. Hence, this algorithm produces the desired effects. Currently, there is no method to impose the record choice priorities on clients of `Onedata`, which are all based on HTTP protocol. The SRV DNS records could be an answer to this, but the HTTP protocol does not assume its use and popular web browsers do not support it. The load of a node, L, is calculated using a weighted average, as in Equation (1).

$$L = \frac{\alpha * net\_load + \beta * cpu\_load + \gamma * mem\_load}{\alpha + \beta + \gamma} \tag{1}$$

where *net_load* is network interfaces load, *cpu_load* is the CPU usage and *mem_load* is the memory usage. They are expressed in per cents and so is the resulting load. The network usage ratio depends on maximum interface throughput. The values of α, β and γ are determined experimentally. If α coefficient is dominating, the DNS server will be eager to lighten the network traffic to nodes with heavily utilized interfaces, which is a desired feature. Nonetheless, computational load must be also taken into consideration so that more clients can be directed to nodes with free resources. In Figure 3, an example DNS instructions has been shown, where one of the nodes has been temporarily removed because of a failure or connection problem. The sequence in exemplary response has been randomized as described before.

4.2 Second Level – Dispatcher

The dispatcher module has been introduced in order to increase control, refine the load balancing algorithm and handle edge cases. The instructions for dispatchers are created based on computational load of a node, according to Equation (2).

$$L = \frac{\beta * cpu\_load + \gamma * mem\_load}{\beta + \gamma} \tag{2}$$

where *cpu_load* is the CPU usage and *mem_load* is the memory usage. The β and γ coefficient values are selected experimentally. The network traffic is omitted, as the dispatcher does not influence the external interfaces usage, regardless of its decisions. In natural circumstances, when the nodes of `Oneprovider cluster` are similarly loaded, the dispatcher has a very low impact on request processing time as it simply follows the incoming requests to handler modules residing on the same node. However, it is crucial in cases when the load is fluctuating sharply. If the node which received the request is significantly more loaded than the others, the dispatcher will delegate such request to another one. This is determined using a threshold coefficient (Equation (3)).

$$\frac{L}{L_{min}} > \rho => overloaded. \tag{3}$$

The ρ coefficient value is selected experimentally. The load of each node – L – is compared to the lowest load in the cluster – L_{min}, and if the ratio exceeds the ρ threshold, the node is considered *overloaded* and the generated instructions for dispatchers will strive to correct that situation. They include information how often and to which nodes should requests be delegated (see Figure 3). The target node for delegation is chosen in weighted random manner, where the least loaded nodes are most probable to be chosen. The randomization approach introduces very low overheads while giving satisfying results, and allows for parallelization of request redirecting as the load balancing instructions are processed in read-only mode. Naturally, the request rerouting itself increases its processing time. However, given that the `Oneprovider cluster` nodes are interconnected with high performance interfaces and communication mechanisms in Erlang are greatly optimized, it is still beneficial to delegate the request as it will be processed faster than on a heavily loaded node. This is especially true for requests that consume a lot of resources to be processed.

4.3 Request Flow in TLLB

Figure 4 presents a `Oneprovider cluster` composed of two nodes, outlines all modules and entities that take part in load balancing process and shows the requests flow during web GUI usage.

As mentioned before, the CCM module prepares instructions for DNS and dispatcher modules based on monitoring data and propagates them periodically (step (0) in Figure 4). When a user wants to use the web GUI, firstly his browser has to resolve the `Oneprovider cluster` domain into an IP address. The domain must be public and recognizable by global DNS servers. The client performs a DNS request to a public DNS server, and receives a response indicating where it should

a) DNS instructions

	IP	probability
Node A	200.0.0.1	0.26
Node B	200.0.0.2	0.31
Node C	200.0.0.3	0.43
Node D	---	---

Exemplary DNS response:

```
200.0.0.3
200.0.0.1
200.0.0.2
```

b) dispatcher instructions

Node OK

	probability
Node A	1.0

All requests processed locally

Node overloaded

	probability
Node A	0.20
Node B	0.32
Node C	0.48

80 % of requests delegated to other nodes

Figure 3. a) DNS and b) dispatcher instructions

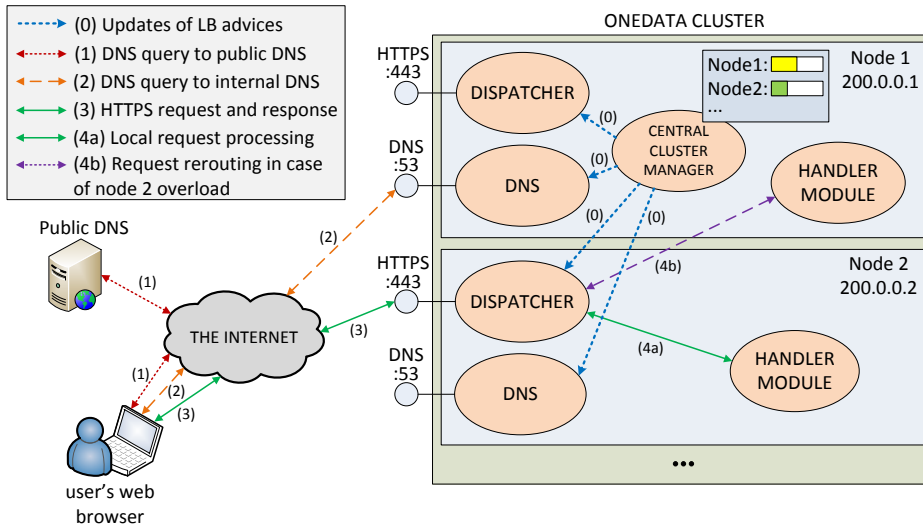


Figure 4. Request flow in Two-Level Load Balancing

ask again (step (1)). The response list contains hostnames of all the `Oneprovider cluster` nodes. The client continues to resolve the domain by asking one of the internal DNS servers (step (2)). The queried server returns a list of IP addresses of cluster nodes (sorted in a manner mentioned before), and one of the addresses is finally chosen. The client performs a HTTPS request, which reaches the dispatcher (step (3)). If the target node is not highly loaded, the request is processed locally (step (4a)) and the response is returned to the client. However, in case of an overload, delegation is performed and another, less loaded node evaluates the request (step (4b)). Eventually, the client receives the response, which has been processed

unnoticeably longer. It is important that all nodes in the cluster are able to handle any request, so the delegation algorithm is uncomplicated.

4.4 Synergy of the Two Levels

Ultimately, it should be justified why both load balancing levels shall be used instead of just one. To start with, DNS servers are indispensable in distributed web systems, i.e., when the system is reachable under multiple IP addresses. This issue could be settled by using a static, third party DNS server that uses Round-Robin shuffling. However, the proposed solution gives great elasticity and ability to quickly react to changes in the cluster structure and possible node failures. Moreover, by having access to the system status, the load balancing can be finer and better distribute incoming connections. In addition, the integration of DNS servers increases maintainability of the whole system. Secondly, the presence of dispatchers is necessary, as DNS load balancing has too high inertia. It means that it cannot responsively control the flow of requests, and it is mostly because of its responses Time To Live (TTL). For `Oneprovider cluster` it is set to a very low value of 60 seconds, but during this time the cluster status might change dramatically while the clients will still be using cached DNS responses. Hence, dispatchers are indispensable for finer and more responsive load balancing. Arguably, the two levels of load balancing create a synergy, which utilizes their best features.

5 TEST RESULTS

To evaluate the TLLB algorithm, a test environment has been set up. It was composed of multiple, homogeneous virtual nodes with enabled network emulation. `Oneprovider cluster` instances of different sizes were deployed on some nodes, while other served as clients that generated requests. The conducted tests included scalability tests based on throughput measurements. In addition, a test scenario which emphasises the two-level synergy has been evaluated. The obtained results have been normalized for more convenient analysis.

The clients' behaviour was simulated, using requests of various types and sizes – and the configurations were constant in the scope of each test. Each test was repeated multiple times and the outcomes were averaged. The repeatability of test results was high.

The aim of these tests was to assess the behaviour of the TLLB algorithm in a virtual environment, identical with target physical environment that the system could be deployed on. The tests were designed to ensure assumed features of the TLLB algorithm, such as the ability to maintain efficient system scaling or delegate requests on dispatcher level. This way, the algorithm can be safely introduced in production environment, where further testing and tuning will be performed.

The purpose of the first test was to examine the scalability of `Oneprovider cluster` depending on the use of different load balancing levels. This way, four combinations were obtained:

1. none,
2. dns,
3. disp,
4. dns_disp (see Figure 5).

Combination 1. included a DNS server working in RR mode and disabled dispatcher, i.e. all requests were processed on the target node. It served as a reference measurement as the simplest, static solution with none of proposed load balancing algorithms enabled. The next combinations were: 2. enabled DNS level and disabled dispatcher, 3. disabled DNS (RR mode) and enabled dispatcher and 4. Two-Level Load Balancing. The results are presented in Figure 5.

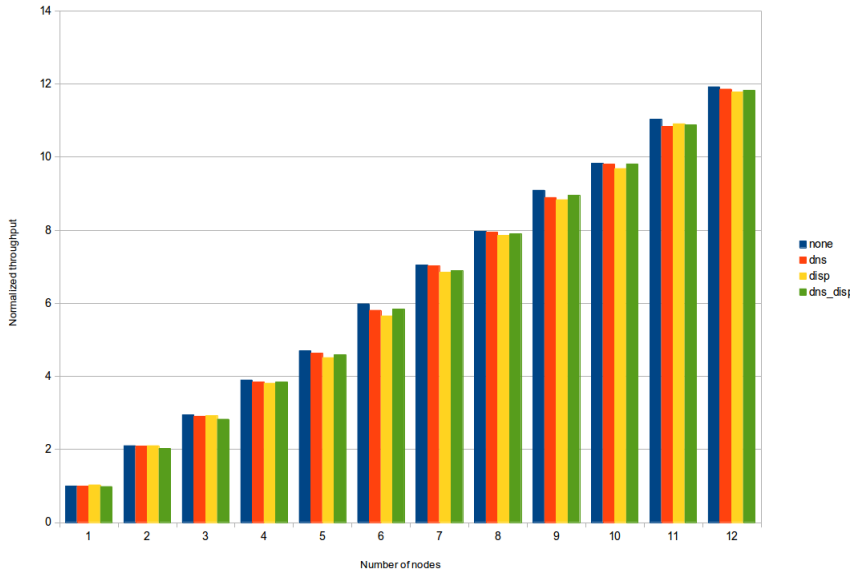


Figure 5. Scalability – throughput (normalized)

The results show that none of the proposed load balancing levels introduce significant overheads as their performance is comparable to static algorithms. In this case, an improvement over static algorithms was not anticipated because all the nodes were similarly loaded during the tests. Nevertheless, the test results justify the use of dedicated DNS servers in `Oneprovider cluster`. They achieve similar performance as standard RR algorithms, but also ensure elasticity and HA of the system by the ability to temporary exclude nonoperational nodes.

For the next test, a specific scenario was designed to evaluate the dispatcher efficiency. Only half of the nodes were receiving requests to verify if the dispatchers can cope with such situation. For reference, the tests were repeated with dispatcher

load balancing turned off, which is indicated by 'none' data series. The 'dispatcher' data series includes results when dispatchers were enabled. The results are depicted in Figure 6.

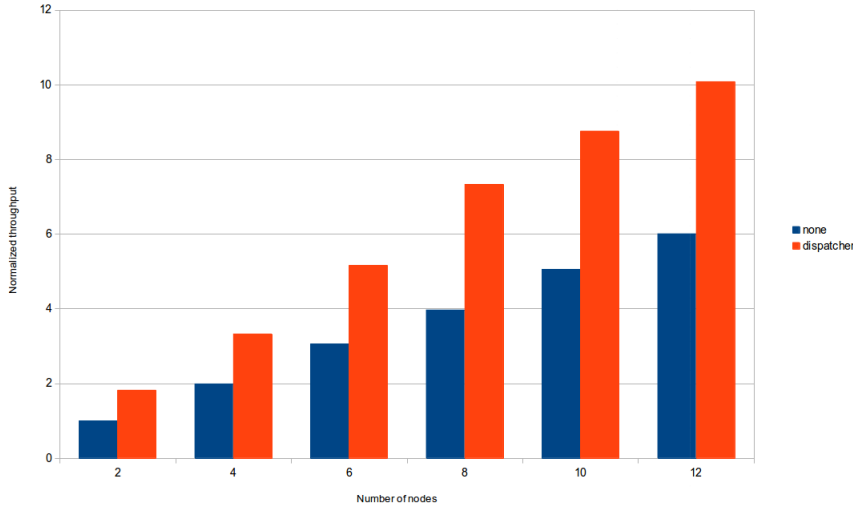


Figure 6. Dispatcher performance with uneven node loads – throughput (normalized)

Obtained results show that dispatchers were able to use free resources on the nodes that did not receive requests. The overall system throughput was nearly two times greater compared to the case when dispatchers were disabled – as requests were rerouted internally to the free nodes, they could be processed more quickly. It was not possible to double the throughput as the requests rerouting introduces overheads on the processing time, but the results prove the value of dispatchers. Their use ensures resistance to load fluctuations and efficient use of collective resources of the whole `Oneprovider cluster`.

6 CONCLUSIONS AND FUTURE WORK

The complexity and size of distributed web systems increases and so does the demand for effective load balancing algorithms. Their requirements have been identified using the case of `Onedata`, considering the various use cases of the system and diversity of requests that it processes. Finally, an innovative solution has been designed, called Two-Level Load Balancing. The first level refers to DNS servers, and the second to dispatchers that operate on application level and are able to reroute requests inside a cluster of nodes. Both levels are encapsulated in the `Oneprovider cluster` application, which allows for low-cost use of monitoring data. The DNS servers distribute load among network interfaces of the cluster in a balanced manner, while dispatchers correct load fluctuations.

The proposed Two-Level Load Balancing has been implemented and evaluated. It has proven to be a suitable choice for the `Onedata` system. The obtained results show that a `Oneprovider cluster` instance can benefit from the use of TLLB in cases of unbalanced or fluctuating loads, while during smoother periods it does not impair the performance of the system. One of the reasons the overheads are negligible is the simplicity of the algorithms used on both levels. While the two level approach is not novel, the proposed algorithm features a highly distributed architecture based on Erlang language, where load balancing modules are fully integrated with the system. This way, it becomes a unique approach. What is more, it has the potential to be used in other scalable web systems, not only in the field of global data access.

The proposed algorithm still has the potential for improvements. It has been tested in environment with simulated clients and yielded satisfying results. However, further testing and tuning should be performed in production environment. What is more, there are techniques that could potentially improve its performance. They include probabilistic load prediction based on history of requests, admission control or content awareness. Importantly, their introduction must be careful and well thought out, as it might cause overheads that could impair the system's performance.

REFERENCES

[1] Martin, A.: OneDrive vs. Google Drive vs. Dropbox: The Best Cloud Storage Service of 2017. Online. Accessed 13.09.2017. `http://www.alphr.com/dropbox/7034/onedrive-vs-google-drive-vs-dropbox-the-best-cloud-storage-service-of-2017`.

[2] Amazon Web Services (AWS) for HPC: Online. Accessed 13.09.2017. `http://aws.amazon.com/hpc/`.

[3] IBM HPC Cloud: Online. Accessed 13.09.2017. `https://ibm.com/systems/spectrum-computing/solutions/hpccloud.html`.

[4] facebook: Online. Accessed 13.09.2017. `https://facebook.com`.

[5] twitter: Online. Accessed 13.09.2017. `https://twitter.com`.

[6] gmail: Online. Accessed 13.09.2017. `https://gmail.com`.

[7] onedata: Online. Accessed 13.09.2017. `https://onedata.org`.

[8] Dutka, Ł.—Wrzeszcz, M.—Lichoń, T.—Słota, R.—Zemek, K.—Trzepla, K.—Opioła, Ł.—Słota, R.—Kitowski, J.: Onedata – A Step Forward Towards Globalization of Data Access for Computing Infrastructures. Procedia Computer Science, Vol. 51, 2015, pp. 2843–2847, doi: 10.1016/j.procs.2015.05.445.

[9] Bubak, M.—Kitowski, J.—Wiatr, K. (Eds.): eScience on Distributed Computing Infrastructure. Springer, Lecture Notes in Computer Science, Vol. 8500, 2014. ISBN 978-3-319-10893-3, doi: 10.1007/978-3-319-10894-0.

[10] Gilly, K.—Juiz, C.—Puigjaner, R.: An Up-to-Date Survey in Web Load Balancing. World Wide Web, Vol. 14, 2011, No. 2, pp. 105–131.

[11] Tiwari, A.—Kanungo, P.: Dynamic Load Balancing Algorithm for Scalable Heterogeneous Web Server Cluster with Content Awareness. Trendz in Information Sciences Computing (TISC), 2010, pp. 143–148, doi: 10.1109/TISC.2010.5714626.

[12] Sharifian, S.—Motamedi, S. A.—Akbari, M. K.: An Approximation-Based Load-Balancing Algorithm with Admission Control for Cluster Web Servers with Dynamic Workloads. The Journal of Supercomputing, Vol. 53, 2010, No. 3, pp. 440–463, doi: 10.1007/s11227-009-0303-8.

[13] Bao, L.—Zhao, D.—Zhao, Y.: A Dynamic Dispatcher-Based Scheduling Algorithm on Load Balancing for Web Server Cluster. Web Information Systems and Mining (WISM 2010). Springer, Lecture Notes in Computer Science, Vol. 6318, 2010, pp. 95–102, doi: 10.1007/978-3-642-16515-3_13.

[14] Moon, J.-B.—Kim, M.-H.: Dynamic Load Balancing Method Based on DNS for Distributed Web Systems. E-Commerce and Web Technologies (EC-Web 2005). Springer, Lecture Notes in Computer Science, Vol. 3590, 2005, pp. 238–247, doi: 10.1007/11545163_24.

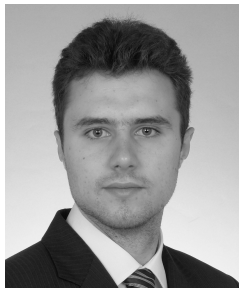
[15] Cardellini, V.—Colajanni, M.—Yu, P. S.: Geographic Load Balancing for Scalable Distributed Web Systems. Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2000, pp. 20–27, doi: 10.1109/MASCOT.2000.876425.

[16] RFC 2136: Online. Accessed 13.09.2017. https://ietf.org/rfc/rfc2136.txt.

[17] PL-Grid Infrastructure: Online. Accessed 13.09.2017. http://plgrid.pl/en.

[18] European Grid Infrastructure: Online. Accessed 13.09.2017. https://egi.eu/.

[19] Słota, R.—Dutka, Ł.—Wrzeszcz, M.—Kryza, B.—Nikolow, D.—Król, D.—Kitowski, J.: Storage Management Systems for Organizationally Distributed Environments – PLGrid PLUS Case Study. Parallel Processing and Applied Mathematics (PPAM 2013). Springer, Lecture Notes in Computer Science, Vol. 8384, 2014, pp. 724–733, doi: 10.1007/978-3-642-55224-3_68.



Łukasz Opioła received his Master's degree in computer science from the University of Science and Technology (AGH), Cracow, Poland in 2015. He is currently a Ph.D. student at the Faculty of Computer Science, Electronics and Telecommunication at AGH and an employee of the Academic Computer Centre CYFRONET-AGH. His research areas include data synchronization in distributed systems, as well as authentication and authorization infrastructures.

Łukasz Dutka has significant expertise in cloud systems, large-scale systems, development of application for business purposes, team and project management in commercial projects as well as EU IST projects. He received his Ph.D. in computer science from the AGH University of Science and Technology, Cracow, Poland. He has a longstanding experience with managing large development teams. His scientific interests include large-scale computer system, system architectures, component approaches. He is the author of a modern software development architecture called the Component-Expert Architecture combining expert systems with component architectures, with successful applications in commercial and scientific environments. He has actively participated in a number of EU funded projects including Indigo DataCloud, EGI Engage, Helix Nebula Science Cloud and many others. Currently he is the Technical Director of PL-GRID Plus project and leader of Onedata team.

Michał Wrzeszcz is a Ph.D. student of computer science at the AGH University of Science and Technology in Cracow, Poland and an employee of the Academic Computer Centre CYFRONET-AGH. He is the author or co-author of over 20 scientific papers and conference contributions. His research interests are transparent data access and distributed computing as well as artificial intelligence and social networks.

Renata Slota Ph.D., D.Sc., works at the Department of Computer Science of the AGH University of Science and Technology (AGH) in Cracow, Poland. She is the author or co-author of about 130 scientific papers. Topics of interest include parallel and distributed computing, distributed systems, grid and cloud environments, data management and storage systems, knowledge engineering. She has been involved in many national (recently: PL-Grid Core, PL-Grid NG) and international projects most notably in EU IST, recently: PaaSage, and VirtRoll. Among others, she worked on the development of Onedata and Scalarm systems. Member of the Program Committee of the International Conference on Computational Science (ICCS), and the International Conference on Parallel Processing and Applied Mathematics (PPAM). Reviewer of: Future Generation Computer Systems (FGCS), Computing and Informatics (CAI), and Computer Science (CSCI) journals. Currently, she is the Deputy Dean of the Faculty of Computer Science, Electronics and Telecommunication of AGH.

Jacek Kitowski is Full Professor of computer science and Head of Computer Systems Group at the Department of Computer Science of the AGH University of Science and Technology in Cracow, Poland and Head International Affairs at the Academic Computer Centre CYFRONET-AGH, responsible for international collaboration and for developing high-performance systems and grid/cloud environments. He is author or co-author of about 240 scientific papers. His topics of interest include large-scale computations, Grid services and Cloud computing, distributed storage systems, high availability systems, network computing, knowledge engineering. He is a member of program committees of many conferences, participant of many international and national projects, funded by the European Commission, European Defense Agency, Polish National Centre for Research and Development and Polish National Science Centre. Director of PLGrid Consortium running PLGrid e-infrastructure in Poland for scientific computing. Member of Ministry Expert Body for Scientific Investments.

Computing and Informatics, Vol. 37, 2018, 23–48, doi: 10.4149/cai_2018_1_23

DIFFUSION OF FALSE INFORMATION DURING PUBLIC CRISES: ANALYSIS BASED ON THE CELLULAR AUTOMATON METHOD

Xiaoxia Zhu, Jiajia Hao, Yuhe Shen
Tuo Liu, Mengmeng Liu

Economics Management School
Yanshan University
Western of Hebei Street No. 438
066004 Qinhuangdao, Hebei Province, China
e-mail: zhuxiaoxia@ysu.edu.cn

Abstract. The progress of false information diffusion in the public crisis is harmful to the society. When the public crisis occurs, the public respond in different ways and the public also want to tell others what they think right. But what they think is right is not recognized by the government. Thus the false information forms and it begins to diffuse. As the false information spreads, the harm to society magnifies gradually. Particularly in network society, false information diffusion can easily cause secondary hazards and accelerate public crises to a devastating degree. Thus intervening and controlling the false information diffusion is an important aspect of the public crisis management. From the perspective of the social network theory, this study analyzes the progress of false information diffusion in terms of different public crisis management strategies and presents the result of false information diffusion through simulation on cellular automaton of different public crisis management strategies. In simulations on cellular automaton, interventions are also carried to control false information diffusion and alternatives are proposed to help reduce public crises. This study also extends the theory of false information management, which is significant for the government to improve the ability to evaluate the false information and carry out interventions effectively to control the false information when it begins to diffuse.

Keywords: Cellular automaton, public crisis, false information, intervention mechanism, network interactions

Mathematics Subject Classification 2010: 37Fxx, 93Cxx

1 INTRODUCTION

1.1 Background

When public crises occur, they typically develop rapidly and in unexpected directions, leading to high uncertainty about future trends, and increasing public concern and interest. Indeed, in recent years, the rapid development of the Internet and growing number of Internet users through both traditional and mobile platforms has meant that more and more people are willing to participate in virtual discussions to express their views, exacerbating the upsurge of public opinion in addition to traditional channels.

When vast amounts of information are collected, discordant voices are inevitable, and this creates false information. The outbreak and evolution of recent public crises has notably been accompanied by the diffusion and dissemination of false information, which greatly increases the difficulty of managing such events. Examples of the diffusion of false information during public crises include the frenzy of buying radix during the SARS outbreak in 2003, the Wen'an incident in 2008, the panic to buy salt following the nuclear leak in Japan in 2011, and the DiaoYu Island incident in 2012. The diffusion of false information during the evolution of a public crisis not only increases public uncertainty, it also creates obstacles for the crisis recovery and may even endanger social and political stability. Therefore, the effective management of false information (FIM hereafter) during public crises has become an important element of research on public crisis management.

1.2 Related Research

Because the diffusion of false information such as spreading rumors in interpersonal networks is similar to the proliferation of a virus, most rumor spreading models are drawn from epidemic models. Rapoport and Rebhun [1] were the first to use an epidemic model to discuss information diffusion problems, while Goffman and Newill [2] also compared the spread of disease and rumors and used the stability of the infectious disease spread model to explain the final state of rumors. In the 1960s, Daley and Kendall [3] proposed a mathematical model for rumor spreading in which individuals are categorized based on their health status, infection status, and removed or immune status. Their model analyzes the rumor problem by using random processes and assumes that the probability of role conversion between the statuses above satisfies a certain mathematical distribution. Although this model does not fully comply with the actual diffusion of false information, it is a reasonable approximation under certain conditions.

In recent years, along with the in-depth study of dynamics, rumor spreading based on dynamics has drawn more scholarly attention. For instance, Dickinson et al. [4] was another author to compare the spread of infectious diseases with that of rumors in order to analyze and summarize the correlations in their communication processes. Similarly, Thompson [5], by using the Daley-Kendall model, took account

of the impact of the differences between susceptible persons and disseminators during the spread of rumors. Based on statistical data derived from MSN chatrooms, the author found that population activity is the most sensitive parameter that affects rumor spreading and that the number of communicators in the 18-34 age group is far more than those over 55 years old. In addition, this finding shows that increasing activity can control the size of rumors, but encouraging people to spread rumors faces many ethical issues. Indeed, according to simulation experiments based on the collected data, a second wave of rumors tends to spread, which will gradually become weaker and disappear.

Kawachi [6], by considering the different parties involved, established finite and infinite dimensional dynamical models and determined the threshold at which rumors spread. In the same vein, Lebensztayn et al. [7] discussed the final state of rumor spreading, but assumed differences in individual behavior (i.e., randomly deciding to spread rumors or not).

With a specific focus on the diffusion of false information in public crises, Zhang and Zhang [8] established an interactive model of rumor spreading and crisis management, finding that spreading rumors may have a negative effect on public crises and providing appropriate recommendations for FIM in such incidents. Huo et al. [9], who studied the spread of rumors after the occurrence of unexpected events and the effectiveness of the government response, used simulation data to draw a phase diagram of the system. Finally, Zhong et al. [10] divided information into real and false to discuss FIM from an ecological perspective, discovering that the diffusion of false information during public crises has different diffusion rates. Thereafter, Maki and Thomson [11], and Murray [12] have been carrying out studies using a mathematical model of rumors that focus on the theoretical analysis. Kawachi et al. [13] consider the impact of different contacts on the ultimate spread in another paper and used the mathematical modeling methods to explore the results of the final spread.

Zanette [14] studied the spread of rumors, based on the small-world networks he established a rumor spread model and drew some conclusions, including the critical value of the rumor spread. Moreno et al. [15] also established a rumor propagation model based on scale-free networks and compared conclusions obtained by random analysis method and by computer simulation. Chinese scholar Wang Xiaofan paid more attention to clustering coefficient of the network, and she found that increasing clustering coefficient of the network can effectively inhibit the spread of rumors. Since then, on the basis of two basic network models – small-world networks and scale-free networks, some other scholars have proposed many modified models of networks, such as Newman and Park proposed NP model, Boguna et al. proposed social distance model, Jin, Girvan and Newman et al. proposed JGN model, Jackson and Rogers proposed JR model, as well as Vazquez proposed CNN model. Seeger [16] used chaos theory to explain the deep complex problems of information dissemination in crisis and put forward some universal problems about the complexity of information dissemination in crisis, which is extremely important for us to understand the process of information spread in crisis essentially. Monge and Noshir [17] from the perspective of complex adaptive systems theory proposed a "multi-theory

and multi-level framework" model. They used mathematical modeling and computer simulation to study coevolution among variable parameters of crisis diffusion network and they also discussed the structural mechanisms of chaos emergence.

Meanwhile, on the basis of Zanette's [14] and Moreno's [15] work, many scholars extended study about the false information diffusion in public crisis. Pan et al. [18] from Shanghai Jiaotong University applied Moreno's [15] rumors spread model on the variable clustering coefficient scale-free networks, and studied rumors spread though scale-free networks. Their scale-free networks obey a power law degree distribution and have variable clustering coefficient. By changing the parameters of a special network clustering coefficient to observe changes of rumors spread, they came to the conclusion that spread rumors and clustering coefficient of network have a negative correlation. Chu and Tong [19] from the theoretical perspective of crisis diffusion, reviewing the latest theory of crisis diffusion, explored the chaotic prediction, co-evolution, complex adaptive systems and other complex features of crisis diffusion.

However, although rumor spreading models are based on mathematical principles that have a high degree of abstraction, rigorous logic and application breadth, describing the process of diffusion of false information is somewhat non-intuitive. In other words, although rumor spreading can be represented by a mathematical model, dynamics differential equations of infectious diseases is a mathematical model based on macroeconomic statistics where no microscopic mechanisms are used. So, such models cannot explain the process completely.

The cellular automaton is a discrete mathematical model, its physical image is clear, it is fully parallel without truncation errors, etc., such that in recent years it has become a powerful tool for exploring nonlinear complex systems. Xuan and Zhang [20] detailed the rumors diffusion model based on the cellular automata. Their model can reproduce the process of rumors spread through local interaction among individuals. The individuals to spread false information in public crisis are discrete and independent, false information diffusion in public crisis is scenario-dependent, so this article is based on the forecast method and uses cellular automata to build management simulation model of false information diffusion in public crisis. According to the thought on policy test, the designed test solution is converted to the corresponding code. The code is generated into the management simulation model. Different codes represent different management schemes. Thus the simulation model reflects phenomenologically the process of false information diffusion in public crisis.

2 RESEARCH HYPOTHESE

FIM during public crises is a process of repeatedly comparing goals and correcting deviations effectively. Further, the process of FIM during public crises consists of a series of interconnected, continuous management activities as well as discrete management activities that are time independent. Meanwhile, the interventions about false information in public crisis also are simple, complete, coordinate, open, adap-

tive, direct and indirect. Specifically, simplicity means that the relevant government department manages the false information unilaterally, whereas completeness means that various management policies, systems, and commands are used in the process of false information management without their own errors or major defects. In other words, the fewer the number of such defects or errors generated during implementation, the higher the degree of completeness of management's countermeasures; coordination means that the various strategies used to manage false information during public crises support and cooperate with each other without contradiction and conflict in management activities. In addition, open and adaptive management means that the initiatives of potential recipients can be mobilized. In other words, the ability to identify potential recipients can be controlled and improved.

To examine the diffusion of false information in different intervention scenarios during public crises, this study puts forward the following hypothese:

Hypothesis 1. In the process of FIM, coordinated management is more effective than a simple management strategy.

Hypothesis 2. Open, adaptive management during public crises can effectively prevent or even eliminate the diffusion of false information, but cannot control the diffusion direction of false information.

During public crises the frequency of correcting deviations represents management skills. In other words, in the process of FIM during public crises, the more (less) the corrective deviation, the stronger (weaker) are the skills to manage information. According to ascending order of performing frequency, the execution is divided into defensive, discrete and continuous. Hence:

Hypothesis 3. The greater the enforcement the better the false information in public crisis can be controlled.

Any management behavior can effectively correct the deviation to a certain extent and achieve management objectives. FIM assumes that an individual that has accepted false information provides false information, whereas an individual that has not accepted false information maintains the original state.

Hypothesis 4. The lower the acceptance probability of false information in public crisis the more easily the false information diffusion can be controlled.

3 MODEL DESIGN

3.1 Research Method

Cellular Automaton (CA) is used to simulate and predict the behavior of complex systems. CA was dating back to the study of Ulam and Von Neumann (the father of modern computers) in the 1940s. They found a simple iterative calculation algorithm could replace complex models to explain many phenomena in the nature.

CA uses some very simple local rules to effectively simulate the spatial pattern formation process of complex systems. In 1980s, Wolfram made a lot of contribution to the development of CA [21]. Now CA is used to simulate natural and artificial complex systems in physics, chemistry, biology, geography, and other fields [22, 23]. CA's "bottom-up" research philosophy fully reflects that in complex systems local individual behavior affects the overall order and direction so it is very suitable for CA to simulate and predict the formation of public opinion. At the same time, more and more scholars use CA to simulate and predict the geographic phenomenon. In 1980s, Couceleis used CA to simulate geographic phenomenon and got some important conclusions [24, 25, 26]. Some scholars also used CA to simulate urban sprawl [27], land use dynamic evolution [28], volcano spread [29], etc.

Wolfrarm [30] divided CA into four categories according to the dynamic behavior of cellular automaton:

1. Smooth type: from any initial state, after running a certain period of time, cellular space keeps a stabilized configuration, that is each cellular refers to a fixed state and it does not change with time.
2. Cycle type: after running a certain time, a series of simple cellular space tends to be a fixed structure or periodic structure.
3. Chaos type: from any initial state, after running a certain period of time, every cellular shows the chaotic non-periodic behavior.
4. Complex: local structure becomes complex, or partial chaos attends, some cellular will continue to spread.

Chaos theory is fundamentally linked to the management of false information in public crises [31]. As a complicated social phenomenon, the generation and diffusion of false information during public crises is essentially a kind of complex nonlinear evolutionary process, which has chaotic characteristics typical of complex systems, such as the evolution of diffusion randomness, the information variation of the butterfly effect, and the propagation path of fractal characteristics [32]. Chaos theory is an emerging science about nonlinear systems, and it is different from traditional scientific thoughts that explain social phenomena. It reveals both the unity and the opposition between inevitability and contingency and between disorder and order, while also providing two of the most basic complex research paradigms for FIM in public crises. These two paradigms, namely the simple-complex paradigm (simple system explaining complex behavior) and the complex-simple paradigm (complex system driven by simple rules), provide a new methodology to support the theory of FIM in public crises.

The chaos forecasting method based on chaos theory is a type of method for predicting nonlinear system evolution [33]. Chaos prediction mainly adopts bottom-up modeling ideas that assume that the evolution of complex systems is the result of many primitive interactive roles that aim to simulate macroscopic disorderly phenomena by using micro-level orderly rules. The basic idea is that under the assumed conditions, a micro-level dynamic mechanism (e.g., rule or policy) aggregates the

results of individual behavior, thus predicting complex scenarios (patterns) at the macro level. As a general modeling method, predicting chaotic situations focuses more on the application of chaos theory. Because it mainly explains and analyzes the formation and evolution of complex phenomena, it is widely used to simulate, evaluate, and imitate complex socioeconomic phenomena and formulate public management policy [34]. Further, compared with traditional mathematical models and simulation methods, the chaos forecasting method can easily describe interactions among elements. Although some complex phenomena may be difficult to analyze and express, it can thus accurately simulate them and their evolution. Therefore, it can vividly and truly reflect the detailed structure and pattern of a large number of individual interactions in order to predict the future prospects of the evolution of false information in public crises. In summary, the chaos forecasting method is an effective research tool for simulating FIM during public crises.

False information management in public crisis depends on situations. There exists a complex, dynamic relationship among Public crisis management strategies, executive skill and FIM performance. So it is very suitable to use Chaos forecasting method to simulate and imitate. In this paper, a management simulation model of false information diffusion in public crisis will be constructed, which is based on chaos forecasting method. According to the thought of policy test, the designed test schemes are converted into the corresponding code, this code is generated into the management simulation model, and we use this method to simulate evolution of the false information diffusion in public crisis phenomenologically.

3.2 Model Building

A management simulation model of the diffusion of false information during public crises is as follows:

1. Primitive and space (L_d): Space refers to the entire space in which false information during public crises spreads. In this space, each primitive is a potential recipient in the real world that is likely to accept false information during public crises. Each primitive has its own state and behavior, and it exchanges information with the external environment and other cellulars to update the status of the entire system.
2. Neighbors (N): This study uses the Moore type of neighbors. Specifically, the primitive makes eight adjacent primitives (three above, three below, one on each side) as its neighbors, as is shown in Figure 1. The neighboring state of each primitive depends on its own and on the surrounding neighbors' current states. As the first step in this paper we regard all neighbors' acceptance probability in the same way and that will be considered such as neighbors' acceptance probability depending on places, space, their characteristics and so on.
3. State (S): There are two kinds of primitive states, 0 and 1. The primitive state of 0 means that potential recipients refuse the false information, whereas 1 means that they accept the false information.

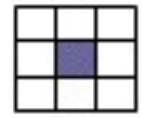


Figure 1. MOORE-type neighbors

4. Decision rules (f): A decision rule generalizes and abstracts potential recipients' behaviors and decisions when diffusing false information during public crises. The local decision rules are defined as follows:

   (a) has the false information been accepted by a neighbor;
   (b) if yes, then the individual decides whether to accept the false information according to a certain probability.

This study adopts a two-dimensional interface as a system space for the diffusion of false information during public crises. Hence, we assume that the false information diffuses from the center to the surrounding area. If a primitive accepts the false information being spread, the primitive's state is marked on the screen. The program flow is shown in Figure 2.

3.3 Research Plan

In this study, the influence of FIM strategies and management skills during public crises on the diffusion of false information is discussed. A public crisis management system is based on institutional systems, policy, and security. For example, the institutional system includes the relevant organizational settings, interdepartmental operations, functions, and coordination, while policy represents the policymaking decisions by various government departments. In addition, management strategies can be divided into

1. simple, complete management,
2. coordinated, direct management, and
3. open, adaptive management.

In the space of the diffusion of false information during public crises, these strategies correspond to

1. a whole line, when execute simple, complete management strategy, the number of affected individuals is that all individuals on a line in false information diffusion system space;
2. one ring (including a central point), when execute coordinated, direct management strategy, the number of affected individuals is that all individuals on one ring (including a central point) in false information diffusion system space, and

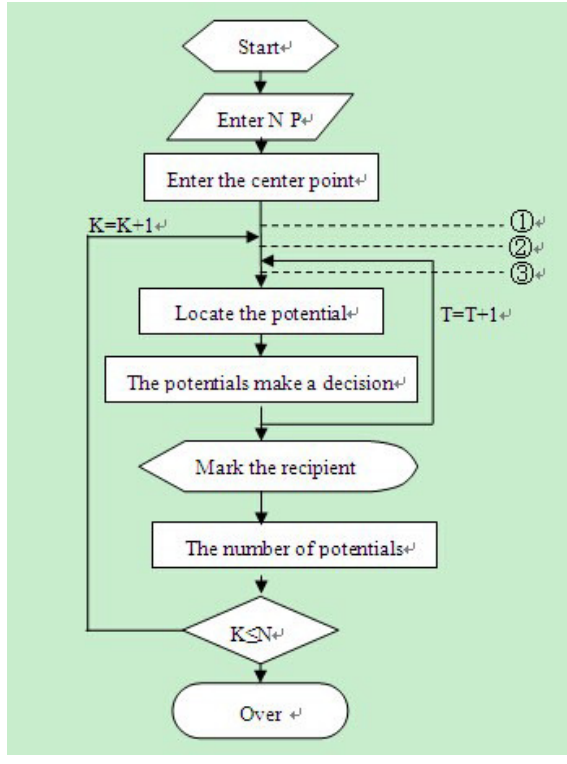


Figure 2. The process of fake information

3. several randomly distributed points, when execute open, adaptive management strategy, the number of affected individuals is several randomly distributed points in false information diffusion system space respectively.

In the simulation, we write the corresponding code of these three management strategies to make the potential recipient not accept the false information.

Player	1. Simple and Complete	2. Coordinate and Direct	3. Open and Adaptive
Defensive management	AI	BI	CI
Discrete management	AII	BII	CII
Sustainable management	AIII	BIII	CIII

Table 1. Simulation scheme of fake information management in public crisis

From the aspect of the simulation of management skills, ①②③ three insertion points are set to represent defensive management, discrete management, and sustainability management, respectively. From the perspective of the control func-

tion of management, and according to the different positions into which the code is inserted, these three cases may be constructed as feed forward control, feedback control, and field control, respectively. As for the difference between emergency management and crisis management, insertion point ① means simple precautions beforehand, insertion point ② represents a certain lag in the emergency management mode, and insertion point ③ represents the new public crisis management mode affected by non-linear thought (i.e., "try to do").

First, we assume that the probability for potential recipients to accept false information in public crisis is P and the total simulation time is N. When the total simulation time is beyond N, the simulation process is over. We use ①②③ three different interventions to impact the potential recipients of false information and ①②③ three interventions are set to represent defensive management, discrete management, and sustainability management, respectively. After impacted by interventions, the potential recipients of false information make decision whether accept the false information or not. And we calculate the number of the recipients of false information and mark them in the simulation process. If simulation time is less the total simulation time assumed at the beginning of the experiment, we continue to execute the interventions and calculate the number of the recipients. When the total simulation time is beyond N, the experiment is over.

The simulation process shows the different execution frequency of the code in each of these three locations. The code inserted into ① (simple precautions beforehand) only works at the beginning of the simulation, that inserted into ② (a certain lag in the emergency management mode) is executed N times throughout the process (import N in the program initialization), and that inserted into ③ (try to do) is executed $T \times N$ times throughout the process. In the simulation, the code of the corresponding management strategy is inserted into ①②③, respectively, which is used to simulate phenomenologically the evolution and management performance of the nine management strategies shown in Table 1.

There are 9 kinds of management simulation schemes, which are AI, AII, AIII, BI, BII, BIII, CI, CII and CIII in Table 1. Actually, the false information diffuses freely when the schemes AI, BI and CI are executed. So this article will not study these 3 schemes AI, BI and CI and only consider the other 6 schemes AII, AIII, BII, BIII, CII and CIII.

4 RESULTS AND ANALYSIS

In order to make general observations, the acceptance probability P of false information was set to 0.3. The simulation clock was set to 25 K and the simulation interface to 100×100 (i.e., the number of potential recipients is 10 000), while potential recipients of false information were defined as $(0, 0)$–$(100, 100)$ (the positive integer points out the coordinates in the two-dimensional space). Moreover, to simplify the simulation process, we ensured that:

1. The horizontal axis was X and vertical axis was Y, the X axis represents the simulation time and the Y axis represents the number of the false information receivers;
2. The source of false information during public crises was located at the point $(50, 50)$, namely the diffusion distance T was 50;
3. The evolution scheme in a circular area was centered on $(50, 50)$ with a radius as inspected. If the primitive in a circular area accepted the false information during public crises, the corresponding coordinates were marked and the number of recipients recorded.

The nine kinds of management simulations shown in Table 1 were conducted several times, changing the spread direction, speed, quantity, and spatial distribution of the receiver of the false information under these various scenarios. Then, the simulation results and management performance were observed and measured. Through these simulation tests, the most typical evolution distribution and quantity variation were recorded. The results of the simulations are summarized next.

In scheme A, the FIM function area was represented by the 25^{th} line. The simulation results show that scheme AII can delay the diffusion of false information, as illustrated in Figure 3. Although the spread speed from the center to the function area lags significantly behind that in the other directions, it is difficult to block the spread of false information effectively in terms of quantity, as shown in Figure 4. In this scenario, when $K = 10$, the diffusion of false information during public crises is affected by FIM. Because the number of receivers decreases, false information "bypasses" the region and continues to spread outward. Then, the number of receivers soars to more than 800, close to the number of false information receivers in the free diffusion scheme.

Having observed the diffusion of false information and quantity variation in scheme AIII, we know that scheme AIII effectively prevents false information from spreading outside the function area. As shown in Figure 5, all the parts on the left-hand side of the 25^{th} line are blank, while diffusion in the other direction is unaffected, meaning that the diffusion of false information during public crises is shared by function area. Moreover, in scheme AIII the number change curve of the false information receiver at $K = 10$ and $K = 15$ has clear fluctuations, but the overall curve retains a rapid growth trend. Indeed, the ultimate recipient number is below 700, fewer than the number of false information receivers in scheme AII.

Comparing the evolution of the distribution of false information during public crises and quantity change in both schemes allows us to conclude that scheme AIII is more effective at preventing false information from spreading outside the function area compared with scheme AII. Further, this scheme is more effective at reducing the number of false information recipients during public crises than scheme AIII, which suggests that administrative skills are important for the diffusion of false information during such events.

In scheme B, the layer 4 neighbors outside the center are used for the action area, namely the circle (marked with red in Figures 7 and 9) is made up of line 46,

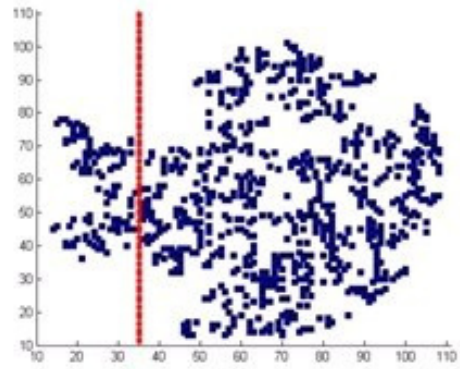


Figure 3. The evolution of distribution in AII scheme

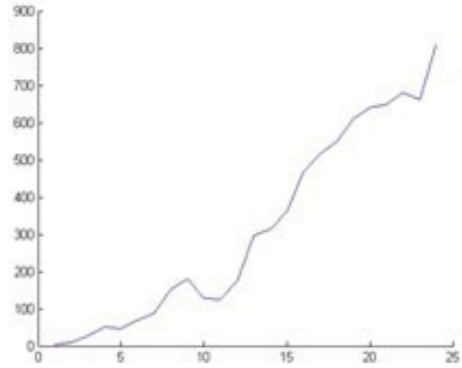


Figure 4. Changes of the number of recipients in AII scheme

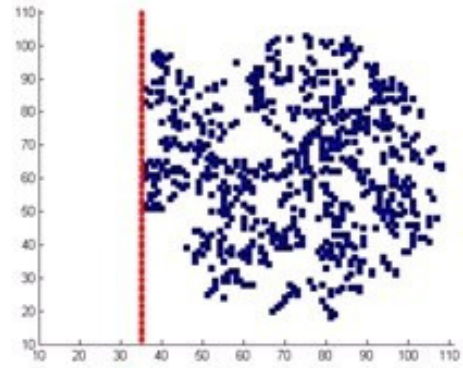


Figure 5. The evolution of distribution in AIII scheme

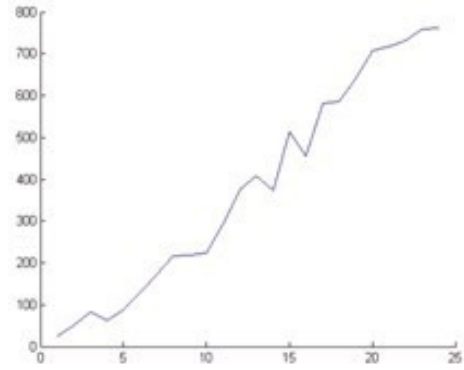


Figure 6. Changes of the number of recipients in AIII scheme

row 54, and columns 46 and 54. The simulation results show that scheme BII has a clear influence on the diffusion of false information during public crises, which mainly manifests in the following two aspects. First, compared with the free diffusion scheme, the diffusion speed of false information in each direction from the center to the surrounding no longer converges (high in some directions and low in others; Figure 7). Second, in terms of quantity changes, the change curve of the receiver number is coarser, with jagged peaks appearing many times, as shown in Figure 8. Further, at the beginning of the spread, the growth rate of false information recipients is far lower than that in the other scenarios (except for scheme BIII), while the final number of false information recipients is only 450, far below that in the free diffusion scenario, showing that scheme BII has more impact on the diffusion of false information during public crises.

In scheme BIII, false information is strictly controlled within the function areas (i.e., it cannot penetrate the area; Figure 9) and the number of false information recipients changes irregularly and randomly, as shown in Figure 10. This finding suggests that this scheme can effectively control and limit the spread direction of false information during public crises. Comparing the effects of schemes BIII and BII thus allows us to conclude that both influence false information at the beginning of the diffusion process. In other words, in the early stages, the gap in the number change is unclear, but the ultimate evolution results are different because of the differences in management skills. Hence, the former continue to spread out to reach 450, while the latter are strictly limited in function area and the largest number of recipients is only 15. This finding shows that FIM during public crises must receive constant attention and that FIM skills must not be lax in order to effectively prevent and eliminate the diffusion of false information.

In scheme C, the random distribution rate (pp) is set to 0.01, 0.05, and 0.1 in order to examine the sensitivity of the diffusion of false information during public crises to the random distribution. The simulation results show that in scheme CII, as shown in Figures 11, 13, 15, with an increasing random distribution rate, the false

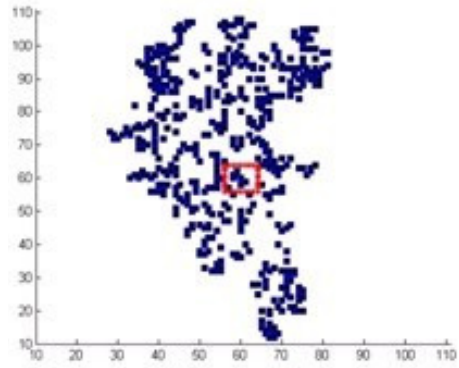


Figure 7. The evolution of distribution in BII scheme

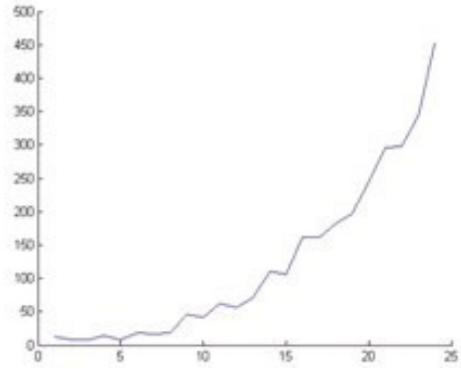


Figure 8. Changes of the number of recipients in BII scheme

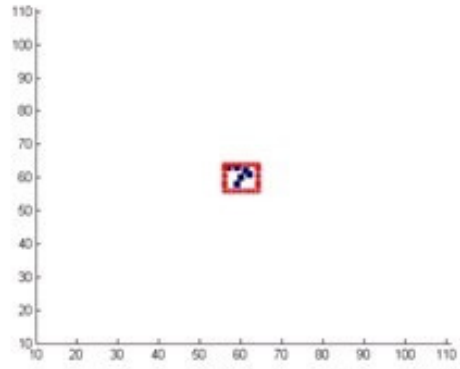


Figure 9. The evolution of distribution in BIII scheme

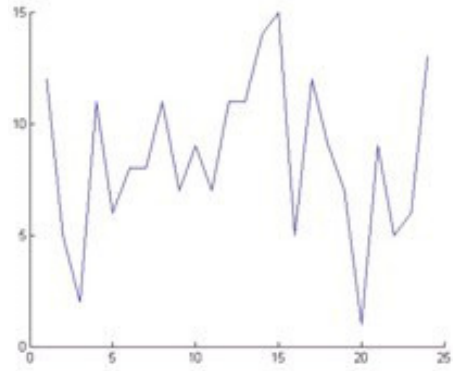


Figure 10. Changes of the number of recipients in BIII scheme

information spread becomes more irregular and the diffusion speed gradually slows down. Thus, the growth rate and number of false information recipients during public crises gradually diminishes, too. As presented in Figures 12, 14, 16, the quantity change curve shows that when K = 15 in all three schemes, the number of recipients is more than 300, fewer than 300, and close to 200, respectively. By contrast, when K = 24 in these three schemes, the number of recipients decreases greatly to more than 600, just fewer than 600, and just over 500, respectively.

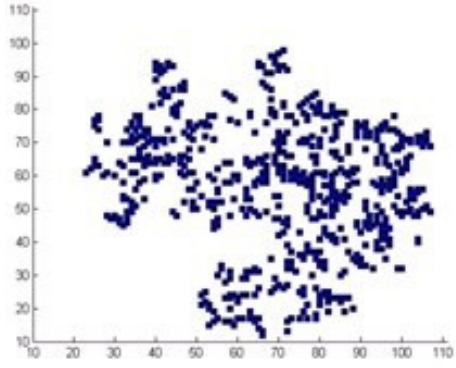


Figure 11. Evolution distribution of spread when pp = 0.01 in CII

The test results of scheme CIII suggest that an increase in the random distribution rate makes it more and more difficult to spread false information during public crises. Compared with scheme CII when pp = 0.01, the diffusion of false information during public crises in scheme CIII when pp = 0.01 is very slow. In addition, its number change curve fluctuates and the receiver number is only about 250 at the end, far less than in scheme CII (pp = 0.01). When pp = 0.05, the spread of false information is impacted strongly: only the individual points become false

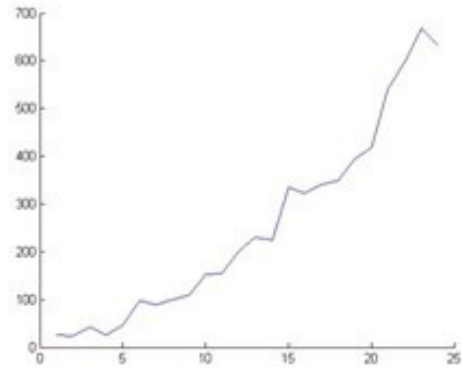


Figure 12. Changes of the number of recipients in CII scheme when pp = 0.01

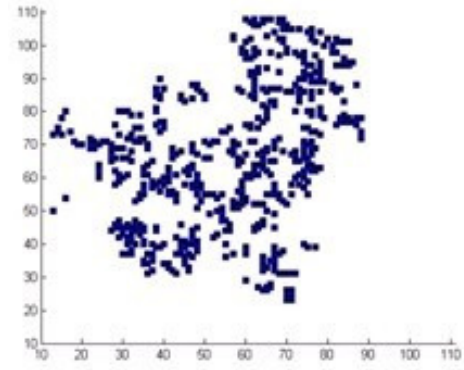


Figure 13. Evolution distribution of spread when pp = 0.05 in CII

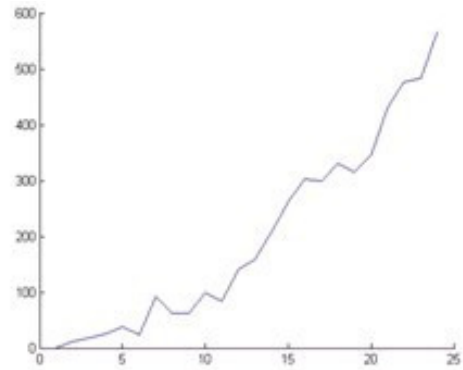


Figure 14. Changes of the number of recipients in CII scheme when pp = 0.05

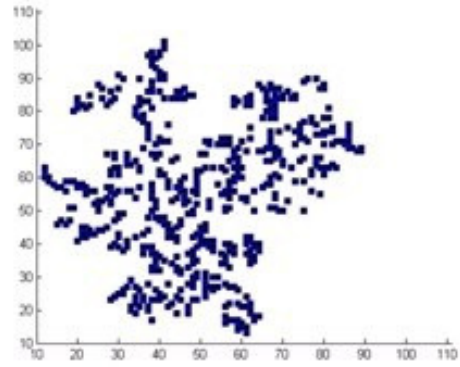


Figure 15. Evolution distribution of spread when pp = 0.1 in CII

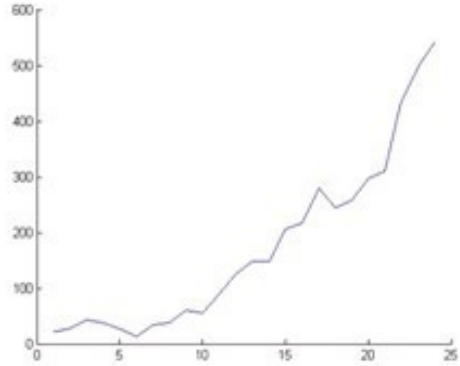


Figure 16. Changes of the number of recipients in CII scheme when pp = 0.1

information recipients (as shown in Figure 18), the changes in number tend to be smooth, and the largest number of receivers is only 12 (as shown in Figure 21). When pp = 0.1, false information is rarely able to spread (the center is just selected as the initial random distribution point). This finding indicates that this scheme can prevent the generation of false information to a certain extent. Even if false information somehow manages to spread (as shown in Figure 19), the number of false information receivers in this scheme is extremely low, where the largest number of false information receivers is only three (as shown in Figure 22).

These simulation results show that the adaptive and sustainable management of false information during public crises can control its spread in a timely and effective manner. In addition, the higher the openness and adaptability, the lower is the diffusion speed of false information during public crises. When openness and adaptability occur to a certain extent, such as mobilizing 0.01 of the public to participate in the prevention and control of false information during public crises (pp = 0.01),

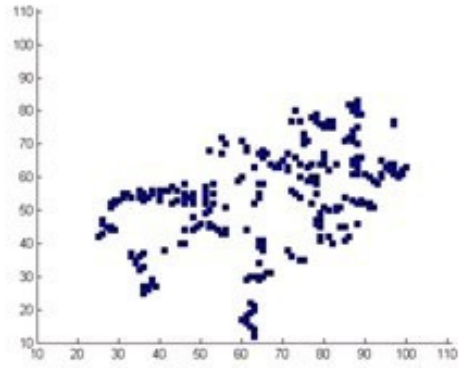


Figure 17. Evolution distribution of spread when pp = 0.01 in CIII

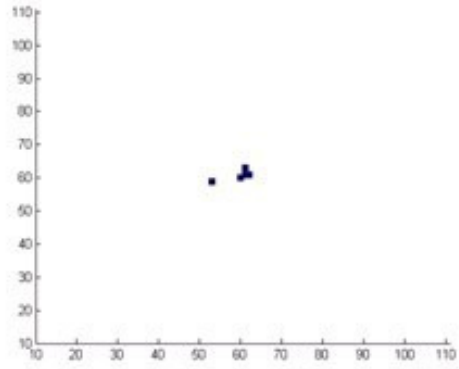


Figure 18. Evolution distribution of spread when pp = 0.05 in CIII

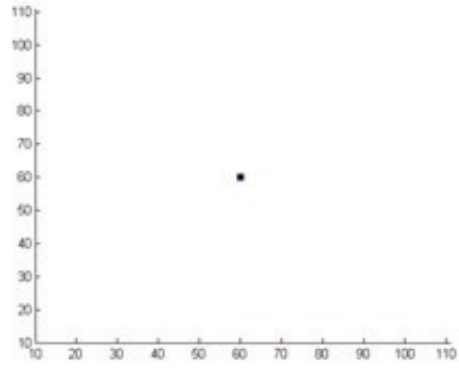


Figure 19. Evolution distribution of spread when pp = 0.1 in CIII

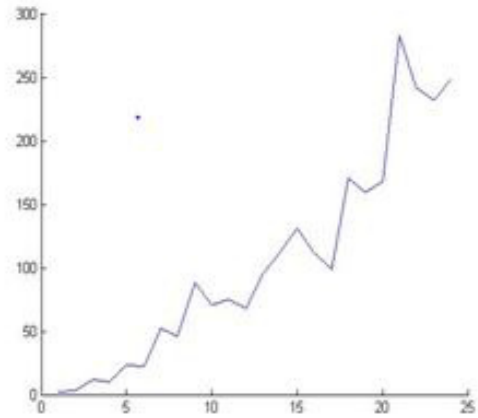


Figure 20. Changes in the number of recipients in scheme CIII when pp = 0.01

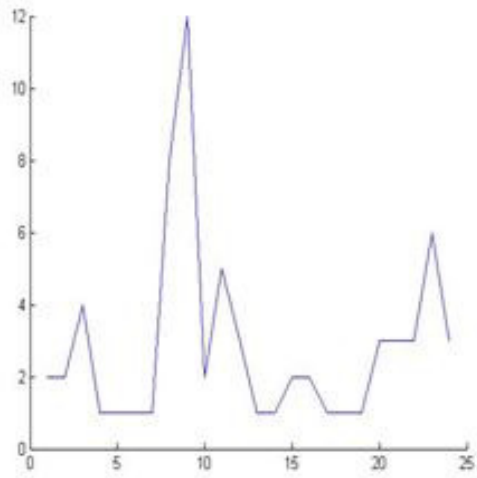


Figure 21. Changes in the number of recipients in scheme CIII when pp = 0.05

the proliferation and spread of false information can be inhibited significantly. This result has important managerial value for FIM during public crises, suggesting that nongovernmental organizations and their members should actively join FIM during such events. When openness and adaptability improve even further (pp = 0.05), a repressive influence on the diffusion of false information appears.

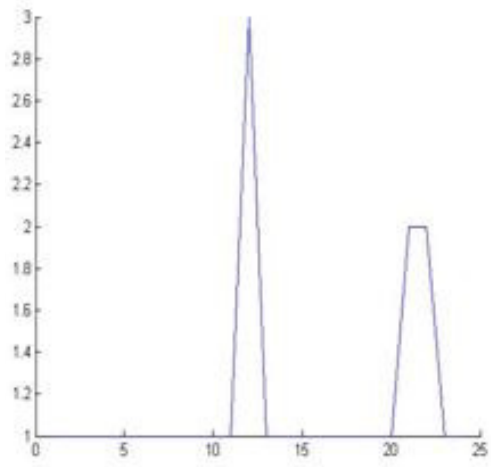


Figure 22. Changes in the number of recipients in scheme CIII when pp = 0.1

5 CONCLUSIONS

From the perspective of cellular automata method, this study analyzed the diffusion of false information in terms of public crisis management, drawing the following four main conclusions.

First, coordinated management strategy is more effective than simple management strategy, which confirms the hypothesis 1 as shown in Figures 4 and 6. Because in the coordinated management mode, the direction and quantity of the diffusion of false information can be effectively controlled, whereas simple management can only prevent the diffusion of false information in a certain direction. The diffusion speed of false information during public crises is very high, which is harmful for political and even social stability. False information diffusion sources are diverse and uncertain, the distribution of potential is irregular. A department only knows a little information about the false information so it is beyond its power to control the false information diffusion process from all aspects. The coordinated management among many departments can control the false information diffusion process better from many aspects because every department knows a lot different information and possess a variety of human and material resources. Therefore, FIM during public crises should pay more attention to coordinating the various countermeasures in addition to their performance and capabilities. Coordination is not only among staff but also among resources of various departments. The crisis education and training should be often carried out to enhance the awareness of cooperation among various departments. To make sure resources are shared among different departments, related departments should establish a unified command center to allocate resources reasonably.

Second, the open, adaptive management during public crises can effectively prevent or even eliminate the diffusion of false information, but cannot control the diffusion direction of false information, which confirms the hypothesis 2 as shown in Figures 12, 14, 16. Because with the increasing of random distribution probability, the false information diffusion rate becomes quicker and quicker and the number of infections becomes larger and larger. But, the diffusion direction remains unchanged. The false information diffusion resources and the potential recipients are dependent individuals in real life, those dependent individuals are easier to be impacted by other individuals who are same with the dependent individuals in some aspects. So the public and the influential in group are the key for the government to control the false information in public crisis. But distribution of the public and the influential in group is irregular, it is difficult to control the false information diffusion direction. This finding implies that management organizations should consider countermeasures to mobilize the public and improve individual and group initiatives in order to increase active participation in preventing and controlling false information.

Third, among the various kinds of schemes examined here, sustainable management was found to be superior to discrete management which confirms the hypothesis 3 as shown in Figures 5, 6, 9, 10 and 17, 18, 19, 20, 21, 22. Because sustainable management can affect both the direction and the speed of the diffusion of false information during public crises as well as the number of false information receivers to a significant degree, while the effects of discrete management are relatively weak in all these aspects. Discrete management strategy can only control the false information for a short time in some degree, which is harmful to the society in a long run. Sustainable management strategy can control the whole process of the false information diffusion for a long time from the beginning to the end. Thus the government has a chance to know more about the false information diffusion mechanism and gain experience from the feedback. This finding suggests that in the process of FIM during public crises, FIM skills and the execution of various management strategies must be strengthened and the consistency and continuity of management measures and countermeasures in each stage should receive enough attention. It will take some time to organize stuff and resources to control crisis. Relevant departments establish special emergency response teams in advance. Emergency response teams can take actions to control the situation immediately when crisis occurs which can reduce losses caused by lag.

Finally, the most fundamental and effective management method is reducing the probability of accepting false information during public crises, which confirms the hypothesis 4 as shown in Figure 17, 18, 19 and 20, 21. Because the false information diffusion range and direction is smaller and controlled better, the number of false information recipients is lower. The public is the subject to accept and diffuse the false information, when they refused to accept the false information, grapevine and other unknown information, the source of infection and the transmission of infection are well controlled and even the susceptible population. So reducing the acceptance probability of false information is the key to solve the problem. If this probability cannot be reduced, using a timely adaptive management strategy can effectively

control information diffusion despite incurring high workload and management costs. In addition, targeted measures to isolate information can control the number of false information receivers and diffusion scope during public crises, although this is often difficult in practice. Therefore, combining several programs could be considered. When crisis occurs, the official media should release timely and accurate information on the status of the crisis continuously to strengthen the authority. When faced with gossip and news released by official media, the public are more willing to be convinced by official news.

FIM during public crises is complex, involving sociology, management, psychology, communication, information science, political science, informatics, linguistics, anthropology, and public relations among many other fields. In future researchers will study the diffusion of false information in public crises from other aspects, such as type, sources, lifecycle and so on. In more complex evolution schemes, how to combine management strategy with management skills more effectively to control the diffusion of false information during public crises should also be studied more in-depth.

Acknowledgements

The authors would like to thank for the support from the National Natural Science Foundation of China under Grants 71301140, 71171174, and 71271187. The authors are also thankful for the support of the Foundation of Hebei Educational Committee under Grant QN20131090.

Fund assistance: Study of diffusion mechanism and control of false information in public crisis of National Natural Science Fund (71301140); Natural Science Fund of Hebei Province (G2015203425); Hebei higher school science and technology research project "Mechanism and Evolution of False Information Intervention Study in Public Emergency" (QN20131090); Electronic commerce and supply chain key laboratory open fund project in Chongqing (1456027).

REFERENCES

[1] Rapoport, A.—Rebhun, L. I.: On the Mathematical Theory of Rumor Spread. The Bulletin of Mathematical Biophysics, Vol. 4, 1952, No. 14, pp. 375–383, doi: 10.1007/BF02477853.

[2] Goffman, W.—Newill, V. A.: Generalization of Epidemic Theory. Nature, Vol. 204, 1964, No. 4955, pp. 225–228.

[3] Daley, D. J.—Kendall, D. G.: Stochastic Rumours. IMA Journal of Applied Mathematics, Vol. 1, 1965, No. 1, pp. 42–55, doi: 10.1093/imamat/1.1.42.

[4] Dickinson, R. E.—Pearce, C. E. M.: Rumours, Epidemics, and Processes of Mass Action: Synthesis and Analysis. Mathematical and Computer Modelling, Vol. 38, 2003, No. 11-13, pp. 1157–1167, doi: 10.1016/S0895-7177(03)90116-6.

[5] Thompson, K.—Castro Estrada, R.—Daugherty, D.—Cintrón-Arias, A.: A Deterministic Approach to the Spread of Rumors. Technical Report BU-1642-M, Cornell University, 2003.

[6] Kawachi, K.: Deterministic Models for Rumor Transmission. Nonlinear Analysis: Real World Applications, Vol. 9, 2008, No. 5, pp. 1989–2028.

[7] Lebensztayn, E.—Machado, F. P.—Rodríguez, P. M.: On the Behaviour of a Rumour Process with Random Stifling. Environmental Modelling and Software, Vol. 26, 2011, No. 4, pp. 517–522, doi: 10.1016/j.envsoft.2010.10.015.

[8] Zhang, Z.—Zhang, Z.: An Interplay Model for Rumour Spreading and Emergency Development. Physica A: Statistical Mechanics and Its Applications, Vol. 388, 2009, No. 19, pp. 4159–4166.

[9] Huo, L. A.—Huang, P. Q.—Fang, X.: An Interplay Model for Authorities' Actions and Rumor Spreading in Emergency Event. Physical A: Statistical Mechanics and Its Applications, Vol. 390, 2011, No. 20, pp. 3267–3274.

[10] Zhong, Q.—Qi, W.—Zhang, L.: Social-Pattern Crisis Information Diffusion Model under Lotka-Volterra System. System Engineering Theory and Practice, Vol. 32, 2012, No. 1, pp. 104–110 (in Chinese).

[11] Maki, D.—Thomson, M.: Mathematical Models and Applications. Prentice-Hall, Englewood Cliff, New Jersey, 1973.

[12] Murray, D.: Mathematical Modeling in Epidemiology. Springer, Berlin, 1980.

[13] Kawachi, K.—Seki, M.—Yoshida, H.—Otake, Y.—Warashina, K.—Ueda, H.: A Rumor Transmission Model with Various Contact Interactions. Journal of Theoretical Biology, Vol. 253, 2008, No. 1, pp. 55–60, doi: 10.1016/j.jtbi.2007.11.024.

[14] Zanette, D. H.: Dynamics of Rumor Propagation on Small-World Networks. Physical Review Letters, Vol. 65, 2002, No. 4, Art. No. 041908, doi: 10.1103/PhysRevE.65.041908.

[15] Moreno, Y.—Nekovee, M.—Pacheco, A. F.: Dynamics of Rumor Spreading in Complex Networks. Physical Review E, Vol. 69, 2004, No. 6, Part 2, pp. 66–130, doi: 10.1103/PhysRevE.69.066130.

[16] Seeger, M. W.: Chaos and Crisis: Propositions for a General Theory of Crisis Communication. Public Relations Review, Vol. 28, 2002, No. 4, pp. 329–337, doi: 10.1016/S0363-8111(02)00168-6.

[17] Monge, P. R.—Contractor, N.: Theories of Communication Networks. Oxford University Press, NY, 2003.

[18] Pan, Z.—Wang, X.—Li, X.: Simulation of Rumors Spread on the Variable Clustering Coefficient Scale-Free Networks. Journal of System Simulation, Vol. 18, 2006, No. 8, pp. 2346–2348 (in Chinese).

[19] Chu, J.—Tong, S.: Build the Public Crisis Management System Based on CAS and Analysis the Complexity of the Public Crisis Management System. The First Session of the China Management Annual Meeting, Beijing, 2006 (in Chinese).

[20] Xuan, H.—Zhang, F.: Simulation of Complex Systems and Applications. Tsinghua University Press, BeiJing, 2008.

[21] Wolfram, S.: Cellular Automata as Models of Complexity. Nature, Vol. 311, 1984, pp. 419–424, doi: 10.1038/311419a0.

[22] Goles, E.: Cellular Automata, Dynamics and Complexity. In: Manneville, P., Boccara, N., Vichniac, G. Y., Bidaux, R. (Eds.): Cellular Automata and Modeling of Complex Physical Systems. Springer-Verlag Berlin, Heidelberg, Springer Proceedings in Physics, Vol. 46, 1990, pp. 10–20, doi: 10.1007/978-3-642-75259-9_2.

[23] Binder, P.: Evidence of Lagrangian Tails in a Lattice Gas. In: Manneville, P., Boccara, N., Vichniac, G. Y., Bidaux, R. (Eds.): Cellular Automata and Modeling of Complex Physical Systems. Springer-Verlag Berlin, Heidelberg, Springer Proceedings in Physics, Vol. 46, 1990, pp. 155–160, doi: 10.1007/978-3-642-75259-9_14.

[24] Couclelis, H.: Cellular Worlds: A Framework for Modeling Micro-Macro Dynamics. Environment and Planning A: Economy and Space, Vol. 17, 1985, No. 5, pp. 585–596.

[25] Couclelis, H.: Of Mice and Men: What Rodent Populations Can Teach Us About Complex Spatial Dynamics. Environment and Planning A: Economy and Space, Vol. 20, 1988, No. 1, pp. 99–109.

[26] Couclelis, H.: Macrostructure and Microbehavior in a Metropolitan Area. Environment and Planning B: Urban Analytics and City Science, Vol. 16, 1989, No. 2, pp. 141–154.

[27] Batty, M.—Xie, Y.: From Cells to Cities. Environment and Planning B: Urban Analytics and City Science, Vol. 21, 1994, No. 7, pp. 531–548.

[28] White, R.—Engelen, G.: Cellular Automata and Fractal Urban Form: A Cellular Modelling Approach to the Evolution of Urban Land-Use Patterns. Environment and Planning A: Economy and Space, Vol. 25, 1993, No. 8, pp. 1175–1199.

[29] Clarke, K. C.—Brass, J. A.—Riggan, P. J.: A Cellular Automata Model of Wildfire Propagation and Extinction. Photogrammetric Engineering and Remote Sensing, Vol. 60, 1994, No. 11, pp. 1355–1367.

[30] Wolfram, S.: Statistical Mechanics of Cellular Automata. Reviews of Modern Physics, Vol. 55, 1983, No. 3, pp. 601–644, doi: 10.1103/RevModPhys.55.601.

[31] Fu, Y. W.—Liu, T.—Zhu, F. G.: Research on the Application Background of Chaos Theory in Public Crisis Management. Journal of Modern Management Science, 2008, No. 2, pp. 7–9 (in Chinese).

[32] Yang, H. Y.: The Research on the of Festival Interpersonal Communication in Mass Media Age – SMS Communication. Master's Thesis, Sichuan University, Chengdu, 2006 (in Chinese).

[33] Liu, H.: Chaos Theory Principle and Method for Prediction of the Economic System. Science Press, Beijing, 2003 (in Chinese).

[34] Xuan, H.—Gao, B.: Management and Social Economy System Simulation. Wuhan University Press, Wuhan, 2002 (in Chinese).

Xiaoxia Zhu received her Ph.D. degree in management science and engineering from Harbin Engineering University, China in 2008. She is currently Associate Professor at the Economics and Management School at Yanshan University, China. Now she is working for the Ministry of Industry and Information Technology as Economic Analysts. Her main research interests include complex networks, information dissemination and data mining.

Jiajia Hao is pursuing her Master's degree in management science and engineering in Yanshan University. Her main research interests include complex networks and crisis management.

Yuhe Shen works for The State Radio Monitoring-Center. Her main research interests include complex networks, and crisis management.

Tuo Liu received his Ph.D. degree in management science and engineering from Harbin Engineering University, China in 2009. He is currently Senior Engineer in the State Grid Energy Research Institute, China. His main research interests include crisis management and energy economics.

Mengmeng Liu is pursuing her Master's degree in management science and engineering in Yanshan University. Her main research interests include complex networks and crisis management.

Computing and Informatics, Vol. 37, 2018, 49–75, doi: 10.4149/cai_2018_1_49

RANKING-BASED DIFFERENTIAL EVOLUTION FOR LARGE-SCALE CONTINUOUS OPTIMIZATION

Li Guo

School of Economics and Management
China University of Geosciences
Wuhan, 430074, China
e-mail: guoli_cn@163.com

Xiang Li, Wenyin Gong

School of Computer Science
China University of Geosciences
Wuhan, 430074, China
e-mail: {lixiang, wygong}@cug.edu.cn

Abstract. Large-scale continuous optimization has gained considerable attention in recent years. Differential evolution (DE) is a simple yet efficient global numerical optimization algorithm, which has been successfully used in diverse fields. Generally, the vectors in the DE mutation operators are chosen randomly from the population. In this paper, we employ the ranking-based mutation operators for the DE algorithm to improve DE's performance. In the ranking-based mutation operators, the vectors are selected according to their rankings in the current population. The ranking-based mutation operators are general, and they are integrated into the original DE algorithm, GODE, and GaDE to verify the enhanced performance. Experiments have been conducted on the large-scale continuous optimization problems. The results indicate that the ranking-based mutation operators are able to enhance the overall performance of DE, GODE, and GaDE in the large-scale continuous optimization problems.

Keywords: Differential evolution, ranking-based mutation, vector selection, large-scale continuous optimization

1 INTRODUCTION

During the last few decades, evolutionary algorithms and metaheuristics have been successfully used for the optimization problems. However, they are mainly applied for the low- or moderate-dimensional problems. Since there are many real-world problems (such as neural network training, bio-computing, etc.) that have large problem size, in recent years, large-scale continuous optimization has gained more attention [17, 31, 37, 18, 16].

Differential evolution (DE), which was proposed by Storn and Price in 1995 [27, 28], is a simple and powerful evolutionary algorithm for global optimization. Due to its simplicity, robustness, ease of use, and efficiency, DE has obtained many successful applications in diverse fields, such as data mining, engineering design, geophysical inversion, and so on [22, 14]. More details on the state-of-the-art research within DE can be found in two surveys [20] and [5] and the references therein.

In the original DE algorithm, the core operator is the *differential* mutation, and generally, the parents in the mutation are always randomly chosen from the current population. For example, in the classical "DE/rand/1" mutation, three parent vectors $\mathbf{x}_{r_1}$, $\mathbf{x}_{r_2}$, and $\mathbf{x}_{r_3}$ are selected randomly from the current population. The indexes r_1, r_2, and r_3 satisfy $r_1, r_2, r_3 \in [1, NP]$ and $r_1 \neq r_2 \neq r_3 \neq i$. Since the parent vectors in the mutation are selected randomly, it may lead to DE be good at exploring the search space and locating the region of global minimum, but be slow at exploitation of the solutions [21]. Based on this motivation, in this paper, we modify our previous proposed ranking-based mutation operators [11] to enhance the exploitation ability of DE and employ it for the large-scale continuous optimization problems.

In the proposed ranking-based mutation operators, each parent vector has a selection probability, which is calculated according to its ranking in the population. Then, the parent vectors in the mutation are proportionally selected based on the selection probabilities. The major advantage of our proposed ranking-based mutation operators is that they are very simple and do not introduce any new parameters at all. In addition, the ranking-based mutation operators are general, they can be easily incorporated into most of existing DE variants. In this paper, they are integrated into the original DE algorithm, GODE [32], and GaDE [36] to verify the enhanced performance. Experiments have been conducted on the large-scale continuous optimization problems. The results indicate that the ranking-based mutation operators are able to enhance the overall performance of DE, GODE, and GaDE in the large-scale continuous optimization problems.

The rest of this paper is organized as follows. In Section 2, we briefly introduce the related work, including the DE algorithm and large-scale optimization in DE. Section 3 describes the ranking-based mutation operators for the DE algorithm in detail. The experimental results and analysis are shown in Section 4. Finally, in Section 5, we draw the conclusions from this work. In addition, the detailed experimental results of rank-DE, rank-GODE, and rank-GaDE are described in Appendix A.

2 RELATED WORK

Without loss of generality, in this work, we consider the following numerical optimization problem:

$$\text{Minimize} \quad f(\mathbf{x}), \qquad \mathbf{x} \in S \tag{1}$$

where $S \subseteq \mathbb{R}^D$ is a compact set, $\mathbf{x} = [x_1, x_2, \ldots, x_D]^T$, and D is the dimension, i.e. the number of decision variables. Generally, for each variable x_j, it satisfies a boundary constraint, such that:

$$\underline{x}_j \leq x_j \leq \overline{x}_j, \qquad j = 1, 2, \ldots, D \tag{2}$$

where $\underline{x}_j$ and $\overline{x}_j$ are respectively the lower bound and upper bound of x_j.

2.1 Differential Evolution

Similar to other evolutionary algorithms, differential evolution, which is mainly used for the numerical optimization problems, is a population-based optimization algorithm. The population consists of NP vectors. Each vector $\mathbf{x}_i$, $i = 1, \ldots, NP$ is initialized within the boundary. There are three operators in the DE algorithms, i.e. differential mutation, crossover, and selection. DE creates new candidate solutions through the differential mutation and crossover operations. The selection is applied between the target solution and its corresponding trial solution, and a candidate replaces the parent only if it has an equal or better fitness value. The pseudocode of the original DE algorithm is shown in Algorithm 1, where D is the number of decision variables; NP is the population size; F is the mutation scaling factor; CR is the crossover rate; $x_{i,j}$ is the j^{th} variable of the solution $\mathbf{x}_i$; $\mathbf{u}_i$ is the offspring. The function rndint$(1, D)$ returns a uniformly distributed random integer number between 1 and D, while rndreal$[0, 1)$ gives a uniformly distributed random real number in $[0, 1)$. $\langle\cdot\rangle_D$ is the modulo operation with divisor D. In Algorithm 1, the "DE/rand/1/exp" is illustrated, since the exponential crossover obtains very promising results in large-scale optimization. The binomial crossover and other mutation operators can be found in [22]. As for the terminal conditions, we can either fix the maximum number of fitness function evaluations (Max_NFFEs) or define a desired solution value-to-reach (VTR).

2.2 Large-Scale Optimization in DE

Many real-world problems can be formulated as numerical optimization problems, and many of them are large-scale, such as bio-computing, data mining, neural network training, etc. [18]. Due to the importance of the large-scale optimization, using the evolutionary algorithms and metaheuristics for the large-scale continuous optimization problems has gained considerable attention in recent years, such as the special sessions in conference [30, 29] and special issue in journal [12].

Algorithm 1 The DE algorithm with "DE/rand/1/exp"

```
Generate the initial population randomly
Evaluate the fitness for each individual in the population
while the stop criterion is not satisfied do
  for i = 1 to NP do
    Select uniform randomly r1 ≠ r2 ≠ r3 ≠ i
    v_i = x_{r1} + F · (x_{r2} − x_{r3})
    u_i = x_i
    j_rand = rndint(1, D)
    u_{i,j_rand} = v_{i,j_rand}
    L = 0
    while rndreal[0, 1) < CR and L < D do
      j_rand = ⟨j_rand + 1⟩_D
      L = L + 1
      u_{i,j_rand} = v_{i,j_rand}
    end while
  end for
  for i = 1 to NP do
    Evaluate the offspring u_i
    if f(u_i) is better than or equal to f(x_i) then
      Replace x_i with u_i
    end if
  end for
end while
```

Since DE has obtained very promising performance in the numerical optimization [5], many researchers employed it for the large-scale continuous optimization recently. Yang et al. [34] presented two DE algorithms based on the cooperative coevolution framework for large-scale optimization problems. Later on, in order to handle the high-dimensional nonseparable problems, they extended their work in [34] and proposed a new cooperative coevolution framework [35], where the random grouping scheme and adaptive weighting are introduced. In [19], Muelas et al. proposed a hybrid memetic algorithm based on DE for large-scale optimization problems. Brest et al. [4] presented a self-adaptive DE, jDElsgo, on large-scale optimization. In [32], the authors presented a neighborhood search based sequential DE for the CEC2010 Special Session on Large Scale Global Optimization. Stanarevic [26] hybridized the artificial bee colony with DE for the large scale optimization problems.

Recently, in the special issue of Soft Computing on the large-scale continuous optimization, there are seven papers related to the DE algorithm [18]. Brest and Maučec proposed jDElscop [3], where parameter self-adaptation, three strategies, and a population size reduction mechanism are combined. In [10], García-Martínez et al. proposed the role differentiation mechanism and malleable mating for DE. The

role differentiation mechanism differentiates the DE population into four groups, i.e., *receiving*, *placing*, *leading*, and *correcting* groups. The malleable mating ensures some similarity relations between chosen vectors. LaTorre et al. [15] proposed a memetic algorithm that combines the explorative and exploitative strength of differential evolution and MTS-LS1. In addition, the multiple offspring sampling framework has also been used in the hybrid memetic algorithm. Wang et al. [32] presented an improved DE algorithm based on generalized opposition-based learning (GOBL) for high dimensional optimization problems, where the opposition-based population initialization and generation jumping are applied with GOBL. SOUPDE, proposed by Weber et al. [33], is a shuffle or update parallel DE, where a structured population algorithm characterized by sub-populations is employed. Based on the analysis of the similarities and pitfalls of existing parameter adaptation techniques in DE, Yang et al. [36] proposed a generalized parameter adaptation method in DE for large-scale optimization problems. In [39], the authors presented the SaDE-MMTS algorithm to solve large-scale continuous optimization problems. In SaDE-MMTS, the strategy adaptation along with control parameter values presented in SaDE [23], the JADE mutation strategy [38], and the modified multi-trajectory search (MMTS) algorithm are hybridized.

3 RANKING-BASED DE

In this work, we modified our previous proposed ranking-based mutation operators in [11] and combine them with the original DE algorithm, GODE [32], and GaDE [36] to improve their performance on the large-scale continuous optimization problems.

3.1 Ranking-Based Mutation

3.1.1 Rankings Assignment

In order to utilize the information of good vectors in the DE population, in this work, we assign a ranking for each vector according to its fitness. Firstly, the population is sorted in ascendent order (i.e., from the best to the worst) based on the fitness of each vector. Then, the ranking of a vector is assigned as follows:

$$R_i = NP - i, \quad i = 1, 2, \ldots, NP \tag{3}$$

where NP is the population size. According to Equation (3), the best vector in the current population will obtain the highest ranking.

3.1.2 Selection Probability

After assigning the ranking for each vector, the selection probability p_i of the i^{th} vector $\mathbf{x}_i$ is calculated based on the quadratic model as follows:

$$p_i = \left(\frac{R_i}{NP}\right)^2. \tag{4}$$

Different models can be used to calculate the selection probabilities, and they may lead to different selection pressure on the better solutions. Note that, in this work, the quadratic model is used, since it is able to provide better results than the linear and sinusoidal models. Interested readers can refer to our recent paper in [11].

Algorithm 2 Ranking-based vector selection for "DE/rand/1" mutation

Input: The target vector index i
Output: The selected vector indexes r_1, r_2, r_3
Randomly select $r_1 \in [1, NP]$
while rndreal$[0, 1) > p_{r_1}$ **or** $r_1 == i$ **do**
  Randomly select $r_1 \in [1, NP]$
end while
Randomly select $r_2 \in [1, NP]$
while rndreal$[0, 1) > p_{r_2}$ **or** $r_2 == r_1$ **or** $r_2 == i$ **do**
  Randomly select $r_2 \in [1, NP]$
end while
Randomly select $r_3 \in [1, NP]$
while rndreal$[0, 1) <= p_{r_3}$ **or** $r_3 == r_2$ **or** $r_3 == r_1$ **or** $r_3 == i$ **do**
  Randomly select $r_3 \in [1, NP]$
end while

3.1.3 Ranking-Based Vector Selection

Inspired by the role differentiation mechanism proposed in [10], in our proposed ranking-based mutation operators, the vectors are selected based on their rankings and roles. Also, the vector can be classified into four different roles (i.e. *placing*, *leading*, *correcting*, and *receiving* vectors) as proposed in [10]. Solutions in the population with higher selection probabilities are more likely to be chosen as the placing and leading vectors, while poor solutions are more likely to be selected as the correcting vectors in the DE mutation. As an illustration, the ranking-based vector selection for the "DE/rand/1" mutation is shown in Algorithm 2. From Algorithm 2, different from the vector selection in the original DE algorithm, in the ranking-based vector selection the selection probabilities, which are calculated based on the rankings, are used to control the selection of different vectors. For

example, in ranking-based "DE/rand/1" mutation, the placing vector $\mathbf{x}_{r_1}$ and the leading vector $\mathbf{x}_{r_2}$ try to select good solutions, but the correcting vector $\mathbf{x}_{r_3}$ proportionally chooses the poor solution. Different from the vector selection presented in [10], our approach does not introduce any new parameters, while in [10] there are three new parameters, i.e. N^P, N^L, and N^C. In addition, in our ranking-based vector selection there are no explicit groups to differentiate the vectors in the population.

It is worth pointing out that Algorithm 2 is only an illustration for the "DE/rand/1" mutation, our proposed ranking-based vector selection is simple and general. It is also applicable to other mutation operators. Compared with our previous work in [11], the major difference is that in this work the correcting vector is also selected according to its ranking, while in [11] it is only selected randomly.

Algorithm 3 rank-DE: ranking-based differential evolution

```
1:  Generate the initial population randomly
2:  Evaluate the fitness for each individual in the population
3:  while the stop criterion is not satisfied do
4:     Sort the population based on the fitness of each individual          ⇐
5:     Calculate the selection probability for each individual according to Equation (4)  ⇐

6:     for i = 1 to NP do
7:        Select r1, r2, r3 as shown in Algorithm 2                         ⇐
8:        Generate the trial vector u_i with ranking-based "DE/rand/1/exp" strategy
9:     end for
10:    for i = 1 to NP do
11:       Evaluate the offspring u_i
12:       if f(u_i) is better than or equal to f(x_i) then
13:          Replace x_i with u_i
14:       end if
15:    end for
16: end while
```

3.2 DE with Ranking-Based Mutation

By combing the ranking-based mutation operators, we propose the ranking-based DE algorithm, referred to as rank-DE. The pseudo-code of rank-DE is shown in Algorithm 3. The differences between Algorithm 1 and Algorithm 3 are highlighted in "⇐". Note that in line 8 of Algorithm 3 other ranking-based DE strategies can also be used to generate trial vector $\mathbf{u}_i$. Compared with the original DE algorithm, Algorithm 3 indicates that our proposed ranking-based DE algorithm is very simple, it does not increase the overall complexity of DE. Additionally, rank-DE

enhances the exploitation ability of DE due to its ranking-based mutation operator.

4 EXPERIMENTAL RESULTS AND ANALYSIS

In order to evaluate the performance of our proposed ranking-based DE for large-scale optimization problems, we employ the test suite presented for the special issue of Soft Computing on scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems [12]. The test suite contains 19 test functions, which can be categorized into four groups:

- Shifted unimodal functions: F1–F2;
- Shifted multimodal functions: F3–F6;
- Other shifted unimodal functions: F7–F11;
- Hybrid composite functions: F12–F19.

All of these functions are tested at $D = 50$, 100, 200, 500, and 1 000. More details of these functions can be found in [13].

Algorithm	Parameter Settings
DE, rank-DE	$NP = 60$, $CR = 0.9$, $F = 0.5$
GODE, rank-GODE	$NP = 60$, $CR = 0.9$, $F = 0.5$
GaDE, rank-GaDE	$NP = 60$, $p = 0.2$, $c = 0.1$ $F_m = 0.5$, $CR_m = 0.9$

Table 1. Parameter settings for all compared DE variants

4.1 Parameter Settings

As mentioned above, our proposed ranking-based mutation operators are general, they can be used in different DE variants. In this work, the ranking-based mutation operators are integrated into the original DE algorithm, GODE [32], and GaDE [36], and they are respectively named as rank-DE, rank-GODE, and rank-GaDE. In order to make a fair comparison between rank-DE and its corresponding non-rank DE, we adopt the same parameter settings as used in their original literature. The parameter settings for all compared algorithms are shown in Table 1. The maximal number of fitness function evaluations (Max_NFFEs) are set to $5\,000 \times D$ as suggested in [12]. All algorithms are performed over 25 independent runs. In addition, in both rank-DE and rank-GODE the "DE/rand/1/exp" strategy is used as adopted in DE and GODE. In rank-GaDE, the same strategies are also employed as originally used in GaDE.

F	DE	rank-DE	GODE	rank-GODE	GaDE	rank-GaDE
F1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F2	3.60E−01	**8.15E−04**	2.57E−01	**1.33E−03**	1.46E+01	**2.69E+00**
F3	2.89E+01	**1.59E−01**	3.06E+01	**1.87E−09**	1.18E+01	**3.24E−12**
F4	3.98E−02	3.98E−02	**1.05E−13**	3.98E−02	0.00E+00	0.00E+00
F5	**0.00E+00**	9.85E−04	0.00E+00	0.00E+00	**0.00E+00**	8.88E−04
F6	1.43E−13	**0.00E+00**	1.24E−14	**0.00E+00**	0.00E+00	0.00E+00
F7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F8	3.44E+00	**3.45E−03**	1.67E−01	**4.42E−08**	1.08E−08	**0.00E+00**
F9	2.73E+02	**9.91E−09**	7.77E−06	**4.39E−10**	6.24E−07	**0.00E+00**
F10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F11	6.23E−05	**1.05E−08**	6.44E−06	**6.93E−10**	1.31E−06	**0.00E+00**
F12	5.35E−13	**0.00E+00**	1.33E−13	**0.00E+00**	0.00E+00	0.00E+00
F13	2.45E+01	**4.98E−02**	2.55E+01	**5.05E−02**	1.19E+01	**6.24E−01**
F14	4.16E−08	**3.35E−14**	6.24E−09	**5.79E−13**	9.78E−13	**0.00E+00**
F15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F16	1.56E−09	**0.00E+00**	1.57E−10	**5.35E−14**	4.78E−12	**0.00E+00**
F17	7.98E−01	**2.21E−01**	1.17E+00	**3.96E−02**	4.97E−01	**2.49E−01**
F18	1.22E−04	**1.18E−10**	2.97E−07	**6.30E−10**	4.82E−08	**2.40E−10**
F19	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

All the results below 1.00E−14 have been approximated to 0.

Table 2. Comparison of the mean error values between DEs and their corresponding rank-DEs for functions F1–F19 at $D = 50$

4.2 Influence of Ranking-Based Mutation

In this section, we evaluate the influence of ranking-based mutation operators to DE, GODE, and GaDE. The ranking-based DE is compared with its corresponding non-ranking-based DE, i.e., rank-DE vs. DE, rank-GODE vs. GODE, and rank-GaDE vs. GaDE. The results for all functions at $D = 50$, 100, 200, 500, and 1 000 are reported in Tables 2–6, respectively. Note that the results of DE, GODE, and GaDE are obtained from `http://sci2s.ugr.es/eamhco/SOCO-results.xls`. In Tables 2–6, the better results are highlighted in **boldface** compared between rank-DEs and their corresponding non-rank-DEs. In addition, as stated in [8, 9], the multiple-problem statistical analysis is also important to check the behavior of the stochastic algorithms. Therefore, in order to further prove statistical significance of the results, we also use the Wilcoxon's test to compare rank-DEs with their corresponding non-rank-DEs. The Wilcoxon's test is a non-parametric statistical hypothesis test, which can be used as an alternative to the paired t-test when the results cannot be assumed to be normally distributed [25]. The results, which are calculated by OriginPro software, are shown in Table 7. In addition, the detailed results of rank-DE, rank-GODE, and rank-GaDE for all functions at different dimensions are shown in Tables 15–17 in the Appendix A.

F	DE	rank-DE	GODE	rank-GODE	GaDE	rank-GaDE
F1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F2	4.45E+00	**1.69E−01**	3.65E+00	**2.10E−01**	3.88E+01	**4.74E+00**
F3	8.01E+01	**3.39E+01**	8.14E+01	**4.14E+01**	5.89E+01	**2.22E+00**
F4	**7.96E−02**	1.19E−01	8.32E−14	**0.00E+00**	0.00E+00	0.00E+00
F5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F6	3.10E−13	**1.42E−14**	2.60E−14	**1.48E−14**	0.00E+00	0.00E+00
F7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F8	3.69E+02	**1.75E+01**	7.53E+01	**8.50E−06**	1.23E−03	**3.34E−06**
F9	5.06E+02	**1.04E−07**	1.46E−05	**7.32E−10**	3.87E−07	**0.00E+00**
F10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F11	1.28E−04	**1.13E−07**	1.58E−05	**7.30E−10**	4.34E−07	**0.00E+00**
F12	5.99E−11	**0.00E+00**	7.57E−12	**0.00E+00**	0.00E+00	0.00E+00
F13	6.17E+01	**2.49E+01**	6.32E+01	**2.87E+01**	4.99E+01	**8.96E−01**
F14	4.79E−02	**3.98E−02**	**4.13E−08**	3.98E−02	7.90E−13	**0.00E+00**
F15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F16	3.58E−09	**1.46E−13**	3.75E−10	**1.24E−12**	2.45E−12	**4.21E−13**
F17	1.23E+01	**1.03E−01**	1.11E+01	**8.98E−02**	3.28E+00	**7.19E−01**
F18	2.98E−04	**2.66E−09**	1.11E−06	**1.30E−08**	1.96E−08	**2.47E−09**
F19	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

All the results below 1.00E−14 have been approximated to 0.

Table 3. Comparison of the mean error values between DEs and their corresponding rank-DEs for functions F1–F19 at $D = 100$

4.2.1 Comparison Between DE and Rank-DE

First, the results of rank-DE is compare with those of DE. From the results shown in Tables 2–6, we can see that:

- For all functions at $D = 50$, in 5 functions (F1, F7, F10, F15, and F19) both DE and rank-DE are able to find the global optimum over all runs. In 12 out of 19 functions, our proposed rank-DE obtains better mean error values than DE. Only in one function (F5), DE is better than rank-DE. In F5, rank-DE occasionally converges to the local optima. The reason might be that the ranking-based mutation operator in rank-DE leads to over-exploitation in this problem. Therefore, this motivates us to study more sophisticated ranking technique that can control the selection pressure adaptively. We will leave it in our future work.
- When $D = 100$, there are 6 functions (F1, F5, F7, F10, F15, and F19) whose global optimum are obtained by both DE and rank-DE over all runs. rank-DE provides better results than DE in 12 out of 19 functions, but only loses in one function (F4).
- With respect to $D = 200$, similar to the results at $D = 100$, both DE and rank-DE get the global optimum in 6 functions (F1, F5, F7, F10, F15, and F19). In

F	DE	rank-DE	GODE	rank-GODE	GaDE	rank-GaDE
F1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F2	1.92E+01	**3.22E+00**	1.53E+01	**3.59E+00**	5.76E+01	**2.86E+01**
F3	1.78E+02	**1.36E+02**	1.80E+02	**1.42E+02**	1.61E+02	**9.03E+01**
F4	**1.27E−01**	1.59E−01	**4.17E−13**	3.98E−02	0.00E+00	0.00E+00
F5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	**0.00E+00**	5.91E−04
F6	6.54E−13	**3.09E−14**	5.45E−14	**3.24E−14**	0.00E+00	0.00E+00
F7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F8	5.53E+03	**1.15E+03**	2.10E+03	**9.33E−07**	3.02E+00	**6.94E−01**
F9	1.01E+03	**8.19E−07**	3.23E−05	**9.66E−11**	**4.53E−09**	7.09E−07
F10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.20E−02	4.20E−02
F11	2.62E−04	**8.00E−07**	3.12E−05	**1.18E−10**	**1.85E−07**	2.21E−06
F12	9.76E−10	**2.38E−14**	1.20E−10	**2.30E−13**	4.92E−14	**0.00E+00**
F13	1.36E+02	**1.09E+02**	1.38E+02	**1.11E+02**	1.24E+02	**7.63E+01**
F14	1.38E−01	**1.19E−01**	**8.17E−02**	1.59E−01	2.87E−12	**2.17E−13**
F15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F16	7.46E−09	**1.84E−12**	9.54E−10	**1.35E−11**	**1.58E−12**	5.96E−12
F17	3.70E+01	**1.13E+01**	3.74E+01	**1.26E+01**	2.45E+01	**7.54E−01**
F18	**4.73E−04**	7.96E−02	**1.91E−06**	3.98E−02	2.53E−08	**2.39E−08**
F19	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00

All the results below 1.00E−14 have been approximated to 0.

Table 4. Comparison of the mean error values between DEs and their corresponding rank-DEs for functions F1–F19 at $D = 200$

11 functions, rank-DE is better than DE in terms of the mean error values. In 2 functions (F4 and F18), DE provides better results than rank-DE.

- For all function at $D = 500$, also in 6 functions (F1, F5, F7, F10, F15, and F19) both DE and rank-DE obtain the global optimum over all runs. rank-DE is capable of providing better results in 10 out of 19 functions, but loses in three functions (F4, F14, and F18).
- When the dimension is scaled up to $D = 1\,000$, for functions F1, F5, F7, F10, F15, and F19, their global optimum are found by both rank-DE and DE over all 25 runs. In 12 out of 19 functions, rank-DE improves the results of DE. DE only gets better mean error value in one function (F18) than that of rank-DE.

4.2.2 Comparison Between GODE and Rank-GODE

In this section, the ranking-based mutation operator is integrated into GODE [32] to verify the enhanced performance of our approach. The mean error values of rank-GODE and GODE are shown in Tables 2–6. From the results, it can be observed that:

F	DE	rank-DE	GODE	rank-GODE	GaDE	rank-GaDE
F1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F2	5.35E+01	**2.35E+01**	5.81E+01	**2.31E+01**	7.42E+01	**4.69E+01**
F3	4.76E+02	**4.35E+02**	4.76E+02	**4.34E+02**	4.40E+02	**3.80E+02**
F4	**3.20E−01**	4.38E−01	**1.62E−03**	2.39E−01	0.00E+00	0.00E+00
F5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F6	1.65E−12	**8.22E−14**	1.43E−13	**8.88E−14**	**1.46E−14**	3.44E−14
F7	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F8	6.09E+04	**2.68E+04**	3.93E+04	**0.00E+00**	1.33E+03	**1.32E+03**
F9	2.52E+03	**6.28E−06**	7.84E−05	**4.20E−14**	**0.00E+00**	4.44E−05
F10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	3.78E−01	**1.26E−01**
F11	6.76E−04	**6.22E−06**	8.25E−05	**3.72E−14**	**0.00E+00**	4.04E−05
F12	7.07E−09	**2.43E−12**	7.39E−10	**1.81E−11**	**1.07E−12**	7.04E−12
F13	3.59E+02	**3.31E+02**	3.59E+02	**3.34E+02**	3.34E+02	**3.07E+02**
F14	**1.35E−01**	3.18E−01	**7.67E−02**	2.79E−01	2.79E−11	**8.42E−12**
F15	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F16	2.04E−08	**2.72E−11**	2.24E−09	**1.72E−10**	**1.67E−12**	1.38E−10
F17	1.11E+02	**8.69E+01**	1.12E+02	**8.84E+01**	9.26E+01	**5.24E+01**
F18	**1.22E−03**	3.98E−02	5.06E−06	**1.49E−06**	5.59E−08	**3.99E−10**
F19	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.20E−02	**0.00E+00**

All the results below 1.00E−14 have been approximated to 0.

Table 5. Comparison of the mean error values between DEs and their corresponding rank-DEs for functions F1–F19 at $D = 500$

- In 6 functions (F1, F5, F7, F10, F15, and F19) at $D = 50$, 100, 200, and 500, both rank-GODE and GODE consistently get the global optimum over all runs. At $D = 1\,000$, in 9 functions (F1, F5, F7–F11, F15, and F19) rank-GODE still obtains the global optimum over all 25 runs. While GODE finds the global optimum only in 4 functions.
- Regardless of the influence of dimensionality, in the majority of the test functions, our proposed rank-GODE consistently provides better results than those of GODE. In 12, 12, 10, 11, and 14 functions, rank-GODE respectively gets better mean error values than GODE at $D = 50$, 100, 200, 500, and 1 000.
- Rank-GODE is only worse than GODE in 1, 1, 3, 2, and 1 out of 19 functions at $D = 50$, 100, 200, 500, and 1 000, respectively.

4.2.3 Comparison Between GaDE and Rank-GaDE

GaDE, proposed by Yang et al. [36], is an adaptive DE variant with a new proposed generalized parameter adaptation scheme and strategy adaptation. In this section, our proposed ranking-based vector selection technique is integrated into both of the mutation operators used in GaDE. The results of rank-GaDE and GaDE are

F	DE	rank-DE	GODE	rank-GODE	GaDE	rank-GaDE
F1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F2	8.46E+01	**5.03E+01**	9.02E+01	**4.79E+01**	8.93E+01	**4.34E+01**
F3	9.69E+02	**9.27E+02**	9.70E+02	**9.30E+02**	9.45E+02	**8.76E+02**
F4	1.44E+00	**5.97E−01**	1.03E+00	**7.56E−01**	0.00E+00	0.00E+00
F5	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F6	3.29E−12	**1.75E−13**	2.88E−13	**1.86E−13**	**1.66E−14**	5.41E−14
F7	0.00E+00	0.00E+00	INF	**0.00E+00**	0.00E+00	0.00E+00
F8	2.46E+05	**1.37E+05**	1.86E+05	**0.00E+00**	1.77E+04	**1.59E+04**
F9	5.13E+03	**2.26E−05**	1.70E−04	**0.00E+00**	**0.00E+00**	1.80E−04
F10	0.00E+00	0.00E+00	0.00E+00	0.00E+00	4.62E−01	**8.40E−02**
F11	1.35E−03	**2.29E−05**	1.73E−04	**0.00E+00**	**0.00E+00**	1.73E−04
F12	1.68E−08	**2.30E−11**	1.87E−09	**1.57E−10**	**3.85E−12**	1.49E−10
F13	7.30E+02	**7.06E+02**	7.31E+02	**7.08E+02**	7.15E+02	**6.80E+02**
F14	6.90E−01	**3.98E−01**	6.06E−01	**3.98E−01**	8.82E−11	**7.18E−12**
F15	0.00E+00	0.00E+00	INF	**0.00E+00**	0.00E+00	0.00E+00
F16	4.18E−08	**1.28E−10**	4.59E−09	**8.00E−10**	**2.35E−12**	6.78E−10
F17	2.36E+02	**2.11E+02**	2.36E+02	**2.14E+02**	2.19E+02	**1.80E+02**
F18	**2.37E−03**	3.98E−02	**3.29E−05**	3.98E−02	1.30E−07	**1.62E−08**
F19	0.00E+00	0.00E+00	0.00E+00	0.00E+00	3.78E−01	**0.00E+00**

All the results below 1.00E−14 have been approximated to 0.

Table 6. Comparison of the mean error values between DEs and their corresponding rank-DEs for functions F1–F19 at $D = 1\,000$

reported in Tables 2–6. All of the results are averaged over 25 independent runs. The results in Tables 2–6 show that:

- When $D = 50$, both rank-GaDE and GaDE can solve 8 functions (F1, F4, F6, F7, F10, F12, F15, and F19) over all runs. rank-GaDE improves the mean error values of GaDE in 10 out of 19 functions. GaDE only obtains better results than rank-GaDE in function F5.
- For all functions at $D = 100$, in 9 functions (F1, F4–F7, F10, F12, F15, and F19), their global optimum are obtained by rank-GaDE and GaDE consistently. In the rest of 10 functions, rank-GaDE gets better results than GaDE.
- For all functions at $D = 200$, rank-GaDE obtains better results in 8 functions, but loses in 4 functions compared with GaDE. In the rest of 7 functions, both rank-GaDE and GaDE get the same mean error values.
- With respect to $D = 500$, in 9 out of 19 functions rank-GaDE is capable of provide better results than GaDE. rank-GaDE is worse than GaDE in 5 functions. Both GaDE and rank-GaDE consistently find the global optimum in the rest of 5 functions (F1, F4, F5, F7, and F15).

$D = 50$					
Algorithm	R^+	R^-	p-value	significance at $\alpha = 0.05$	significance at $\alpha = 0.1$
rank-DE vs. DE	97	8	3.05E−03	+	+
rank-GODE vs. GODE	83	8	6.10E−03	+	+
rank-GaDE vs. GaDE	59	7	1.86E−02	+	+
$D = 100$					
Algorithm	R^+	R^-	p-value	significance at $\alpha = 0.05$	significance at $\alpha = 0.1$
rank-DE vs. DE	84	7	4.64E−03	+	+
rank-GODE vs. GODE	83	8	6.10E−03	+	+
rank-GaDE vs. GaDE	55	0	1.95E−03	+	+
$D = 200$					
Algorithm	R^+	R^-	p-value	significance at $\alpha = 0.05$	significance at $\alpha = 0.1$
rank-DE vs. DE	78	13	2.15E−02	+	+
rank-GODE vs. GODE	70	21	9.42E−02	=	+
rank-GaDE vs. GaDE	57	21	1.76E−01	=	=
$D = 500$					
Algorithm	R^+	R^-	p-value	significance at $\alpha = 0.05$	significance at $\alpha = 0.1$
rank-DE vs. DE	73	18	5.74E−02	=	+
rank-GODE vs. GODE	76	15	3.27E−02	+	+
rank-GaDE vs. GaDE	85	20	4.19E−02	+	+
$D = 1\,000$					
Algorithm	R^+	R^-	p-value	significance at $\alpha = 0.05$	significance at $\alpha = 0.1$
rank-DE vs. DE	86	5	2.44E−03	+	+
rank-GODE vs. GODE*	114	6	8.54E−04	+	+
rank-GaDE vs. GaDE	84	21	4.94E−02	+	+

$\star$ In GODE, for functions F7 and F15 "INF" is approximated to 1.00E+20 to make the multiple-problem Wilcoxon's test.

Table 7. Results of the multiple-problem Wilcoxon's test for all DE variants on the mean error values of functions F1–F19

- When $D = 1\,000$, similar to the results at $D = 500$, in 5 functions (F1, F4, F5, F7, and F15) GaDE and rank-GaDE get the global optimum over all runs. rank-GaDE improves GaDE in 9 functions, but loses in 5 functions.

4.2.4 Summary

To summarize the results shown in Tables 2–6, the multiple-problem analysis on the mean error values in all functions is tabulated in Table 7. From Table 7, it is clear to

F	DE	CHC	G-CMA-ES	rank-DE	rank-GODE	rank-GaDE
F1	**0.00E+00**	**1.67E−11**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F2	3.60E−01	6.19E+01	**2.75E−11**	**8.15E−04**	1.33E−03	2.69E+00
F3	2.89E+01	1.25E+06	7.97E−01	1.59E−01	**1.87E−09**	**3.24E−12**
F4	**3.98E−02**	7.43E+01	1.05E+02	**3.98E−02**	**3.98E−02**	**0.00E+00**
F5	**0.00E+00**	1.67E−03	**2.96E−04**	9.85E−04	**0.00E+00**	8.88E−04
F6	**1.43E−13**	6.15E−07	2.09E+01	**0.00E+00**	**0.00E+00**	**0.00E+00**
F7	**0.00E+00**	2.66E−09	**1.01E−10**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F8	3.44E+00	2.24E+02	**0.00E+00**	3.45E−03	**4.42E−08**	**0.00E+00**
F9	2.73E+02	3.10E+02	1.66E+01	9.91E−09	**4.39E−10**	**0.00E+00**
F10	**0.00E+00**	7.30E+00	**6.81E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F11	6.23E−05	2.16E+00	3.01E+01	1.05E−08	**6.93E−10**	**0.00E+00**
F12	**5.35E−13**	9.57E−01	1.88E+02	**0.00E+00**	**0.00E+00**	**0.00E+00**
F13	2.45E+01	2.08E+06	1.97E+02	**4.98E−02**	**5.05E−02**	6.24E−01
F14	4.16E−08	6.17E+01	1.09E+02	**3.35E−14**	5.79E−13	**0.00E+00**
F15	**0.00E+00**	3.98E−01	**9.79E−04**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F16	1.56E−09	2.95E−09	4.27E+02	**0.00E+00**	**5.35E−14**	**0.00E+00**
F17	7.98E−01	2.26E+04	6.89E+02	**2.21E−01**	**3.96E−02**	2.49E−01
F18	1.22E−04	1.58E+01	1.31E+02	**1.18E−10**	6.30E−10	**2.40E−10**
F19	**0.00E+00**	3.59E+02	**4.76E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**

All the results below 1.00E−14 have been approximated to 0.

Table 8. Comparison of the mean error values among three baseline algorithms and rank-DEs for functions F1–F19 at $D = 50$

see that the ranking-based DE variants consistently provides higher R^+ values than those of non-ranking-based DEs, which means that the ranking-based DE variants are consistently better than the original DE mutation based methods. At $\alpha = 0.05$, in 12 out of 15 cases rank-DEs get significantly better results than non-rank-DEs according to the Wilcoxon's test. In addition, when $\alpha = 0.1$, rank-DEs significantly outperforms non-rank-DEs in 18 out of 19 cases.

In general, from the results shown in Tables 2–7 and the above analysis, we can conclude that our proposed ranking-based vector selection technique is really capable of improving the performance of DE. The reason is that the ranking-based mutation operators enhance the exploitation ability and make ranking-based DE balance the exploration and exploitation abilities.

4.3 Comparison with Baseline Algorithms

In the previous section, we have verified the enhanced performance of our proposed ranking-based mutation operators. In this section, in order to make an analysis of the scalability behavior of our proposed rank-DEs, the comparison to three baseline evolutionary algorithms for continuous optimization problems is performed as suggested in [12]. The three baseline algorithms are

F	DE	CHC	G-CMA-ES	rank-DE	rank-GODE	rank-GaDE
F1	**0.00E+00**	**3.56E−11**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F2	4.45E+00	8.58E+01	**1.51E−10**	**1.69E−01**	2.10E−01	4.74E+00
F3	8.01E+01	4.19E+06	**3.88E+00**	3.39E+01	4.14E+01	**2.22E+00**
F4	**7.96E−02**	2.19E+02	2.50E+02	1.19E−01	**0.00E+00**	**0.00E+00**
F5	**0.00E+00**	3.83E−03	**1.58E−03**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F6	3.10E−13	4.10E−07	2.12E+01	**1.42E−14**	1.48E−14	**0.00E+00**
F7	**0.00E+00**	1.40E−02	**4.22E−04**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F8	3.69E+02	1.69E+03	**0.00E+00**	1.75E+01	8.50E−06	**3.34E−06**
F9	5.06E+02	5.86E+02	1.02E+02	1.04E−07	**7.32E−10**	**0.00E+00**
F10	**0.00E+00**	3.30E+01	**1.66E+01**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F11	1.28E−04	7.32E+01	1.64E+02	1.13E−07	**7.30E−10**	**0.00E+00**
F12	**5.99E−11**	1.03E+01	4.17E+02	**0.00E+00**	**0.00E+00**	**0.00E+00**
F13	6.17E+01	2.70E+06	4.21E+02	**2.49E+01**	2.87E+01	**8.96E−01**
F14	4.79E−02	1.66E+02	2.55E+02	**3.98E−02**	**3.98E−02**	**0.00E+00**
F15	**0.00E+00**	8.13E+00	**6.30E−01**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F16	3.58E−09	2.23E+01	8.59E+02	**1.46E−13**	1.24E−12	**4.21E−13**
F17	1.23E+01	1.47E+05	1.51E+03	**1.03E−01**	**8.98E−02**	7.19E−01
F18	2.98E−04	7.00E+01	3.07E+02	**2.66E−09**	1.30E−08	**2.47E−09**
F19	**0.00E+00**	5.45E+02	**2.02E+01**	**0.00E+00**	**0.00E+00**	**0.00E+00**

All the results below 1.00E−14 have been approximated to 0.

Table 9. Comparison of the mean error values among three baseline algorithms and rank-DEs for functions F1–F19 at $D = 100$

- DE: the original DE algorithm with "DE/rand/1/exp" strategy, $CR = 0.9$, and $F = 0.5$ [28];
- CHC: the real-coded CHC proposed by Eshelman and Schaffer [7];
- G-CMA-ES: a restart CMA-ES with increasing population size [2].

We obtained the results of DE, CHC and G-CMA-ES from `http://sci2s.ugr.es/eamhco/SOCO-results.xls`. The results of DE, CHC, G-CMA-ES, rank-DE, rank-GODE, and rank-GaDE for all functions at $D = 50$, 100, 200, 500, and 1 000 are respectively reported in Tables 8–12. In these tables, the best and second best results are highlighted in **grey boldface** and **boldface**, respectively. In addition, the average rankings obtained by each above algorithm in the Friedman test[1] are tabulated in Table 13.

From the results shown in Tables 8–12, we can observe that regardless of the dimensionality the ranking-based DE variants always get the 1st best mean error values than the three baseline algorithms in the majority of the functions. For example, for all functions at $D = 500$, rank-DE, rank-GODE, rank-GaDE, DE, and

[1] The KEEL software [1] (`http://www.keel.es/`) is used to get the average rankings obtained by each algorithm based on the Friedman test.

F	DE	CHC	G-CMA-ES	rank-DE	rank-GODE	rank-GaDE
F1	**0.00E+00**	**8.34E−01**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F2	1.92E+01	1.03E+02	**1.16E−09**	**3.22E+00**	3.59E+00	2.86E+01
F3	1.78E+02	2.01E+07	**8.91E+01**	1.36E+02	1.42E+02	**9.03E+01**
F4	1.27E−01	5.40E+02	6.48E+02	1.59E−01	**3.98E−02**	**0.00E+00**
F5	**0.00E+00**	8.76E−03	**0.00E+00**	**0.00E+00**	**0.00E+00**	**5.91E−04**
F6	6.54E−13	1.23E+00	2.14E+01	**3.09E−14**	3.24E−14	**0.00E+00**
F7	**0.00E+00**	2.59E−01	**1.17E−01**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F8	5.53E+03	9.38E+03	**0.00E+00**	1.15E+03	**9.33E−07**	6.94E−01
F9	1.01E+03	1.19E+03	3.75E+02	8.19E−07	**9.66E−11**	**7.09E−07**
F10	**0.00E+00**	7.13E+01	4.43E+01	**0.00E+00**	**0.00E+00**	**4.20E−02**
F11	2.62E−04	3.85E+02	8.03E+02	**8.00E−07**	**1.18E−10**	2.21E−06
F12	9.76E−10	7.44E+01	9.06E+02	**2.38E−14**	2.30E−13	**0.00E+00**
F13	1.36E+02	5.75E+06	9.43E+02	**1.09E+02**	1.11E+02	**7.63E+01**
F14	1.38E−01	4.29E+02	6.09E+02	**1.19E−01**	1.59E−01	**2.17E−13**
F15	**0.00E+00**	2.14E+01	**1.75E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F16	7.46E−09	1.60E+02	1.92E+03	**1.84E−12**	1.35E−11	**5.96E−12**
F17	3.70E+01	1.75E+05	3.36E+03	**1.13E+01**	1.26E+01	**7.54E−01**
F18	**4.73E−04**	2.12E+02	6.89E+02	7.96E−02	3.98E−02	**2.39E−08**
F19	**0.00E+00**	2.06E+03	**7.52E+02**	**0.00E+00**	**0.00E+00**	**0.00E+00**

All the results below 1.00E−14 have been approximated to 0.

Table 10. Comparison of the mean error values among three baseline algorithms and rank-DEs for functions F1–F19 at $D = 200$

G-CMA-ES provides the 1st best results in 8, 9, 11, 6 and 3 functions, respectively. There are no functions that CHC obtains the best overall results.

The p-values computed by Iman-Davenport test on the mean error values shown in Tables 8–12 are respectively 1.50E−11, 1.00E−12, 2.30E−11, 2.90E−11, and 1.00E−11 at $D = 50$, 100, 200, 500, and 1 000. The results indicate that there are significant differences in the behavior of the compared six algorithms for all the functions at $\alpha = 0.05$, regardless of the dimensionality of the test functions.

According to the average rankings obtained by each algorithm in the Friedman test shown in Table 13, the results show that all of our proposed ranking-based DE variants obtain better rankings than the three compared baseline algorithms. Regardless of the dimensionality, in all cases, rank-GaDE gets the 1st ranking, followed by rank-GODE, rank-DE, DE, G-CMA-ES (except $D = 500$ and $D = 1\,000$)[2], and CHC.

[2] In G-CMA-ES, when $D = 500$ and $D = 1\,000$ the average error values of some functions are greater than $1.00E + 100$, therefore, the average rankings obtained by the Friedman test do not include the G-CMA-ES in these two cases.

F	DE	CHC	G-CMA-ES	rank-DE	rank-GODE	rank-GaDE
F1	**0.00E+00**	**2.84E−12**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F2	5.35E+01	1.29E+02	**3.48E−04**	2.35E+01	**2.31E+01**	4.69E+01
F3	4.76E+02	1.14E+06	**3.58E+02**	4.35E+02	4.34E+02	**3.80E+02**
F4	**3.20E−01**	1.91E+03	2.10E+03	4.38E−01	2.39E−01	**0.00E+00**
F5	**0.00E+00**	6.98E−03	**2.96E−04**	**0.00E+00**	**0.00E+00**	**0.00E+00**
F6	1.65E−12	5.16E+00	2.15E+01	**8.22E−14**	8.88E−14	**3.44E−14**
F7	**0.00E+00**	**1.27E−01**	7.21E+153	**0.00E+00**	**0.00E+00**	**0.00E+00**
F8	6.09E+04	7.22E+04	**2.36E−06**	2.68E+04	**0.00E+00**	1.32E+03
F9	2.52E+03	3.00E+03	1.74E+03	**6.28E−06**	**4.20E−14**	4.44E−05
F10	**0.00E+00**	1.86E+02	1.27E+02	**0.00E+00**	**0.00E+00**	**1.26E−01**
F11	6.76E−04	1.81E+03	4.16E+03	**6.22E−06**	**3.72E−14**	4.04E−05
F12	7.07E−09	4.48E+02	2.58E+03	**2.43E−12**	1.81E−11	**7.04E−12**
F13	3.59E+02	3.22E+07	2.87E+03	**3.31E+02**	3.34E+02	**3.07E+02**
F14	**1.35E−01**	1.46E+03	1.95E+03	3.18E−01	2.79E−01	**8.42E−12**
F15	**0.00E+00**	**6.01E+01**	2.82E+262	**0.00E+00**	**0.00E+00**	**0.00E+00**
F16	2.04E−08	9.55E+02	5.45E+03	**2.72E−11**	1.72E−10	**1.38E−10**
F17	1.11E+02	8.40E+05	9.59E+03	**8.69E+01**	8.84E+01	**5.24E+01**
F18	1.22E−03	7.32E+02	2.05E+03	3.98E−02	**1.49E−06**	**3.99E−10**
F19	**0.00E+00**	**1.76E+03**	2.44E+06	**0.00E+00**	**0.00E+00**	**0.00E+00**

All the results below 1.00E−14 have been approximated to 0.

Table 11. Comparison of the mean error values among three baseline algorithms and rank-DEs for functions F1–F19 at $D = 500$

4.4 Comparison with Reported Results

In the special issue of Soft Computing [12], there are 13 papers published therein. All of the results are available online at `http://sci2s.ugr.es/eamhco/SOCO-results.xls`. In this subsection, we compare the results of rank-DE, rank-GODE, and rank-GaDE with those of the 13 advanced methods. With respect to the mean error values, the average rankings obtained by each algorithm in the Friedman test are reported in Table 14. From the results, it can be seen that MOS [15], which is a multiple offspring sampling method containing different search strategies, consistently obtains the best ranking regardless of the dimensionality. The ranking of jDElscop and rank-GaDE in different dimensions of problems are twisted: in $D = 50$, 200, and 500, jDElscop is better than rank-GaDE; while in $D = 100$ and 1 000, rank-GaDE provides better rankings than jDElscop. However, in overall, rank-GaDE obtains the 2$^{\text{nd}}$ ranking, following by jDElscop, rank-GODE, and rank-DE. It is worth noting that although the ranking-based DE variants are not the best one among all compared algorithms, they can provide promising results. More importantly, they improves their non-ranking-based DEs markedly, for example, the overall ranking of rank-GaDE is 2, while GaDE only ranks 7.

F	DE	CHC	G-CMA-ES	rank-DE	rank-GODE	rank-GaDE
F1	**0.00E+00**	**1.36E−11**	NA	**0.00E+00**	**0.00E+00**	**0.00E+00**
F2	8.46E+01	1.44E+02	NA	5.03E+01	**4.79E+01**	**4.34E+01**
F3	9.69E+02	8.75E+03	NA	**9.27E+02**	9.30E+02	**8.76E+02**
F4	1.44E+00	4.76E+03	NA	**5.97E−01**	7.56E−01	**0.00E+00**
F5	**0.00E+00**	**7.02E−03**	NA	**0.00E+00**	**0.00E+00**	**0.00E+00**
F6	3.29E−12	1.38E+01	NA	**1.75E−13**	1.86E−13	**5.41E−14**
F7	**0.00E+00**	**3.52E−01**	NA	**0.00E+00**	**0.00E+00**	**0.00E+00**
F8	2.46E+05	3.11E+05	NA	1.37E+05	**0.00E+00**	**1.59E+04**
F9	5.13E+03	6.11E+03	NA	**2.26E−05**	**0.00E+00**	1.80E−04
F10	**0.00E+00**	3.83E+02	NA	**0.00E+00**	**0.00E+00**	**8.40E−02**
F11	1.35E−03	4.82E+03	NA	**2.29E−05**	**0.00E+00**	1.73E−04
F12	1.68E−08	1.05E+03	NA	**2.30E−11**	1.57E−10	**1.49E−10**
F13	7.30E+02	6.66E+07	NA	**7.06E+02**	7.08E+02	**6.80E+02**
F14	6.90E−01	3.62E+03	NA	**3.98E−01**	**3.98E−01**	**7.18E−12**
F15	**0.00E+00**	**8.37E+01**	NA	**0.00E+00**	**0.00E+00**	**0.00E+00**
F16	4.18E−08	2.32E+03	NA	**1.28E−10**	8.00E−10	**6.78E−10**
F17	2.36E+02	2.04E+07	NA	**2.11E+02**	2.14E+02	**1.80E+02**
F18	**2.37E−03**	1.72E+03	NA	3.98E−02	3.98E−02	**1.62E−08**
F19	**0.00E+00**	**4.20E+03**	NA	**0.00E+00**	**0.00E+00**	**0.00E+00**

All the results below 1.00E−14 have been approximated to 0.

Table 12. Comparison of the mean error values among three baseline algorithms and rank-DEs for functions F1–F19 at $D = 1\,000$. "NA" means the results are not available.

Algorithm	Ranking ($D = 50$)	Ranking ($D = 100$)	Ranking ($D = 200$)	Ranking ($D = 500$)	Ranking ($D = 1\,000$)
DE	3.5	3.5263	3.3158	3.2368	3.3421
CHC	5.5	5.5	5.6316	4.8947	4.8947
G-CMA-ES	4.5789	4.6579	4.4474	NA	NA
rank-DE	2.6579	2.6579	2.6316	2.6053	2.3947
rank-GODE	**2.3947**	**2.5263**	**2.5263**	**2.1316**	**2.3421**
rank-GaDE	**2.3684**	**2.1316**	**2.4474**	**2.0893**	**2.0263**

Table 13. Average rankings obtained by each algorithm in the Friedman test. "NA" means not available.

5 CONCLUSIONS

In this paper, we employ our proposed modified ranking-based mutation operators to enhance the performance of differential evolution. In the ranking-based mutation operators, the vectors in the mutation operators are selected according to their rankings in the current population. Better solutions are more likely to be selected to be the placing and leading vectors, while worse solutions have more chance to be chosen as the correcting vector(s). In general, the proposed ranking-based vector technique

Algorithm	Ranking ($D = 50$)	Ranking ($D = 100$)	Ranking ($D = 200$)	Ranking ($D = 500$)	Ranking ($D = 1\,000$)	Average	Overall
SOUPDE	7.7632	8.2105	8.2632	8.0263	7.3684	7.9263	9
DE−D^{40}+M^{m}	8.3947	8.5263	8.4737	7.8421	7.3158	8.1105	10
GODE	8.9211	8.9737	8.3947	8.3947	6.5789	8.2526	11
GaDE	7.6316	7.5000	6.7895	7.3158	NA	7.3092	7
jDElscop	**5.9211**	6.5789	**6.2105**	**6.2632**	6.5000	6.2947	3
SaDE−MMTS	6.6579	7.3421	7.3421	7.8421	7.1316	7.2632	6
MOS	**5.6053**	**5.7632**	**5.0263**	**5.0000**	**4.5526**	**5.1895**	**1**
MA-SSW-Chains	9.3684	10.1842	10.6579	12.0526	10.6053	10.5737	15
RPSO-vm	11.5526	10.9211	10.8684	10.3684	8.3684	10.4158	14
Tuned IPSOLS	9.6842	7.7105	7.7105	7.6053	6.5263	7.8474	8
EvoPROpt	15.1316	15.0000	14.8421	14.0000	12.8421	14.3632	16
EM323	10.1842	9.1053	9.1842	9.7895	NA	9.5658	12
VXQR1	10.3947	10.6316	11.0263	10.6842	8.9211	10.3316	13
rank-DE	6.5526	7.3158	7.5263	7.7105	6.5526	7.1316	5
rank-GODE	6.1842	6.4474	6.8421	6.6579	6.0000	6.4263	4
rank-GaDE	6.0526	**5.7895**	6.8421	6.4474	**5.7368**	**6.1737**	**2**

Table 14. Average rankings obtained by different algorithms in the Friedman test. "NA" means not available.

is very simple, and it does not introduce any new parameters. In order to verify the performance of our proposed ranking-based mutation operators, they are integrated into the original DE, GODE, and GaDE; rank-DE, rank-GODE, and rank-GaDE are evaluated on the large-scale continuous optimization problems presented in the special issue of Soft Computing. Experimental results verify our expectation that the ranking-based mutation operators are consistently able to enhance the performance of DE, GODE, and GaDE. Regardless of the dimensionality, ranking-based DEs achieve very promising results in the large-scale continuous optimization. Compared with the three baseline algorithms, statistical results show that ranking-based DEs still obtain better rankings.

The ranking-based mutation operators may also be useful in the constrained optimization and multiobjective optimization. For example, the stochastic ranking technique [24] and non-dominated sorting method [6] can be possibly used to rank solutions in the constrained optimization and multiobjective optimization. In our future, we will try to verify these expectations.

A APPENDIX

In this section, the detailed results of rank-DE, rank-GODE, and rank-GaDE are reported in Tables 15–17, respectively. In each function, each algorithm is performed over 25 independent runs. In Tables 15–17, the median value is highlighted in **boldface** when it is better than or equal to the mean value in the same function.

D	F	Best	Median	Worst	Mean	D	F	Best	Median	Worst	Mean
50	F1	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F11	6.17E−09	**9.93E−09**	1.77E−08	1.05E−08
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		6.25E−08	**1.13E−07**	1.75E−07	1.13E−07
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		6.75E−07	**7.90E−07**	9.43E−07	8.00E−07
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		5.51E−06	6.25E−06	7.06E−06	6.22E−06
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		2.09E−05	**2.26E−05**	2.53E−05	2.29E−05
50	F2	5.21E−04	**7.79E−04**	1.40E−03	8.15E−04	50	F12	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		1.34E−01	**1.69E−01**	2.03E−01	1.69E−01	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		2.82E+00	3.24E+00	3.49E+00	3.22E+00	200		1.72E−14	2.47E−14	3.34E−14	2.38E−14
500		2.23E+01	2.36E+01	2.50E+01	2.35E+01	500		1.58E−12	2.49E−12	3.17E−12	2.43E−12
1 000		4.91E+01	**5.02E+01**	5.16E+01	5.03E+01	1 000		2.01E−11	**2.27E−11**	2.85E−11	2.30E−11
50	F3	1.77E−14	**2.61E−11**	3.99E+00	1.59E−01	50	F13	6.16E−08	**3.85E−07**	3.94E−01	4.98E−02
100		3.12E+01	**3.38E+01**	3.68E+01	3.39E+01	100		1.91E+01	2.52E+01	2.78E+01	2.49E+01
200		1.31E+02	**1.35E+02**	1.75E+02	1.36E+02	200		1.06E+02	**1.07E+02**	1.44E+02	1.09E+02
500		4.29E+02	**4.32E+02**	4.74E+02	4.35E+02	500		3.28E+02	**3.31E+02**	3.35E+02	3.31E+02
1 000		9.24E+02	**9.27E+02**	9.30E+02	9.27E+02	1 000		7.01E+02	**7.02E+02**	7.45E+02	7.06E+02
50	F4	0.00E+00	**0.00E+00**	9.95E−01	3.98E−02	50	F14	0.00E+00	**2.36E−14**	9.57E−14	3.35E−14
100		0.00E+00	**0.00E+00**	9.95E−01	1.19E−01	100		4.29E−12	**7.30E−12**	9.95E−01	3.98E−02
200		0.00E+00	**0.00E+00**	9.95E−01	1.59E−01	200		1.39E−10	**3.43E−10**	9.95E−01	1.19E−01
500		0.00E+00	**0.00E+00**	1.99E+00	4.38E−01	500		2.81E−09	**4.15E−09**	1.99E+00	3.18E−01
1 000		0.00E+00	**0.00E+00**	3.98E+00	5.97E−01	1 000		1.10E−08	**1.38E−08**	1.99E+00	3.98E−01
50	F5	0.00E+00	**0.00E+00**	1.48E−02	9.85E−04	50	F15	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
50	F6	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F16	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		1.11E−14	1.47E−14	1.47E−14	1.42E−14	100		9.08E−14	**1.31E−13**	2.27E−13	1.46E−13
200		2.89E−14	3.24E−14	3.24E−14	3.09E−14	200		1.38E−12	**1.84E−12**	2.49E−12	1.84E−12
500		7.86E−14	**8.22E−14**	8.57E−14	8.22E−14	500		2.23E−11	2.74E−11	3.06E−11	2.72E−11
1 000		1.75E−13	**1.75E−13**	1.82E−13	1.75E−13	1 000		1.12E−10	**1.28E−10**	1.40E−10	1.28E−10
50	F7	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F17	3.46E−07	**1.07E−02**	3.99E+00	2.21E−01
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		1.95E−05	**7.81E−02**	2.84E−01	1.03E−01
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		7.69E+00	1.16E+01	1.62E+01	1.13E+01
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		8.50E+01	8.72E+01	8.90E+01	8.69E+01
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		2.10E+02	**2.11E+02**	2.16E+02	2.11E+02
50	F8	1.29E−03	**3.45E−03**	8.21E−03	3.45E−03	50	F18	5.53E−11	**1.15E−10**	1.79E−10	1.18E−10
100		8.89E+00	**1.67E+01**	3.01E+01	1.75E+01	100		1.84E−09	**2.43E−09**	5.00E−09	2.66E−09
200		8.92E+02	**1.11E+03**	1.44E+03	1.15E+03	200		2.50E−08	**3.54E−08**	9.95E−01	7.96E−02
500		2.30E+04	**2.66E+04**	3.21E+04	2.68E+04	500		3.15E−07	**3.60E−07**	9.95E−01	3.98E−02
1 000		1.27E+05	**1.37E+05**	1.46E+05	1.37E+05	1 000		1.28E−06	**1.44E−06**	9.95E−01	3.98E−02
50	F9	5.68E−09	**9.27E−09**	2.06E−08	9.91E−09	50	F19	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		7.35E−08	1.09E−07	1.40E−07	1.04E−07	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		6.13E−07	**8.19E−07**	9.60E−07	8.19E−07	200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
500		5.04E−06	6.42E−06	7.55E−06	6.28E−06	500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
1 000		2.05E−05	**2.24E−05**	2.66E−05	2.26E−05	1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
50	F10	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	All the results below 1.00E−14 have been					
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	approximated to 0.					
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00						
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00						
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00						

Table 15. Experimental results of rank-DE for functions F1–F19 at $D = 50$, 100, 200, 500, and 1 000, where the median value is highlighted in **boldface** when it is better than or equal to the mean value in the same function

D	F	Best	Median	Worst	Mean	D	F	Best	Median	Worst	Mean
50	F1	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F11	2.29E−10	**6.18E−10**	1.48E−09	6.93E−10
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		8.30E−11	**6.86E−10**	1.58E−09	7.30E−10
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		1.53E−11	**7.12E−11**	5.04E−10	1.18E−10
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		0.00E+00	**1.46E−14**	2.11E−13	3.72E−14
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
50	F2	8.02E−04	**1.31E−03**	2.19E−03	1.33E−03	50	F12	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		1.72E−01	**2.05E−01**	2.57E−01	2.10E−01	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		2.93E+00	**3.62E+00**	3.94E+00	3.59E+00	200		1.21E−13	2.33E−13	3.28E−13	2.30E−13
500		2.23E+01	**2.30E+01**	2.42E+01	2.31E+01	500		1.42E−11	**1.77E−11**	2.46E−11	1.81E−11
1 000		4.68E+01	**4.78E+01**	4.96E+01	4.79E+01	1 000		1.37E−10	1.58E−10	1.79E−10	1.57E−10
50	F3	2.94E−13	**7.41E−10**	8.27E−09	1.87E−09	50	F13	1.23E−06	**4.12E−06**	3.48E−01	5.05E−02
100		3.28E+01	**3.60E+01**	8.42E+01	4.14E+01	100		2.49E+01	2.89E+01	3.31E+01	2.87E+01
200		1.34E+02	**1.37E+02**	1.82E+02	1.42E+02	200		1.07E+02	**1.10E+02**	1.49E+02	1.11E+02
500		4.32E+02	**4.34E+02**	4.38E+02	4.34E+02	500		3.31E+02	**3.32E+02**	3.72E+02	3.34E+02
1 000		9.28E+02	**9.30E+02**	9.32E+02	9.30E+02	1 000		7.02E+02	**7.05E+02**	7.41E+02	7.08E+02
50	F4	0.00E+00	**0.00E+00**	9.95E−01	3.98E−02	50	F14	1.03E−13	**5.08E−13**	1.34E−12	5.79E−13
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		2.28E−11	**9.93E−11**	9.95E−01	3.98E−02
200		0.00E+00	**0.00E+00**	9.95E−01	3.98E−02	200		1.18E−09	**2.67E−09**	1.99E+00	1.59E−01
500		0.00E+00	**0.00E+00**	1.99E+00	2.39E−01	500		1.92E−08	**2.68E−08**	9.95E−01	2.79E−01
1 000		2.38E−13	9.95E−01	2.98E+00	7.56E−01	1 000		7.16E−08	**9.30E−08**	1.99E+00	3.98E−01
50	F5	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F15	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
50	F6	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F16	2.45E−14	**4.99E−14**	1.36E−13	5.35E−14
100		1.47E−14	**1.47E−14**	1.82E−14	1.48E−14	100		6.95E−13	**1.21E−12**	1.76E−12	1.24E−12
200		2.89E−14	**3.24E−14**	3.60E−14	3.24E−14	200		8.54E−12	1.36E−11	1.72E−11	1.35E−11
500		8.57E−14	8.93E−14	9.28E−14	8.88E−14	500		1.44E−10	1.75E−10	1.95E−10	1.72E−10
1 000		1.82E−13	**1.85E−13**	1.92E−13	1.86E−13	1 000		7.25E−10	8.02E−10	8.98E−10	8.00E−10
50	F7	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F17	4.39E−07	**6.60E−06**	2.36E−01	3.96E−02
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		1.66E−05	**6.28E−02**	2.50E−01	8.98E−02
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		9.17E+00	1.30E+01	1.45E+01	1.26E+01
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		8.54E+01	8.88E+01	9.03E+01	8.84E+01
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		2.11E+02	**2.14E+02**	2.16E+02	2.14E+02
50	F8	1.38E−10	**8.45E−09**	3.13E−07	4.42E−08	50	F18	2.93E−10	**6.11E−10**	1.08E−09	6.30E−10
100		9.28E−09	**2.84E−07**	7.14E−05	8.50E−06	100		7.10E−09	**1.25E−08**	2.45E−08	1.30E−08
200		1.03E−11	**5.35E−08**	1.42E−05	9.33E−07	200		9.94E−08	**1.36E−07**	9.95E−01	3.98E−02
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		1.16E−06	**1.47E−06**	1.79E−06	1.49E−06
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		4.39E−06	**5.13E−06**	9.95E−01	3.98E−02
50	F9	1.47E−10	**3.64E−10**	1.13E−09	4.39E−10	50	F19	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		1.87E−10	**6.01E−10**	2.68E−09	7.32E−10	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		7.27E−12	**7.92E−11**	3.24E−10	9.66E−11	200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
500		4.14E−15	**2.19E−14**	2.06E−13	4.20E−14	500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
50	F10	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	All the results below 1.00E−14 have been approximated to 0.					
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00						
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00						
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00						
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00						

Table 16. Experimental results of rank-GODE for functions F1–F19 at $D = 50$, 100, 200, 500, and 1 000, where the median value is highlighted in **boldface** when it is better than or equal to the mean value in the same function

D	F	Best	Median	Worst	Mean	D	F	Best	Median	Worst	Mean
50	F1	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F11	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		0.00E+00	**4.33E−07**	1.15E−05	2.21E−06
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		2.64E−05	**3.57E−05**	7.15E−05	4.04E−05
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		1.05E−04	**1.71E−04**	2.59E−04	1.73E−04
50	F2	9.10E−01	**2.49E+00**	7.88E+00	2.69E+00	50	F12	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		3.59E−01	**3.65E+00**	1.79E+01	4.74E+00	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		1.84E+01	**2.64E+01**	3.80E+01	2.86E+01	200		0.00E+00	**0.00E+00**	2.59E−14	0.00E+00
500		2.89E+01	4.86E+01	6.32E+01	4.69E+01	500		3.01E−12	**6.40E−12**	1.78E−11	7.04E−12
1 000		2.85E+01	4.48E+01	6.03E+01	4.34E+01	1 000		8.11E−11	**1.40E−10**	2.51E−10	1.49E−10
50	F3	0.00E+00	**0.00E+00**	6.32E−11	3.24E−12	50	F13	4.10E−02	**2.94E−01**	4.32E+00	6.24E−01
100		5.04E−10	**1.04E−01**	1.21E+01	2.22E+00	100		1.73E−01	**8.50E−01**	2.26E+00	8.96E−01
200		4.80E+01	**8.57E+01**	1.39E+02	9.03E+01	200		6.02E+01	**6.59E+01**	1.13E+02	7.63E+01
500		3.38E+02	**3.79E+02**	4.11E+02	3.80E+02	500		2.89E+02	**3.06E+02**	3.30E+02	3.07E+02
1 000		7.95E+02	**8.71E+02**	9.67E+02	8.76E+02	1 000		6.22E+02	**6.78E+02**	7.40E+02	6.80E+02
50	F4	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F14	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		5.32E−14	**1.98E−13**	6.29E−13	2.17E−13
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		4.13E−12	**8.06E−12**	1.25E−11	8.42E−12
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		4.05E−12	7.24E−12	1.02E−11	7.18E−12
50	F5	0.00E+00	**0.00E+00**	7.40E−03	8.88E−04	50	F15	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		0.00E+00	**0.00E+00**	1.48E−02	5.91E−04	200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
50	F6	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F16	0.00E+00	**0.00E+00**	5.48E−14	0.00E+00
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		1.16E−13	**3.37E−13**	1.12E−12	4.21E−13
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		2.98E−12	**5.54E−12**	1.25E−11	5.96E−12
500		2.84E−14	**3.20E−14**	6.75E−14	3.44E−14	500		7.88E−11	1.42E−10	1.92E−10	1.38E−10
1 000		4.26E−14	**4.97E−14**	1.28E−13	5.41E−14	1 000		4.13E−10	**6.66E−10**	1.00E−09	6.78E−10
50	F7	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F17	8.12E−08	**2.35E−01**	4.71E−01	2.49E−01
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		2.82E−01	**6.94E−01**	1.53E+00	7.19E−01
200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	200		3.85E−01	**7.39E−01**	1.10E+00	7.54E−01
500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	500		4.44E+01	**5.20E+01**	5.96E+01	5.24E+01
1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	1 000		1.73E+02	**1.80E+02**	1.85E+02	1.80E+02
50	F8	0.00E+00	**0.00E+00**	5.48E−14	0.00E+00	50	F18	5.10E−11	**1.76E−10**	5.92E−10	2.40E−10
100		3.89E−07	**2.41E−06**	9.87E−06	3.34E−06	100		9.64E−10	**2.41E−09**	5.62E−09	2.47E−09
200		1.58E−01	**5.69E−01**	3.68E+00	6.94E−01	200		1.74E−08	**1.99E−08**	7.88E−08	2.39E−08
500		7.36E+02	**1.21E+03**	2.04E+03	1.32E+03	500		2.05E−10	**3.83E−10**	5.98E−10	3.99E−10
1 000		1.33E+04	**1.55E+04**	1.93E+04	1.59E+04	1 000		1.17E−08	**1.51E−08**	2.11E−08	1.62E−08
50	F9	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	50	F19	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
200		0.00E+00	**0.00E+00**	5.27E−06	7.09E−07	200		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
500		1.97E−05	**4.39E−05**	7.60E−05	4.44E−05	500		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
1 000		9.73E−05	**1.60E−04**	4.11E−04	1.80E−04	1 000		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00
50	F10	0.00E+00	**0.00E+00**	0.00E+00	0.00E+00	All the results below 1.00E−14 have been approximated to 0.					
100		0.00E+00	**0.00E+00**	0.00E+00	0.00E+00						
200		0.00E+00	**0.00E+00**	0.00E+00	1.05E+00						
500		0.00E+00	**0.00E+00**	1.05E+00	1.26E−01						
1 000		0.00E+00	**0.00E+00**	1.05E+00	8.40E−02						

Table 17. Experimental results of rank-GaDE for functions F1–F19 at $D = 50$, 100, 200, 500, and 1 000, where the median value is highlighted in **boldface** when it is better than or equal to the mean value in the same function

From the results it is clear to see that in the majority of the cases the median values are much better than or equal to the corresponding mean values. For example, for rank-GODE there are 85 out of 95 cases where the median values are much better than or equal to the corresponding mean values. The results show that the ranking-based DEs sometimes occasionally converge to the local optima in some functions. But, in general, our proposed ranking-based DEs are able to obtain good solutions within the specified Max_NFFEs.

Acknowledgments

The source codes of GODE and GaDE are obtained available online at `http://sci2s.ugr.es/EAMHCO/contributionsSOCO.php`.

REFERENCES

[1] Alcalá-Fdez, J.—Sánchez, L.—García, S.—del Jesus, M. J.—Ventura, S.—Garrell, J. M.—Otero, J.—Romero, C.—Bacardit, J.—Rivas, V. M.—Fernández, J. C.—Herrera, F.: KEEL: A Software Tool to Assess Evolutionary Algorithms for Data Mining Problems. Soft Computing, Vol. 13, 2009, No. 3, pp. 307–318, doi: 10.1007/s00500-008-0323-y.

[2] Auger, A.—Hansen, N.: A Restart CMA Evolution Strategy with Increasing Population Size. The 2005 IEEE Congress on Evolutionary Computation, 2005, Vol. 2, pp. 1769–1776, doi: 10.1109/CEC.2005.1554902.

[3] Brest, J.—Sepesy Maučec, M.: Self-Adaptive Differential Evolution Algorithm Using Population Size Reduction and Three Strategies. Soft Computing – A Fusion of Foundations, Methodologies and Applications, Vol. 15, 2011, No. 11, pp. 2157–2174, doi: 10.1007/s00500-010-0644-5.

[4] Brest, J.—Zamuda, A.—Fister, I.—Sepesy Maučec, M.: Large Scale Global Optimization Using Self-Adaptive Differential Evolution Algorithm. 2010 IEEE Congress on Evolutionary Computation (CEC), 2010, pp. 1–8, doi: 10.1109/CEC.2010.5585927.

[5] Das, S.—Suganthan, P. N.: Differential Evolution: A Survey of the State-of-the-Art. IEEE Transactions on Evolutionary Computation, Vol. 15, 2011, No. 1, pp. 4–31, doi: 10.1109/TEVC.2010.2059031.

[6] Deb, K.—Pratap, A.—Agarwal, S.—Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation, Vol. 6, 2002, No. 2, pp. 182–197, doi: 10.1109/4235.996017.

[7] Eshelman, L. J.—Schaffer, J. D.: Real-Coded Genetic Algorithms and Interval-Schemata. In: Whitley, D. L. (Ed.): Foundation of Genetic Algorithms, Vol. 2, 1993, pp. 187–202.

[8] García, S.—Fernández, A.—Luengo, J.—Herrera, F.: A Study of Statistical Techniques and Performance Measures for Genetics-Based Machine Learning: Accuracy and Interpretability. Soft Computing, Vol. 13, 2009, No. 10, pp. 959–977, doi: 10.1007/s00500-008-0392-y.

[9] García, S.—Molina, D.—Lozano, M.—Herrera, F.: A Study on the Use of Non-Parametric Tests for Analyzing the Evolutionary Algorithms' Behaviour: A Case Study on the CEC'2005 Special Session on Real Parameter Optimization. Journal of Heuristics, Vol. 15, 2009, No. 6, pp. 617–644, doi: 10.1007/s10732-008-9080-4.

[10] García-Martínez, C.—Rodríguez, F.—Lozano, M.: Role Differentiation and Malleable Mating for Differential Evolution: An Analysis on Large-Scale Optimisation. Soft Computing – A Fusion of Foundations, Methodologies and Applications, Vol. 15, 2011, No. 11, pp. 2109–2126, doi: 10.1007/s00500-010-0641-8.

[11] Gong, W.—Cai, Z.: Differential Evolution with Ranking-Based Mutation Operators. IEEE Transactions on Cybernetics, Vol. 43, 2013, No. 6, pp. 2066–2081.

[12] Herrera, F.—Lozano, M.—Molina, D. (Eds.): Special Issue on Scalability of Evolutionary Algorithms and Other Metaheuristics for Large Scale Continuous Optimization Problems. Soft Computing – A Fusion of Foundations, Methodologies and Applications, Vol. 15, 2011, No. 11. Published online: 2010.

[13] Herrera, F.—Lozano, M.—Molina, D.: Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and Other Metaheuristics for Large Scale Continuous Optimization Problems. Technical report, Department of Computer Science and Artificial Intelligence, University of Granada, 2010.

[14] Korošec, P.—Šilc, J.: Using Stigmergy to Solve Numerical Optimization Problems. Computing and Informatics, Vol. 27, 2008, No. 3, pp. 377–402.

[15] LaTorre, A.—Muelas, S.—Peña, J.-M.: A MOS-Based Dynamic Memetic Differential Evolution Algorithm for Continuous Optimization: A Scalability Test. Soft Computing, Vol. 15, 2011, No. 11, pp. 2187–2199.

[16] Leong, W. J.—Hassan, M. A.: Scaled Memoryless Symmetric Rank One Method for Large-Scale Optimization. Applied Mathematics and Computation, Vol. 218, 2011, No. 2, pp. 413–418.

[17] Li, H.-M.—Zhang, K.-C.: A Decomposition Algorithm for Solving Large-Scale Quadratic Programming Problems. Applied Mathematics and Computation, Vol. 173, 2006, No. 1, pp. 394–403.

[18] Lozano, M.—Molina, D.—Herrera, F.: Editorial Scalability of Evolutionary Algorithms and Other Metaheuristics for Large-Scale Continuous Optimization Problems. Soft Computing – A Fusion of Foundations, Methodologies and Applications, Vol. 15, 2011, No. 11, pp. 2085–2087.

[19] Muelas, S.—La Torre, A.—Peña, J.-M.: A Memetic Differential Evolution Algorithm for Continuous Optimization. Ninth International Conference on Intelligent Systems Design and Applications (ISDA'09), 2009, pp. 1080–1084.

[20] Neri, F.—Tirronen, V.: Recent Advances in Differential Evolution: A Survey and Experimental Analysis. Artificial Intelligence Review, Vol. 33, 2010, No. 1-2, pp. 61–106.

[21] Noman, N.—Iba, H.: Accelerating Differential Evolution Using an Adaptive Local Search. IEEE Transactions on Evolutionary Computation, Vol. 12, 2008, No. 1, pp. 107–125.

[22] Price, K.—Storn, R. M.—Lampinen, J. A.: Differential Evolution: A Practical Approach to Global Optimization. Springer-Verlag, Berlin, 2005.

[23] Qin, A. K.—Huang, V. L.—Suganthan, P. N.: Differential Evolution Algorithm with Strategy Adaptation for Global Numerical Optimization. IEEE Transactions on Evolutionary Computation, Vol. 13, 2009, No. 2, pp. 398–417.

[24] Runarsson, T. P.—Yao, X.: Stochastic Ranking for Constrained Evolutionary Optimization. IEEE Transactions on Evolutionary Computation, Vol. 4, 2000, No. 3, pp. 284–294, doi: 10.1109/4235.873238.

[25] Shaw, C.—Williams, K. S.—Assassa, R. P.: Patients' Views of a New Nurse-Led Continence Service. Journal of Clinical Nursing, Vol. 9, 2003, No. 4, pp. 574–582.

[26] Stanarevic, N.: Hybridizing Artificial Bee Colony (ABC) Algorithm with Differential Evolution for Large Scale Optimization Problems. International Journal of Mathematics and Computers in Simulation, Vol. 1, 2012, No. 6, pp. 194–202.

[27] Storn, R.—Price, K.: Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical report TR-95-012, Berkeley, CA, 1995.

[28] Storn, R.—Price, K.: Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. Journal of Global Optimization, Vol. 11, 1997, No. 4, pp. 341–359.

[29] Tang, K.—Li, X.—Suganthan, P. N.—Yang, Z.—Weise, T.: Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2009.

[30] Tang, K.—Yao, X.—Suganthan, P. N.—MacNish, C.—Chen, Y. P.—Chen, C. M.—Yang, Z.: Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007.

[31] Wang, H.—Ni, Q.: A New Method of Moving Asymptotes for Large-Scale Unconstrained Optimization. Applied Mathematics and Computation, Vol. 203, 2008, No. 1, pp. 62–71.

[32] Wang, H.—Wu, Z.—Rahnamayan, S.: Enhanced Opposition-Based Differential Evolution for Solving High-Dimensional Continuous Optimization Problems. Soft Computing – A Fusion of Foundations, Methodologies and Applications, Vol. 15, 2011, No. 11, pp. 2127–2140.

[33] Weber, M.—Neri, F.—Tirronen, V.: Shuffle or Update Parallel Differential Evolution for Large-Scale Optimization. Soft Computing – A Fusion of Foundations, Methodologies and Applications, Vol. 15, 2011, No. 11, pp. 2089–2107.

[34] Yang, Z.—Tang, K.—Yao, X.: Differential Evolution for High-Dimensional Function Optimization. IEEE Congress on Evolutionary Computation (CEC 2007), 2007, pp. 3523–3530.

[35] Yang, Z.—Tang, K.—Yao, X.: Large Scale Evolutionary Optimization Using Cooperative Coevolution. Information Sciences, Vol. 178, 2008, No. 15, pp. 2985–2999.

[36] Yang, Z.—Tang, K.—Yao, X.: Scalability of Generalized Adaptive Differential Evolution for Large-Scale Continuous Optimization. Soft Computing – A Fusion of Foundations, Methodologies and Applications, Vol. 15, 2011, No. 11, pp. 2141–2155.

[37] Yu, G.—Zhao, Y.—Wei, Z.: A Descent Nonlinear Conjugate Gradient Method for Large-Scale Unconstrained Optimization. Applied Mathematics and Computation, Vol. 187, 2007, No. 2, pp. 636–643.

[38] Zhang, J.—Sanderson, A. C.: JADE: Adaptive Differential Evolution with Optional External Archive. IEEE Transactions on Evolutionary Computation, Vol. 13, 2009, No. 5, pp. 945–958.

[39] Zhao, S.-Z.—Suganthan, P. N.—Das, S.: Self-Adaptive Differential Evolution with Multi-Trajectory Search for Large-Scale Optimization. Soft Computing – A Fusion of Foundations, Methodologies and Applications, Vol. 15, 2011, No. 11, pp. 2175–2185.

Li Guo received his Ph.D. degree from the China University of Geosciences, Wuhan, China, in 2015. He is currently working at CCCC Infrastructure Maintenance Group Co., Ltd., China.

Xiang Li received the Ph.D. degree from the China University of Geosciences, Wuhan, China, in 2008. He is currently Associate Professor with School of Computer Science, China University of Geosciences. He has published over 30 research papers in journals and international conferences.

Wenyin Gong received his B.Eng., M.Eng. and Ph.D. degrees in computer science from the China University of Geosciences, Wuhan, China, in 2004, 2007, and 2010, respectively. He is Professor with the School of Computer Science, China University of Geosciences. He has published over 50 research papers in journals and international conferences. His current research interests include evolutionary algorithms, evolutionary optimization, and their applications. He served as a reviewer for over 20 international journals, such as the IEEE Transactions on Evolutionary Computation, the IEEE Transactions on Cybernetics, the IEEE Computational Intelligence Magazine, the ACM Transactions on Intelligent Systems and Technology, the Information Sciences, the European Journal of Operational Research, the Applied Soft Computing, and the International Journal of Hydrogen Energy.

Computing and Informatics, Vol. 37, 2018, 76–108, doi: 10.4149/cai_2018_1_76

ENERGY AWARE RESOURCE ALLOCATION FOR CLOUDS USING TWO LEVEL ANT COLONY OPTIMIZATION

Ashok Kumar, Rajesh Kumar, Anju Sharma

Department of Computer Science & Engineering
Thapar University
Patiala-147004, India
e-mail: ashok.khunger@gmail.com, {rakumar, anju.sharma}@thapar.edu

Abstract. In cloud environment resources are dynamically allocated, adjusted, and deallocated. When to allocate and how many resources to allocate is a challenging task. Resources allocated optimally and at the right time not only improve the utilization of resources but also increase energy efficiency, provider's profit and customers' satisfaction. This paper presents ant colony optimization (ACO) based energy aware solution for resource allocation problem. The proposed energy aware resource allocation (EARA) methodology strives to optimize allocation of resources in order to improve energy efficiency of the cloud infrastructure while satisfying quality of service (QoS) requirements of the end users. Resources are allocated to jobs according to their QoS requirements. For energy efficient and QoS aware allocation of resources, EARA uses ACO at two levels. First level ACO allocates Virtual Machines (VMs) resources to jobs whereas second level ACO allocates Physical Machines (PMs) resources to VMs. Server consolidation and dynamic performance scaling of PMs are employed to conserve energy. The proposed methodology is implemented in CloudSim and the results are compared with existing popular resource allocation methods. Simulation results demonstrate that EARA achieves desired QoS and superior energy gains through better utilization of resources. EARA outperforms major existing resource allocation methods and achieves up to 10.56 % saving in energy consumption.

Keywords: Energy efficiency, resource allocation in cloud, dynamic voltage frequency scaling, ant colony optimization, quality of service

1 INTRODUCTION

Cloud computing is a paradigm that has huge potential in enterprise and business. It has a large pool of configurable resources which can be acquired and used on demand [1, 20]. The acquired resources can be accessed over the network. In cloud, everything is provided as a service. Cloud has three service models, namely: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). In IaaS, fundamental computing resources like processing, storage, networks, etc. are provisioned to the consumers for deployment and execution of arbitrary software [20]. The resources are provisioned and allocated according to consumers' demands. Furthermore, resource allocation mechanism is to guarantee that requirements of all applications are suitably met. Due to all these reasons, resource allocation in cloud computing is one of the important challenges. Apart from resource allocation for performance, i.e., allocating sufficient resources to user applications in order to satisfy QoS parameters, another challenge posed to researchers and industry is to minimize the energy consumption and carbon footprints. According to Koomey [18]: *"Total data center power consumption from servers, storage, communications, cooling, and power distribution equipment accounts for 1.7–2.2 % of total electricity used in U.S. in 2010"*. With their enormous appetite for energy, today's data centers emit as much carbon dioxide as whole of Argentina. If left on their current path, data center carbon dioxide output will quadruple by the year 2020 [15]. While the cloud energy appetite is growing quickly, industrial organizations and researchers are finding ways to reduce the energy consumption. Several methods to reduce the energy consumption of a data center exists. Data centers infrastructure is generally over-provisioned to sustain availability of resources during peak hours. But due to dynamic nature of load average resource utilization is approximately 15–20% [26, 15]. Energy efficiency can be improved by better management of resources. One such area for reduction in energy consumption is efficient resource allocation. A large extent of energy can also be saved by server consolidation and turning off idle servers. Sometimes consolidation is not economically feasible due to constraints such as communication cost of migration, QoS violations due to interruption in service while consolidating, or unavailability of PM with sufficient free resources where the VM can be migrated. In such cases energy consumed by PMs can be saved by adjusting its operating voltage/frequency.

In this paper, we proposed EARA methodology that uses ACO for resource allocation. Resources are allocated to the jobs with the goal to minimize total cost of execution, total execution time and total energy consumption while satisfying QoS requirements of the end users. Each QoS parameter of the job is associated with some weight value that indicates its priority over the others. ACO is applied at two levels for efficient allocation of resources. The first level ACO allocates VM resources to jobs whereas the second level ACO allocates PM resources to VMs. Server consolidation and dynamic performance scaling is employed to conserve energy. Dynamic performance scaling is used when server consolidation is economically unfeasible because of high communication cost, QoS violations due to interruption

in service or unavailability of destination machine with sufficient free resources. The proposed methodology is implemented in CloudSim and its effectiveness is evaluated with jobs having different resource demands and QoS requirements.

The rest of the paper is organized as follows: Related work is presented in Section 2. Section 3 discusses the energy aware resource allocation methodology and its mathematical representation. Section 4 explains the technique used for EARA that is ant colony optimization. Comparative performance analysis of EARA with the first fit decreasing (FFD), and multi-objective grouping genetic algorithm (MGGA) is presented in Section 5. Conclusion and the scope of future work is detailed in Section 6.

2 RELATED WORK

Beloglazov et al. [3] proposed power efficient and QoS aware resource allocation heuristics. An algorithm for minimization of number of VM migrations is also proposed. Upper and lower threshold utilization levels are set to detect overloaded and underloaded machines. When the resource utilization of a particular server falls below the lower threshold value, all the VMs running on the machine are shifted to some other machine. If utilization of a machine is above upper threshold, one or more VMs are shifted to other machines to keep the utilization between the threshold values. They proposed algorithms for single core machines. In real cloud environment heterogeneous multicore systems are used. Gao et al. [11] proposed the linear programming based multi-objective ant colony based system for virtual machine placement to minimize resource wastage and power consumption. Initial pheromone value is assigned to VM-host movement. The pheromone value indicates probability of a host to be selected for allocation of VM under consideration. The authors used only CPU processing speed and memory requirements of VM while allocating resources to the VMs. Kinger et al. [17] proposed event driven prediction based proactive temperature aware VM scheduling to keep temperature of a server below the specified upper threshold temperature. Temperature predictor constantly monitors temperature of the physical machine. The authors used "unified list" to store current as well as threshold temperature of each node. The unified list is updated after a fixed interval of time, which would cause network congestion, performance degradation and limited scalability. Quarati et al. [23] proposed two level brokering algorithm for hybrid cloud with the objective to maximize broker's revenue and user satisfaction. First level scheduler schedules the requested services on private or public cloud based on reserved quota of private resources. The authors proposed three first level scheduling techniques namely feasible, static reservation, and maximum occupation. Second level of scheduling uses less consuming resource and dynamic less consuming resource techniques to allocate resources to the services. The requested services are run on physical machine having maximum availability of free resources. The proposed technique causes uneven distribution of workload among

the servers and overloading of high performance machines. Overloading results in creation of hot spots and increase in rate of failure. Lee et al. [19] proposed performance analysis based resource allocation strategy for green cloud. Every PM in a data center is assigned a performance value based on CPU processing speed, number of cores, and memory capacity relative to machine having maximum number of cores, CPU processing speed, and memory capacity. A PM is allocated to a VM if its performance value fits best the VM requirements. The proposed method results in hot spots and in overloading of high performance machines. The improper distribution of load among servers would cause wastage of energy. Raycroft et al. [24] analyzed the effect of global VM allocation policy on energy consumption. Simulation is performed for the same type of applications but real cloud hosts diverse type of applications. Communication cost between VMs and QoS is not taken into account. Moreover, the authors proposed movement of VMs between regions which is impractical in case of large sized VM. Feller et al. [10] proposed multi-dimensional ant colony optimization based workload consolidation algorithm. The algorithm uses resource utilization history to predict future resource demands and dynamically overbooks the resources. The authors have tested the algorithm on PM having the same capacity, i.e. in homogeneous environment. Real cloud environment is heterogeneous in nature, having machines with different resource capacity. Gao et al. [11] proposed multi-objective ant colony system algorithm for virtual machine placement that minimizes total resource wastage and power consumption. The algorithm attempts to utilize server to its full capacity which would result in creation of hot spots and increase in number of service-level agreement (SLA) violations. Moreover, using server near full capacity causes more heat dissipation which results in decrease in server reliability. Nathani et al. [21] proposed modified immediate and advance reservation algorithms for deadline sensitive leases. The proposed algorithms try to schedule new lease as a deadline sensitive lease in a single or multiple time slots. If a new lease cannot be scheduled in a single or multiple time slots, the algorithm reschedules the already scheduled deadline sensitive leases to make a room for new lease. In case, rescheduling fails to generate deadline constrained schedule, then backfilling is applied to accommodate new lease. The drawback of the proposed algorithm is its high lease preemption rate which, in turn, increases the allocation overhead. Ant colony optimization technique for assigning real-time tasks to heterogeneous processors is proposed by Chen et al. [7]. Local search technique is applied to improve energy efficiency of the feasible assignment solution generated by the proposed assignment algorithm. The authors have claimed that their algorithm saves 15.8 % energy over prototyped version of ant colony optimization. Huang et al. [14] proposed adaptive sub-optimal resource management scheme. In the proposed scheme, global resource allocation module uses remaining resource table and resource utilization rate table to estimate number of VMs required to provide desired level of service. Genetic algorithm (GA) is proposed for reallocation of resources to achieve better performance. The proposed technique suffers from a single point failure. Moreover, centralized global resource al-

location module, remaining resource table, and resource utilization rate table will cause performance degradation when the number of requests is large. In [6, 16], the authors proposed methods to calculate energy consumption of a PM. Castañé et al. [6] modeled four basic subsystems: computing, memory, storage, and network of a PM to calculate the amount of energy consumed by it. Kim et al. [16] proposed a methodology for estimating energy consumption of a VM based on its in-processor events without using a dedicated energy measuring instrument. Performance counters available in modern processor are used to keep the track of specific type floating point instructions issued by VM. Based on the instruction type and its count, the energy consumption of VM is estimated. The number of performance counters available in a processor is limited, so limited number of instructions can be tracked, and that results in erroneous estimation of energy consumption. The authors also proposed the energy credit scheduler. The proposed scheduler assigns resources to the VM based on its energy credit. The resources allocated to VM are preempted when its energy credit vanishes. Garg et al. [12] proposed green cloud computing framework for reducing carbon footprint without sacrificing QoS. The authors used Green Offer Directory and Carbon Emission Directory to offer green services to the users. The Carbon Emission Directory maintains data related to the energy efficiency of cloud services. Based on the information in these two directories, the cost and carbon footprint of leasing a particular cloud service are calculated. The providers are supposed to publish carbon footprint and energy efficiency of their services in public directories. There is no check on the data that is published by the providers. The service provider can publish manipulated data in order to earn more and for building its reputation in the market. Xu and Fortes [28] proposed multi-objective VM allocation algorithm. The authors have taken CPU, and memory parameters for VMs and have claimed reduction in power consumption, thermal dissipation costs, and resource wastage. Disk utilization and inter VM communication cost is not taken into consideration. Wu et al. [27] proposed energy efficient priority job scheduling for cloud computing. The requirements of a job are given in terms of maximum and minimum CPU frequencies. Every server is assigned some weight based on its performance/Watt. Servers are selected for jobs according to assigned weight and SLA level required by the the users. A job is assigned to VM running on a selected server that meets its requirements. Frequency of the server is then tuned to reduce energy consumption. However, the authors have not considered memory, input/output and other requirements of the job. In [25, 4, 22, 13], the authors proposed energy-conscious consolidation heuristics in order to conserve energy and maximize resource utilization without affecting the performance of the system. Takeda and Takemura [25] proposed ranking of physical servers for consolidation and VM placement. Servers with higher priorities are considered more reliable than the servers with lower priority value. Higher priorities are assigned to newly installed servers. The main drawback of the server ranking strategy is that the priorities are to be assigned to the server by the operator manually.

3 ENERGY AWARE RESOURCE ALLOCATION METHODOLOGY

The proposed EARA methodology allocates resources to jobs using ant colony optimization. It utilizes the resources efficiently to save energy besides fulfilling the operational demands of the jobs. Each job has some resource and QoS requirements. Each QoS parameter of a job is associated with a weight value. EARA allocates resources to jobs in accordance with their resources demands and weight values of QoS parameters. The key features of proposed EARA are:

- It deliberately assigns resources to jobs in order to improve the utilization of resources, thereby increases the energy efficiency of the cloud infrastructure.
- Idle PMs are switched to sleep mode to conserve energy.
- Monitoring of utilization of resources viz. processor, memory, network bandwidth of a PM for energy efficient resource allocation and management.
- Dynamic performance scaling of servers to conserve energy.
- Server consolidation to minimize number of active servers.

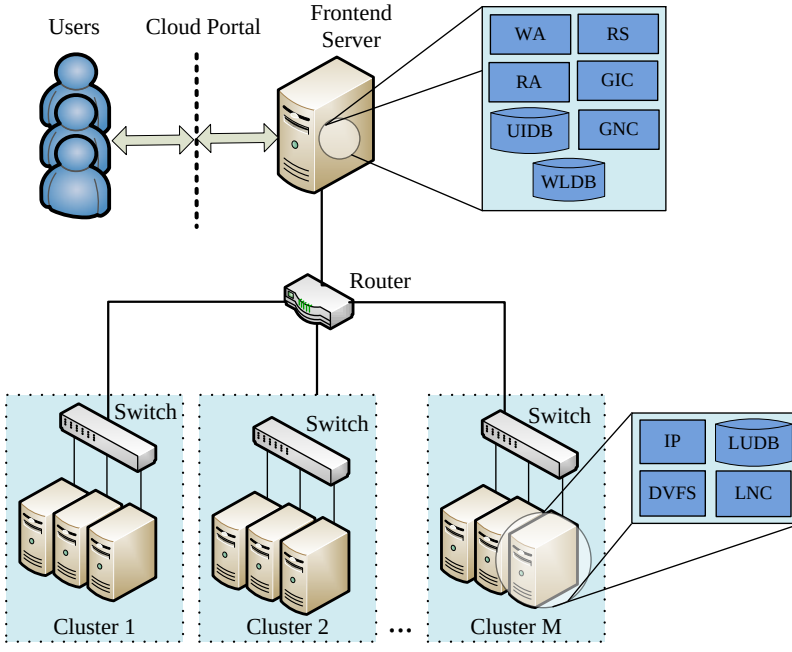


Figure 1. Energy aware resource allocation

The various components of EARA as shown in Figure 1 are:

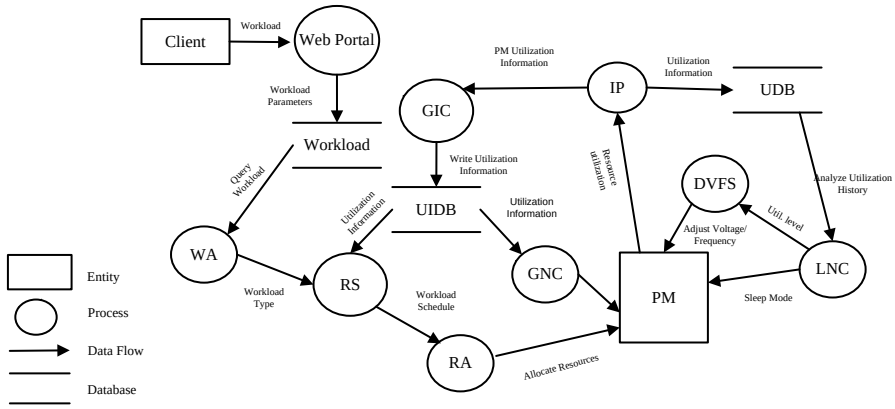


Figure 2. Data flow representation for energy aware resource allocation methodology

Cloud Portal: It provides an interface to the cloud users to input their jobs and desired QoS.

Workload Analyzer (WA): It analyses QoS requirements of the jobs and classifies them into different classes using k-means cluster algorithm.

Resource Scheduler (RS): It generates schedule of jobs to be executed.

Resource Allocation (RA): It applies ant colony optimization to allocate jobs to VMs and VMs to PMs. Resources are allocated in accordance with resource demands and weight values of QoS parameters associated with a job.

Global Information Collector (GIC): It receives the resource utilization data from information probes (IP) of every PM and stores it in utilization information database (UIDB).

Utilization Information Database (UIDB): Resource utilization data of every PM is kept in UIDB for future resource allocation and VM migration decisions.

Global Node Controller (GNC): It initiates live migration of VMs running on a PM when resource utilization of PM violates lower green threshold (LGT) or upper green threshold (UGT) limit.

Workload Database (WLDB): It stores the information associated with each job.

Information Probes (IP): It monitors the utilization of resources viz. processor, memory, network bandwidth of a PM and records observed values in local utilization database (LUDB).

Local Utilization Database (LUDB): It keeps record of utilization of resources of PMs.

Dynamic Voltage Frequency Scaling (DVFS): It adjusts the voltage and frequency of the PM in order to save power and to reduce heat dissipation. The

voltage/frequency of PM is adjusted in accordance to resource demands of the VMs/jobs running over it.

Local Node Controller (LNC): It switches the PM to sleep mode if it is found idle for specific period of time.

The overview of various components of EARA and information flow among them is depicted through data flow diagram using Yourdon/DeMarco notation [35] (Figure 2). The directional lines (arrows) show information exchange between components. They do not give any information about timing and sequence of execution of processes. The client inputs jobs from cloud portal. The frontend server stores the job's requirements in workload database for analyzing, energy efficient scheduling, and allocation. Each job has a different resource and QoS requirements. For example, batch job may require storage and computing resources (i.e. memory, CPU), whereas for online job, network bandwidth may be more critical. WA analyses the jobs and classifies them into different categories based on weight values of their QoS parameters. Jobs are then mapped to VMs. Resources are allocated to VMs and then scheduled on PMs. Once the VM is deployed on the PM, its resource utilization is monitored by IP after a customizable fixed interval and observed values are stored in LUDB. When utilization of PM remains below the lower green threshold (LGT) for two consecutive monitoring intervals, either of the two methods is applied to save energy consumed by a PM. First, offloading PM by migrating the VM running on it to some other PM and then switching it to sleep mode. Sometimes, it is economically unfeasible to migrate VM running on PM to some other PM, either due to high migration cost, or deadline violation due to migration, or unavailability of PM that can host the VM due to insufficient available free resources. When VM migration is not economically feasible, the second method, that is dynamic performance scaling of PM, is applied. In dynamic performance scaling, voltage/frequency of the PM is intentionally varied to change its performance. DVFS helps to cut power cost but at the price of a slower job execution. It is applied when the user deadlines can be achieved at slower execution speed.

For implementation purposes and subsequent evaluation of EARA, mathematical equations for energy aware resource allocation, DVFS, and fitness function are formulated as discussed in the forthcoming subsections.

3.1 Mathematical Modeling of Energy Aware Resource Allocation

EARA contemplates the energy efficient usage of resources. EARA is considered from both the provider's and clients' point of view. It minimizes:

1. total energy consumption for the benefit of service provider, and
2. total execution cost and total time of execution for the end users' satisfaction.

The following assumptions are taken into considerations while formulating mathematical representation of EARA.

1. Physical nodes can be unilaterally switched on/off, or put to sleep mode as and when required.
2. Every PM supports advanced configuration and power interface (ACPI). The operating system can adjust voltage and frequency to any level supported by the hardware.
3. PMs and VMs are characterized by processing speed, memory, and network bandwidth.
4. Transition from one voltage/frequency level to other is instant.
5. Energy consumed by PM during sleep mode is negligible.
6. All the cores of a PM can be operated on the same voltage/frequency level at a time.

Total power consumption of a PM [2] at any instant is given by Equation (1).

$$\begin{aligned} P_{total} &= P_{dynamic} + P_{static} \\ &= \underbrace{ACV^2f}_{\text{DPC}} + \underbrace{V * I_{leak}}_{\text{SPC}}, \end{aligned} \tag{1}$$

where A is switching activity, C is capacitance, V is voltage, I_{leak} is the leakage current, and f is the clock frequency applied to the cores of PM. P_{total}, P_{static}, and $P_{dynamic}$ are total power consumption, static power consumption (SPC), and dynamic power consumption (DPC) of a PM, respectively. SPC is due to leakage current that is present in any active circuit, and it is independent of clock frequency and usage scenario. It can be reduced by switching PM to sleep mode [2]. Whereas, DPC is due to circuit activity and it depends on usage scenario or resource utilization. EARA reduces SPC by switching PM to sleep mode as and when required, whereas DPC is optimized by efficient utilization of resources as explained below.

Suppose s is processing speed, and p is DPC of a PM. Then, $s \propto f$, and $f \propto V$, which implies $p \propto f^3$ and $p \propto s^3$ [34]. Suppose operational requirements of N_t tasks (jobs) can be fulfilled by N_v number of VMs, and each VM has resources to fulfill needs of n_g tasks. R_g is execution requirement and E_{ijg} total energy consumption of the tasks deployed on VM g that is instantiated on PM j of cluster i. If r_{ij}^q is the execution requirement of task q on this PM, then the execution time t_{ij}^q of the task q on this PM with power p_{ij}^q and processing speed s_{ij}^q can be calculated from Equation (2).

$$t_{ij}^q = \frac{r_{ij}^q}{s_{ij}^q} = \frac{r_{ij}^q}{\left(p_{ij}^q\right)^{\frac{1}{3}}}, \tag{2}$$

and the energy consumed by this PM to execute task is given by e_{ij}^q as shown in Equation (3).

$$e_{ij}^q = p_{ij}^q.t_{ij}^q = r_{ij}^q \left(p_{ij}^q\right)^{\frac{2}{3}} = r_{ij}^q \left(s_{ij}^q\right)^2. \tag{3}$$

With time constraint T_g, the objective is:

1. to minimize energy consumed (E_{ijg}) by VM g, and
2. the execution time of all tasks $T_{ijg}= t_{ij}^1+t_{ij}^2+\ldots+t_{ij}^{n_g}$ should not exceed deadline time T_g.

Energy consumption (E_{ijg}) and execution time (T_{ijg}) are calculated as shown in Equations (4) and (5).

$$E_{ijg}\left(p_{ij}^1, p_{ij}^2, \ldots, p_{ij}^n\right) = r_{ij}^1 {p_{ij}^1}^{\frac{2}{3}} + r_{ij}^2 {p_{ij}^2}^{\frac{2}{3}} + \ldots + r_{ij}^{n_g} {p_{ij}^{n_g}}^{\frac{2}{3}}, \tag{4}$$

and

$$T_{ijg}\left(p_{ij}^1, p_{ij}^2, \ldots, p_{ij}^n\right) = \frac{r_{ij}^1}{{p_{ij}^1}^{\frac{1}{3}}} + \frac{r_{ij}^2}{{p_{ij}^2}^{\frac{1}{3}}} + \ldots + \frac{r_{ij}^{n_g}}{{p_{ij}^{n_g}}^{\frac{1}{3}}} \leq T. \tag{5}$$

Both the energy consumption and execution time are functions of $p_{ij}^1, p_{ij}^2, \ldots, p_{ij}^{n_g}$. So, solving Equations (4) and (5) using Lagrange multiplier system [37], we get execution time as represented in Equation (6), and energy consumption, as shown in Equation (7).

$$T_{ijg} = \frac{R_g^{\frac{3}{2}}}{\sqrt{E_{ijg}}}, \tag{6}$$

$$E_{ijg} = P_{ijg}T_{ijg} = \left(\frac{R_g}{T_{ijg}}\right)^3 T_{ijg} = \frac{R_g^3}{(T_{ijg})^2} \tag{7}$$

where $R_g = r_{ij}^1 + r_{ij}^2 + \ldots + r_{ij}^{n_g}$ is the total execution requirement of all the tasks deployed on VM g.
Total energy consumption (TEC) by all the VMs can be calculated as shown in Equation (8).

$$TEC = \sum_{g=1}^{N_v} E_{ijg}. \tag{8}$$

Total execution time (TET) of N_t tasks running on N_v VMs is calculated as shown in Equation (9).

$$TET = T_{ij1} + T_{ij2} + \ldots + T_{ijN_v}. \tag{9}$$

In order to minimize TET and TEC, every VM has to be carefully mapped to suitable PM of a cluster.

A set $CL = \{cl_i \mid 1 \leq i \leq N_c\}$ of clusters is considered. Each cluster i has a pool of physical machines $PM_i = \{h_{ij} \mid 1 \leq j \leq H_i\}$. Here, PM j of cluster i has computing power represented by $h_{ij} = \{hs_{ij}, hm_{ij}, hn_{ij}\}$, where hs_{ij}, hm_{ij}, and hn_{ij} are CPU processing speed, memory, and network bandwidth, respectively. VM computing requirements are represented by $v_g = \{vs_g, vm_g, vn_g\}$, where vs_g, vm_g, and vn_g are processing speed, memory, and network bandwidth of VM g, respectively. The symbolic notations used in mathematical formulation of EARA are depicted in Table A1 of Appendix A.

Suppose C_{ijg} is total cost (Processing cost + Memory cost + Bandwidth cost) of running VM g on PM j of cluster i, x_{ijg} equal to 1 if VM g is assigned to PM j of cluster i and 0 otherwise.

Execution cost (EC_{ij}) of all the VMs running on PM j of cluster i can be calculated from Equation (10):

$$EC_{ij} = C_{ijg} * x_{ijg}, \forall g | x_{ijg=1}, \tag{10}$$

and total cost of execution (COE) of all the VMs can be evaluated using Equation (11):

$$COE = \sum_{i=1}^{N_c} \sum_{j=1}^{H_i} EC_{ij}. \tag{11}$$

A VM should be assigned to PM of a cluster in such a way that all the three conditions, represented by Equations (12), (13), and (14) are satisfied.

$$\sum_{g=1}^{N_v} vs_{ijg} * x_{ijg} \leq hs_{ij} * CPU_{UGT}, \qquad \forall i, j | x_{ijg} = 1, \tag{12}$$

$$\sum_{g=1}^{N_v} vm_{ijg} * x_{ijg} \leq hm_{ij} * MEM_{UGT}, \qquad \forall i, j | x_{ijg} = 1, \tag{13}$$

$$\sum_{g=1}^{N_v} vn_{ijg} * x_{ijg} \leq hn_{ij} * BW_{UGT}, \qquad \forall i, j | x_{ijg} = 1. \tag{14}$$

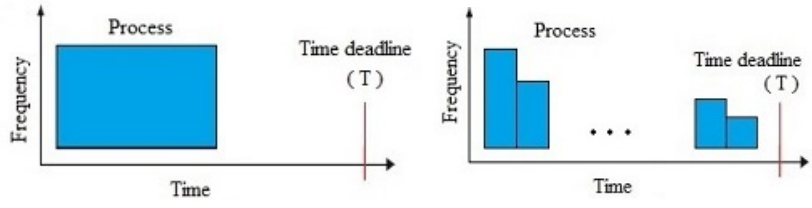
These three conditions put a cap on the maximum utilization of PM resources. Utilization of resources determines power consumption of a PM. In fact, power consumed by a PM increases linearly with its utilization [3]. EARA uses relationship between power consumption of a PM and its utilization (Equation (15)) as a basis for energy consumption calculation.

$$P = P_{idle} + (P_{max} - P_{idle})U \tag{15}$$

where P_{idle} is the power consumption when PM is idle, P_{max} is the power consumption at 100 % utilization, and P is the power consumption at utilization $U \in (0, 1)$ of the PM. An idle PM consumes 70 % of the power consumed at full load [3]. Whereas, high utilization of PM resources causes performance degradation because tasks running on it do not get sufficient resources [3]. So, PM should not be operated at too low or very high utilization in order to improve its energy efficiency.

EARA assigns PM resources to VMs using ACO. The resource allocation using ACO is discussed in Section 4. Besides the optimal allocation and management of PM resources, DVFS is applied as recommended by Wu et al. in [27] to further reduce the energy consumption of a PM. In DVFS, energy is conserved by intentionally scaling down the performance of a PM as discussed in the next subsection.

3.2 Dynamic Voltage Frequency Scaling



a) Execution at maximum frequency b) Execution at different frequencies

Figure 3. Dynamic voltage frequency scaling

Modern processors support ACPI, and thus can be operated at different levels of voltage/frequency. This feature of modern processors has been extensively used in [37, 36, 34, 33, 32, 27] to reduce energy consumption as well as the heat dissipated by them. In this work, DVFS is used to execute a process/task using different combination of frequencies/voltages to conserve energy. Figure 3 shows the effect of executing a task using different combination of supported frequencies. Figure 3 a) depicts the case of executing a process/task at maximum frequency. The process may finish well in advance of its deadline time T. The completion of a task sometimes lowers the utilization of a PM below LGT. In such cases energy can be saved by either server consolidation or dynamic performance scaling of PM. Sometimes consolidation is not economically feasible due to constraints such as communication cost of migration, QoS violations due to interruption in service while consolidating, or unavailability of PM with sufficient free resources where the VM can be migrated. In such cases energy consumed by PMs can be saved by executing the process using different combinations of voltage/frequency [37, 36, 34, 33, 32, 27] as shown in Figure 3 b). In EARA, when the utilization of a PM drops below LGT and remains less than LGT for two consecutive monitoring periods, voltage/frequency of PM is adjusted to lower supported level to conserve energy. In case of low utilization, LNC sends current utilization value 'U' to DVFS module shown in Figure 4.

DVFS module calculates voltage/frequency level 'VF' that can complete the workload before user deadline. Voltage/frequency of PM is then adjusted to 'VF' to conserve energy. Energy is saved at the cost of slower process execution. Elongated process execution time does not affect user satisfaction because frequencies are selected in such a manner to complete the process before desired time deadline. The different frequencies are calculated to complete the task by deadline time T. The PMs in EARA are assumed to have to ACPI support and can be operated at any of the discrete h frequencies $f_1 < f_2 < \ldots < f_{h-1} < f_h$, supported by it. The key idea behind applying DVFS is to execute tasks using linear combination of supported frequencies. The minimization of power consumption is formulated as shown in Equation (16).

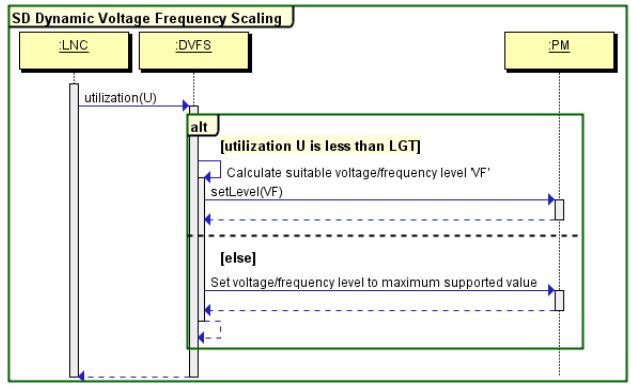


Figure 4. Sequence diagram for dynamic voltage frequency scaling

$$\min E_{ij} = \begin{cases} \sum_{l=1}^{h} t_l A C V^2 f_l, & \text{s.t.} \\ \sum_{l=1}^{h} t_l \leq T, t_l \geq 0, & \text{for } l = 1, 2, \ldots, h \end{cases} \tag{16}$$

where h is number of supported frequencies, t_l is the time period for which the task is executed at frequency f_l.

3.3 Fitness Function

The main goal is to minimize total energy consumption, total execution time, and cost of execution by the efficient utilization of resources. We used weighted sum method to scalarize multiple objectives into a single objective. The weighted fitness function is calculated as shown in Equation (17).

$$F(N_t) = \xi(TEC_{min}) + \zeta(TET_{min}) + \gamma(COE) \tag{17}$$

where

$$TEC_{min} = \sum_{g=1}^{N_v} min(E_{ijg}), \qquad \forall i, j, \tag{18}$$

$$TET_{min} = \sum_{g=1}^{N_v} min(T_{ijg}), \qquad \forall i, j, \tag{19}$$

$0 \leq \xi, \zeta, \gamma < 1$, and $\xi + \zeta + \gamma = 1$. The value of weights ξ, ζ, and γ depends on the importance of each objective in the context of resource allocation problem. Mathematical model formulated for EARA is implemented using ACO metaheuristic as discussed in the next section.

4 ANT COLONY OPTIMIZATION BASED ENERGY AWARE RESOURCE ALLOCATION

Ant colony optimization is a metaheuristic optimization technique proposed by Dorigo in 1992 [9]. Ants secrete a chemical substance called pheromone while foraging. Pheromone gets deposited on the paths followed by ants. The amount of pheromone deposited on the path depends on the number of ants that followed the path. So a path that is used by large number of ants will have a higher quantity of the pheromone deposit. Initially, the ants choose random path while searching for food. But with time the difference in the quantity of pheromone deposited on the paths guides them to choose the path marked with a strong pheromone concentration. The larger amount of pheromone on a path attracts more ants to choose that path again, and finally all the ants converge to the single path. Pheromone evaporates with time, thus reducing the attractive strength of the path, causing ants to explore more paths to the food source. Pheromone evaporation has the advantage of avoiding the convergence to locally optimal solution.

Reasons behind choosing ACO for resource allocation are:

1. It can solve certain NP-hard problems in polynomial time.
2. It maintains a balance between acquired knowledge and exploring new solutions exploiting pheromone evaporation.
3. It gives near to optimal solution.
4. It performs distributed computation to avoid premature convergence.

Ant colony optimization has been applied to solve wide range of combinatorial optimization problems [7, 9, 10, 11, 29]. To solve a combinatorial optimization problem using ant colony optimization, an instance of the problem has to be mapped to a graph $G = (N, L)$, called construction graph. Node set N of the graph represents components of the problem instance and edge set L fully connects the components. Each edge (u, v) of the graph is associated with pheromone trail τ_{uv} and heuristic information η_{uv}. Heuristic information can be cost, distance, etc. that is associated with the edge. Each ant uses a pheromone trail and heuristic information, probabilistically, to construct its own solution of the problem instance. Once the solution is constructed the pheromone trail associated with every edge is updated to reflect evaporation, in order to enable ants to forget previously taken bad decision. The pheromone trail on each edge that belongs to the best solution is then updated.

In this paper, we have applied ACO at two levels for:

1. allocation of VM resources to jobs, and
2. allocation of PM resources to VMs.

Detailed description of the graph construction, pheromone and heuristic information, solution construction, pheromone evaporation, and pheromone trail update for allocation of VM resources to jobs and PM resource to VMs is as follows.

4.1 Allocation of VM Resources to Jobs

Each job of end users has some resource demands and QoS requirements. Each QoS parameter is associated with some value that indicates its priority over the others. The weights of QoS properties can be specified by three ways: absolute weighting, relative weighting, and arbitrary weighting [31]. In this work, we used relative weighting for QoS attributes. ACO metaheuristic for allocation of VM resources to jobs is explained below:

Construction Graph: The problem of VM resource allocation to jobs is mapped to construction graph $G_1 = (N_1, L_1)$. The node set N_1, consists of all VMs and jobs. Set L_1 of edges fully connects the nodes of the graph G_1. Each edge (a, g) of the graph G_1 is assigned pheromone value given by Equation (20).

$$\tau_{ag} = \frac{1}{\ell_a} \tag{20}$$

where a is unique identification number of a job, g is unique identification number of a VM, ℓ_a is the length of the job a. In general, length of job (cloudlet) is in millions of instructions. As inverse of job length is used as pheromone value so shorter jobs will be given preference over the longer ones. Heuristic information assigned to edge (a, g) is given by Equation (21).

$$\eta_{ag} = \prod_{x=1}^{X} (W_{ax})^{\alpha_x} \tag{21}$$

where X is number of QoS parameters associated with job a, W_{ax} is weight value of QoS parameter x, and α_x is control parameter for QoS attribute x of job a.

Solution Construction: Each ant is initially provided with the list of jobs to be mapped on VMs. For each job a that is mapped to VM g, variable y_{ag} is set to 1. Variable y_{ag} is used to keep record of jobs that have already been assigned. The probability that an ant k maps job a on VM g is:

$$\wp_{ag}^{k} = \begin{cases} \frac{(\tau_{ag})^{\alpha 1}(\eta_{ag})^{\beta 1}}{\sum_{t \in \mathcal{N}_g^k}(\tau_{tg})^{\alpha 1}(\eta_{tg})^{\beta 1}}, & \text{if } t \in \mathcal{N}_g^k, \\ 0, & \text{otherwise,} \end{cases} \tag{22}$$

where $\mathcal{N}_g^k$, consisting of all the jobs remaining to be mapped, is called feasible neighborhood of VM g. A job a which has maximum value of $\wp_{ag}^k$ is mapped to VM g. The probability of a job selection for mapping on a particular VM depends on the value of the pheromone trail and heuristic information of the associated edge. $\alpha 1$ is the parameter to control influence of pheromone trail, and $\beta 1$ is parameter to control the overall influence of weight values of QoS parameters. Both $\alpha 1$ and $\beta 1$ can have any value between 0 and 1, and their sum should be equal to one.

Pheromone Trail Evaporation: The pheromone deposited on all the arcs evaporates with time by a constant factor $0 \leq \rho_1 \leq 1$, called evaporation rate. Evaporation avoids unlimited accumulation of pheromone trails on the edges and enables ants to forget allocation decisions previously taken. Pheromone evaporation on all the edges of graph G_1 is realized by Equation (23).

$$\tau_{ag} = (1 - \rho_1)\tau_{ag}, \quad \forall (a, g) \in L_1. \tag{23}$$

Pheromone Trail Update: In order to reflect usage of an arc during solution construction, the pheromone trail on it is updated by an amount equal to inverse of the number of VMs used by an ant for the solution construction. The update makes pheromone concentration on some of the arcs stronger than the others. Strong pheromone concentration on an edge increases the probability of selection of the associated job. Pheromone update by ant k is realized as:

$$\tau_{ag} = \tau_{ag} + \sum_{k=1}^{N_a^1} \Delta\tau_{ag}^k, \quad \forall (a, g) \in S1^k \tag{24}$$

where N_a^1 is the number of ants, $\Delta\tau_{ag}^k$ is the amount of additional pheromone to be deposited on edge (a, g) which is traversed by ant k while constructing the allocation solution, and $S1^k$ is jobs to VMs mapping solution constructed by ant k. The amount of pheromone deposited on arc (a, g) by ant k is defined as follows:

$$\Delta\tau_{ag}^k = \begin{cases} \frac{1}{D1^k}, & (a, g) \in S1^k, \\ 0, & \text{otherwise}, \end{cases} \tag{25}$$

where $D1^k$ is number of VMs used for mapping of all jobs and calculated as length of solution $S1^k$.

4.2 Allocation of PM Resources to VM

Construction Graph: The resource allocation problem is mapped to construction graph $G_2 = (N_2, L_2)$. The node set N_2, consists of all VMs and PMs. L_2 is set of edges that fully connects nodes. Each edge (g, j) of the graph G_2 is associated with pheromone trail τ_{gj} and heuristic information η_{gj}, where g is the identification number of a VM and j is identification number of a PM. Pheromone trail associated with an edge(g, j) is given by Equation (26):

$$\tau_{gj} = \frac{1}{\frac{vm_g}{FM_j}} \tag{26}$$

where vm_g is memory requirements of VM g, and FM_j is available memory space of PM j which can be calculated for given g and j using Equation (27):

$$FM_j = hm_{ij} - \sum_g vm_{ijg} * x_{ijg}, \quad \forall i, j | x_{ijg} = 1. \tag{27}$$

Heuristic information assigned to the edge (g, j) for PM j of cluster i is given by Equation (28):

$$\eta_{gj} = \frac{1}{d_{ij}} \tag{28}$$

where d_{ij} is the distance between the frontend server and PM j of cluster i.

Solution Construction: Each ant is initially provided with the list of VMs to be deployed. x_{ijg} is set to 1 for each VM g that is assigned to PM j of cluster i. Variable x_{ijg} is used to keep record of VM that have already been assigned. The probability that an ant k deploys VM g on PM j of cluster i is:

$$\wp_{ijg}^k = \begin{cases} \frac{(\tau_{gj})^{\alpha 2}(\eta_{gj})^{\beta 2}}{\sum_{t \in \mathcal{N}_j^k}(\tau_{tj})^{\alpha 2}(\eta_{tj})^{\beta 2}}, & \text{if } t \in \mathcal{N}_j^k, \\ 0, & \text{otherwise,} \end{cases} \tag{29}$$

where $\mathcal{N}_j^k$ is the feasible neighborhood of PM j, comprising all those VMs which can still be deployed on it. The probability of choosing a PM j for VM g increases with the value of associated pheromone trail τ_{gj} and heuristic information η_{gj}. $\alpha 2$ and $\beta 2$ are the parameters to control the influence of pheromone trail and heuristic information, and can have any value between 0 and 1.

Pheromone Trail Evaporation: The pheromone deposited on all the arcs graph G_2 evaporates with time by a constant factor $0 \leq \rho_2 \leq 1$, is called the evaporation rate. Evaporation avoids unlimited accumulation of pheromone trails on the edges and enables ant to forget allocation decisions previously taken. Pheromone evaporation on all the edges of graph G_2 is realized by Equation (30).

$$\tau_{gj} = (1 - \rho_2)\tau_{gj}, \quad \forall (g, j) \in L_2. \tag{30}$$

Pheromone Trail Update: In order to reflect usage of an arc during solution construction, pheromone trail on it is updated by the amount equal to inverse of the length of solution path. The update makes pheromone concentration on some of the arcs stronger than the others. Strong pheromone concentration increases the probability of arc selection, which is exploited to optimize the utilization of PM. Pheromone update by ant k is realized as:

$$\tau_{gj}^k = \tau_{gj}^k + \sum_{k=1}^{N_a^2} \Delta\tau_{gj}^k, \quad \forall (g, j) \in S2^k \tag{31}$$

where N_a^2 is the number of ants, $\Delta\tau_{gj}^k$ is the amount of additional pheromone to be deposited on arc (g, j) which is traversed by ant k while constructing the solution, and $S2^k$ is solution constructed by ant k, consisting of VMs to PMs mapping. The amount of pheromone deposited on arc (g, j) by ant k is defined as follows:

$$\Delta\tau_{gj}^k = \begin{cases} \frac{1}{D2^k}, & (g,j) \in S2^k, \\ 0, & \text{otherwise}, \end{cases} \tag{32}$$

where $D2^k$ is computed as sum of lengths of all arcs belonging to solution $S2^k$.

Algorithm 1 EARA

```
1: procedure EARA(VM)                                   ▷ set of VMs
2:     initialization
3:     while not Termination Condition do
4:         Allocation Solution Construction
5:         Pheromone Evaporation
6:         Pheromone Trail Update
7:     end while
8:     return VM to PM map                              ▷ VM to PM mapping
9: end procedure
```

Algorithm 2 *Initialization*

```
1: procedure INITIALIZATION
2:     Create construction graph G2(N2, L2) of resource allocation problem
3:     Set VM = ID of nodes representing VMs in the construction graph G2(N2, L2)
4:     Set PM = ID of nodes representing PMs in the construction graph G2(N2, L2)
5:     for all g in VM do
6:         for all j in PM do
7:             Set τ_gj = 1 / (vm_g / FM_j)
8:             i = getClusterID(j)                      ▷ get cluster ID of PM j
9:             Set η_gj = 1 / d_ij
10:        end for
11:    end for
12: end procedure
```

4.3 Algorithms for EARA

The pseudo code for energy aware resource allocation (EARA) using ACO shown in Algorithm 1, is composed of four phases namely; *initialization, allocation solution*

Algorithm 3 *Allocation Solution Construction*

procedure RESOURCE ALLOCATION(k) ▷ Resource allocation solution construction for ant k
  input: Construction graph $G_2(N_2, L_2)$ of the resource allocation problem
  Set $S2^k$ = NULL ▷ Initialize solution set $S2^k$ of ant k
  while not (Are all VMs Alloted?) **do**
    if Available(activePM) **then**
      Set j = ID of randomly selected PM from list of active PMs
      i = getClusterID(j) ▷ get cluster ID of PM j
    else
      Set j = ID of randomly selected PM from list of newly added PMs
    ▷ Randomly selects PM representing dummy node of construction graph
      i = getClusterID(j) ▷ get cluster ID of PM j
    end if
    Set $\mathcal{N}_j^k$ = NULL ▷ feasible neighborhood of PM j
    Set VMIDs = IDs of VMs yet not assigned to any PM
    for all g in { VMIDs } **do**
      if PM j fulfills processing, memory and network bandwidth requirements of VM g **then**
        Add VM g to $\mathcal{N}_j^k$ ▷ Add VM g to feasible neighborhood of PM j
        Evaluate $\wp_{ijg}^k = \frac{(\tau_{gj})^{\alpha 2}(\eta_{gj})^{\beta 2}}{\sum_{t \in \mathcal{N}_j^k}(\tau_{tj})^{\alpha 2}(\eta_{tj})^{\beta 2}}$,
      end if
    end for
    VM_{id}^f = NULL ▷ set the fittest VM to NULL
    $P^f = 0$ ▷ Probability of the fittest VM
    for all VM g in $\mathcal{N}_j^k$ **do**
      if $\wp_{ijg}^k > P^f$ **then**
        $VM_{id}^f = g$ ▷ set the fittest VM ID to g
        $P^f = \wp_{ijg}^k$ ▷ Change probability of the fittest VM
      end if
    end for
    if VM_{id}^f # NULL **then**
      Set $g = VM_{id}^f$ ▷ assign ID of the fittest VM to g
      Set $x_{ijg} = 1$ ▷ Assign VM g to PM j of Cluster i
      Add x_{ijg} to $S2^k$ ▷ update allocation solution of ant k
    end if
  end while
  return ($S2^k$)
end procedure

Algorithm 4 *Pheromone Evaporation*

1: **procedure** PHEROMONEEVAPORATION
2: **Set** VM = ID of nodes representing VMs in the construction graph $G_2(N_2, L_2)$
3: **Set** PM = ID of nodes representing PMs in the construction graph $G_2(N_2, L_2)$
4: **for all** g in VM **do**
5: **for all** j in PM **do**
6: $\tau_{gj} = (1 - \rho_2)\tau_{gj}$
7: **end for**
8: **end for**
9: **end procedure**

Algorithm 5 *Pheromone Trail Update*

1: **procedure** PHEROMONEUPDATE(k)
2: **input:** k ▷ ant identifier
3: $D2^k = \text{Length}(S2^k)$ ▷ Calculate length of solution $S2^k$
4: $\Delta\tau^k = \frac{1}{D2^k}$
5: **for all** s in $S2^k$ **do** ▷ for each element s in solution $S2^k$
6: $g = \text{getVMID(s)}$ ▷ get VM ID from solution element s
7: $j = \text{getPMID(s)}$ ▷ get PM ID from solution element s
8: $\tau_{gj} = \tau_{gj} + \Delta\tau^k$ ▷ update pheromone information on edge (g, j)
9: **end for**
10: **end procedure**

construction, pheromone evaporation, and *pheromone trail update*. As discussed earlier ACO is applied at two levels. At first level, ACO is applied to allocate VM resources to jobs and at second level it is applied to allocate PM resources to VMs. We have discussed here the pseudo code for allocation of PM resources to VMs only. The given pseudo code can be easily modified for allocation of VM resources to jobs.

For allocation of PM resources to VMs, list of VMs is passed to the procedure EARA (Algorithm 1). In the *initialization* phase as shown in Algorithm 2, construction graph of the problem is created and every edge of the graph is assigned initial pheromone trail and heuristic information as discussed earlier. The function getClusterID(), in line number 8, is used to get cluster ID of a PM. Algorithm 3 outlines *allocation solution construction* phase. It probabilistically maps the VMs to the appropriate PMs. A PM is selected randomly and VMs from its feasible neighborhood are assigned to it one by one till UGT is observed. The process is repeated for each ant until all the VMs are mapped. When all the ants have finished assigning VMs to PMs, pheromone trail evaporation on all the edges is performed by Algorithm 4. Pheromone trail on every edge used by an ant for solution construction is then updated as shown in Algorithm 5.

5 PERFORMANCE EVALUATION AND COMPARATIVE ANALYSIS

The performance evaluation of EARA is performed on CloudSim. CloudSim is a framework for modeling and simulation of cloud computing infrastructure and services [5]. CloudSim simulation toolkit is used for implementation, testing, and validation because of large-scale nature of real cloud environment. For performance analysis, EARA is compared with existing resource allocation algorithms, i.e. FFD [30] and MGGA [28]. Five data centers are created with specification as shown in Table 1. In each data center, PMs complying with specifications, as shown in Table 2, are created. To conduct comparative analysis, four types of VMs, as shown in Table 3, are used. FFD, MGGA and EARA are rigorously tested with jobs having different QoS requirements.

Name	*PC*	*MC*	*SC*	*BC*	*Time Zone*
DC1	3	0.05	0.10	0.10	3.0
DC2	3.5	0.07	0.10	0.11	5.0
DC3	4	0.09	0.10	0.07	5.5
DC4	5	0.10	0.10	0.13	8.0
DC5	5.25	0.12	0.10	0.15	10.0

Name – data center name; PC – processing cost; MC – memory cost per MB; SC – storage cost per MB; BC – bandwidth cost; Time Zone – time zone of data center location

Table 1. Specification of data centers

PM Type	*CPU*	*Cores*	*RAM*	*Storage*	*BW*
1	1 000	4	8	2	10
2	1 500	8	16	2	10
3	2 000	12	32	2	10
4	3 000	20	64	4	10
5	5 000	36	64	4	10

CPU – processing speed in mips; Cores – number of processing cores; RAM – random access memory in GB; Storage – permanent storage capacity in TB; BW – network bandwidth in gbps

Table 2. Specification of physical machines

The aim of executing jobs with different QoS requirements is to test the effectiveness of EARA in terms of energy efficiency, number of PMs required, and quality of service. Performance of EARA is analyzed by varying number of jobs in every simulation run. Simulation is repeated 25 times and in each simulation run, parameters are set to a value from the range of values given in Table 4.

VM Type	*CPU*	*PEs*	*RAM*	*BW*
1	500	1	512	1
2	1 000	2	1 024	2
3	2 000	4	2 048	4
4	4 000	8	4 096	8

CPU – processing speed in mips; PEs – number of cores; RAM – random access memory in MB; BW – network bandwidth in gbps

Table 3. Specification of virtual machines

Number of jobs	200–1 200	Varied in every simulation run
Number of datastores	2–5	Stores instances of VMs
Number of ants	10–50	Construct allocation solution
Idle Time	10 min.	Time to switch PM to sleep mode
CPU_{UGT}	0.85	UGT for CPU utilization
MEM_{UGT}	0.85	UGT for memory utilization
BW_{UGT}	0.85	UGT for bandwidth utilization
CPU_{LGT}	0.30	LGT for CPU utilization
MEM_{LGT}	0.30	LGT for memory utilization
BW_{LGT}	0.30	LGT for bandwidth utilization

Table 4. Simulation parameters

5.1 Comparative Analysis

Figure 5 shows the comparison of the number of PMs used by FFD, MGGA, and EARA to fulfill the computational requirements of a given number of jobs. EARA outperforms both FFD and MGGA in terms of the number of PMs used to deploy a given number of jobs. On an average, EARA uses 11.36 % and 7.68 % lesser number of PMs than FFD and MGGA, respectively.

Figure 6 shows the comparison of total energy consumed by FFD, MGGA, and EARA. Total energy consumption of EARA is less than FFD and MGGA because it uses lesser number of PMs to deploy given number of jobs. It is experimentally established that EARA is 10.56 %, and 5.43 % more energy efficient than FFD and MGGA, respectively.

Figure 7 depicts the comparison of percentage resource utilization of PMs by FFD, MGGA, and EARA. In case of EARA, utilization of 85 % of PMs is between 41–80 %. In case of FFD and MGGA the utilization of more than one third of PMs is above 80 % which caused the performance degradation of user applications and resulted in creation of hot spots. Moreover, only 2 % of the PMs are there in EARA where the utilization is 0–20 % as compared to 5 % and 7 % in FFD and MGGA, respectively. Therefore, EARA is capable of managing the resources efficiently.

Figure 8 shows the comparison of the average number of VM migrations. The number of migrations in EARA is less than MGGA but more than FFD. In EARA, when the utilization of a PM falls below LGT value migration of VMs running over it is performed. Migration is performed for PMs consolidation so that some of the PMs can be switched to sleep mode to conserve energy. We observed approximately two migrations for 1200 jobs, such a small number of migrations does not impact the performance of the system. Moreover, EARA compensates the energy loss due to migrations by switching idle PMs to sleep mode.

Figure 9 shows the comparison of number of hot spots created by FFD, MGGA, and EARA as the number of jobs vary from 200 to 1200. We define a PM as a hot spot if its resource utilization is 100 %. Maximum number of hot spots are created by FFD methodology because it tries to utilize PM to its full capacity. However, EARA does not create any hot spot because it keeps resource utilization of PMs between LGT and UGT. Hot spots adversely affect the performance and reliability of PM. Moreover, creation of hot spots demands better cooling arrangements and also increases chances of hardware failure. Hence, EARA is more reliable and energy efficient.

Figure 10 shows the comparison of percentage workload of data centers, when $\alpha 2 = 0$. All the data centers have exactly same computing infrastructure but are at unequal distance from frontend server. Distance between a datacenter and frontend server is calculated from time zone of the datacenter. When $\alpha 2 = 0$, EARA gives weightage to distance while mapping VMs to PMs. As distance of DC1 is least so EARA distributes the VMs to PMs of DC1 first. Once DC1 resources are used upto UGT of their capacity, EARA starts assigning jobs to PMs of the datacenter whose distance is next to distance of DC1 and so on. EARA saves energy by deploying jobs/VMs over PMs which are nearer to frontend server because more energy is consumed to transmit jobs/VMs over longer distances. Moreover, deploying VM over nearer datacenter also improves response time.

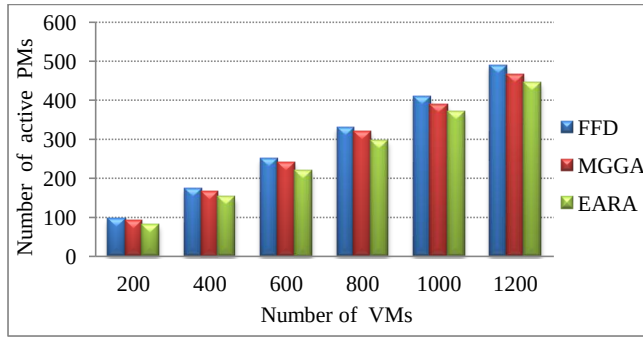


Figure 5. Comparison of number of PMs required by FFD, MGGA, and EARA

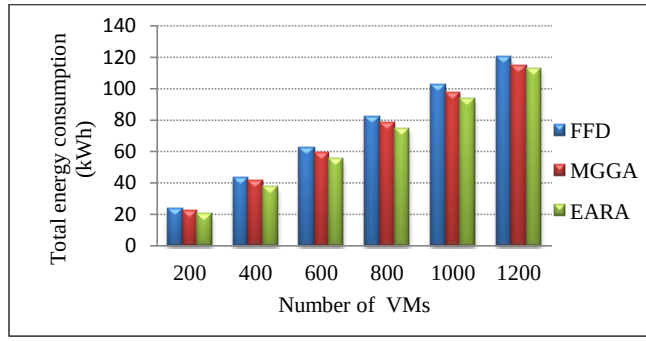


Figure 6. Comparison of total energy consumption by FFD, MGGA, and EARA

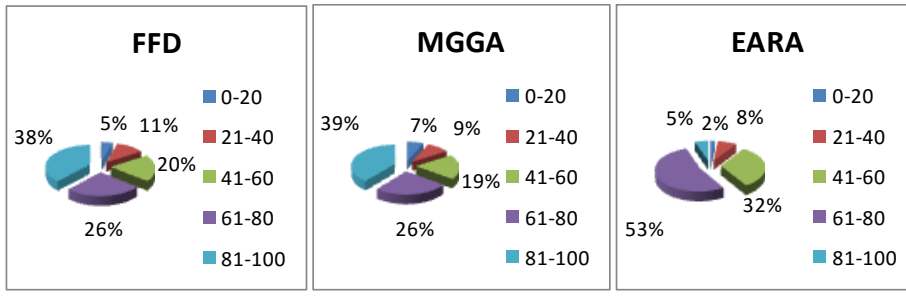


Figure 7. Comparison of PMs utilization in FFD, MGGA, and EARA

Figure 11 shows the comparison of percentage workload of data centers, when $\beta 2 = 0$. All the data centers are having five types of PMs with specification as per Table 2, and the data centers have equal number of specific type of PMs. In case of EARA, variance of the percentage workload of data centers is less than that of FFD and MGGA. This is due to the fact that EARA gives more weight to resource availability when $\beta 2$ is set to 0. As all the data centers have exactly the same computing infrastructure, so EARA distributes almost equal workload among them. This feature of EARA can be exploited for load balancing among datacenters.

Figure 12 shows the comparison of total energy consumption by the cloud computing infrastructure between EARA and EARA-DVFS. In case of EARA-DVFS, dynamic voltage frequency scaling is not applied. The result shows that EARA which is employing DVFS saves 8.15 % more energy than EARA-DVFS.

Figure 13 depicts the comparison of computational energy for FFD, MGGA, and EARA. It is the total energy consumed, measured in Watt hours (Wh), for finding suitable resources for all the jobs. This figure shows that the EARA consumes less energy in computation than MGGA but a little more than FFD. On an average, EARA consumes 0.42 % of total energy consumption if the allocation of resources is made efficiently. Therefore, EARA is better than FFD and MGGA as the compu-

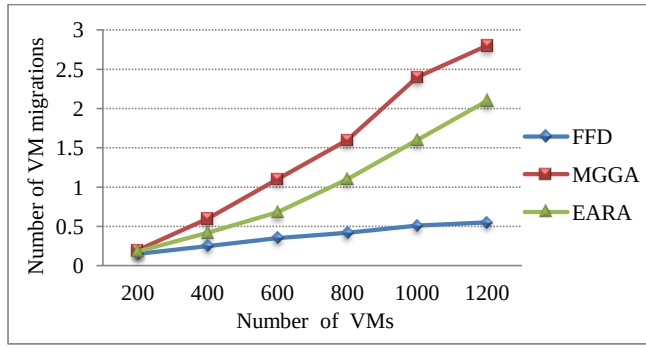


Figure 8. Comparison of number of VM migrations in FFD, MGGA, and EARA

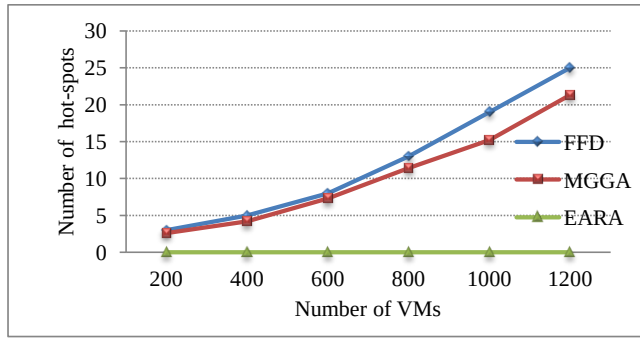


Figure 9. Comparison of number of hot spots in FFD, MGGA, and EARA

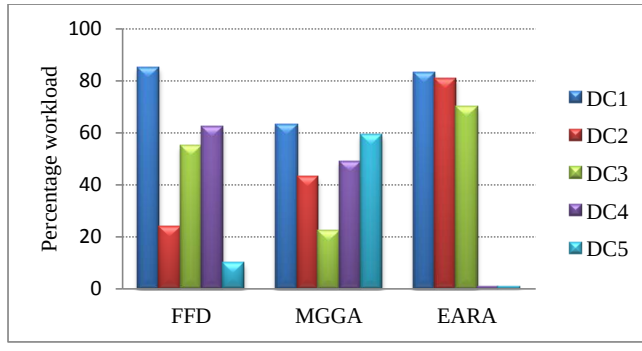


Figure 10. Comparison of percentage workload of data centers in FFD, MGGA, and EARA, when $\alpha 2 = 0$

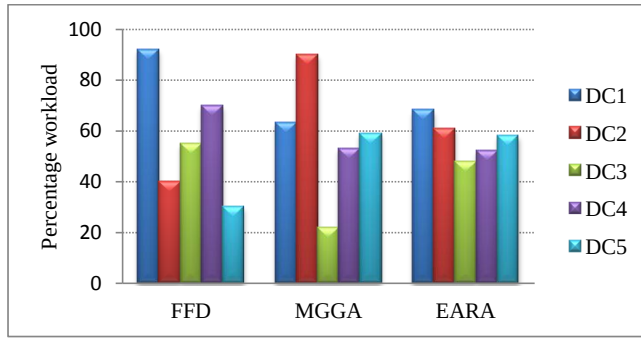


Figure 11. Comparison of percentage workload of data centers in FFD, MGGA, and EARA, when $\beta 2 = 0$

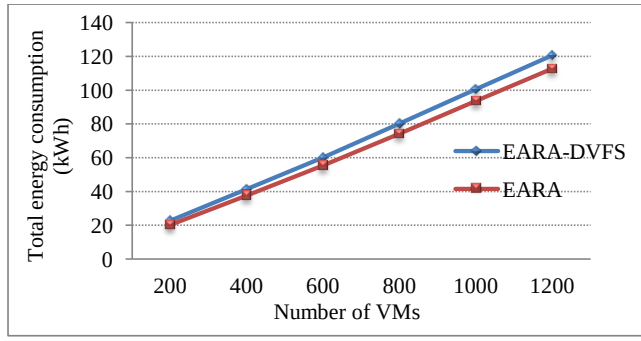


Figure 12. Comparison of total energy consumption between EARA and EARA-DVFS

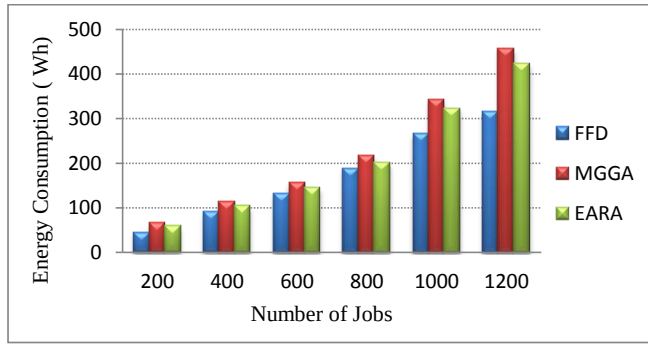


Figure 13. Comparison of computational energy consumption in FFD, MGGA, and EARA

tational energy consumption of 0.42 % is very small compared to the overall energy gain of 10.56 %.

6 CONCLUSION AND FUTURE WORK

In this paper, energy aware resource allocation methodology using the ant colony optimization has been proposed. ACO is applied at two levels for efficient allocation of resources. The first level ACO allocates VMs resources to jobs whereas the second level ACO allocates PMs resources to VMs. Resources are allocated to jobs on the basis of their QoS requirements. Server consolidation and dynamic performance scaling of PMs are employed to conserve energy. The proposed methodology is implemented in CloudSim and the results are compared FFD and MGGA resource allocation methods. It is experimentally established that the proposed EARA achieves up to 10.56 % saving in energy consumption through a better utilization of resources and desired QoS.

In future, EARA can be tested on OpenNebula based private cloud environment comprising water cooled CPU, and CPU with self steering frequency to confirm its capability of reducing the energy consumption. Furthermore, temperature aware resource allocation, and fault tolerant features such as check pointing and replication can be added to make EARA more robust and reliable.

Acknowledgement

This work is partially supported by the major project granted by the University Grants Commission, New Delhi, India. File No. 41-652/2012(SR).

Appendix A SYMBOLIC NOTATIONS USED IN EARA

Table A1: List of Symbols

Symbol	Definition
CL	Set of clusters
cl_i	i^{th} cluster
PM_i	Set of physical machines in i^{th} cluster
N_c	Number of clusters
N_v	Number of VMs
N_t	Number of tasks/jobs
N_a^1	Number of ants used in allocating jobs to VMs
N_a^2	Number of ants used in allocating VMs to PMs
H_i	Number of PMs in cluster i

Continued on next page

Table A1 – continued from previous page

Symbol	Definition
h_{ij}	PM j of cluster i
$hs_{ij}, hm_{ij}, hn_{ij}$	CPU speed, memory, and network bandwidth of PM j of cluster i
E_{ijg}	Energy consumed by VM g if executed on PM j of cluster i
E_{ij}	Energy consumption of PM j of cluster i
VM	Set of VMs
R_g	Requirement of VM g
r_{ij}^q	Resource requirement of task q executing on PM j of cluster i
t_{ij}^q	Execution time of task q on PM j of cluster i
p_{ij}^q	Power consumed for executing task q on PM j of cluster i
s_{ij}^q	CPU speed allocated to task q on PM j of cluster i
e_{ij}^q	Energy consumed for executing task q on PM j of cluster i
T_{ijg}	Total execution time of all tasks deployed on VM g
h	Number of voltage/frequency levels supported by PM
C_{ijg}	Total cost of running VM g on PM j of cluster i
x_{ijg}	1 if VM g is assigned to PM j of cluster i, 0 otherwise
EC_{ij}	Execution cost of VMs run on PM j of cluster i
v_g	VM g
vs_g, vm_g, vn_g	CPU speed, memory, and network bandwidth of VM g
n_g	Number of jobs allocated to VM g
G_1	Construction graph for jobs to VMs mapping
N_1	Set of nodes of construction graph G_1
L_1	Set of edges of construction graph G_1
α_1	Parameter to control influence of pheromone trail in G_1
β_1	Parameter to control influence of heuristic information in G_1
ρ_1	Pheromone evaporation rate for graph G_1
$\wp_{ag}^k$	Probability of mapping job a to VM g
W_{ax}	Weight value of QoS parameter x of job a
ℓ_a	Length of job a
X	Number of QoS parameters
y_{ag}	is 1 if job a is assigned to VM g, 0 otherwise
$\mathcal{N}_g^k$	Feasible neighborhood of VM g for ant k
$S1^k$	Ant k's jobs to VMs mapping solution
$D1^k$	Number of VMs used in solution k
G_2	Construction graph for VMs to PMs mapping
N_2	Set of nodes of construction graph G_2
L_2	Set of edges of construction graph G_2
$\tau_{u,v}$	Pheromone value associated with edge (u, v)
$\eta_{u,v}$	Heuristic information associated with edge (u, v)

Continued on next page

Table A1 – continued from previous page

Symbol	Definition
α_2	Parameter to control influence of pheromone trail in G_2
β_2	Parameter to control influence of heuristic information in G_2
ρ_2	Pheromone evaporation rate for graph G_2
$\mathcal{N}_j^k$	Feasible neighborhood of PM j for ant k
FM_j	Available memory space of PM j
$S2^k$	Ant k's VMs to PMs mapping solution
$D2^k$	Length of solution k
d_{ij}	Distance of PM j of cluster i from frontend server
i, j, g, k, q, k	Identifier for cluster, PM, VM, ant, task, and ant, respectively
ξ, ζ, γ	Weight values for TEC, TET, and COE, respectively
P_{idle}	Power consumption of idle PM
P_{max}	Power consumption of PM at 100 % utilization
U	Utilization of a PM
P	Power consumption of PM at U % utilization
CPU_{UGT}	Upper Green Threshold value for CPU
MEM_{UGT}	Upper Green Threshold value for Memory
BW_{UGT}	Upper Green Threshold value for Bandwidth
CPU_{LGT}	Lower Green Threshold value for CPU
MEM_{LGT}	Lower Green Threshold value for Memory
BW_{LGT}	Lower Green Threshold value for Bandwidth
$P_{dynamic}$	Dynamic power consumption
P_{static}	Static power consumption
A	Switching activity
C	Capacitance
V	Voltage
I_{leak}	Leaking current
f	Frequency

REFERENCES

[1] Armbrust, M.—Fox, A.—Griffith, R.—Joseph, A. D.—Katz, R. H.—Konwinski, A.—Lee, G.—Patterson, D. A.—Rabkin, A.—Stoica, I.—Zaharia, M.: Above the Clouds: A Berkeley View of Cloud Computing. Technical report No. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[2] Beloglazov, A.: Energy-Efficient Management of Virtual Machines in Data Centers for Cloud Computing. Ph.D. thesis, University of Melbourne, 2013.

[3] Beloglazov, A.—Abawajy, J.—Buyya, R.: Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing.

Future Generation Computer Systems, Vol. 28, 2012, No. 5, pp. 755–768, doi: 10.1016/j.future.2011.04.017.

[4] Beloglazov, A.—Buyya, R.: Managing Overloaded Hosts for Dynamic Consolidation of Virtual Machines in Cloud Data Centers under Quality of Service Constraints. IEEE Transactions on Parallel and Distributed Systems, Vol. 24, 2013, No. 7, pp. 1366—1379, doi: 10.1109/TPDS.2012.240.

[5] Calheiros, R. N.—Ranjan, R.—Beloglazov, A.—De Rose, C. A. F.—Buyya, R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Software: Practice and Experience (SPE), Vol. 41, 2011, No. 1, pp. 23–50.

[6] Castañé, G. G.—Núñez, A.—Llopis, P.—Carretero, J.: E-mc2: A Formal Framework for Energy Modeling in Cloud Computing. Simulation Modeling Practice and Theory, Vol. 39, 2013, pp. 56–75.

[7] Chen, H.—Cheng, A. M. K.—Kuo, Y.-W.: Assigning Real-Time Tasks to Heterogeneous Processors by Applying Ant Colony Optimization. Journal of Parallel and Distributed Computing, Vol. 71, 2011, No. 1, pp. 132–142, doi: 10.1016/j.jpdc.2010.09.011.

[8] Chen, J.-J.—Yang, C.-Y.—Kuo, T.-W.—Shih, C.-S.: Energy-Efficient Real-Time Task Scheduling in Multiprocessor DVS Systems. Asia and South Pacific Design Automation Conference (ASP-DAC '07), 2007, pp. 342–349, doi: 10.1109/ASP-DAC.2007.358009.

[9] Dorigo, M.—Stützle, T.: Ant Colony Optimization. The MIT Press, Cambridge, MA, USA, 2004.

[10] Feller, E.—Rilling, L.—Morin, C.: Energy-Aware Ant Colony Based Workload Placement in Clouds. 12th IEEE/ACM International Conference on Grid Computing (GRID), 2011, pp. 26–33, doi: 10.1109/Grid.2011.13.

[11] Gao, Y.—Guan, H.—Qi, Z.—Hou, Y.—Liu, L.: A Multi-Objective Ant Colony System Algorithm for Virtual Machine Placement in Cloud Computing. Journal of Computer and System Sciences, Vol. 79, 2013, pp. 1230–1242, doi: 10.1016/j.jcss.2013.02.004.

[12] Garg, S. K.—Yeo, C. S.—Buyya, R.: Green Cloud Framework for Improving Carbon Efficiency of Clouds. Euro-Par 2011 Parallel Processing. Lecture Notes in Computer Science, Vol. 6852, 2011, pp. 491–502, doi: 10.1007/978-3-642-23400-2_45.

[13] Hsu, C.-H.—Chen, S.-C.—Lee, C.-C.—Chang, H.-Y.—Lai, K.-C.—Li, K.-C.—Rong, C.: Energy-Aware Task Consolidation Technique for Cloud Computing. IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom), 2011, pp. 115–121, doi: 10.1109/CloudCom.2011.25.

[14] Huang, C. J.—Guan, C.-T.—Chen, H.-M.—Wang, Y.-W.—Chang, S.-C.—Li, C.-Y.—Weng, C.-H.: An Adaptive Resource Management Scheme in Cloud Computing. Engineering Applications of Artificial Intelligence, Vol. 26, 2013, No. 1, pp. 382–389.

[15] Kaplan, J. M.—Forrest, W.—Kindler, N.: Revolutionizing Data Center Energy Efficiency. Technical report, McKinsy & Company, 2008.

[16] Kim, N.—Cho, J.—Seo, E.: Energy-Credit Scheduler: An Energy-Aware Virtual Machine Scheduler for Cloud Systems. Future Generation Computer Systems, Vol. 32, 2014, pp. 128–137, doi: 10.1016/j.future.2012.05.019.

[17] Kinger, S.—Kumar, R.—Sharma, A.: Prediction Based Proactive Thermal Virtual Machine Scheduling in Green Clouds. The Scientific World Journal, Vol. 2014, 2014, Art. No. 208983, pp. 92–103, doi: 10.1155/2014/208983.

[18] Koomey, J.: Growth in Data Center Electricity Use 2005 to 2010. 2014, `http://www.analyticspress.com/datacenters.html`.

[19] Lee, H. M.—Jeong, Y.-S.—Jang, H. J.: Performance Analysis Based Resource Allocation for Green Cloud Computing. The Journal of Supercomputing, Vol. 69, 2014, No. 3, pp. 1013–1026.

[20] Mell, P.—Grance, T.: The NIST Definition of Cloud Computing, 2011.

[21] Nathani, A.—Chaudhary, S.—Somani, G.: Policy Based Resource Allocation in IaaS Cloud. Future Generation Computer Systems, Vol. 28, 2012, No. 1, pp. 94–103, doi: 10.1016/j.future.2011.05.016.

[22] Nathuji, R.—Schwan, K.: VirtualPower: Coordinated Power Management in Virtualized Enterprise Systems. ACM SIGOPS Operating Systems Review, Vol. 41, 2007, No. 6, pp. 265–278, doi: 10.1145/1294261.1294287.

[23] Quarati, A.—Clematis, A.—Galizia, A.—D'Agostino, D.: Hybrid Clouds Brokering: Business Opportunities, QoS and Energy-Saving Issues. Simulation Modeling Practice and Theory, Vol. 39, 2013, pp. 121–134, doi: 10.1016/j.simpat.2013.01.004.

[24] Raycroft, P.—Jansen, R.—Jarus, M.—Brenner, P. R.: Performance Bounded Energy Efficient Virtual Machine Allocation in the Global Cloud. Sustainable Computing: Informatics and Systems, Vol. 4, 2014, pp. 1–9.

[25] Takeda, S.—Takemura, T.: A Rank-Based VM Consolidation Method for Power Saving in Datacenters. IPSJ Transactions on Advanced Computing Systems, Vol. 3, 2010, No. 2, pp. 138–146, doi: 10.2197/ipsjtrans.3.88.

[26] Vogels, W.: Beyond Server Consolidation. Queue, Vol. 6, 2008, No. 1, pp. 20–26, doi: 10.1145/1348583.1348590, doi: 10.1145/1348583.1348590.

[27] Wu, C.-M.—Chang, R.-S.—Chan, H.-Y.: A Green Energy-Efficient Scheduling Algorithm Using the DVFS Technique for Cloud Datacenters. Future Generation Computer Systems, Vol. 37, 2014, pp. 141–147, doi: 10.1016/j.future.2013.06.009.

[28] Xu, J.—Fortes, J. A. B.: Multi-Objective Virtual Machine Placement in Virtualized Data Center Environments. IEEE/ACM International Conference on Green Computing and Communications (GreenCom), 2010, pp. 179–188, doi: 10.1109/GreenCom-CPSCom.2010.137.

[29] Yin, P.-Y.—Wang, J.-Y.: Ant Colony Optimization for the Nonlinear Resource Allocation Problem. Applied Mathematics and Computation, Vol. 174, 2006, No. 2, pp. 1438–1453.

[30] Yue, M.: A Simple Proof of the Inequality FFD (L) $\leq$ 11/9 OPT (L) + 1, $\forall$L for the FFD Bin-Packing Algorithm. Acta Mathematicae Applicatae Sinica, Vol. 7, 1991, No. 4, pp. 321–331.

[31] Tran, V. X.—Tsuji, H.—Masuda, R.: A New QoS Ontology and Its QoS-Based Ranking Algorithm for Web Services. Simulation Modelling Practice and Theory, Vol. 17, 2009, No. 8, pp. 1378–1398.

[32] Wang, P.—Qi, Y.—Liu, X.: Power-Aware Optimization for Heterogeneous Multi-Tier Clusters. Journal of Parallel and Distributed Computing, Vol. 74, 2014, No. 1, pp. 2005–2015.

[33] Guérout, T.—Monteil, T.—Da Costa, G.—Calheiros, R. N.—Buyya, R.—Alexandru, M.: Energy-Aware Simulation with DVFS. Simulation Modelling Practice and Theory, Vol. 39, 2013, pp. 76–91, doi: 10.1016/j.simpat.2013.04.007.

[34] Zhai, B.—Blaauw, D.—Sylvester, D.—Flautner, K.: Theoretical and Practical Limits of Dynamic Voltage Scaling. Proceedings of the 41st Design Automation Conference, 2004, pp. 868–873, doi: 10.1145/996566.996798.

[35] Data Flow Diagram. 2015, `https://en.wikipedia.org/wiki/Data_flow_diagram`. Accessed: January 5, 2015.

[36] Marzolla, M.—Mirandola, R.: Dynamic Power Management for QOS-Aware Applications. Sustainable Computing: Informatics and Systems, Vol. 3, 2013, No. 4, pp. 231–248.

[37] Bertsekas, D. P.: Chapter 4 – Exact Penalty Methods and Lagrangian Methods. In: Bertsekas, D. P. (Ed.): Constrained Optimization and Lagrange Multiplier Methods. Academic Press, 1982, pp. 179–301, doi: 10.1016/B978-0-12-093480-5.50008-8.

Ashok Kumar received his M.Sc. degree in information technology from Punjab Technical University, Jalandhar. Currently he is pursuing his doctoral degree in cloud computing from Thapar University, Patiala. His research interests include cloud computing, internet of things and fog computing. He has five research publications in reputed journals and conferences.

Rajesh Kumar is currently working as Professor in the Computer Science and Engineering Department, Thapar University, Patiala. He received his M.Sc., M.Phil. and Ph.D. degrees from IIT Roorkee. He has more than 21 years of UG & PG teaching and research experience. He wrote over 101 research papers for various international and national journals and conferences. He has, so far, guided 10 Ph.D. and 23 M.E./M.Sc. theses. His current areas of research interests include FANETs, resource scheduling and fault tolerance in clouds.

Anju Sharma is working as Assistant Professor in the Computer Science and Engineering Department, MRSPTU, Bathinda. Her research interests include smart grid computing, cloud computing, IoT and fog computing. She has varied numbers of publications in international journals and conferences of repute. She is Senior Member of International Association of Computer Science and Information Technology (IACSIT) and professional member of ACM India, IEEE. She is an active member (TCM and reviewer) of varied conferences.

Computing and Informatics, Vol. 37, 2018, 109–141, doi: 10.4149/cai_2018_1_109

TOWARDS A FORMALIZATION OF A FRAMEWORK TO EXPRESS AND REASON ABOUT SOFTWARE ENGINEERING METHODS

Miguel Morales-Trujillo*

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Mexico City, Mexico
e-mail: migmor@ciencias.unam.mx

Hanna Oktaba, Francisco Hernández-Quiroz

Facultad de Ciencias, Universidad Nacional Autónoma de México
Mexico City, Mexico
e-mail: {hanna.oktaba, fhq}@ciencias.unam.mx

Boris Escalante-Ramírez

Facultad de Ingeniería, Universidad Nacional Autónoma de México
Mexico City, Mexico
e-mail: boris@servidor.unam.mx

Abstract. Software Engineering is considered a knowledge-intensive discipline, in which knowledge creation, collection and sharing is an uninterrupted process. However, a large part of this knowledge exists in a tacit form and depends on practitioners. Therefore defining a mechanism to transform tacit knowledge into explicit one is of upmost importance. This paper presents a formalization approach to represent Software Engineering practitioners' tacit knowledge, which is related to their ways of working, as a set of explicit statements. The formalization is based on KUALI-BEH, which is a normative kernel extension of ESSENCE formal specification, and consists of three parts: an ontology to share a common representation of knowledge

* corresponding author

as a set of concepts; a Situational Method Engineering based algebra that represents well-defined method properties and operations; and a knowledge representation of the ontology and algebra using Description Logics. The main objectives of this initial formalization are to improve communication among humans and machines, computational inference and reuse of knowledge.

Keywords: Software engineering, situational method engineering, ontology, description logics, ESSENCE, KUALI-BEH

Mathematics Subject Classification 2010: 68-N30

1 INTRODUCTION

Precisely specifying the process by which a Software Engineering activity takes place is a challenging task [1]. Every aspect of software development, particularly in large systems, demands a great deal of knowledge and understanding of the software practitioner [2].

Moreover, according to [3] and [4], creating software is one of the most knowledgeintensive professions. Software Engineering community has been motivated to collect all the knowledge that practitioners possess; the activity of knowledge gathering has become a relevant research line for the discipline.

Since knowledge creation is an uninterrupted process, it is true about its collection as well, thus we need a process to manage it. In [5] it is established that knowledge management process should address all of the following tasks:

1. to acquire new knowledge;
2. to transform the knowledge from tacit or implicit into explicit knowledge;
3. to systematically store, disseminate, and evaluate knowledge; and
4. to apply knowledge in new situations.

Providing Software Engineering with a knowledge-based approach allows us to create models and reason about them. At this point the wide scope of the discipline becomes an obstacle. Presentation and integration of the knowledge-based approach into the everyday working world of software engineers is a critical challenge for the Knowledge-Based Software Engineering (KBSE) community [4].

In 1992 [4] identified three crucial questions to give Software Engineering a knowledge-based focus:

1. What part of the software process is targeted?
2. What knowledge is applicable and how can it be represented, acquired, and maintained?

3. How can we present the knowledge to developers, teams, and managers to improve the quality, cost, and timeliness of software development?

By finding answers to these questions, that are still valid, researchers will be able to create a model to represent the knowledge of a targeted software process. Once the model of a process is precisely defined in a formal manner, process analysis techniques can be applied to such a model to identify problematic and erroneous steps, or to leverage efficiency improvements [1].

The objective of this paper is to present a formalization, which was created as a way of improvement of the communication among humans and machines, and of reasoning about the tacit knowledge possessed by Software Engineering practitioners. In particular, the paper is focused on the practitioners' ways of working during software projects. The proposed formalization is built on KUALI-BEH [6], a Normative Annex of ESSENCE – Kernel and Language for Software Engineering Methods [7], which is an Object Management Group (OMG) formal specification. The proposal uses three types of formalization: an ontology to share a common representation of knowledge as a set of concepts; an algebra based on Situational Method Engineering (SME) to represent well-defined method properties and operations; and a knowledge representation of the ontology and algebra using Description Logics (DL).

The motivation behind creating a formalization based on an ESSENCE Kernel extension is the lack of reasoning mechanisms in metamodels, like SPEM [8], or standards like ESSENCE itself. Moreover, the main reason that motivated the formalization was to provide Software Engineering practitioners with a mechanism to reason about the knowledge they possess. As [9] stated, there is a need to provide *"a simple specification language to describe any type of activity in a company, in a concurrent and modular fashion"*, which is also useful for software engineers.

This paper is organized as follows: the background is presented in Section 2, Section 3 demonstrates the proposed formalization, Section 4 describes its validation, Section 5 mentions the intended usage of the formalization, and conclusions and future work are discussed in the final section.

2 BACKGROUND

This section presents KUALI-BEH as the object to be formalized, and the following formalization approaches: Ontologies, SME and DL.

2.1 ESSENCE – Kernel and Language for Software Engineering Methods

In 2011 OMG initiated a standard project, the outcome of which was ESSENCE: a four layer approach that defines a kernel, a language and permits the construction of software engineering methods, see Figure 1 (left side).

The ESSENCE kernel is a set of universal components involved in software engineering efforts, which are expressed in a language by the syntax and semantics

rules. The path that guided the definition of ESSENCE was the separation of concerns, and it therefore defines three areas of concern: Customer, Solution and Endeavor. Another relevant concept is activity space, which according to [7], is the representation of the essential things to do; it describes the challenges a team faces when developing, maintaining, and supporting software systems. Within each area of concern an activity space is provided, in which practitioners can model the state of a particular software project with the help of a previously defined set of Abstract-Level Project Health Attributes (ALPHAs), see Figure 1 (right side).

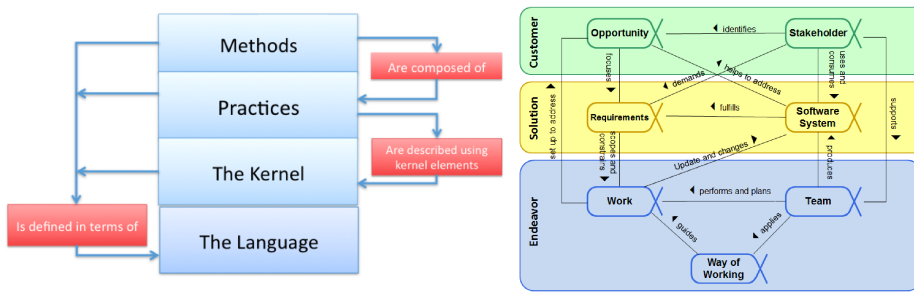


Figure 1. ESSENCE architecture and its areas of concern [7]

The Endeavor area of concern contains everything to do with team members and the way in which they approach their work [7]. Two ALPHAs are associated with the Endeavor: Work and Way-of-Working. In the context of software engineering, work is everything that the team does to meet the goals of producing a software system. The work is guided by the practices that make up the team's way-of-working. The Way-of-Working is the tailored set of practices and tools used by a team to guide and support their work, which evolves according to specific working contexts [7].

In particular, although the Endeavor area of concern addresses the practitioners' practical knowledge, it does not indicate how to express it nor how to reason about it. This lack of expressive mechanisms stimulated the extension of ESSENCE in the form of KUALI-BEH as an alternative to allow practitioners to transform their tacit knowledge into explicit knowledge. KUALI-BEH covers the top two layers of the ESSENCE architecture, which are Methods and Practices, and offers mechanisms to identify, express, agree, execute, optimize and consolidate ways of working. Consequently, KUALI-BEH became an extension of the ESSENCE Kernel, presented as a Normative Annex of the formal specification.

2.2 KUALI-BEH

KUALI-BEH is based on a set of common concepts involved in software projects and provides a framework for authoring Software Engineering Methods. KUALI-BEH is composed of two views: static and operational.

The static view defines the common concepts needed for the definition of the practitioners' diverse ways of working (see Figure 2), and arranges them into methods composed of practices. This knowledge creates an infrastructure of methods and practices that is built and used by practitioners.

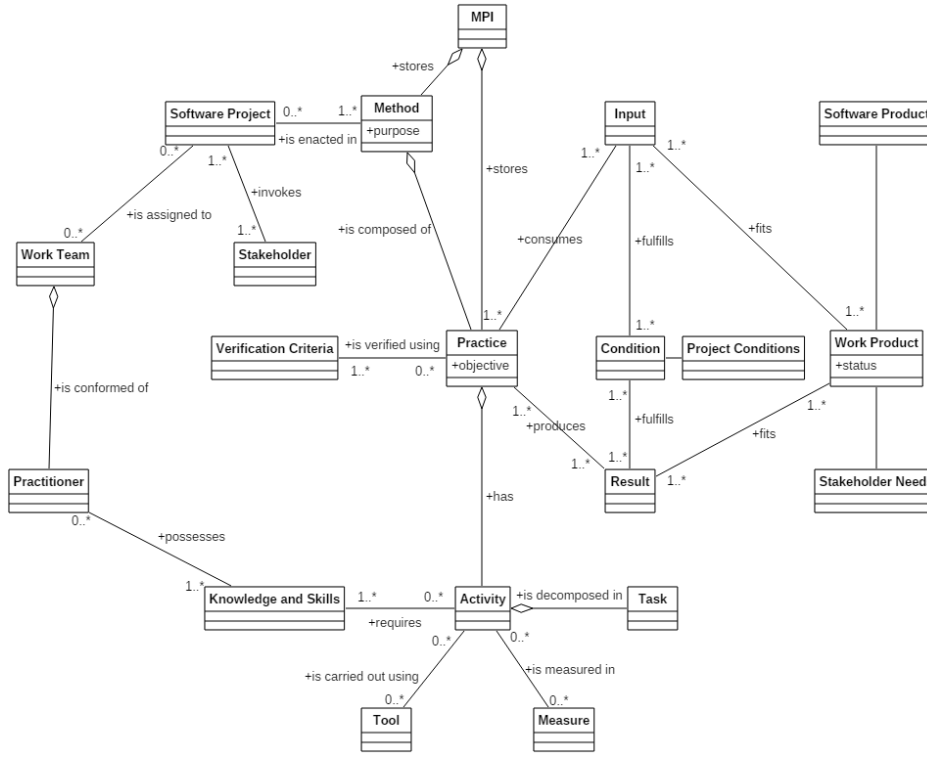


Figure 2. KUALI-BEH concepts and their relationships and attributes

The operational view is related to the software project execution. It provides work teams with mechanisms how to enact a method and adapt its practices to a specific context and stakeholder needs.

The KUALI-BEH static view target audience represents Software Engineering practitioners, who will be able to express their actual ways of working using KUALI-BEH to author methods and practices. The knowledge produced by practitioners should be validated and approved before being accumulated and shared, both inside and outside the organization.

During the process of authoring of methods and practices KUALI-BEH advises practitioners with regard to certain attributes. The set of practices that comprise a method should preserve the properties of coherency, consistency and sufficiency [6], which are formally defined in Section 3.2.3.

Having expressed methods and practices, practitioners can identify the method to be followed during software project execution. Due to the fact that every project is different, the work team will need to adapt the selected method, for which KUALI-BEH defines the following adaptations operations [6]:

1. substitution,
2. concatenation,
3. combination and
4. splitting.

These operations are explained in Section 3.2.4.

To carry out a formalization process of the software project common concepts, method properties and adaptation operations presented in KUALI-BEH, we used ontologies for the definition of common concepts, SME and Set Theory to state method properties and adaptation operations and DL for knowledge representation. They are described in more detail in the next subsections.

2.3 Ontologies

According to [10] an important challenge faced by current communities of researchers and practitioners in the field of Software Engineering and Technology is the lack of explicit knowledge shared among members of a group/project, with other groups and with other stakeholders.

The ambiguity of natural language implies potential mistakes and nonproductive efforts. Ontologies can mitigate these problems and, furthermore, some authors have intended to use ontologies as the back-bone of software tools and environments [10].

An ontology, defined by [5], is a data model that represents a set of concepts within a domain and relationships between those concepts, and it is used to reason about objects within that domain.

In [11] the main usages of ontologies in Software Engineering are

1. to clarify the knowledge structure,
2. to reduce conceptual and terminological ambiguity, and
3. to allow the sharing of knowledge.

Ontologies are meant to conceptualize; in the words of [11], conceptualization is understood to be "an abstract and simplified version of the world to be represented: a representation of knowledge based on objects, concepts and entities existing within the studied area, as well as the relationships existing among them". In [12] the authors have also considered it important to enrich this definition with the requirements of being formalized so that a machine can process it, and being shared, where the acquired knowledge is the consensus of a community of experts.

Using ontologies brings the following benefits:

1. they can be checked for inconsistencies;
2. reasoning can help to detect derived relationships or implicit class memberships;
3. errors can be removed, thus improving the quality of a knowledge base;
4. ontologies can be imported and shared [5].

Implementation of ontologies in Software Engineering in order to understand a specific field of knowledge is quite broad. For example, engineering of the ontology for the Software Engineering Body of Knowledge [13], software development methodologies and endeavours [14], software maintenance ontology [15], software measurement [16], an ontological approach to the SQL:2003 [17].

As we mentioned before, ontologies represent knowledge items in the form of concepts, relationships and attributes, which must be expressed as statements. There are different formats and languages to represent statements, e.g. Resource Description Framework (RDF) [18] or Web Ontology Language (OWL) [19].

RDF is a general-purpose language for representing and referencing information on the Web, and is intended for situations in which this information needs to be processed by applications rather than be presented to people directly [5]. RDF represents simple statements as a graph of resources, their properties and values. Based on [5] an RDF statement is composed of three elements:

1. Subject that is someone or something considered as a resource, it may be any person or item represented by a Uniform Resource Identifier (URI);
2. Predicate that indicates the subject's relation to another concept or the subject's activity; and
3. Object that defines what the subject is related to or what the subject is doing.

As for OWL, it is a markup language built on RDF and is used to publish and share ontologies on the web [5].

The usage of standardized languages like RDF and OWL helps to define and share ontologies. However, an important part of ontologies is the possibility of generating new knowledge, which is called reasoning. A tool that supports applying logic, querying and reasoning with ontologies is called a reasoner. Examples of reasoners are RACER[1], HermiT[2], FaCT++[3].

It is important to mention that most of the tools that support management of ontologies comprise more than one module, one of which is a reasoner. For the purposes of this formalization HermiT was chosen as a reasoner.

[1] `http://www.ifis.uni-luebeck.de/~moeller/racer/`

[2] `http://hermit-reasoner.com/`

[3] `http://owl.man.ac.uk/factplusplus/`

2.4 Situational Method Engineering

SME focuses on configuration of system development methods tuned to the situation of a project at hand [20]. SME aims to support software engineering by providing means for appropriate method engineering. That includes all aspects of creating, using and adapting a software development method based on local conditions, it is focused on formalising the use of methods for systems development [21]. Any coherent product, activity, or tool being part of an existing generic or situational method is a method fragment [20]. In other words, method fragments are building blocks of a situational method. It is important to mention that, in SME, a formalized method is usually an ideal type created as an abstraction of existing 'good practices' [22].

According to [23], SME contributes to reach the requirements of flexibility, experience accumulation, integration and communication, and quality. By flexibility it is assumed that the method to be used in a certain development project is situational, that is, completely tuned to the project situation at hand. The controlled adaptability of the method allows for the addition and accumulation of the project experience. All methods are based on one common repository, in which the building blocks of methods are also stored; this contributes to integration and communication. The fact that flexibility should be controlled guarantees that the constructed situational method meets the same quality requirements as standard methods.

The relevance of these requirements is present in the current research; in a recent study [21] discusses issues like tailoring a constructed method in order to apply it to a particular context (see flexibility), comparing chunks, fragments and components, and creating a methodology from them. Besides, the author addresses such questions as how to consider methods as action knowledge (see experience accumulation), and how to assess the quality of the method parts, the constructed method and its effectiveness in practice (see quality).

An initial formalization of method fragments is developed by [23] and organized in four groups: Sets, Predicates, Functions and Rules. This particular approach was used to formalize KUALI-BEH.

2.5 Description Logics

DL [24] is a family of knowledge representation (KR) formalisms that represents the knowledge of an application domain [25]. It is a popular formalism for ontologies and is regarded as the foundation of some ontology languages due to the fact that ontologies contain knowledge about a domain in a precise and unambiguous manner [1].

DL has been shown as common language for ontologies appearing in a Software Engineering process that needs support from Knowledge Engineering and is the natural successor in terms of evolution of UML [26]. The basic DL family is the $\mathcal{AL}$-languages [27], and it follows the syntax rule of:

$$C, D \rightarrow A \mid \top \mid \bot \mid \neg A \mid C \sqcap D \mid \forall R.C \mid \exists R.\top.$$

It is possible to create statements and build a knowledge base using this language. A knowledge base in DL comprises two areas: the Terminological Knowledge (TBox) and the Assertional Knowledge (ABox).

The TBox is a collection of concepts and roles. Concepts represent the entities of a *"universe"*, while Roles denote the relations (properties or associations) between these concepts. In other words, the TBox introduces vocabulary of a specific domain. The basic form of a declaration in a TBox is a concept definition, that is, the definition of a new concept in terms of other previously defined concepts. For example, in the context of KUALI-BEH we can define a Practitioner as a Person who is also a Software Engineer, so in DL this concept acquires the following representation:

$$\text{Practitioner} \equiv \text{Person} \sqcap \text{SoftwareEngineer}.$$

On the other hand, the ABox contains assertions about named individuals in terms of this vocabulary, that is, it contains extensional knowledge about the domain of interest. For example:

$$\text{Person}(\text{“Miguel”}) \sqcap \text{SoftwareEngineer}.$$

It states that the individual “Miguel” is a Person and also a Software Engineer. Given the above example of a Tbox, we obtain the assertion that *Miguel is a Practitioner*, which now belongs to the ABox.

There are some restrictions when working with DL and according to [25] the most important are that:

1. only one definition for a concept name is allowed, and
2. definitions are acyclic in the sense that concepts are neither defined in terms of themselves nor in terms of other concepts that indirectly refer to them.

At this point we can observe that Person and Software Engineer are atomic concepts, but DL offers the possibility of building complex descriptions inductively using concept constructors [25]. By adding more constructors to $\mathcal{AL}$ we obtain more expressive languages. Table 1 shows the spectrum of $\mathcal{AL}$ families.

Family	Added Constructor	Expression
$\mathcal{U}$	Union	$C \sqcup D$
$\mathcal{E}$	Full existential quantification	$\exists R.C$
$\mathcal{N}$	At most and at least restrictions	$\leq n$ R, $\geq n$ R
$\mathcal{C}$	Negation	$\neg C$
$\mathcal{O}$	One of	$a_1, \ldots, a_n$
$\mathcal{I}$	Inverse relation	$P, Q, R \rightarrow \text{R}^-$
$\mathcal{Q}$	Qualified number restriction	$\leq nR.C$, $\geq nR.C$
$\mathcal{R}$	Complex role inclusion	$P \circ Q \subseteq \text{R}$

Table 1. DL family of languages, adapted from [26]

Due to the fact that DL belongs to a KR formalism and to the assumption that a KR system always answers user's queries in a reasonable time, the reasoning and decision procedures of DL are its strength [25]. Besides, a knowledge representation system based on DL and derived from a KR formalism assures four main elements: the TBox and the ABox, together with a reasoner (Inference System) and a user interface (Interface), see Figure 3.

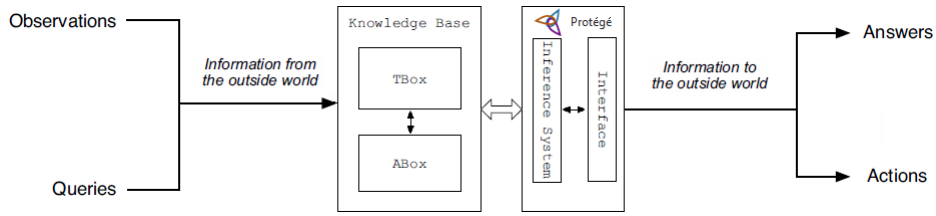


Figure 3. Architecture of a system based on DL, adapted from [1] and [2]

3 KUALI-BEH FORMALIZATION

This section presents the KUALI-BEH formalization, which is divided into three parts: the ontology of common concepts; the representation of method properties and adaptation operations using SME; and the representation of the ontology through DL.

3.1 KUALI-BEH Language

KUALI-BEH language is an initial approach to share a common representation of knowledge as a set of concepts, attributes and relationships of a domain in the form of ontology, abbreviated as KB-O. This ontology offers method engineers the means how to describe, analyze and reason about software projects and information related to them.

Due to its simplicity and the fact that it was created specifically for Software Engineering, Representation Formalism for Software Engineering Ontologies (REFSENO) [28] was chosen to define the KB-O ontology. REFSENO provides constructs to define concepts with their attributes and relationships between them. The construction of REFSENO ontologies is based on three tables that use text and, optionally, diagrams and contain a glossary of concepts, attributes and relationships, respectively. REFSENO allows definition of cardinalities for the relationships and value ranges for the attributes.

The specification of an ontology should contain the modeled domain, the purpose of the ontology, the scope, and administrative information like the authors and knowledge sources [28]. Table 2 displays the KB-O requirements specification. Below

is the KB-O definition based on the general background and the KB-O requirements specification mentioned above.

Domain	Software Projects
Last modified date	January 14, 2016 (updated)
Conceptualized by	KUALI-KAANS Research Group
Purpose	To describe the common concepts involved in software projects and their relationships
Level of formality	Semi-formal (UML Diagrams, text and REFSENO tables)

Table 2. KB-O requirements specification

3.1.1 Definition of KB-O

After establishing the KB-O requirements specification, we carried on with the development of the ontology itself, using the suggested by REFSENO process model and a Unified Modeling Language (UML) [29] Class diagram. Note that for the purpose of this paper, a reduced version of REFSENO is presented in order to maintain it readable and easy to assimilate.

The resulting ontology consists of a graphical representation, a UML class diagram, and a textual semi-formal representation of knowledge using REFSENO. Figure 2 shows the corresponding UML class diagram.

3.1.2 Concepts Glossary

The concepts glossary lists alphabetically all the concepts of the ontology. One row of the concepts glossary corresponds to one concept. The columns are labeled Name, Definition and Example, denoting the respective components of the concept definition. REFSENO requires an extra column named References; however, in this case it was omitted and, instead, a reference list with all the sources considered to create the respective concept definition is reported in [6].

Table 3 displays a fragment of the glossary from the KUALI-BEH ontology[4], showing the specific concepts used in this paper in order to illustrate the proposed formalization. The full version can be consulted in [6].

3.1.3 Relationships

Relationships model the way in which a particular software engineering entity is related to other software engineering entities and are labeled as follows: Name, Con-

[4] This definitions were created in the context of ESSENCE standardization process. ISO/IEC 24744:2007 (now 2014) was not considered because of important differences between both efforts, for example in the clabject and powertype concepts. As future work we consider to make a comparison between the concepts of both standards.

Name	Definition	Example
Method	A method is a composition of a coherent, consistent and sufficient set of practices, with a specific purpose that fulfills the stakeholder needs under specific conditions.	Software Implementation
MPI	The methods and practices infrastructure (MPI) is a set of methods and practices learned by the organization members by experience, abstraction or apprehension. This base of knowledge is continuously expanded and modified by the practitioners.	Organizational Base of Knowledge
Practice	A practice is work guidance, with a specific objective, that advises how to produce a result originated from an input. This guide provides a systematic and repeatable set of activities focused on the achievement of the practice objective and result.	Software Requirements Analysis
Practitioner	A practitioner is a professional in Software Engineering that is actively engaged in the discipline. The practitioner should have the ability to make a judgment based on his or her experience and knowledge.	Miguel

Table 3. KB-O concepts glossary

cepts (cardinality) and Description. The relationships of this ontology are equivalent to the non-terminal concept attributes defined in REFSENO. Table 4 shows a subset of the 29 relationships of KB-O.

Name	Concepts (Cardinality)	Description
Consumes	Practice (*) – Input (*)	A practice consumes an input.
Is assigned to	Work Team (*) – Software Project (*)	A work team is assigned to a software project.
Is composed of	Method (*) – Practice (*)	A method is composed of practices.
Is formed of	Work Team (*) – Practitioners (*)	A work team is formed of practitioners.
Produces	Practice (*) – Result (*)	A practice produces a result.

Table 4. KB-O relationships

3.1.4 Attributes

An attribute is represented using the concept attribute table, which is concept-specific and contains one row for every attribute. The columns are labeled as follows: Name, Description, Mandatory and Type. The attributes of this ontology are equivalent to the terminal concept attributes defined in REFSENO. Table 5 presents the attributes of KB-O.

Attribute (of Concept)	**Description**	**Mandatory**	**Type**
Objective (Practice)	Description of the goal that a practice pursues.	Yes	Text
Purpose (Method)	Description of the goal that a method pursues.	Yes	Text
Status (Work Product)	Description of the actual state or situation of a work product.	No	Text

Table 5. KB-O attributes

3.2 KUALI-BEH Algebra

Based on the ideas outlined in [20] and [23], we defined KUALI-BEH algebra (KB-A), which is a set of axioms, predicates, functions and operations to represent the KUALI-BEH method properties and operations. KB-A is defined in the next subsections.

3.2.1 KB-A Axioms and Definitions

The methods and practices infrastructure (MPI) contains all the elements that are built using the common concepts. Therefore, it is the first axiom of KB-A.

Axiom 1. Let $\mathcal{MPI}$ be all the *things* that can be created using KUALI-BEH.

$$\mathcal{MPI} = \{\mathcal{M} \oplus \mathcal{P} \oplus \mathcal{J} \oplus \mathcal{W} \oplus \mathcal{C}\}$$

where

$$\begin{aligned}\mathcal{M} &= \{m \mid m \text{ is a method}\},\\ \mathcal{P} &= \{p \mid p \text{ is a practice}\},\\ \mathcal{J} &= \{j \mid j \text{ is a software project}\},\\ \mathcal{W} &= \{w \mid w \text{ is a work product}\},\\ \mathcal{C} &= \{c \mid c \text{ is a condition}\}.\end{aligned}$$

Axiom 2. $\mathcal{W}$ and $\mathcal{C}$ are disjoint sets.

$$\mathcal{W} \cap \mathcal{C} = \emptyset.$$

Definition 1. Let $p_1, \ldots, p_n$ be practices, a method m composed of this set of practices can be expressed as $m = \{p_1, \ldots, p_n\}$.

Definition 2. The definition of any element of the KUALI-BEH static view is a conceptual definition and is denoted by the letter "c" as super-index. For example, a conceptual definition of a work product w is denoted as w^c, which is related to its characteristics.

Definition 3. The definition of any element of the KUALI-BEH operational view is a technical definition and is denoted by the letter "t" as super-index. For example, a technical definition of a work product w is denoted as w^t, which is its instance.

Definition 4. The practitioners are the only individuals who can determine whether:

1. similarity between inputs and results is held;
2. the method purpose is fully achieved; and
3. the objective of a practice supports a method purpose.

This ability corresponds to the practitioner's judgment.

3.2.2 KB-A Functions and Predicates

In this section we discuss the functions defined in the KB-A.

The objective of a practice p is obtained through the function *objective* applied to the conceptual definition of the practice.

$$\text{objective} : \mathcal{P} \longrightarrow \text{Text}.$$

In a similar way, the purpose of a method m is obtained through the function *purpose* applied to the conceptual definition of the method.

$$\text{purpose} : \mathcal{M} \longrightarrow \text{Text}.$$

In the same way, the status of a work product w is obtained through the function *status* applied to the technical definition of the work product.

$$\text{status} : \mathcal{W} \longrightarrow \text{Text}.$$

In order to decide whether the purpose of a method is fully achieved, the function *fully_achieved* is defined.

$$\text{fully\_achieved} : \mathcal{M} \longrightarrow \mathbb{B}.$$

The function *similarity* is defined in order to decide whether a work product or condition fits the characteristics required by an input or result. Comparing the

technical definition against the conceptual definition of a work product or condition, practitioners can decide on their similarity.

$$\text{similarity} : (\mathcal{W}^c \cup \mathcal{C}^c) \times (\mathcal{W}^t \cup \mathcal{C}^t) \longrightarrow \mathbb{B}.$$

Likewise, the functions *input* and *result* are defined for methods and practices. These functions receive a practice or a method and return a set of work products and/or conditions.

$$\text{input} : \mathcal{M} \cup \mathcal{P} \longrightarrow \mathcal{W} \cup \mathcal{C},$$

$$\text{result} : \mathcal{M} \cup \mathcal{P} \longrightarrow \mathcal{W} \cup \mathcal{C}.$$

The KUALI-BEH predicates are expressed in the following way:

- produces(p_i, r) denotes that a practice p_i produces a result r:

$$\text{produces} : \mathcal{P} \times (\mathcal{W} \cup \mathcal{C}),$$

- consumes(p_j, i) denotes that a practice p_j consumes an input i:

$$\text{consumes} : \mathcal{P} \times (\mathcal{W} \cup \mathcal{C}),$$

- precedes(p_i, p_j) denotes that a practice p_i precedes a practice p_j:

$$\text{precedes} : \mathcal{P} \times \mathcal{P},$$

- follows(p_j, p_k) denotes that a practice p_j follows a practice p_k:

$$\text{follows} : \mathcal{P} \times \mathcal{P},$$

- supports(p, m) denotes that the objective of a practice p supports the purpose of a method m:

$$\text{supports} : \mathcal{P}^c \times \mathcal{M}^c.$$

3.2.3 KB-A Method Properties

In order to represent method properties, as stated in Definition 1, let us define a method $m \in \mathcal{M}$ as a set:

$$m = \{p \mid p \ \in \mathcal{P}\}.$$

The coherency, consistency and sufficiency properties of a method are defined below.

Coherency. Let us define a function named *coherency*, which receives a method and returns *true* if it is coherent or *false* if it is not.

$$\text{coherency} : \mathcal{M} \longrightarrow \mathbb{B}.$$

This function **coherency(m)** is evaluated as follows:

$$\text{coherency}(m) = \begin{cases} \textit{true} & \text{if for all practices } p_i \text{ of a method } m, \text{ the objective} \\ & \text{of } p_i \text{ supports the purpose of } m, \\ \textit{false} & \text{otherwise.} \end{cases} \tag{1}$$

In other words we have:

$$\text{if } \forall p \in m, \text{supports}(\text{objective}(p^c), \text{purpose}(m^c)) = \textit{true}.$$

Consistency. Let us define the function *consistency*, which receives a method and returns *true* if it is consistent or *false* if it is not.

$$\text{consistency} : \mathcal{M} \longrightarrow \mathbb{B}.$$

This function **consistency(m)** is evaluated as follows:

$$\text{consistency}(m) = \begin{cases} \textit{true} & \text{if all the practice inputs are produced and all} \\ & \text{the practice results are consumed, except for} \\ & \text{the method input and result;} \\ \textit{false} & \text{otherwise.} \end{cases} \tag{2}$$

In other words we have:

$$\begin{aligned} \text{if } \forall p_1 \in m\ \exists p_2, p_3 \in m(&\text{produces}(p_1, r) \rightarrow \text{consumes}(p_2, r) \vee \text{result}(m^c) = r) \\ &\wedge (\text{consumes}(p_1, i) \rightarrow \text{produces}(p_3, i) \vee \text{input}(m^c) = i). \end{aligned}$$

Sufficiency. Let us define the function *sufficiency*, which receives a method and returns *true* if it is sufficient or *false* if it is not.

$$\text{sufficiency} : \mathcal{M} \longrightarrow \mathbb{B}.$$

This function **sufficiency(m)** is evaluated as follows:

$$\text{sufficiency}(m) = \begin{cases} \textit{true} & \text{if the method } m \text{ is coherent, consistent and its} \\ & \text{purpose is fully achieved,} \\ \textit{false} & \text{otherwise.} \end{cases} \tag{3}$$

In other words we have:

$$\text{if coherency}(m) \wedge \text{consistency}(m) \wedge \text{fully\_achieved}(m) = \textit{true}.$$

3.2.4 KB-A Adaptation Operations

In order to express the operations of adaptation, let us define a practice P as a triple formed by an Input (I), an Objective (O) and a Result (R)

$$P = (I, O, R).$$

The operations of substitution, concatenation, combination and splitting are defined below.

Substitution. The *substitution* of practices consists in replacing a practice by another equivalent practice.

$$\begin{gathered}\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices.}\\ P_1 \text{ can be } \textit{substituted} \text{ by } P_2 \text{ if and only if}\\ P_1 \equiv P_2.\end{gathered}$$

Notice that similarity is recognized and dictated by the practitioner's judgment. After applying the adaptation operation the original properties of a method are preserved, since the new practice holds an objective, input and result similar to the substituted practice.

Concatenation. If one practice has a result similar to the input of another practice, both can be integrated into one practice, applying the *concatenation* operation, which is defined as follows:

$$\begin{gathered}\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices and } R_1 \text{ is similar to } I_2.\\ \text{A practice } P_3 \text{ is a correct } \textit{concatenation} \text{ of the practices } P_1 \text{ and } P_2 \text{ if}\\ P_3 = (I_1, O_1 \wedge O_2, R_2).\end{gathered}$$

Combination. A *combination* of practices consists of bringing two different practices into one and is defined as follows:

$$\begin{gathered}\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices.}\\ P = (I, O, R) \text{ is a correct } \textit{combination} \text{ of } P_1 \text{ and } P_2 \text{ if}\\ I \text{ is similar to } I_1 \cup I_2 \text{ and}\\ R \text{ is similar to } R_1 \cup R_2 \text{ and}\\ O \equiv O_1 \wedge O_2.\end{gathered}$$

Splitting. A *splitting* of practices consists in the partition of the original practice into two different practices preserving the original objective and similar inputs and results. Formally, the splitting operation is defined as follows:

$$\text{Let } P_1 = (I_1, O_1, R_1) \text{ and } P_2 = (I_2, O_2, R_2) \text{ be practices.}$$
$$P_1 \text{ and } P_2 \text{ are a correct } \textit{split} \text{ of } P = (I, O, R) \text{ if}$$
$$I_1 \cup I_2 \text{ is similar to } I \text{ and}$$
$$R_1 \cup R_2 \text{ is similar to } R \text{ and}$$
$$O_1 \wedge O_2 \equiv O.$$

Strictly following these rules while applying the adaptation operations to practices assures the preservation of the original properties of coherency, consistency and sufficiency of a method.

3.3 KUALI-BEH Knowledge

In this section we discuss KB-K, short for the knowledge representation of KUALI-BEH using DL and based on KB-O. To make a clear picture of the process of creating KB-K, we will provide some examples. Let us consider the concept practitioner defined in Table 3 of KB-O. This concept can be defined using sets as:

$$\{x \mid \text{Practitioner}(x)\}.$$

To express roles we proceed in a similar way. Let us consider the relationship *isFormedOf* presented in Table 4. This relationship expresses that a work team is formed of practitioners, and it can be denoted as follows:

$$\{(x, y) \mid \text{isFormedOf}(x, y)\}.$$

Now let us transform these sets into First Order Logic (FOL) predicates, which is a common transformation process of semantic networks and ontologies. Let us assume that every work team in our organization must be formed of practitioners. This assertion can be rewritten in FOL as:

$$\forall x.\text{WorkTeam}(x) \rightarrow \exists y.\text{isFormedOf}(x, y) \wedge \text{Practitioner}(y).$$

Then, this predicate can be rewritten again, but now in DL:

$$\text{WorkTeam} \sqsubseteq \forall \text{isFormedOf.Practitioner}.$$

In a similar way, the inverse relationship of *isFormedOf* can be represented as the relationship *belongsTo*. This expresses that each practitioner in the organization belongs to a work team, and can be represented as follows:

$$\text{Practitioner} \sqsubseteq \exists \text{belongsTo.WorkTeam}.$$

The creation process of KB-K used two DL association types:

has-part: This type can be used to denote aggregation and composition in UML [30]. Although these associations are completely different in UML, since they denote strong and weak relationships, in DL has-part can represent both of them.

is-a: This type is equivalent to the generalization in Entity-Relationship model or to inheritance in Object Oriented paradigm. It is a general association that can be interpreted as is-a-kind-of and is-an-instance-of.

For example, we identified in KUALI-BEH that an Activity consists of four elements: Knowledge and Skills, Tasks, Tools and Measures (see Figure 2). So, expressing in DL that an Activity *has-parts* we have:

$$\begin{aligned}\text{Activity} \sqsubseteq (&\exists\, \text{requires.KnowledgeAndSkills}) \wedge \\ &((\exists\, \text{isDecomposedIn.Task}) \vee \\ &(\exists\, \text{isCarriedOutUsing.Tool}) \vee \\ &(\exists\, \text{isMeasuredIn.Measure}) \vee \mathit{true}).\end{aligned}$$

Notice that only Knowledge and Skills are mandatory, while Task, Tool and Measure are not.

In KUALI-BEH the Stakeholder Needs is a specialization of a WorkProduct. Then, we can say that *StakeholderNeeds is-a WorkProduct*, which is written in DL as:

$$\text{StakeholderNeeds} \sqsubseteq \text{WorkProduct}.$$

This association is used for the three instances defined in KB-O: Stakeholder Needs, Project Conditions and Software Product.

To represent an attribute, for example, we know that a *WorkProduct* has an attribute named *status* and its datatype is String, so we have:

$$\{x \mid \text{WorkProduct}(x) \wedge (\exists s.\text{Status}(x, s) \wedge \text{String}(s))\}.$$

In DL the datatype equivalent to String is Text, so we can represent a Work Product in DL as follows:

$$\text{WorkProduct} \sqsubseteq (= 1\ (\text{status.Text})).$$

Let us examine a more general example: if we have a work product named *Database Model* and its status is *Draft*, in DL we have:

$$\begin{aligned}&\text{WorkProduct}(\text{“DatabaseModel”}) \wedge \\ &\text{Status}(\text{“DatabaseModel”}, \text{“Draft”}) \wedge \text{Text}(\text{“Draft”}).\end{aligned}$$

3.3.1 Definition of KB-K

After having transformed KB-O into DL expressions, we generated KB-K, which is defined as follows:

Method:

$$\sqsubseteq (= 1\ (\text{mpi.MPI})) \wedge (\exists\, \text{isComposedOf.Practice}) \wedge (\exists\, \text{isEnactedIn.SoftwareProject}) \wedge (= 1\ (\text{purpose.Text})).$$

MPI:

$$\sqsubseteq (\exists\, \text{stores.SoftwareProject}) \vee (\exists\, \text{stores.Method}) \vee (\exists\, \text{stores.Practice}) \vee (\exists\, \text{stores.WorkProduct}) \vee (\exists\, \text{stores.Condition}).$$

Practice:

$$\sqsubseteq (\exists\, \text{method.Method}) \wedge (\exists\, \text{consumes.Input}) \wedge (\exists\, \text{produces.Result}) \wedge (\exists\, \text{isVerifiedUsing.VerificationCriteria}) \wedge (\exists\, \text{has.Activity}) \wedge (= 1\ (\text{objective.Text})).$$

Practitioner:

$$\sqsubseteq (\exists\, \text{workTeam.WorkTeam}) \wedge (\exists\, \text{possesses.KnowledgeAndSkills}).$$

WorkTeam:

$$\sqsubseteq (\exists\, \text{isFormedOf.Practitioner}) \wedge (\exists\, \text{isAssignedTo.SoftwareProject}).$$

Due to space restrictions, only 4 definitions are presented. The full KB-K is available online on WebProtege site[5].

[5] http://webprotege.stanford.edu/#Edit:projectId=d846a165-258c-4ce9-b703-44a29dd690c8

4 VALIDATION

This section presents a KB-K proof of concept and exemplifies the KB-K usage through a method, which was expressed in a real organization during the validation stage of KUALI-BEH. At the end, we present a comparison with related work.

4.1 Proof of Concept

As it has been mentioned before, DL systems not only store terminologies and assertions, but also offer services to reason about them. Typical reasoning tasks are to determine whether a description is satisfiable (i.e., non-contradictory) [25]. In order to show the usage of KB-K, we offer a proof of concept through an example in the next subsections.

4.1.1 Source of the Example

To prove the usefulness of the proposed formalization, we need an example of an expressed way of working that would contain a description of a software project activity. In this case, the example was taken from Annex C: Case Studies and Examples of [8] and is a fragment of a typical information system delivery process done in Fujitsu DMR Macroscope modeled in SPEM:

The *Information System Delivery Process* is developed during the *Preliminary Analysis* phase and consists of only one iteration that is the *First Joint Requirements Planning Workshop*. In this iteration the *Define Owner Requirements* task is developed by the *System Architect* role in three steps: *Define objectives based on stated needs*, *Define the key issues* and *Determine the relevant enterprise principles*. Besides, one work product is required to start the task and two new items are produced, the *Enterprise Architecture*, *Assessment of Current System* and *Owner Requirements*, respectively.

The specific SPEM elements appear in bold fonts. Acronyms of each element of the process are presented in the parenthesis, this with the purpose of facilitating process representations in KB-K.

Activity {kind: Phase}: Preliminary Analysis (PA)
  Process: Information System Delivery Process (ISDP)
    Activity {kind: Iteration}: First Joint Requirements Planning Workshop (FJRPW)
      TaskUse: Define Owner Requirements (DOR)
        RoleUse: System Architect (SA)
        WorkDefinitionParameter {kind: in}
          WorkProductUse: Enterprise Architecture (EA)
        WorkDefinitionParameter {kind: out}
          WorkProductUse: Assessment of Current System (ACS) {state: initial draft}

WorkProductUse: Owner Requirements (OR)

{state: initial draft}

Steps

Step: Define objectives based on stated needs (DOBOSN)

Step: Define the key issues (DTKI)

Step: Determine the relevant enterprise principles (DREP)

At this point we have an expressed way of working, but how well is it defined? It was implied in [1] that the process was not fully completed and five inconsistencies related to two major classes of problems were pointed out. Firstly, *Phase* and *Iteration* have no performer. Since both elements are a specialization of an Activity, they must have at least one performer each. Secondly, the *WorkProductUse*'s kinds are not clear, because they are defined through the *WorkDefinitionParameter*. And it is not possible to know precisely if a specific work product is input or output.

In order to prove that KUALI-BEH solves these inconsistencies, the Fujitsu way of working was modeled using the formalization presented in this paper.

4.1.2 Example of KB-K Usage

To provide a comparison between SPEM and KUALI-BEH, a partial mapping between the KUALI-BEH concepts and the SPEM elements was done. Table 6 shows a subset of the mapping and contains only the concepts required for the purpose of the example.

SPEM	KUALI-BEH
Process	Method
Activity	Practice
TaskUse	Activity
Step	Task
RoleUse	KnowledgeAndSkills
WorkProductUse	WorkProduct

Table 6. Mapping between KUALI-BEH and SPEM concepts

This research followed the idea of [31] who propose the use of ontologies for the evaluation of metamodels. In their study, they create a reference ontology, i.e., an ontology of method in general against which they can then compare any given 'branded' method or SME approach [21].

Based on the mapping (Table 6), we obtain the following expressions, which are equivalent to the process fragment presented in the previous subsection:

$$\begin{gathered}\text{Method}(\text{“ISDP”}) \wedge \text{isComposedOf}(\text{“ISDP”}, \text{“FJRPW”}) \wedge \\ \text{Practice}(\text{“FJRPW”}) \wedge \text{has}(\text{“FJRPW”}, \text{“DOR”}) \wedge \\ \text{Activity}(\text{“DOR”}) \wedge \text{requires}(\text{“DOR”}, \text{“SA”}) \wedge \\ \text{KnowledgeAndSkills}(\text{“SA”}) \wedge \text{isDecomposedIn}(\text{“DOR”}, \text{“DOBOSN”}) \wedge \\ \text{isDecomposedIn}(\text{“DOR”}, \text{“DTKI”}) \wedge \text{isDecomposedIn}(\text{“DOR”}, \text{“DREP”}) \wedge \\ \text{Task}(\text{“DOBOSN”}) \wedge \text{Task}(\text{“DTKI”}) \wedge \text{Task}(\text{“DREP”}) \wedge \\ \text{input}(\text{“DOR”}, \text{“EA”}) \wedge \text{WorkProduct}(\text{“EA”}) \wedge \\ \text{Status}(\text{“EA”}, \text{“initial draft”}) \wedge \text{Text}(\text{“initial draft”}) \wedge \\ \text{result}(\text{“DOR”}, \text{“ACS”}) \wedge \text{WorkProduct}(\text{“ACS”}) \wedge \\ \text{Status}(\text{“ACS”}, \text{“initial draft”}) \wedge \text{Text}(\text{“initial draft”}) \wedge \\ \text{result}(\text{“DOR”}, \text{“OR”}) \wedge \text{WorkProduct}(\text{“OR”}).\end{gathered}$$

It is worth mentioning that using Protégé (version 5.0.0) and HermiT (version 1.3.8.413) we demonstrated the satisfiability of this example. Figure 4 shows the resulting KB-K graph generated using Protégé.

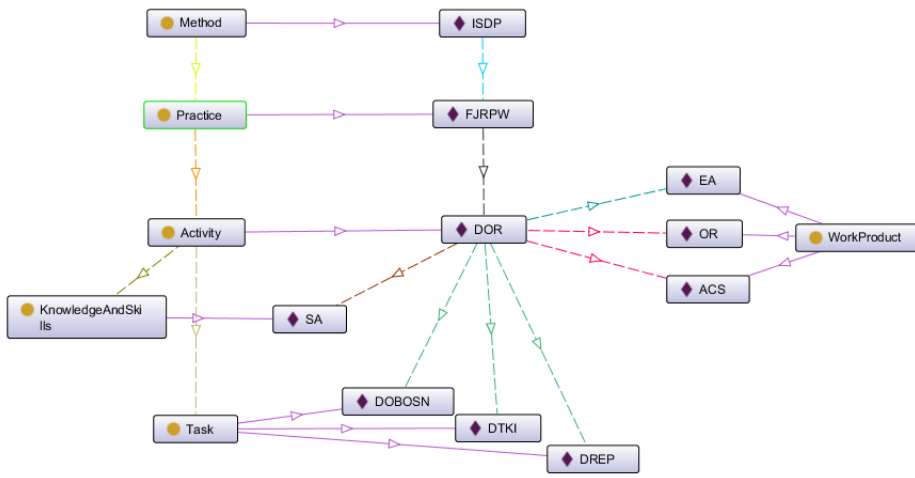


Figure 4. KB-K inferred model

It can be noticed that DOR has a performer (SA) and each of the work products associated with DOR can be differentiated as inputs (blue arrow) or results (pink arrows).

4.1.3 Results of the Example

It was pointed out by Wang [1] that direct reasoning about models built using metamodels like SPEM is difficult and it is hard to keep these models consistent, thus making SPEM to be not the best choice when analyzing models. Nevertheless, we found that KUALI-BEH is capable of representing ways of working through a clear model that preserves its structure [32].

Our example has demonstrated how the SPEM inconsistencies found by [1] can be corrected. These inconsistencies are solved by KUALI-BEH because

1. KUALI-BEH has a well-defined hierarchical approach, and
2. KUALI-BEH does not have reflexive associations.

Since KUALI-BEH's structure is a tree, the dependencies of its elements can be hierarchically stated. As a consequence, the practitioners' ways of working are modeled as a fixed structure avoiding ambiguities and allowing a uniform interpretation of data.

Importantly, KUALI-BEH has no cyclical dependencies and no element can be defined recursively, which is a major drawback in SPEM. For example, in the above described Fujitsu process, an *Activity* can represent either a process or an iteration using the stereotype *kind*; however, the elements that form an activity remain the same regardless of the fact that different kinds of objects are modeled. In fact, this constitutes the main cause of inconsistencies found in [1].

Last but not least, the coherency, consistency and sufficiency properties, defined in KUALI-BEH, can be evaluated and are meant to decide if the expressed way of working is well or ill-formed, thus giving practitioners an initial means of verifying and improving their own ways of working.

4.2 Examples Obtained from Case Studies

During the validation of KUALI-BEH, specifically during case studies, we identified situations where the formalization was used in order to improve practices and methods created by participants. It is important to mention that the validation focus was not the formalization per se, however, the case study participants naturally used properties and operations to improve their practices and methods.

The first appearance of a formalization element occurred when practitioners of case study 1, reported in [32], evaluated the consistency of its method, called IB. In particular, one practice produced two results that consequently became input for another practice. The problem was that the practice did not consume one of the results. This behavior occurred because the practices were executed by different practitioners. After having expressed explicitly the whole method, both practitioners realized that there exists a practice which result (a subset of it) is not similar to the input of the rest of the method's practices. In other words, the practice was

producing a result that nobody else consumed:

$$\exists p_1 \in IB(\text{produces}(p_1, r) \text{ but } \nexists p_i \in IB \text{ consumes}(p_i, r).$$

Therefore:

$$\text{consistency}(IB) = \textit{false}.$$

Another instance of the formalization occurred during the case study 3. At a particular moment of method authoring, the participants decided to adapt the defined method using adaptation operations. On the one hand, they needed to split a practice in order to make it easier to execute and distribute the work. On the other hand, they wanted to merge four practices having similar objectives and create a more generic practice that could be applied in different contexts. When this was done empirically, we evaluated the rules defined by the adaptation operations against the method properties confirming that the latter were totally preserved.

4.3 Comparison with Related Work

Method engineering approaches offer guidance to express methods and methods fragments with the purpose of formalizing knowledge and tailoring methods to particular situations. Aharoni [33] identifies four approaches: the OPEN Process Framework (OPF) [34], the assembly-based SME approach [35], the scenario-based approach [36] and the application-based approach [37].

These fragment representation approaches were evaluated by Aharoni in terms of expressiveness, consistency, formalism and comprehensibility. He reports the lack of comprehensibility of the obtained representations where only 1 of 4 approached supports assembly operations and formalisms. Besides, all the four approaches rely on visual semi-formal languages. There is a special need to provide a formal representation and to define adequate operations to manipulate methods.

On the other hand, these approaches require the means to represent methods or processes, i.e. a process modeling language (PML). For example, an extensively used PML in software engineering is UML. In [38] several other PMLs, such as Petri nets, Business Process Model and Notation (BPMN) [39] and SPEM are mentioned.

Analyzing the variety of alternatives to modelling methodologies and hence many aspects of SME, it is generally agreed that there are at least three core elements: producer, work unit and work product [21]. SPEM and ISO/IEC 24744 [40] follow this line and, together with BPMN and UML, are the most representative alternatives for modeling processes.

However, some drawbacks can be identified. Firstly, SPEM, which is also a OMG standard specially created for Software Engineering methods, is perceived as very complex and hard to tackle making it difficult to learn [41]. Second, despite formalism and structure advantages of the ISO/IEC 24744, the introduced concepts are not only unfamiliar to practitioners, but also are distanced from their context [42]. Finally, as reported by [43], the important drawbacks of UML, BPMN

and other PMLs remain deficiency of evolution, evaluation and human decision support.

KUALI-BEH formalization supports a comprehensive representation of knowledge and permits to reason about created methods. On the one hand, it is based on a simpler set of concepts (KB-O), allows a formal representation and manipulation of methods (KB-A) and provides a method rationale mechanism (KB-K). On the other hand, the KUALI-BEH concepts are compatible with alternatives used in the software engineering community, e.g. SPEM, BPMN or UML. Therefore, any process that is modelled with these alternatives can be formalized through KUALI-BEH taking advantage of its simplicity, common language and method reasoning.

5 INTENDED USAGE

According to [44], the main objectives of formalizations are to improve the communication, computational inference and reuse of knowledge. As a whole, the formalization of KUALI-BEH (KB-K, KB-O and KB-A) is a starting point to motivate and improve these aspects, giving practitioners and organizations a viable alternative to structure their wide tacit knowledge.

KB-O offers a unified vocabulary (terms and definitions) and improves communication among humans, consequently discussions and agreements over a specific domain are possible.

Applying KB-A, its rules and properties, we can manipulate knowledge in a uniform and standardized manner and achieve communication between humans and computers.

With KB-K, comprising a knowledge base and inference rules, the communication between software tools will foster the exchange and analysis of data. Besides, providing a structure and a management mechanism to the knowledge involved in software projects, we will be able to make computational inferences. Moreover, practitioners will have means to evaluate and enrich their knowledge.

It is important to keep in mind that the target audience of KUALI-BEH formalization is process engineers; this formalization aims at providing them with the means of description, analysis and reasoning about software projects.

According to [45], an advantage for knowledge representation is the coupling between theory and practice. In fact, KUALI-BEH was born as a proposal to bridge the gap between theory and practice by structuring and reasoning about practitioners' knowledge and enriching the Software Engineering body of knowledge. It has also been validated by practitioners directly involved in software developer organizations with solid results [32], which has encouraged its formalization.

Finally, the main intended usage of the formalization is to establish the basis for creating a Computer-Aided Method Engineering (CAME) tool. There is a necessity for creating a tool that would fulfill the needs defined by [23] or [46], and which were

never completely achieved, based on the words of [47]: "Unfortunately none of these tools can express the process part of a method and support the enactment of the method process mode". We hope that KB-K, -A, and -O will become the basis of a tool that really helps practitioners to carry out the processes of method authoring and enactment.

The tool, named KB-Tool, is currently being developed. We already released two modules: the first module expresses and shares ways of working using KB-O; the second module uses KB-A to determine the accomplishment of method properties. The module that integrates KB-K with the reasoning process is still a prototype and is under development. The idea is to go from manual to systematic (semi-automatic) and then to automatic approaches [9]. As it was mentioned before, the proof of concept used the functionalities provided by Protégé.

6 CONCLUSIONS AND FUTURE WORK

KUALI-BEH formalization preserves the foundations of SME. On the one hand it defines the common concepts required to express the practitioners' ways of working by taking advantage of KUALI-BEH ontology. On the other hand, it permits the adaptation of its elements, practices and methods, in a controlled manner using the KUALI-BEH adaptation operations and properties defined as a set of axioms, definitions, predicates and functions. Following these rules it is possible to customize, modify and assemble complex elements from other elements.

Moreover, the hierarchical organization of KUALI-BEH common concepts allows for the consistency on the granularity level of its elements, no matter if it is analyzed separately or as a part of a whole.

Finally, through DL, a knowledge representation formalism, which is able to capture virtually almost all class-based representation formalisms used in Artificial Intelligence, Software Engineering, and Databases [45], we can improve communication among humans and machines, allow for computational inference and promote the reuse of knowledge.

We can conclude that with KUALI-BEH formalization it is possible to build a knowledge base with the following characteristics:

1. it has the necessary knowledge (completeness);
2. the knowledge is reliable to the real world (correctness);
3. the knowledge is not self-contradictory (consistency); and
4. the system has efficient algorithms to perform inferences (competence), which, according to [2], are the stringent requirements for a knowledge base.

Besides, as stated in [48], the practical usefulness of a formal semantics for a language is that it provides a rigorous standard that can be used to judge the correctness of an implementation, in our case the correct forming of ways of working.

This work is at the initial stage, and the DL representation of KUALI-BEH is the first step to provide method engineers with an alternative to reason directly about the actual knowledge, which is expressed by practitioners themselves, so several lines arise as future work. In the first place our research group will work on:

1. enhancing the robustness and completeness of KB-A;
2. proving the usefulness and applicability of KB-K applying it to more real life cases and ways of working;
3. making the formalization available to method engineers for feedback and improvements; and
4. use existing theory expression methods, like Petri nets or fuzzy logic, to capture and check inconsistencies of expressed ways of working.

Second, but not less important is to develop and release a fully functional technological environment, which will motivate more practitioners to use KUALI-BEH and will result in the spread of the proposal and its formalization.

Acknowledgment

The authors thank Pascual Julian Iranzo, Ph.D. for his expert advice in Description Logics.

This work has been funded by the Postdoctoral Fellowships Program of the General Directorate of the Academic Staff (DGAPA-UNAM) and the PAPIIT project IN113013 (DGAPA-UNAM); the Graduate Science and Engineering Computing (PCIC-UNAM) and CONACYT (Mexico).

REFERENCES

[1] Wang, S.—Jin, L.—Jin, C.: Represent Software Process Engineering Metamodel in Description Logic. Proceedings of World Academy of Science, Engineering and Technology, Vol. 11, 2006, pp. 109–113.

[2] Devanbu, P. T.—Jones, M. A.: The Use of Description Logics in KBSE Systems: Experience Report. Proceedings of the 16th International Conference on Software Engineering (ICSE '94), Los Alamitos, CA, USA. IEEE Computer Society Press, 1992, pp. 23–35.

[3] Edwards, J.: Managing Software Engineers and Their Knowledge. In: Aurum, A., Jeffery, R., Wohlin, C., Handzic, M. (Eds.): Managing Software Engineering Knowledge. Springer, Berlin, Heidelberg, 2003, pp. 5–27, doi: 10.1007/978-3-662-05129-0_1.

[4] Selfridge, P. G.—Hoebel, L. J.—White, D. A.: The Sixth Annual Knowledge-Based Software Engineering Conference (KBSE-91). SIGART Bulletin, Vol. 3, 1992, No. 1, pp. 33–35, doi: 10.1145/130836.130839.

[5] Schneider, K.: Experience and Knowledge Management in Software Engineering. Springer-Verlag, Berlin, Heidelberg, 2009, doi: 10.1007/978-3-540-95880-2.

[6] Morales-Trujillo, M.—Oktaba, H.: KUALI-BEH Software Project Common Concepts. Technical Report, Object Management Group, Needham, MA, USA, 2012.

[7] OMG. ESSENCE – Kernel and Language for Software Engineering Methods. Version 1.0, Formal/2014-11-02, Object Management Group, Needham, MA, USA, 2014.

[8] OMG. Software and Systems Process Engineering Metamodel (SPEM). Version 2.0, Formal/2008-04-01, Object Management Group, Needham, MA, USA, 2008.

[9] Masalagiu, C.—Chin, W.-N.—Andrei, Ş.—Alaiba, V.: A Rigorous Methodology for Specification and Verification of Business Processes. Formal Aspects of Computing, Vol. 21, 2009, No. 5, pp. 495–510, doi: 10.1007/s00165-009-0106-y.

[10] Calero, C.—Ruiz, F.—Piattini, M. (Eds.): Ontologies for Software Engineering and Software Technology. Springer-Verlag, Berlin, Heidelberg, 2006, doi: 10.1007/3-540-34518-3.

[11] Ruiz, F.—Hilera, J.: Using Ontologies in Software Engineering and Technology. In: Calero, C., Ruiz, F., Piattini, M. (Eds.): Ontologies for Software Engineering and Software Technology. Springer, Berlin, Heidelberg, 2006, pp. 62–119.

[12] Gómez-Pérez, A.—Fernández-López, M.—Corcho, O.: Ontological Engineering. Springer-Verlag, London, 2004.

[13] Abran, A.—Cuadrado, J.—García-Barriocanal, E.—Mendes, O.—Sánchez-Alonso, S.—Sicilia, M.: Engineering the Ontology for the SWEBOK: Issues and Techniques. In: Calero, C., Ruiz, F., Piattini, M. (Eds.): Ontologies for Software Engineering and Software Technology. Springer, Berlin, Heidelberg, 2006, pp. 120–138.

[14] González-Pérez, C.—Henderson-Sellers, B.: An Ontology for Software Development Methodologies and Endeavours. In: Calero, C., Ruiz, F., Piattini, M. (Eds.): Ontologies for Software Engineering and Software Technology. Springer, Berlin, Heidelberg, 2006, pp. 139–168.

[15] Dias, M.—Anquetil, N.—De Oliveira, K.: Organizing the Knowledge Used in Software Maintenance. Journal of Universal Computer Science, Vol. 9, 2003, No. 7, pp. 641–658.

[16] García, F.—Bertoa, M. F.—Calero, C.—Vallecillo, A.—Ruíz, F.—Piattini, M.—Genero, M.: Towards a Consistent Terminology for Software Measurement. Information and Software Technology, Vol. 48, 2006, No. 8, pp. 631–644.

[17] Calero, C.—Ruiz, F.—Baroni, A.—Brito, F.—Piattini, M.: An Ontological Approach to Describe the SQL:2003 Object-Relational Features. Computer Standards and Interfaces, Vol. 28, 2006, No. 6, pp. 695–713, doi: 10.1016/j.csi.2005.09.002.

[18] W3C. Web Ontology Language. Standard, World Wide Web Consortium, Cambridge, MA, 2012.

[19] W3C. Resource Description Framework. Standard, World Wide Web Consortium, Cambridge, MA, 2014.

[20] Harmsen, F.—Brinkkemper, S.: Design and Implementation of a Method Base Management System for a Situational CASE Environment. Proceedings of the Asia Pacific Software Engineering Conference, 1995, doi: 10.1109/APSEC.1995.496992.

[21] Henderson-Sellers, B.—Ralyté, J.—Agerfalk, P.–Rossi, M.: Situational Method Engineering. Springer-Verlag, 2014. ISBN 978-3-642-41466-4, doi: 10.1007/978-3-642-41467-1.

[22] Agerfalk, P.—Ahlgren, K.: Modelling the Rationale of Methods. In: Khosrowpour, M. (Ed.): Managing Information Technology Resources in Organizations in the Next Millennium. Proceedings of the 10th Information Resources Management Association International Conference. IDEA Group, Hershey, PA, 1999, pp. 184–190.

[23] Harmsen, F.—Brinkkemper, S.—Oei, H.: Situational Method Engineering for Information System Project Approaches. In: Verrijn Stuart, A. A., Olle, T. W. (Eds.): Methods and Associated Tools for the Information Systems Life Cycle. Proceedings of the IFIP WG 8.1 Working Conference, Maastricht, Netherlands, September 1994. IFIP Transactions A-55, North-Holland, 1994, pp. 169–194. ISBN 0-444-82074-4.

[24] Baader, F.—Calvanese, D.—McGuinness, D. L.—Nardi, D.—Patel-Schneider, P. F. (Eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, USA, 2003.

[25] Baader, F.—Nutt, W.: Basic Description Logics. In: Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Patel-Schneider, P. F. (Eds.): The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003, pp. 47–100.

[26] Kaplanski, P.: Description Logic as a Common Software Engineering Artifacts Language. 1st International Conference on Information Technology (IT 2008), IEEE, 2008, pp. 1–4.

[27] Schmidt-Schauss, M.—Smolka, G.: Attributive Concept Descriptions with Complements. Artificial Intelligence, Vol. 48, 1991, No. 1, pp. 1–26.

[28] Tautz, C.—von Wangenheim, C.: REFSENO: A Representation Formalism for Software Engineering Ontologies. IESE-Report No. 015.98/E, Fraunhofer Institute IESE, 1998.

[29] OMG. Unified Modeling Language (UML) Infrastructure. Version 2.5, Formal/15-03-01, Object Management Group, Needham, MA, USA, 2015.

[30] Sattler, U.: Description Logics for the Representation of Aggregated Objects. In: Horn, W. (Ed.): Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, 2000, pp. 239–243.

[31] Iacovelli, A.—Souveyet, C.: Towards Common Ground in SME: An Ontology of Method Descriptors. In: Ralyté, J., Mirbel, I., Deneckère, R. (Eds.): Engineering Methods in the Serviceoriented Context. Proceedings of 4th IFIP WG8.1 Working Conference on Method Engineering (ME 2011), Paris, France. Springer, Heidelberg, IFIP Advances in Information and Communication Technology, Vol. 351, 2011, pp. 77–90.

[32] Morales-Trujillo, M.—Oktaba, H.—Piattini, M.: Using Technical-Action-Research to Validate a Framework for Authoring Software Engineering Methods. 17th International Conference on Enterprise Information Systems (ICEIS '15), INSTICC, 2015, pp. 15–27, doi: 10.5220/0005338800150027.

[33] Aharoni, A.—Reinhartz-Berger, I.: Representation of Method Fragments: A Comparative Study. In: Ralyté, J., Brinkkemper, S., Henderson-Sellers, B. (Eds.):

Situational Method Engineering: Fundamentals and Experiences. Springer, Boston, MA, IFIP Advances in Information and Communication Technology, Vol. 244, 2007, pp. 130–145.

[34] Firesmith, D.—Henderson-Sellers, B.—Zowghi, D.: Using the OPEN Process Framework to Produce a Situation-Specific Requirements Engineering Method. Software Engineering Institute, 2005.

[35] van de Weerd, I.—Brinkkemper, S.—Souer, J.—Versendaal, J.: A Situational Implementation Method for Web-Based Content Management System-Applications: Method Engineering and Validation in Practice. Software Process: Improvement and Practice, Vol. 11, 2006, No. 5, pp. 521–538.

[36] Rolland, C.—Plihon, V.—Ralyté, J.: Specifying the Reuse Context of Scenario Method Chunks. Proceedings of the 10th International Conference on Advanced Information Systems Engineering (CAiSE '98). Springer, Lecture Notes in Computer Science, Vol. 1413, 1998, pp. 191–218, doi: 10.1007/BFb0054226.

[37] Sturm, A.—Reinhartz-Berger, I.: Applying the Application-Based Domain Modeling Approach to UML Structural Views. International Conference on Conceptual Modeling (ER 2004). Lecture Notes in Computer Science, Vol. 3288, 2004, pp. 766–779, doi: 10.1007/978-3-540-30464-7_57.

[38] Kelemen, Z. D.—Kusters, R. J.—Trienekens, J.—Balla, K.: Selecting a Process Modeling Language for Process Based Unification of Multiple Standards and Models. Technical Report TR201304, Budapest, Hungary, 2013.

[39] OMG. Business Process Model and Notation (BPMN). Version 2.0, Formal/2011-01-03, Object Management Group, Needham, MA, USA, 2011.

[40] ISO/IEC, 24744 Software Engineering – Metamodel for Development Methodologies. International Organization for Standardization, 2007.

[41] Niknafs, A.—Asadi, M.: Towards a Process Modeling Language for Method Engineering Support. 2009 WRI World Congress on Computer Science and Information Engineering, 2009, pp. 674–681, doi: 10.1109/CSIE.2009.956, doi: 10.1109/CSIE.2009.956.

[42] Morales-Trujillo, M.—Oktaba, H.—Piattini, M.: Bottom-Up Authoring of Software Engineering Methods and Practices. Journal of Applied Research and Technology, Elsevier, Submitted 2016.

[43] Zamli, K. Z.—Mat Isa, N. A.: A Survey and Analysis of Process Modeling Languages. Malaysian Journal of Computer Science, Vol. 17, 2004, No. 2, pp. 68–89.

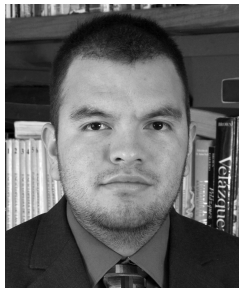
[44] Gruninger, M.—Lee, J.: Ontology: Applications and Design. Communications of the ACM, Vol. 45, No. 2, 2002, pp. 39–41.

[45] Zhang, Y.—Zhang, W.: Description Logic Representation for Requirement Specification. In: Shi, Y., van Albada, G. D., Dongarra, J., Sloot, P. M. A. (Eds.): Computational Science (ICCS 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4488, 2007, pp. 1147–1154.

[46] Abad, Z. S. H.—Sadi, M. H.—Ramsin, R.: Towards Tool Support for Situational Engineering of Agile Methodologies. 2017 17th Asia Pacific Software Engineering Conference (APSEC 2010), IEEE, 2010, pp. 326–335.

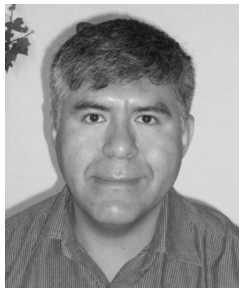
[47] Arni-Bloch, N.: Towards a CAME Tools for Situational Method Engineering. Interoperability of Enterprise Software and Applications, IFIP-ACM, 2005.

[48] Boronat, A.—Meseguer, J.: An Algebraic Semantics for MOF. Formal Aspects of Computing, Vol. 22, 2010, No. 3-4, pp. 269–296, doi: 10.1007/s00165-009-0140-9.



Miguel Morales-Trujillo received his Ph.D. in computer science from the National Autonomous University of Mexico (UNAM). He is the UNAM Representative at the Object Management Group. He has been Assistant Professor at the Science Faculty of the UNAM since 2010. His research interests are software engineering and process engineering.

Hanna Oktaba received her Ph.D. in computer science from the University of Warsaw, Poland. She has been Full Professor at the UNAM since 1983. She was in charge of the MoProSoft project for the Mexican government's PROSOFT program. She is Technical Leader of the Mexican delegation in WG24 of ISO/IEC JCT1 SC7. Nowadays she leads the KUALI-KAANS research group. Her research interests are software engineering and software quality.



Francisco Hernández-Quiroz received his Ph.D. in computer science from the Imperial College of Science, Technology and Medicine in London. He has been Full Professor at the UNAM since 2002. His research interests are computability theory and its practical and philosophical implications, as well as modal logic in computer science and philosophy.

Boris Escalante-Ramírez received his Ph.D. in computer science from the Technical University of Eindhoven. His research interests are computational models of human vision and their applications to digital image processing. He is a member of the National Research System of Mexico.

Computing and Informatics, Vol. 37, 2018, 142–164, doi: 10.4149/cai_2018_1_142

ADAPTIVE AGGREGATION OF FLOW RECORDS

Adrián Pekár, Martin Chovanec

Institute of Computer Technology
Technical University of Košice, Slovakia
e-mail: {adrian.pekar, martin.chovanec}@tuke.sk

Liberios Vokorokos, Eva Chovancová
Peter Feciľak, Miroslav Michalko

Department of Computers and Informatics
Faculty of Electrical Engineering and Informatics
Technical University of Košice, Slovakia
e-mail: {liberios.vokorokos, eva.chovancova,
peter.fecilak, miroslav.michalko}@tuke.sk

Abstract. This paper explores the problem of processing the immense volume of measurement data arising during network traffic monitoring. Due to the ever-increasing demands of current networks, observing accurate information about every single flow is virtually infeasible. In many cases the existing methods for the reduction of flow records are still not sufficient enough. Since the accurate knowledge of flows termed as "heavy-hitters" suffices to fulfill most of the monitoring purposes, we decided to aggregate the flow records pertaining to non-heavy-hitters. However, due to the ever-changing nature of traffic, their identification is a challenge. To overcome this challenge, our proposed approach – the adaptive aggregation of flow records – automatically adjusts its operation to the actual traffic load and to the monitoring requirements. Preliminary experiments in existing network topologies showed that adaptive aggregation efficiently reduces the number of flow records, while a significant proportion of traffic details is preserved.

Keywords: Network traffic monitoring, IPFIX, exporter, flow record, data reduction, adaptive aggregation, heavy-hitter, resource utilization

Mathematics Subject Classification 2010: 68M10, 68M12, 90B18, 90B20

1 INTRODUCTION

Traffic monitoring has a significant role in the design, management and optimization of present computer networks. In order to achieve the full-featured operation of the network it is essential to measure and evaluate various characteristics of the traffic. Nowadays, the most commonly used data measurement methods are based on collecting information about the network and its traffic at the level of flows.

Network monitoring by flow-level based measurement platforms – either implementing the NetFlow v9 [1] or IPFIX [2] protocols – is based on the analysis of information obtained from traffic properties and characteristics. Flow-level information has a wide range of use from analyzing the traffic of the network through anomaly detection up to ensuring QoS. Collecting flow-level information can provide meaningful information about the dynamics of network traffic as well. Further, it is also used for a variety of network monitoring tasks such as user and application monitoring, traffic engineering, capacity planning, accounting, security applications (IDSes and IPSes) and performance analysis. Despite its popularity, flow-level measurement is still surrounded by several issues. As networks are continuously growing in

1. size,
2. connected users and
3. the volume of transmitted data (traffic),

their operation and management are becoming more and more complex. In consequence, current flow-based network monitoring systems generate a huge volume of measurement data what represents one of the most critical issues from the view of both data processing (analysis) and interpretation (visualization) [3].

In the following sections we provide a formal as well as informal description of our systematic approach that have finally led to the design of a yet another adaptive aggregation method. They discuss several methods/techniques and phenomena by the combination of which we achieved the aggregation of flow records and its adaptability to the traffic character.

1.1 State-of-the-Art

Over the past decades many approaches were taken to create a unified standard for monitoring the traffic which flows through the network. In consequence, various techniques and methods were proposed. All of these methods have their advantages and disadvantages. Some of them may excel in data reduction, but, on the other hand, the obtained information may be less detailed and have coarse granularity. On the contrary, other techniques may provide fine granularity, but data reduction may not be sufficient enough. Generally, most of the techniques have been incorporated into unified standards for retrieving network device- and traffic-specific information. Therefore, they are usually associated with a standard or protocol.

The most commonly used protocols in network traffic measurement are *counters* (SNMP counts [4]), *flow-level information* (Netflow/IPFIX [1, 5, 2]) and *sampled flow* (sFlow [6]).

Although SNMP counts are simple and lightweight to process, they do not provide enough details about the network traffic. When details about the traffic semantics are required, three main technologies come into consideration: *packet capture*, *sFlow* and *NetFlow/IPFIX*. Some of them affect the data reduction more and some less. Although NetFlow and IPFIX are surrounded by some challenges, their benefits far outweigh their shortcomings and in comparison with other approaches they provide more flexibility. Thus, collecting flow-level information is currently the most preferred way to perform measurements. It provides measurement data at a relatively high aggregation level while a considerable portion of traffic semantics needed for various network monitoring tasks is still retained. However, gathering flow-level information has to face some issues. The most emerging one is the volume of measurement data [7, 3]. In general, the main aim of the network is to ensure the smooth and fast transfer of traffic data. If the network has to deal with measurement data at the very same time, it can easily cause over-utilization of various network entities (e.g. inter-networking devices) and the network's operation. For example, if the individual inter-networking devices are involved in forwarding of measurement data between the observation point(s) and the monitoring system(s), their performance can be adversely affected. Similar situation occurs in case of routers which besides capturing packets are involved in the creation and export of flow records.

A recent approach to programmable networks is the Software Defined Networking (SDN) architecture, which is aimed, besides other things, at this issue. As described in [8], the main idea behind SDN is to allow developers to rely on network resources in the same easy manner as they do on computing or storage resources. This is reached by decoupling the control and data planes of the network, i.e., the network intelligence is logically centralized in software-based controllers (or control plane), and network devices become simple packet forwarding devices (or data plane) that can be programmed via an open interface (e.g. ForCES [9], OpenFlow [10, 11], etc.).

OpenFlow is currently the most commonly deployed Software Defined Networking (SDN) technology. Since the source code of the software running on the switches is usually inaccessible nor can be modified, it is difficult for the science-research community to test new ideas in current environments. OpenFlow was proposed to standardize the communication between the switches and the software-based controller in an SDN architecture, thus enabling researchers to test new ideas in a production hardware. It provides means to control a switch without requiring the vendors to expose the code of their devices. In the OpenFlow architecture the software-based controller is responsible for managing the forwarding information of one or more switches; while the hardware only handles the forwarding traffic according to the rules set by the controller [11].

Since OpenFlow separates the control plane and data plane of networking devices [11], it should be considered – as also suggested in [12] – a flow-based configura-

tion technology for packet forwarding devices, rather than a flow export technology. However, although it was not specifically developed for tasks related to data export and network monitoring, according to [13], flow-level information available within the OpenFlow control plane (e.g. packet and byte counters) was recently used for performing network measurements as well.

1.2 IPFIX-Based Measurement Platforms

At present, the vast majority of monitoring tools are performing the measurement of flow-level information using either the NetFlow [1] or the IPFIX [2] protocol. Since we expect IPFIX to be the industry standard for flow monitoring in the near future, and considering the fact that between IPFIX and NetFlow is just a slight difference, in the following we will analyze the process of network traffic monitoring in the context of the IPFIX specification.

IPFIX defines a format and a protocol for the export of information about IP flows. IP flow is defined as an unidirectional stream of IP packets identified by a common five-tuple (flow keys); specifically the protocol type, source IP address, destination IP address, source port and destination port. Basically, network traffic monitoring is based on the analysis of the exported information. The *properties* (e.g. the total number of bytes of all packets belonging to a certain flow) and *characteristics* (e.g. source IP address) of the flow are carried in *flow records*. The export of flow records represents a push-based mechanism, where the data are transmitted from the IPFIX exporter(s) to the IPFIX collector(s) over either the TCP, UDP or the SCTP protocol. Actually, the exporters and the collectors are the essential components of any IPFIX-based measurement platform.

Exporter is a device which basically hosts two processes: the *metering* and the *exporting*. Each of these processes can have one or more instances. In general, each exporter sends flow records to one or more *collectors*. The flow records are generated by the metering process(es).

The architecture of the exporter along with the metering and exporting processes is shown in Figure 1. The inputs for flow record generation are the packets themselves. It logically follows that the essential task of the metering process is packet capture. Its further optional functions include timestamping, packet selection (sampling and filtering) and classification.

Another important function of the metering process is maintaining the flow records. Tasks related to the maintenance of flow records include their creation, update, detection of flow expiration, passing the flow records to the exporting process and their removal. In practice, these tasks are performed using a flow cache.

The exporting process is situated a layer higher (see Figure 1). It provides an interface between the metering process(es) and the collecting process(es). In simple terms, it sends (exports) the flow records obtained from the metering

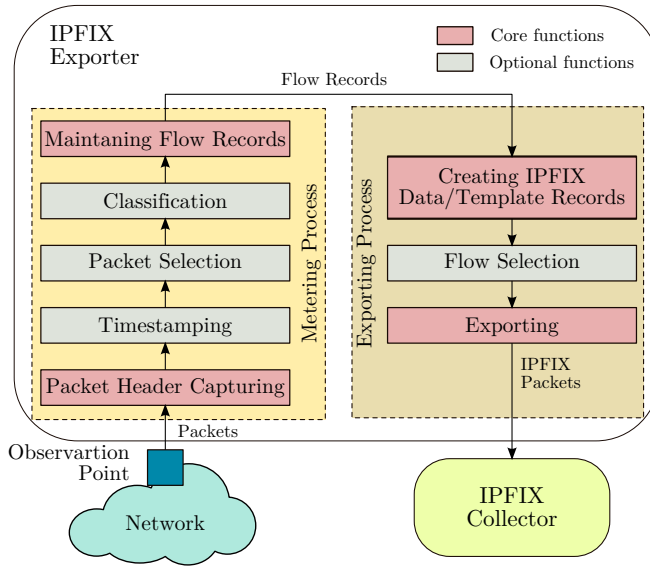


Figure 1. The general architecture of the exporter

process(es) to one or more collectors. The method of exporting the flow records is defined by the IPFIX protocol. Its further functional block is *flow selection* that using sampling or filtering can provide further reduction of data destined for export. This functional block of the exporting process is optional.

The metering process generates the flow records on the basis of the *flow keys*.

Flow keys are used to determine the conditions for creating flow records. In general, flow keys are information elements (IEs) [5] that define IP flows. The most commonly used information elements that serve as flow keys for flow generation are the sourceIPv4Address, destinationIPv4Address, sourceTransportPort, DestinationTransportPort and protocolIdentifier[1]. However, some information elements, for example those which belong to the timestamp and counter groups, cannot serve as flow keys.

Collector is a device which hosts a collecting process. The collecting process receives flow records from one or more exporting processes [5]. The main goal of the collector is to extract the measured properties and characteristics of the flow from the flow records. For efficiency, this information is stored and carried in information elements.

[1] Note that the notation of information elements in this work are in conformity with the IPFIX information model [5].

How the flow records are exported to the collector(s) is defined by the IPFIX protocol. Actually, they are transmitted by two kinds of information: *templates* and *data*. In simple terms, flow records are carried in data records and the structure of these data records is defined by the templates. It follows from the fact that traffic information depends on the purpose of the measurement and the network structure.

The information extracted from the data records can be stored in a *database* and/or directly sent to one or more external *evaluating* entities by mechanisms such as the Analyzer–Collector Protocol (ACP) introduced in one of our previous contributions [14]. Given the base functionality of the external entity (i.e. analysis), in the following we will refer to it as *analyzer*.

Analyzer provides further processing and analysis of the information about flows. More comprehensive analyzers also provide a GUI for both the visualization of the information obtained from the database/collector and the management of the architecture's lower components (i.e. exporter and collector). However, the analyzer itself is not a part of the IPFIX specification. For this reason, neither its requirements and functionalities, nor the communication principles between the analyzer and the other components of the IPFIX-based metering tool are limited.

In conclusion, monitoring and analyzing the network traffic based on the IPFIX protocol, as depicted in Figure 2, can be split into the following steps:

1. The information obtained from the captured packets after timestamping, sampling, classification, etc., are encapsulated into IPFIX messages and sent from the exporter(s) to the collector(s).
2. In the collector, after parsing the currently obtained template/data record, the obtained flow-level data are stored in the database of the metering platform and/or sent directly to the analyzer.
3. The analysis over the flow-level information is performed by an analyzing application. For example, the data obtained from a database can be processed and visualized in a form of plots. Obviously, these plots will vary according to the wanted type of analysis.

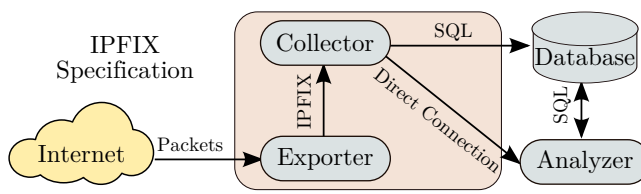


Figure 2. General architecture of a monitoring platform implementing IPFIX

1.3 Problem Statement

When too many flows are present in the traffic, IPFIX-based measurement platforms [15, 16] have to deal with several issues. The root cause of the issues is concerning the exporter(s), however, the immense volume of measurement data (flow records) also has a high impact on the evaluating processes. The issues which can rise during measurements are the following:

- Due to the connectionless nature of the UDP protocol, its use is avoided in many cases. Especially when the loss of information is not affordable (e.g. anomaly detection). It logically follows that TCP can provide a solution. However, when TCP is used and the export rate is too high, due to the (expected) overload of the collector, network congestion may occur between the exporter(s) and the collector(s). In consequence, the performance and accuracy of the evaluation processes might be significantly reduced.
- When using either TCP or SCTP, another issue represents their congestion avoidance mechanism. When the export rates are limited by congestion avoidance, the flow records can overwhelm the exporter. The expectable overflow of the flow cache (buffer) can subsequently result in

  1. the overutilization of the exporter's resources,
  2. the subsequent (radical) degradation of the flow records' accuracy and
  3. even in the exporter's crash.

  In network monitoring, the last issue represents a worst case scenario and its prevention is highly desirable in all circumstances.
- Even if there was a pair of exporter and collector capable of serving the problem-free export of flow records, the immense volume of data still represents an issue in the analyzer. With respect to the results of the experiments stated in [7], the demands related to the storage, processing, analysis, evaluation and visualization of the flow records proportionally grow with their number.

Flow records are obviously not only an issue of two parties (i.e. exporter and collector). Their analysis, evaluation and interpretation in the analyzer represent also a demanding task. We can conclude that the core of the problems is the huge number of flow records, which the exporter has to process and how it provides the data for the upper layers. Since it is unable to measure the traffic on a per-flow basis, in order to optimize the monitoring systems, the measurement data amount has to be reduced.

1.4 Related Work

Initial efforts [17, 18] to monitor the network on a per-flow basis appeared to be unscalable. The cost and speed of the memory required for a simple measurement

of a large number of flows and subsequent generation of flow records proved to be prohibitive. In response, packet and flow sampling techniques started to be used to reduce the complexity of the metering and monitoring devices [19, 20, 21]. Since these methods use sampling, they are exposed to a trade-off between monitoring accuracy and limited resources (e.g. memory size, CPU speed).

The paper by Dressler and Münz [22] introduces an aggregation mechanism for efficient and flexible use in flow accounting scenarios. Their approach is controlled by aggregation rules that allow adapting to several application requirements. However, their proposed approach counts only with the requirements of the monitoring purpose. They do not adapt the aggregation to the traffic behavior.

Hu et al. [23] proposed an entropy-based adaptive flow aggregation algorithm. They claim that their mechanism provides a relief from DoS attacks and other security breaches. Unlike the mechanism designed by Dressler and Münz, this algorithm does not take into account that the arbitrarily defined flow aggregates usually do not meet the requirements of the monitoring purpose.

The most accurate version of the document [24] extending the IPFIX standard provides a common implementation-independent basis for an intermediate component between the exporter and collector to handle flow aggregation. However, since the export of flow records between the exporter and the mediator is performed over the same protocols as between the exporter and the collector (i.e. TCP, UDP, SCTP), this standard still does not address the issues described in Section 1.3. As we already stated, the issue of immense volume of flow records has to be solved at the lowest layer of the IPFIX-based measurement platform. Passing "the core of the problem" to the higher layers is not a good strategy to solve the issue.

A promising way how to deal with large data sets is to deploy various data reduction methods in the storage system of the monitoring tool. Although this solution – as described in one of our previous works [7] – can bring positive results, in a long term – as described in our further work [3] – they still do not represent an appropriate workaround. Mainly due to the fact that the network has a lot more devices than the monitoring system. This results in an incomparable difference between their computation resources, i.e. the resources for traffic generation and traffic measurement. Therefore, the issue of immense volume of flow records have to be solved at the lowest layer of the IPFIX-based measurement platform, i.e. in the exporter. Passing "the core of the problem" to the higher layers is not a good strategy.

We can conclude that all the aforementioned approaches provide a solution to the immense volume of measurement data at an acceptable level. A common denominator of these approaches is that even if they adapt their operation to several conditions, they do so only to one at a time. Intuitively, we can expect that increasing the number of conditions taken into consideration during the adaptation process should bring further improvements.

2 ADAPTIVE AGGREGATION OF FLOW RECORDS

Monitoring mechanisms, whether it comes to the utilization of resources or to the "volume of the measurement data/required granularity of information" ratio, often perform inefficiently. It is mainly due to the absence of capabilities by which they could adapt their "behavior" to the actual state of the network. With no doubt, given the ever changing character of present network traffic, this adaptation is difficult to achieve. Even if adaptability is achieved, like in case of the approaches from Section 1.4, it is usually performed according to only a single criterion/condition. However, if there was a way to adapt the operation of the monitoring tool not only to the traffic character but also to the requirements of the monitoring purpose, the immense volume should not be longer an issue at higher layers (i.e. collector, analyzer). This is the basic idea that evolved to our proposed method – the adaptive aggregation of flow records.

2.1 Aggregation of Flow Records

The general architecture of the exporter (see Figure 1) provides several options where the number of flow records can be reduced, specifically:

- two in the metering process: sampling and filtering (denoted in Figure 1 as Packet Selection);
- and two in the exporting process: sampling and filtering (denoted in Figure 1 as Flow Selection).

As we can see, sampling and filtering techniques can be applied at two different layers on data of two different character. While sampling and filtering represent in the metering process a packet selection task, in the exporting process they are a task of flow selection. In addition, there are several other methods for data reduction whose implementation in the exporter would not violate the IPFIX specification (their most appropriate location is between the metering and exporting processes). However, methods such as K-means or dimensionality reduction (e.g. Cluster Analysis, Principal Component Analysis, etc.) are – due to their complexity and computation demands – not suitable for implementation in the exporter [3]. This leaves us, as suggested in the IPFIX specification [2], with three further data reduction techniques: sampling, filtering and aggregation. Although each of them have their advantages and disadvantages, aggregation provides the most comprehensive solution for data reduction. It combines the advantages of both, sampling and filtering. Indeed, while it can efficiently reduce the number of flow records (like sampling); using subnet masks, it also provides a way to focus only on a specific measurement target (like filtering). Its main advantage is that unlike sampling, which discards an uncertain number of packets, the aggregated flow record reflects all the flow properties. However, we can often observe that even with aggregation, the number of measured flow records within a relatively small time interval is still

too high [20, 23, 25]. Therefore, in order to achieve the reduction of flow records, we had to make a shift away from the classical way of their aggregation.

2.2 Aggregation of Non-Heavy-Hitter Flows

Examination of various phenomena in flows that can be statistically described by either self-similarity, first-order similarity (long-range dependence) or heavy-tailed distribution have been an objective of several research activities [26, 27]. A common observation which can be deduced from these research activities is that a very small percentage of flows carry the main part of the traffic (in bytes). We generally refer to these flows as *heavy-hitter flows*.

Heavy-hitters and non-heavy-hitters enable us to examine the dynamics and semantics of network traffic from a new perspective. Their main advantage is that the differentiation of flows into two main classes (i.e. heavy-hitters and non-heavy-hitters) can radically contribute to the reduction of the volume of flow records. In addition, several research activities [20, 28, 21, 27] report that for many monitoring purposes the accurate knowledge of heavy-hitters is still sufficient enough. These monitoring purposes include anomaly and attack detection, scalable differentiated services, usage-based pricing and accounting, making decisions about network upgrades and peering. Therefore, instead of aggregating flow records in a classical manner, we rather performed this critical tasks of the exporter with respect to the heavy-tailed nature of network traffic.

From the well-know and commonly used flow types [27] we can deduce that the elephant flow[2] is the one for which holds that the smallest percentage of its flows accounts for the largest percentage of the transmitted data. It logically follows that in case of mouse flows, the largest percentage of the flows accounts for the smallest percentage of the transmitted data. Therefore, from the view of data reduction, the highest level of compression of flow records can be achieved by the aggregation of mouse flows. Considering the all above, we proposed the aggregation of flow records on the basis of the separation of flows into elephant flows and mouse flows:

We keep the heavy-hitters (elephants) in their original form and rather aggregate the non-heavy-hitters (mice); where

elephant flow is a flow in which the total number of transferred data expressed in bytes $\left(n_{td}^{f}\right)$ is larger than or equal to a predefined *threshold* T, i.e.

$$\text{elephant} \stackrel{def}{=} n_{td}^{f} \geq T; \text{ and} \tag{1}$$

[2] The method of assigning the names to the flow types depends on the characteristics these flows exhibit in network traffic [28].

mouse flow is a flow in which the total number of transferred data expressed in bytes $\left(n_{td}^{f}\right)$ is less than a predefined *threshold* T, i.e.

$$\text{mouse} \stackrel{def}{=} n_{td}^{f} < T. \tag{2}$$

It is obvious that the evaluation of Equations (1) and (2) requires an accurate knowledge of *the total number of transferred data* $\left(n_{td}^{f}\right)$ of each flow. Fortunately, the *octetTotalCount* information element of the IPFIX information model [5] provides exactly such a measure, according to which:

octetTotalCount is the total number of octets (bytes) in incoming packets for a given flow at an observation point.

In conclusion, flows meeting the following condition will be aggregated:

$$\textit{aggregate the flow record} \stackrel{def}{=} octetTotalCount < T. \tag{3}$$

From the perspective of resource utilization, the accurate estimation of *threshold* T is a critical task. Due to the dynamic nature of the traffic its value cannot be estimated by performing a one-time experiment, because – although it would be suitable for the identification of mice at one point in time – it will be unsuitable at another point in time. This led us to its adaptation to the traffic load.

2.3 Adapting the Aggregation to the Traffic Character

The proposed aggregation adaptability is based on the resource utilization of the exporter. For this purpose we track two parameters, the exporter's CPU utilization and memory utilization. The number of actually processed packets are directly proportional to these two parameters. It means that if the traffic increases, the number of captured packets grows as well. As a result, the exporter experiences an increase in the CPU and memory load. On the contrary, if the traffic decreases, since the number of captured packets in the exporter shrinks, the CPU and memory load decreases as well. We can therefore conclude that:

$$\textit{the number of processed packets} \propto CPU\ load\ (\%) \propto Memory\ load\ (\%). \tag{4}$$

As a result, we can define four different loads (characters) of the network traffic:

- *Weak traffic* – traffic utilizing 1/3 of the exporter's CPU and memory resources.
- *Moderate traffic* – traffic utilizing 2/3 of the exporter's CPU and memory resources.
- *Strong traffic* – traffic utilizing $3/3 - k$ part of the exporter's CPU and memory resources, where the subtraction of k determines the maximum resource utilization while the exporter still reliably processes all the captured packets.

- *Critical traffic* – traffic utilizing 3/3 of the exporter's CPU and memory resources. In other words, this traffic is utilizing the exporter to the maximum of its resources, resulting in packet loss.

These individual network traffic loads are depicted in Figure 3.

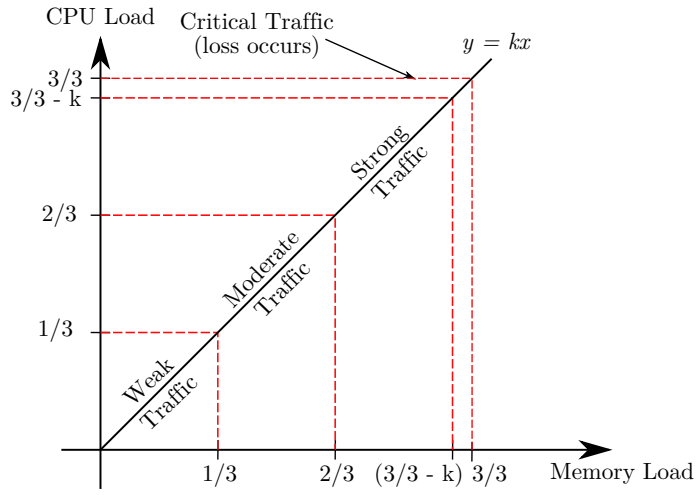


Figure 3. Traffic loads defined by the resource utilization of the exporter

Using these traffic types, instead of estimating the real character of the traffic we rather estimate how it is perceived by the exporter. Since the direct proportionality (Equation (4)) works in both directions, we can observe the network traffic character by estimating the resource utilization of the exporter. If we consider the resource utilization increases (decreases) to be a continuous process, we can define several condition (state) changes by its discretization into time slices using regularly spaced intervals. This allows us to adapt the threshold – for the separation of the flows into heavy-hitters and non-heavy-hitters – to the actual state of traffic. Considering all the above, the process of adapting the threshold to the individual states of the traffic consists of the following steps:

1. Observe the CPU and memory[3] utilization of the exporter in discrete time intervals. For each time slice compute the value by averaging the CPU and memory load. As a result, we get a *set of n observations* $\mathcal{O}$ of the resource utilization of the exporter at each *time slice* t; i.e.

$$\mathcal{O} = \{o_t, o_{t+1}, \ldots, o_{t+n}\} \qquad t = 0, 1, \ldots, k \tag{5}$$

[3] Note that in case of the exporter, by over memory utilization we mean the load of the cache in which it holds the captured packets, i.e. packet cache.

where each *observation* o is computed by averaging the CPU and memory load at time t, i.e.

$$o_t = (cpu_t + memory_t)/2. \tag{6}$$

2. At the initialization of the aggregation process, compute the *average resource utilization* (ARU) from all the observed values since the last process of aggregation; i.e.

$$ARU = \frac{\sum_{i=0}^{k} o_{t+i}}{k}. \tag{7}$$

3. Given the value of ARU, determine the pertaining *traffic character* (TCh), i.e.

$$TCh = \begin{cases} \text{Weak} & \text{if } ARU \in \{1, 2, \ldots, 32\}, \\ \text{Moderate} & \text{if } ARU \in \{33, 34, \ldots, 65\}, \\ \text{Strong} & \text{if } ARU \in \{65, 68, \ldots, 98\}, \\ \text{Critical} & \text{if } ARU \in \{99, 100\}. \end{cases} \tag{8}$$

In words, determine the pertaining actual *traffic character* (TCh) depending on the actual *average resource utilization* of the exporter (ARU).

4. Adjust the value of the *threshold* (T) for the separation of elephant and mouse flows to the actual *traffic character* (TCh) according to the following criteria:

if TCh_t or TCh_{t+1} == Critical then

$$T = \begin{cases} 2T_d & \text{if } TCh_t < TCh_{t+1}, \\ \frac{T_d}{2} & \text{if } TCh_t > TCh_{t+1} \end{cases} \tag{9}$$

else

$$T = \begin{cases} T + (|ARU_t - ARU_{t+1}|)\,\%\text{ of } T_d & \text{if } TCh_t < TCh_{t+1}, \\ T - (|ARU_t - ARU_{t+1}|)\,\%\text{ of } T_d & \text{if } TCh_t > TCh_{t+1} \end{cases}$$

where T_d is the *default threshold* determined by pilot measurements. In words, if there was a transition where either the *traffic character at the previous process of aggregation* (Tch_t) or the *current traffic character* (TCh_{t+1}) was *Critical*, set the *threshold* T to the double or halve of the *default threshold* (T_d) depending on the direction of this transition. Otherwise, i.e. if Tch_t or Tch_{t+1} was *Weak*, *Moderate* or *Strong*, if the *traffic character at the previous process of aggregation* (Tch_t) was smaller than the *current traffic character* (TCh_{t+1}) (i.e., the number of packets increased), T will be increased by as many percent of the *default threshold* (T_d), by as many the *average resource utilizations* (ARU) in time t and $t+1$ differ. However, if TCh_t was larger than TCh_{t+1} (i.e. the number of packets decreased), T will be decreased by as many percent of T_d, by as many the ARUs in time t and $t+1$ differ.

5. Store the current values of the actual *traffic character* (TCh) and the actual *average resource utilization* (ARU) for the next iteration of this process.

All we need to do is to set an initial *threshold* and an estimate of the resource utilization of the exporter at the beginning of the measurement. The aforementioned procedure will automatically adjust the *threshold* to the traffic character in which the individual packets of the flows are captured. In addition, since the individual traffic characters are computed via averaging, our method can also deal with various peaks in the traffic (i.e. they will be averaged out).

2.4 Adapting the Aggregation to the Purpose of the Monitoring

Aggregation obviously causes some information losses. As a result, the information provided by aggregated flow records has a coarse granularity. On the other hand, our aim is to retain as many details of the network traffic as possible. This makes a trade-off between the granularity of information and the volume of flow records. Although our approach neglects the informational value of mouse flows, during the aggregation we still want to preserve as many details about non-heavy-hitters as possible. With respect to the nature of aggregation, this can be achieved only if we take the purpose of monitoring into account. If we know which information elements provide the most valuable information for the monitoring, all we need to do is to take them into account during the process of aggregation. If we order the information elements serving as flow keys [2] from the lowest to the highest according to a ranking indicating the significance of the information element in the context of the network monitoring purpose, we get n different levels of aggregation:

1. The first level of aggregation is performed over the information element with the lowest ranking. As a result, the information carried by the aggregated flow records has the "least coarse" level of granularity[4].
2. The second level of aggregation is performed over the information element with the lowest ranking among the remaining information elements (i.e. the next information element). As a result, the information carried by the aggregated flow records has a coarser level of granularity.
3. The same procedure is repeated iteratively until it gets to the last information element.
4. The last level of aggregation is performed over the n^{th} information element having the highest ranking. As a result, the information carried by the aggregated flow records has the coarsest level of granularity.

This procedure[5] is illustrated in Figure 4.

In consequence, even if we focus only on heavy-hitter flows, this method allows to aggregate efficiently among the mouse flows. As a result, we can further separate mouse flows into different classes that preserve the details of the traffic at various levels of granularity.

[4] The finest level of granularity provides the not aggregated heavy-hitter flow records.

[5] Note that this approach represents a generalized form of the *Gradual Flow Key Reduction* method [29].

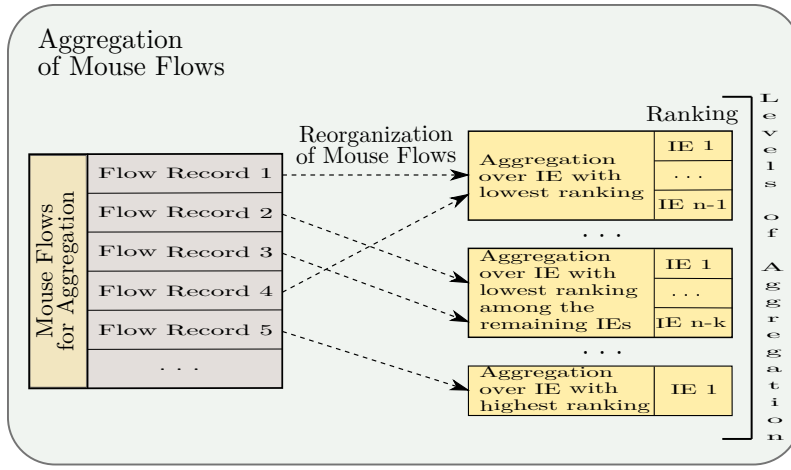


Figure 4. Adapting the aggregation to the purpose of the monitoring based on [29]

2.5 Triggering the Process of Aggregation

The organization of packets into flows is a continuous process. Depending on whether there is a record in the flow cache pertaining to the actually processed packet or not, the existing flow record is updated or a new one is created. Obviously, the value of *octetTotalCount* of those packets that belong to the same flow is conditioned by these creations and updates. However, our approach for the separation of heavy-hitter and non-heavy-hitter flows is based exactly on this value (see Equation (3)). In consequence, the aggregation frequency must not be too low.

This is due to the fact that during a too short period of time the metering process cannot capture enough information about the flows to separate them correctly into elephants and mice. As a result, all the flows will be identified as mouse flows and thus, they will be all aggregated; even those that are in real elephant flows. In addition, a constantly running aggregation process can cause overutilization of the exporter. Therefore, instead of carrying out aggregation at every point in time, we decided for its execution in particular time periods. For this purpose we define an *aggregation frequency* variable.

Aggregation frequency (A_f) is the measure of time period expressed in milliseconds (ms) or the several fractions of it (e.g. microseconds, nanoseconds, etc.) after which the aggregation of non-heavy-hitter flows takes place.

In other words, A_f represents the rate at which the aggregation is performed. The determination of its value, as in case of the *threshold*, highly depends on the character of the traffic. It means that "weak" network traffic requires a different value of A_f than "strong" network traffic. For example, while in the case of weak traffic, aggregation performs properly at higher *aggregation frequency* values (e.g.

$A_f = 500\,\text{ms}$), in the case of strong network traffic it provides accurate results at lower *aggregation frequency* values (e.g. $A_f = 100\,\text{ms}$). However, since the *threshold* is already automatically adapted to the traffic character by the method described in Section 2.3, there is no need to adjust also the aggregation frequency (A_f). In simple words, instead of adapting the value of A_f we rather adapt the *threshold*, by which we should achieve the same result.

3 PRELIMINARY EXPERIMENTS

The module of adaptive aggregation was implemented in the `BEEM` component of the `SLAmeter` IPFIX-based measuring platform [16] as a full-featured functional block between the metering and the exporting processes. The aim of the experiments was to verify the functionality of the SLAmeter with and without the proposed method. For this reason we generated the same artificial traffic by several packet generators.

The accurate operation of adaptive aggregation required some conditions to be satisfied. The individual information elements [5] that were set as flow keys for the generation of flow records were the following: protocolIdentfier (highest rank), sourceTransportPort, destinationTransportPort, destinationIPv4Address, sourceIPv4Address (lowest rank). The pilot measurement showed that the `BEEM` processed the packets in every 35 000 ns. With respect to the predefined size of the allocated memory for the flow records (flow cache = 8 MB), the value of A_f was set to 250 ms in the `BEEM`; where A_f is the time period after which the aggregation of non-heavy-hitter flows takes place. During this time interval, the `BEEM` was able to hold approximately 7 150 records in its flow cache at a time. Measurement results also showed that the total value of transferred octets of almost 92 % of the identified flows was less than 10 000. In other words, during this pilot measurement the `BEEM` did not capture any flow having more than 10 000 B (*bytes*) of transferred data till its passive expiration. Considering all the above, the initial threshold T for the identification of elephant and mouse flows was set to 10 000 octets in the `BEEM`, i.e. $octetTotalCount < 10\,000$.

Assuming these prerequisites were satisfied, the process of adaptive aggregation consisted of the iterative execution of the following two phases:

1. the adaptation phase, in which the threshold T was adjusted according to the method described in Section 2.3; and
2. the aggregation phase, in which the individual flow records were aggregated according to the method described in Section 2.4.

The results of the experiments are summarized in Tables 1 and 2.

Without adaptive aggregation, as shown in Table 1, approximately 120 000 flows (sessions) were generated and as a result created approximately 300 000 flows records in the flow cache. However, since the flow cache after a specific period of time (around time 10:30:00) was utilized to its maximum, an undetermined number of packets was not organized into flow records. Therefore, the values in Table 1 cannot be considered as complete and have a certain bias. Moreover, the average load of

the flow cache was over 90 % during the whole measurement. The flow-rate plot pertaining to this measurement is shown in Figure 5 (denoted with red). From the plot we can see that the average number of flows per a second was between 30 and 50.

Measurement Characteristics	Results
Number of generated packets	1 950 448
Number of flows	121 641
Number of transferred data (in MB)	1 455.1
Total number of flow records	281 537
Average load of Flow Cache	90 %

Table 1. Results without adaptive aggregation

With adaptive aggregation the threshold T was automatically adjusted according to the method described in Section 2.3. The summary of the measurement is shown in Table 2. As we expected, the average load of the flow cache was lower, (i.e. around 20 %) during the whole measurement. The flow-rate plot pertaining to this measurement is shown in Figure 5 (denoted with blue color). From the plot we can see that the average number of flows per a second was between 7 and 15. The plot perfectly emphasizes the coarse granularity of the observed information resulting from the aggregation. In addition, adaptive aggregation also handled the burst in the flows that caused the erroneous operation of `BEEM` during the measurement without adaptive aggregation.

Measurement Characteristics	Results
Number of generated packets	1 950 448
Number of flows	3 460
Number of transferred data (in MB)	1 455.1
Total number of flow records	6 670
Average load of Flow Cache	20 %

Table 2. Results with adaptive aggregation

Discussion

Adaptive aggregation radically reduced the number of flows. However, when we compared the error rate between the measurements, we found that there was a trade-off between the accurate identification of elephant/mouse flows and the load of flow cache. In numbers, although the load of the flow cache was around 20 % during the measurement with adaptive aggregation, the error rate of the identification of mouse flows was around 12–14 %. The determination of the error rate was achieved by the reconstruction of the flows exported and stored in the database from the first

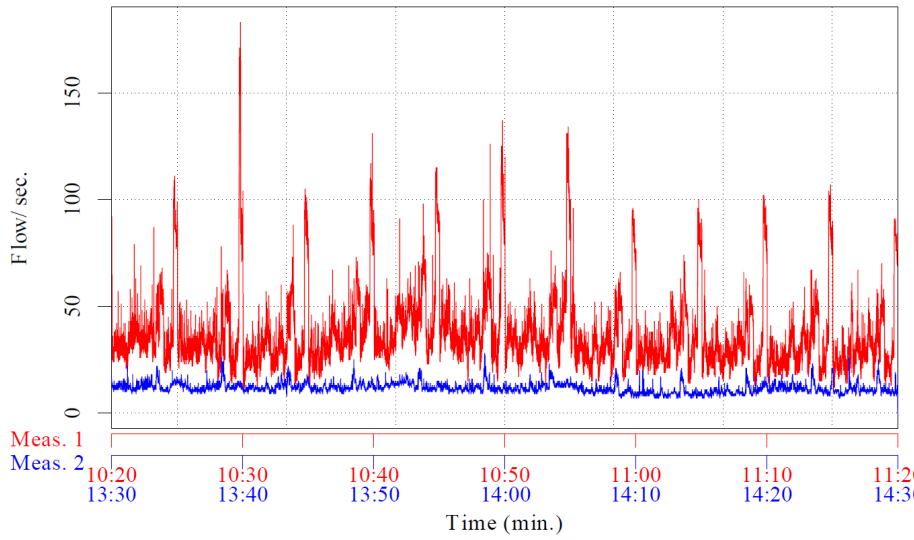


Figure 5. Combined flow-rate plot resulted from the measurements. Red colored flow-rate plot pertains to the measurement without adaptive aggregation (Meas. 1). Blue colored flow-rate plot pertains to the measurement with adaptive aggregation (Meas. 2).

measurement and compared with the results from the second measurement (time intervals with packet loss were discarded).

We can therefore conclude that aggregation involves a certain error rate into the measurements. However, when comparing this bias while all data are preserved and a certain data loss because of memory issues, this error rate can be considered acceptable. In summary, our approach has a significant impact on the monitoring systems that implement IPFIX for collecting information about IP flows. Since in case of the adaptation to the traffic character we did not consider the utilization of the link bandwidth, the accuracy of the measurements still highly depends on the initial value of the estimated threshold.

4 CONCLUSION AND FUTURE WORK

In this paper we introduced yet another approach whose aim is to address the issues related to the volume of measurement data produced during network monitoring. Its main contribution is the method of adaptive aggregation that adjusts its operation to the network character and to the purpose of monitoring. To the best of our knowledge, only a few methods adapts their behavior to more than one factor. The results presented in Section 3 proved that by adaptive aggregation we can efficiently reduce the number of measurement data and it has a positive impact on the measurement platform. We can therefore conclude that our proposed approach

considerably contributes to the research area of the measurement platforms based on the IPFIX protocol.

Since adaptive aggregation, due to the error rate, brings a certain bias into the measurement data, in the future, the method of adjusting the threshold for the identification of (non) heavy-hitters will be extended with the utilization of the link bandwidth as well. The proposed approach will be also examined in network monitoring scenarios with long duration. We will also review the possibility to implement dynamic Bayesian networks, by which the traffic could be classified according to the assumption that the probability distribution of different traffic types are far from each other in the same network.

Acknowledgments

This publication is the result of the Project implementation: University Science Park TECHNICOM for Innovation Applications Supported by Knowledge Technology, ITMS: 26220220182, supported by the Research & Development Operational Programme funded by the ERDF. We support research activities in Slovakia/This project is co-financed by the European Union.

REFERENCES

[1] Claise, B.: Cisco Systems NetFlow Services Export Version 9. RFC3954, 2004, doi: 10.17487/rfc3954.

[2] Claise, B.—Trammell, B.—Aitken, P.: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. RFC7011, 2013, doi: 10.17487/rfc7011.

[3] Pekár, A.—Chovancová, E.—Fanfara, P.—Trelová, J.: Issues in the Passive Approach of Network Traffic Monitoring. Proceedings of the 17th IEEE International Conference on Intelligent Engineering Systems (INES), 2013, pp. 327–332, doi: 10.1109/INES.2013.6632836.

[4] Case, J.—Fedor, M.—Schoffstall, M.—Davin, J.: Simple Network Management Protocol (SNMP). RFC1157, 1990, doi: 10.17487/rfc1157.

[5] Claise, B.—Trammell, B.: Information Model for IP Flow Information Export (IPFIX). RFC7012, 2013, doi: 10.17487/rfc7012.

[6] Phaal, P.—Lavine, M.: sFlow Version 5. Specification, sFlow.org, 2004.

[7] Vokorokos, L.—Pekár, A.—Ádám, N.: Data Preprocessing for Efficient Evaluation of Network Traffic Parameters. Proceedings of the 16th IEEE International Conference on Intelligent Engineering Systems (INES), 2012, pp. 363–367, doi: 10.1109/INES.2012.6249860.

[8] Nunes, B. A. A.—Mendonca, M.—Nguyen, X.-N.—Obraczka, K.—Turletti, T.: A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. IEEE Communications Surveys and Tutorials, Vol. 16, 2014, No. 3, pp. 1617–1634, doi: 10.1109/SURV.2014.012214.00180.

[9] Haleplidis, E.—Salim, J. H.—Halpern, J. M.—Hares, S.—Pentikousis, K.—Ogawa, K.—Wang, W.—Denazis, S.—Koufopavlou, O.: Network Programmability with ForCES. IEEE Communications Surveys and Tutorials, Vol. 17, 2015, No. 3, pp. 1423–1440, doi: 10.1109/COMST.2015.2439033.

[10] Kang, M.—Kang, E.-Y.—Hwang, D.-Y.—Kim, B.-J.—Nam, K.-H.—Shin, M.-K.—Choi, J.-Y.: Formal Modeling and Verification of SDN-OpenFlow. Proceedings of the 6th IEEE International Conference on Software Testing, Verification and Validation (ICST), 2013, pp. 481–482.

[11] Lara, A.—Kolasani, A.—Ramamurthy, B.: Network Innovation Using OpenFlow: A Survey. IEEE Communications Surveys and Tutorials, Vol. 16, 2014, No. 1, pp. 493–512.

[12] Hofstede, R.—Čeleda, P.—Trammell, B.—Drago, I.—Sadre, R.—Sperotto, A.—Pras, A.: Flow Monitoring Explained: From Packet Capture to Data Analysis with NetFlow and IPFIX. IEEE Communications Surveys and Tutorials, Vol. 16, 2014, No. 4, pp. 2037–2064.

[13] Yu, C.—Lumezanu, C.—Zhang, Y.—Singh, V.—Jiang, G.—Madhyastha, H. V.: FlowSense: Monitoring Network Utilization with Zero Measurement Cost. Passive and Active Measurement (PAM 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7799, 2013, pp. 31–41, doi: 10.1007/978-3-642-36516-4_4.

[14] Pekár, A.—Révés, M.—Giertl, J.—Feciľak, P.: Overview and Insight into the MONICA Research Group. Central European Journal of Computer Science, Vol. 2, 2012, No. 3, pp. 331–343.

[15] Jakab, F.—Košč̆o, Ľ.—Potocký, M.—Giertl, J.: Contribution to QoS Parameters Measurement: The BasicMeter Project. Proceedings of the International Conference on Emerging eLearning Technologies and Applications, 2005, pp. 371–377.

[16] Pekár, A.—Feciľak, P.—Michalko, M.—Giertl, J.— Révés, M.: SLAmeter – The Evaluator of Network Traffic Parameters, Proceedings of the 10th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA), 2012, pp. 291–295, doi: 10.1109/ICETA.2012.6418318.

[17] Ramabhadran, S.—Varghese, G.: Efficient Implementation of a Statistics Counter Architecture. ACM SIGMETRICS Performance Evaluation Review (PER), Vol. 31, 2003, No. 1, pp. 261–271, doi: 10.1145/781027.781060.

[18] Shah, D.—Iyer, S.—Prabhakar, B.—McKeown, N.: Analysis of a Statistics Counter Architecture. Hot Interconnects 9, 2001, pp. 107–111, doi: 10.1109/HIS.2001.946701.

[19] Estan, C.—Keys, K.—Moore, D.—Varghese, G.: Building a Better NetFlow. ACM SIGCOMM Computer Communication Review (CCR), Vol. 34, 2004, No. 4, pp. 245–256.

[20] Estan, C.—Varghese, G.: New Directions in Traffic Measurement and Accounting. Proceedings of the 2002 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM), 2002, pp. 323–336, doi: 10.1145/633025.633056.

[21] Lu, Y.—Wang, M.—Prabhakar, B.— Bonomi, F.: ElephantTrap: A Low Cost Device for Identifying Large Flows. Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects, 2007, pp. 99–108, doi: 10.1109/HOTI.2007.13.

[22] Dressler, F.—Münz, G.: Flexible Flow Aggregation for Adaptive Network Monitoring. Proceedings of the 31st IEEE Conference on Local Computer Networks, 2006, pp. 702–709, doi: 10.1109/LCN.2006.322180.

[23] Hu, Y.—Chiu, D.-M.—Lui, J. C. S.: Entropy Based Adaptive Flow Aggregation. IEEE/ACM Transactions on Networking, Vol. 17, 2009, No. 3, pp. 698–711.

[24] Trammell, B.—Wagner, A.—Claise, B.: Flow Aggregation for the IP Flow Information Export (IPFIX) Protocol. RFC7015, 2013, doi: 10.17487/rfc7015.

[25] Cheng, G.—Gong, J.: Adaptive Aggregation Flow Measurement on High Speed Links. Proceedings of the 11th IEEE Singapore International Conference on Communication Systems (ICCS 2008), 2008, pp. 559–563, doi: 10.1109/ICCS.2008.4737246.

[26] Papagiannaki, K.—Taft, N.—Bhattacharyya, S.—Thiran, P.—Salamatian, K.—Diot, C.: A Pragmatic Definition of Elephants in Internet Backbone Traffic. Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment (IMW '02), 2002, pp. 175–176, doi: 10.1145/637201.637227.

[27] Smith, R. D.: The Dynamics of Internet Traffic: Self-Similarity, Self-Organization, and Complex Phenomena. Advances in Complex Systems, Vol. 14, 2011, No. 6, pp. 905–949.

[28] Lan, K.-C.—Heidemann, J.: A Measurement Study of Correlations of Internet Flow Characteristics. Computer Networks, Vol. 50, 2006, No. 1, pp. 46–62.

[29] Irino, H.—Katayama, M.—Chaki, S.: Study of Adaptive Aggregation on IPFIX. Proceedings of the 7th Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT), 2008, pp. 86–91.

Adrián Pekár graduated at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice, Slovakia in 2011. Since then, his scientific research has been focused on the optimization of measurement platforms based on the IPFIX protocol. He defended his Ph.D. thesis in the field of network traffic characteristics measurements and monitoring in 2014. Recently, his scientific research was extended to investigate also the issues related to monitoring and virtualization of cloud networks. His area of interest includes QoS, IPFIX, network traffic management and engineering, cloud computing and virtualization.

Martin Chovanec received his Engineering degree in Informatics in 2005 from the Faculty of Electrical Engineering and Informatics, Technical University of Košice. In 2008 he received his Ph.D. degree at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice and his scientific research was focused on network security and encryption algorithms. Currently, he is Director of the Institute of Computer Technology of the Technical University of Košice.

Liberios Vokorokos graduated (M.Sc.) with honors at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice in 1991. He defended his Ph.D. thesis in the field of programming devices and systems in 2000. In 2005 he became Professor of Computer Science and Informatics. Since 1995 he has worked as an educationist at the Department of Computers and Informatics. His scientific research focuses on parallel computers of the Data Flow type. He also investigates the issues related to the complex systems diagnostics. He is the Dean of the Faculty of Electrical Engineering and Informatics of the Technical University of Košice.

Eva Chovancová graduated (M.Sc.) at the Department of Computers and Informatics at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice in 2009. She defended her Ph.D. thesis in the field of computers and computer systems in 2012; her thesis title was "Specialized Processor for Computing Acceleration in the Field of Computer Vision". Since 2012 she has worked as Assistant Professor at the Department of Computers and Informatics. Her scientific research is focused on the multicore computer architectures.

Peter Feciľak graduated (M.Sc.) at Department of Computers and Informatics at Faculty of Electrical Engineering and Informatics, Technical University of Košice in 2006. In 2009, he finished his Ph.D. studies at the named department with the focus on optimization of computer networks. Currently, he is working as an employee of DCI, FEI, Technical University of Košice. His current teaching and research interests are computer networks, network monitoring, quality of services and smart energy systems.

Miroslav Michalko received his Ph.D. in informatics from the Technical University of Košice (TUKE), Slovakia. For more than 10 years he is a member of a well recognized research institution – the Computer Networks Laboratory at Department of Computers and Informatics (DCI) of TUKE. Currently he is Assistant Professor at DCI TUKE and gives lectures in the field of computer networks. His research includes multimedia content delivery, video streaming services, web and cloud services, innovative teaching and learning techniques and IoE/IoT solutions.

Computing and Informatics, Vol. 37, 2018, 165–185, doi: 10.4149/cai_2018_1_165

EXPLORATION OF COMPILER OPTIMIZATION SEQUENCES USING A HYBRID APPROACH

Tiago Cariolano de Souza Xavier, Anderson Faustino da Silva

Departament of Informatics
State University of Maringá
Maringá, Paraná, Brazil
e-mail: tiago.cariolano@gmail.com, anderson@din.uem.br

Abstract. Finding a program-specific compiler optimization sequence is a challenge, due to the large number of optimizations provided by optimizing compilers. As a result, researchers have proposed design-space exploration schemes. This paper also presents a design-space exploration scheme, which aims to search for a compiler optimization sequence. Our hybrid approach relies on sequences previously generated for a set of training programs, with the purpose of finding optimizations and their order of application. In the first step, a clustering algorithm chooses optimizations, and in the second step, a metaheuristic algorithm discovers the sequence, in which the compiler will apply each optimization. We evaluate our approach using the `LLVM` compiler, and an `I7` processor, respectively. The results show that we can find optimization sequences that result in target codes that, when executed on the `I7` processor, outperform the standard optimization level `O3`, by an average improvement of 8.01 % and 6.07 %, on Polybench and cBench benchmark suites, respectively. In addition, our approach outperforms the method proposed by Purini and Jain, `Best10`, by an average improvement of 24.22 % and 38.81 %, considering the two benchmarks suites.

Keywords: Compilers, optimizations, sequence, performance

Mathematics Subject Classification 2010: 68-N20

1 INTRODUCTION

Optimizing compilers provide a large number of transformations, known as optimizations, which are applied during the compilation process [25]. The aim is to create a target code semantically equal to the source code, but with good performance. Due to the large number of optimizations and the fact that each optimization interacts with each other in complex ways, it is a challenge, even for an expert programmer, to find good optimization sequences. To minimize this challenge, optimizing compilers offer optimization levels (e.g. `O0`, `O1`, `O2` and `O3` in the case of LLVM), which consist of specific sequences.

The choice of optimizations and their order of application has a significant impact on performance [19]. In addition, it is program-specific dependent [4, 6, 10, 11, 21, 22].

Exhaustive design-space exploration, although possible, takes a long time to make it suitable for use in typical iterative compilers. Therefore, researchers engage in proposing design-space exploration schemes to find a program-specific optimization sequence using few evaluations.

In this paper, we propose a hybrid approach to search for good optimization sequences aiming at performance improvements. First of all, a training stage tries to choose good sequences. After the deployment stage, which relies on previously generated sequences, it discovers a program-specific optimization sequence. In our approach, we employ several strategies: random sampling, genetic algorithm, clustering, metaheuristic, and a reduction scheme.

On one hand, the choice of optimizations is based on the premise that similar programs react approximately the same way, when they are compiled using the same sequence. In such manner, a new program can be improved by the optimizations used on a similar program. However, discovering the order of application is based on the premise that this problem is similar to the Traveling Salesman Problem. Therefore, it is possible to develop a reduction algorithm to transform a problem into another, in order to solve it using an existing solution, and then utilizing such result as a solution for the previous conflict.

The experimental results show that our approach finds an optimization sequence that outperforms the standard optimization level `O3`, besides the approach proposed by Purini and Jain [24], `Best10`, considering Polybench and cBench as benchmarks suites.

2 OUR APPROACH

During compilation, the compiler applies several optimizations, in order to improve the target code. However, some optimizations can be useful to a specific program, but not to another. Thus, the most appropriate approach is to choose optimizations and their order of application, considering that it is program-specific dependent.

In this paper we present a design-space exploration scheme that chooses and orders optimizations.

2.1 Overview

It is possible to choose and order optimizations easily, based on the assumption that two *similar* programs react in the same manner, when they are compiled using the same optimizations. Thus, we can compile a new program by applying the optimizations utilized on a *similar* and previously-compiled program. In addition, based on the assumption that a problem can be transformed into another, we can convert the problem of discovering the order of application into the well-known Traveling Salesman Problem (`TSP`) [2], and use a well-known solution to solve the `TSP`.

Our approach can be outlined as:

1. Training stage
   (a) Generate the training data
      i Extract the feature vector f, using compiler level `O0`
      ii Record the feature vector f
      iii Record the benchmark running time, using compiler level `O3`
   (b) Generate the training data for `O3`
      i Instrument each training program
      ii For each training program
         A Select a set of optimizations and apply them
         B Record the running time
2. Deployment stage
   (a) Choose the optimizations
      i Collect feature vector $f1$ from each training program, using compiler level `O0`
      ii Extract feature vector $f2$ from the test program, using compiler level `O0`
      iii Reduce $f1$ and $f2$ to the most significant components using the Principal Component Analysis `PCA`
      iv Cluster the new feature vectors $f1'$ and $f2'$ into N clusters
      v Extract the best set of optimizations from each training program, which belongs to the same cluster of the test program
   (b) Discover the order of application
      i For each set of optimizations selected
         A Reduce the problem of discovering the order of application into the `TSP`
         B Solve the `TSP`
         C Transform the solution into an optimization sequence
   (c) Return the optimizations and their order of application

2.2 Training Stage

The training stage aims to collect pieces of information about several training programs. As a result, this stage provides a small knowledge base (`KB`).

The `KB` can be viewed as a table composed by several entries, where each entry consists in four fields, namely:

1. Program name;
2. Runtime for the program, when it is compiled using optimization level `O3`;
3. Feature vector, when the program is compiled using optimization level `O0`; and
4. Compiler optimization sequences and their runtime.

2.2.1 Generating the Feature Vector

The feature vector is composed of dynamic information, which is collected during program execution. This means that such vector characterizes the dynamic behavior of the program. We use performance counters as feature vectors.

Performance counters are dynamic information that consists of performance data such as the number of issued instructions, completed instructions, cache accesses, cache hits, cache misses, mispredicted branches, and others. They are traditionally used for hardware performance analysis [3, 8, 14, 16].

The work of Cavazos et al. [4] was the first to propose the use of performance counters to characterize programs and measure their similarities. A recent work [7] also demonstrated that performance counters is a good strategy to measure the similarity between two programs. In this paper, we characterize programs in the same manner.

The use of performance counters is attractive, because they do not limit the program class, which the system is able to handle. As a result, our system (strategy) can find a good compiler optimization sequence for any program.

Table 1 presents the features used in our approach.

Type	Features						
Cache	L1_ICM	L1_DCM	L1_STM	L1_TCM	L1_LDM	L2_DCR	L2_TCA
	L2_DCW	L2_STM	L2_TCM	L2_TCR	L2_DCA	L2_TCW	L2_ICR
	L2_DCH	L2_DCM	L2_ICA	L2_ICM	L2_ICH	L3_DCR	L3_TCA
	L3_DCW	L3_TCM	L3_TCR	L3_DCA	L3_TCW	L3_ICR	L3_ICA
Branch	BR_PRC	BR_UCN	BR_NTK	BR_INS	BR_MSP	BR_TKN	BR_CN
SIMD	VEC_SP	VEC_DP					
Floating Point	FDV_INS	FP_INS	DP_OPS	FP_OPS	SP_OPS		
TLB	TLB_DM	TLB_IM					
Cycles	REF_CYC	TOT_CYC	STL_ICY	STL_ICY			
Insts	TOT_INS						

Table 1. Features

In order to standardize the feature vectors of training and test programs, we normalize each feature by `TOT_INS`. To collect these features, we use the tools `PAPI` [18] and `PerfSuite` [12].

2.2.2 Generating Training Data for Optimization Level `O3`

The optimizations used to generate training data belong to the optimization level `O3`. They are presented in Table 2.

inline	prune-eh	scalar-evolution
argpromotion	inline-cost	indvars
gvn	functionattrs	loop-idiom
slp-vectorizer	sroa	loop-deletion
globaldce	domtree	loop-unroll
constmerge	early-cse	memdep
targetlibinfo	lazy-value-info	memcpyopt
no-aa	jump-threading	sccp
tbaa	loop-unswitch	dse
tailcallelim	adce	notti
ipsccp	loop-simplify	block-freq
instcombine	loop-rotate	loop-vectorize
verify	licm	simplifycfg
globalopt	loops	branch-prob
deadargelim	lcssa	basicaa
reassociate	barrier	basiccg
correlated-propagation	strip-dead-prototypes	

Table 2. Optimizations

The process of creating compiler optimization sequences is guided by the following criteria:

- Every optimization appears only once;
- Every optimization can appear in any position;
- Every optimization has to address the compilation infrastructure rules; and
- All sequences, in `KB`, have 40 optimizations.

The first criterion indicates that we do not explore the use of an optimization several times, even though this occurs in all optimization levels, in the case of `LLVM`. The second one indicates that there are no restrictions when a specific optimization should be applied. The third one indicates that a new sequence cannot violate the safety of the `LLVM`. In the fourth criterion, the creation process tries to give the same characteristic to every sequence.

Several strategies can be used to build a base of sequences. We use the strategy proposed by Purini and Jain [24], which consists in using random and genetic algorithms to create effective sets of optimizations. The random algorithm generates

sets utilizing a uniform and random sampling of the search space. While, the genetic algorithms use a sophisticated way to build sets, and explore the search space. The algorithms are described as follows.

Random Algorithm. This iterative algorithm randomly generates 500 sequences.

Genetic Algorithm with Rank Selector. This algorithm generates sets using a genetic process, such as crossover and mutation. A simple genetic algorithm consists in randomly generating an initial population, which will result in an iterative evolution process. Such procedure of evolving a population (or a generation) involves choosing the parents; applying genetic operators; evaluating new individuals; and finally a reinsertion operation deciding which individuals will compose the new generation. This iterative process is performed until a stopping criterion is reached. The first generation is composed of individuals that are generated by a uniform sampling of the optimization space. Evolving a population includes the application of two genetic operators: crossover, and mutation. The first operator has a probability of 90 % for creating a new individual. The second operator, mutation, has a probability of 2 % for transforming an individual. Two types of mutation procedures were proposed:

1. to exchange two optimizations from random points; and
2. to change one optimization in a random point.

Both operators have the same probability of occurrence, though only one mutation is applied over the individual selected to be transformed. This iterative process uses elitism, which maintains the best individual in the next generation. Furthermore, it runs over 100 generations and 60 individuals, and finishes whether the standard deviation of the current fitness score is less than 0.01, or the best fitness score does not change in three consecutive generations.

Genetic Algorithm with Tournament Selector. It is similar to the previous strategy, but instead of using a rank selector it uses a tournament selector ($Tour = 5$).

Each strategy creates two sequences in each round. The first sequence is created utilizing the specific scheme (random or genetic), and the second one is the first sequence modified by human knowledge.

The `LLVM`'s manual suggests that some optimizations should precede and/or succeed a specific optimization for its effectiveness, so that the first sequence is updated to reflect this knowledge. This update follows the criteria:

- *loops* should appear before the first loop optimization;
- *inline-cost* should appear before *inline* and *always-inline*; and
- *verify* should be the last optimization.

After generating several sequences, we select the two best sequences for each training program; one is the best sequence generated by each algorithm, and the other is the best updated sequence.

Each sequence can contain optimizations that do not contribute to the program speedup or have a negative impact on the program. Therefore, the next step is to eliminate these unnecessary optimizations using the `Sequence Reduction Algorithm`, also proposed by Purini and Jain [24].

As the training stage uses 61 training programs, our `KB` has 366 sequences.

2.2.3 Training Benchmarks

The training programs are composed of microkernels, which were taken from `LLVM`'s test-suite. These are programs composed of a single source code, and have short running times. Table 3 shows the training programs.

ackermann	flops-6	matrix
ary3	flops-7	methcall
bubblesort	flops-8	misr
chomp	flops	n-body
dry	fp-convert	nestedloop
dt	hash	nsieve-bits
fannkuch	heapsort	objinst
fbench	himenobmtxpa	oourafft
ffbench	huffbench	oscar
fib2	intmm	partialsums
fldry	lists	perlin
flops-1	lowercase	perm
flops-2	lpbench	pi
flops-3	mandel-2	puzzle
flops-4	mandel	puzzle-stanford
flops-5	queens	queens-mcgill
quicksort	random	realmm
recursive	reedsolomon	richards_bench
salsa20	sieve	spectral-norm
strcat	towers	treesort
whetstone		

Table 3. Microkernels

2.3 Deployment

The deployment stage performs seven steps in order to choose optimizations and their order of application, namely:

1. Extract the feature vector f from the test program, using compiler level `O0`;
2. Cluster the training and test programs, based on their feature vectors;
3. Extract from each training program, which belongs to the same cluster of the test program, their sequences;

4. Reduce the problem of choosing the order of applying compiler optimizations into the TSP;
5. Solve the TSP;
6. Transform the TSP's result into a solution to choose and order optimizations; and
7. Return the best target code.

2.3.1 Choosing Optimizations

The choice of optimizations is based on the premise that we can find similar patterns among programs, which give important insights for determining potential optimizations.

Based on the premise that similar programs react approximately the same way, when they are compiled using the same optimizations, we choose the optimizations that will be enabled during the compilation of the test program from a similar training program. In such manner, each program is represented by a feature vector forming points in a multidimensional space, and a clustering algorithm that operates in this space trying to group points that are proximate.

The task of the clustering algorithm is to group a set of programs, in such a way that programs in the same group (cluster) are more similar to each other than to those that belong to other clusters [28].

Finding similar programs is a task performed in two steps. First, we extract the feature vectors from the training and test programs. Second, the clustering algorithm reduces the feature vectors to the most significant components using PCA [28], and clusters the programs. In this moment, we know which programs are similar.

After clustering the programs, we extract from each training program, which belongs to the same cluster (C) of the test program, their sequences.

Even though the word *sequence* indicates order, in this point the extracted sequences only indicate the optimizations that will be enabled during the compilation of the test program. As ordering optimization is also a program-specific problem, we need to analyze the test program. In our strategy, ordering optimization is based on the insights given by the sequences, which achieve performance on training programs.

In a nutshell, each specific optimization that appears in the extracted sequences forms the set of optimizations that will be enabled by the compiler. In addition, these sequences give insights on when the compiler should apply each optimization.

2.3.2 Discovering the Order of Application

After choosing the optimizations, the next step is to discover the order of application. Such process is performed by extracting knowledge from KB, which is associated with the training programs (their sequences) that belong to C.

This knowledge is obtained by analyzing pairs of optimizations, in order to find patterns that are meaningful to the test program. In fact, these patterns determine

how we will join the pairs to form a new sequence. Therefore, based on these patterns we modify the order of application.

If we consider that optimizations are vertices, and that there is a cost of applying o_i before o_j, and vice-versa, the problem of choosing the order of application can be reduced into the well-known Asymmetric Traveling Salesman Problem (`ATSP`) [2].

It is important to note that the asymmetric version is the appropriate algorithm, because the performance of applying the optimization o_i before o_j can be different from applying o_j before o_i.

The performance of a pair of optimizations can be viewed as a cost. If after analyzing the previously-generated sequences, we consider that applying the optimization o_i before o_j will reduce the performance of the test program, therefore, we provide a high cost to the pair (o_i, o_j) in order to reflect this behavior.

The cost of a pair of optimizations is based on the frequency of all pairs. Given a set of optimizations `S`, we analyze the sequences previously generated in order to find how often all possible pairs formed with `S` optimizations occur in the sequences that belong to C. As the sequences previously generated can provide speedup to the training programs, our approach is guided by the assumption that pairs with high frequency are a potential order.

Reducing Our Problem into ATSP. To reduce the problem of discovering the order of application into `ATSP`, we perform three steps:

1. Create a complete digraph;
2. Map optimizations to vertices; and
3. Weigh the edges.

The first two steps are trivial. To perform the third, we need to infer the cost of the pair (o_i, o_j), which is based on the frequency that o_i appears before o_j in the sequences that belong to the cluster C and is defined as:

$$\mathit{Freq\_Prog}(p, o_i, o_j) = |\{s \in \mathit{Dom}(ES(p)) \mid (o_i \wedge o_j \in s) \wedge o_i \prec o_j\}|.$$

As a result of using $Freq\_Prog$, the function $Freq$ that returns how often o_i appears before o_j is defined as:

$$\mathit{Freq}(o_i, o_j) = \sum_{p \in C} \mathit{Freq\_Prog}(p, o_i, o_j). \tag{1}$$

If $\mathit{Freq}(o_i, o_j) < \mathit{Freq}(o_j, o_i)$, our approach considers that the application of (o_j, o_i) is the best choice. This means that the higher $\mathit{Freq}(o_j, o_i)$ is (implying in a low value of $\mathit{Freq}(o_i, o_j)$), the higher the cost of (o_i, o_j) will be. Therefore, the cost of applying the pair (o_i, o_j) is given by the inverse frequency of that pair, $\mathit{Cost}(o_i, o_j) = \mathit{Freq}(o_j, o_i)$.

Not all pairs of optimizations appear in all C sequences, as a result a high variation occurrence between two different pairs is possible, on their frequencies. To solve this problem, the cost is normalized as follows:

$$Cost(o_i, o_j) = \frac{Freq(o_j, o_i)}{Freq(o_i, o_j) + Freq(o_j, o_i)}. \tag{2}$$

With this standardization, $Cost(o_i, o_j)$ will always range from 0 to 1. In addition, if there are only (o_j, o_i) occurrences, thus $Cost(o_i, o_j) = 1$, which is the highest possible cost.

Solving the ATSP. The algorithm that solves the `ATSP` is based on Ant Colony Optimization (`ACO`) [9].

`ACO` is a metaheuristic of combinatorial optimization, which is based on the behavior of real ants. A metaheuristic is "a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems" [9]. This metaheuristic was well exploited and firstly applied to the Traveling Salesman Problem (TSP) [2, 9].

The execution of an `ACO` algorithm is composed of cycles. Each ant is usually a constructive method and its behavior can be noted when, in order to choose the next vertex to where the ant must go, a probability is used which is calculated based on two factors: pheromone trail and heuristic information [27, 1]. Once the solutions are constructed by the ants, they are used to update the pheromone trail.

In our `ACO`-based algorithm, each ant constructs a solution S, choosing vertices to move to an iterative process. The choice of a vertex v, which was not visited, is based on the probability p, as follows:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_i]^\beta}{\sum_{k \notin Visited_k} [\tau_{ij}]^\alpha [\eta_{ij}]^\beta} & \text{if } j \notin Visited_k, \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

where τ_{ij} is the pheromone on edge (i, j), $\eta_{ij} = 1/d_{ij}$ is the visibility of the vertex j by ant k positioned on i, d_{ij} is the distance between i and j, $Visited_k$ is the set of vertices visited by the ant k, α is the importance of the pheromone and β is the importance of the visibility (heuristic information).

After all ants construct their solutions, the algorithm updates the pheromone trails. This, stored on matrix $P_{|V|\times|V|}$, is initialized with 1 for each edge between non-adjacent vertices and with 0 for each edge between adjacent vertices. Updating the pheromone trail involves the persistence of the current trail by a τ factor, and the evaporation that is based on a ρ factor. The evaporation (Equation (4)), and the general form of depositing pheromone (Equation (5)) are as follows:

$$P_{ij} = \rho P_{ij}, \qquad \forall i, j \in V, \tag{4}$$

$$P_{ij} = P_{ij} + \Delta\tau_{ij}^k, \qquad \Delta\tau_{ij}^k = \frac{Q}{T_k} \tag{5}$$

where Q is an empirical parameter and T_k is the length of the *tour* found by the ant k.

These steps are repeated until the algorithm reaches 100 cycles.

2.3.3 Returning the Best Target Code

After choosing optimizations and their order of application, there are $L*2$ sequences; L sequences extracted from the `KB`, and new L sequences, in which the order of the optimizations was modified. To find the best optimizations and their order, we measure the performance of each target code using each sequence. After this evaluation, we return the best target code.

3 EXPERIMENTAL SETUP AND METHODOLOGY

This section describes the experimental setup and the steps taken to ensure measurement accuracy, besides outlining the methodology used in the experiments.

Platform. The experiments were conducted on a machine with an Intel processor Core I7-3779, 8 MB of cache, and 8 GB of RAM. The operating system is Ubuntu 14.04, with kernel 3.13.0-37-generic.

Compiler. Our technique was implemented on top of `LLVM` 3.5 [13, 15]. The choice of `LLVM` is based on the fact that it allows full control over the optimizations. This means that it is possible to enable a list of optimizations through the command line. In addition, the position of each optimization indicates its order. Neither `GCC` nor `ICC` provides these features, thus, we need to use `LLVM` to show the results of our strategy.

Benchmark Suites. The experiments use the POLYBENCH suite [23] with a large dataset, and the CBENCH suite [5] with dataset 1, as test programs.

Measurement. The results are based on the arithmetic average of five executions. In the experiments, the machine workload was as minimal as possible. In other words, each instance was executed sequentially. In addition, the machine did not have an external interference, and the running time variance was close to zero.

Baseline. The baseline is the `LLVM`'s highest compiler optimization level, `O3`. In terms of running time, the optimization levels `O2` and `O3` have similar performance, on several programs. Therefore, we choose the highest compiler optimization level, `O3`.

Cross-Validation, Clustering, and `ACO`. The experiments use two distinct groups of programs, separating one group for training and the other for testing. Therefore, the experiments perform a holdout cross-validation. The clustering algorithm used was Farthest First, which is implemented on `Weka` [28]. In fact, we evaluate Expectation Maximization, Kmeans, and Farthest First, and the

latter obtained the best results. We use ACO to solve ATSP because it was extensively studied on the TSP. In addition, its way of choosing the next vertex is helpful for our purpose.

Parameters. The parameters used are:

- Clustering: $[10, 15]$, and $[30, 35]$. It indicates that the clustering-based algorithm will try to find at least 10 centroids, and at most 15; or at least 30, and at most 35, respectively. The former tries to gather the training programs, while the latter tries to scatter the training programs.
- ACO: $\alpha = 1$; $\beta = 5$; $\rho = 0.99$; and $Q = 100$.

Metrics. The evaluation uses three metrics to analyze the results, namely:

1. Average Percentage Improvement (API): indicates how much our strategy outperforms the compiler optimization level O3;
2. Average Percentage Improvement Excluding (APIE) Programs: indicates how much our strategy outperforms the compiler optimization level O3, considering only the programs whose performance outperforms the compiler optimization level O3's performance; and,
3. Number of Programs Achieving Improvement (NBI): indicates the number of programs whose performance, obtained with our strategy, was better than using the compiler optimization level O3.

The improvement is calculated as follows:

$$Speedup = baseline\_running\_time / new\_running\_time,$$

$$Improvement = (Speedup - 1) * 100.$$

Training and Deployment Cost. The training, which builds sequences, is a high time-consuming phase. It took several days, which is a significant amount of time. However, it is important to note that it is performed only once, besides performed at the factory. The deployment cost is calculated as follows:

$$Deployment_{cost} = C_{time} + O_{time} + \sum_{S=0}^{Sequences*2} \left(Comp_{time} + \sum_{N=0}^{5} Runtime \right)$$

where C_{time} is the time spent to choose optimizations; O_{time} is the time spent to order optimizations; $Comp_{time}$ is the time spent to compile the program using a specific sequence; and $Runtime$ is the program running time.

Choosing and ordering optimizations takes only 20 % of the system response time in our experiments. It is directly proportional to the size of the ATSP, besides the size of the KB. The other portion of the system response time (80 %) is caused by the need of compiling and running a program several times, in

order to evaluate a sequence and ensure measurement accuracy. In our experiments, choosing and ordering optimizations took from 0.015 (ADI) to 6.755 (GRAMSCHMIDT) seconds, while evaluating sequences took from 0.06 to 9 460.5 seconds.

Best10. In order to evaluate the effectiveness of our approach, we compare it with the one proposed by Purini and Jain [24]. They proposed an approach that extracts the best 10 optimization sequences, from a small search space, and use these sets to compile all programs. They argue that it is possible to outperform the optimization levels using only 10 sets.

Briefly, the approach used to select the best 10 sequences can be summarized in five steps:

1. Generate training data to N programs;
2. Extract the best sequence from each training program;
3. Remove the duplicate sequences;
4. Remove from each sequence the optimizations that do not contribute to reduce the running time; and finally
5. Extract from the search space the best 10 sequences.

4 RESULTS

This section evaluates our hybrid approach that searches for good optimizations and their order, aiming at performance improvements. In other words, this section evaluates our approach that searches for sequences that outperform the optimization level `O3` in terms of running time.

Tables 4 and 5 present the results. In these tables `OT.15` means our approach created at most 15 centroids, `OT.35` our approach produced at most 35 centroids, `Best10` is the algorithm proposed by Purini and Jain, and `BestAll` means the maximum improvement available for compiling the test program with all sequences on `KB`.

Overview. Our approach is able to find sequences that outperform the well-engineered compiler optimization level `O3`, besides `Best10`. Only in five benchmarks (POLYBENCH.ADI, POLYBENCH.LUDCMP, CBENCH.LAME, CBENCH.PATRICIA, and CBENCH.SHA) `Best10` outperforms our approach. In some cases the two approaches have similar performance to `BestAll`. Our approach achieves the maximum available improvement on 26 programs, while `Best10` on 23 benchmarks. In addition, our hybrid approach outperforms `BestAll` on 5 programs: POLYBENCH.JACOBI-2D, POLYBENCH.LU, POLYBENCH.REG_DETECT, CBENCH.TIFF2BW, and CBENCH.PGP_E.

Metrics. `API` means that the gap between our approach and `BestAll` is less than the gap between `Best10` and `BestAll`. This gap is 16.82 %, 41.33 %, and 37.38 %, respectively for `OT.15`, `OT.35` and `Best10`, on POLYBENCH; and 26.42 %, 39.03 %

Benchmark	OT.15	OT.35	Best10	BestAll
2mm	11.72	1.72	11.71	11.75
3mm	12.47	12.47	12.46	12.48
adi	0.002	−18.14	3.75	3.75
atax	6.72	2.58	6.36	7.44
bicg	0.97	0.97	0.97	0.97
cholesky	14.22	14.21	14.26	33.24
correlation	12.42	12.36	12.33	12.44
covariance	12.45	12.44	12.44	12.47
doitgen	12.27	12.27	11.67	12.55
durbin	0.75	0.00	−3.28	3.51
dynprog	12.28	0.00	11.99	12.28
fdtd-2d	6.29	−19.87	−4.89	6.30
fdtd-apml	3.42	0.00	3.58	8.00
floyd-warshall	0.00	0.00	0.00	0.01
gemm	11.04	11.07	0.01	11.09
gemver	1.59	−1.93	−3.76	3.02
gesummv	0.003	−4.74	0.00	1.26
gramschmidt	6.23	6.23	0.01	6.24
jacobi-1d	4.72	0.00	0.00	7.26
jacobi-2d	9.99	15.77	3.96	3.96
ludcmp	0.01	0.01	22.07	22.13
lu	22.13	22.12	0.03	7.03
mvt	1.90	1.90	0.00	1.90
reg_detect	12.35	28.25	12.33	12.44
seidel-2d	41.67	41.75	41.33	41.69
symm	7.14	7.12	7.12	15.3
syr2k	0.00	0.00	0.00	0.01
syrk	0.00	0.00	0.02	0.02
trisolv	4.39	1.59	4.40	7.34
trmm	11.09	0.01	0.01	11.12
API	8.01	5.65	6.03	9.63
APIE	8.29	11.31	7.14	9.63
NBI	30	26	27	30

Table 4. The improvements on POLYBENCH

and 55.27 %, on cBENCH. It means that in general, these gaps are 21.18 %, 40.36 % and 45.37 %, respectively. `APIE` also indicates that our approach outperforms `Best10`. However, `NBI` indicates that the two approaches and `Best10` have a similar performance.

Benchmarks. The general results, mainly `API`, show that our approach and `Best10` perform better on POLYBENCH. It can be explained by the fact that POLYBENCH is composed of kernels, while cBENCH of complete programs. However,

Benchmark	OT.15	OT.35	Best10	BestAll
bitcount	−46.45	−46.45	−45.75	−10.77
qsort1	10.45	10.45	6.71	10.45
susan_c	30.88	30.88	27.04	32.48
susan_e	5.75	6.29	1.86	6.88
susan_s	1.06	0.97	0.78	1.06
bzip2d	49.25	49.25	39.97	49.26
bzip2e	5.73	5.50	4.39	7.03
lame	3.22	6.16	8.75	8.75
mad	2.45	1.99	1.30	2.19
tiff2bw	17.04	14.94	7.45	13.32
tiff2rgba	15.45	15.07	7.49	15.11
tiffdither	2.06	−0.77	0.12	1.13
tiffmedian	25.89	22.14	23.05	26.02
dijkstra	0.59	0.14	0.43	0.64
patricia	0.00	−1.08	0.56	0.56
rsynth	0.46	−1.83	0.38	0.46
stringsearch1	−15.92	−28.91	−18.24	−0.36
blowfish_d	4.12	4.18	4.12	4.18
blowfish_e	4.10	4.10	3.94	4.16
pgp_d	5.13	4.97	1.91	6.51
pgp_e	4.72	0.96	0.59	3.49
rijndael_d	1.97	3.25	0.003	3.54
rijndael_e	−0.32	−2.24	−2.21	−0.32
sha	6.95	6.95	7.605	7.66
adpcm_c	13.12	12.98	5.83	15.05
adpcm_d	16.46	15.08	11.18	16.58
CRC32	2.13	2.13	2.13	2.13
gsm	3.85	3.63	1.97	3.88
API	6.07	5.03	3.69	8.25
APIE	9.70	10.09	6.78	9.70
NBI	25	22	25	25

Table 5. The improvements on cBench

the performance loss on cBench is due to the three programs that BestAll does not outperform with optimization level O3, namely: cBench.bitcount, cBench.stringsearch1 and cBench.rijndael_e. If we remove these three benchmarks, the scenario changes. In this case, API is 9.31 %, 8.73 %, 6.78 % and 9.70 %, respectively for OT.15, OT.35, Best10 and BestAll. The gap decreases from BestAll, and also reinforces that using a hybrid approach is the best choice.

Best10. It is important to remember that our approach and `Best10` have different premises. The former argues that it is necessary to handle individual programs, which means that the choice of optimizations and their order is program-specific dependent. The latter argues that it is possible to cover several sets of programs using the same sequences. The results indicate that handling individual programs tends to decrease the gap between the strategy and the maximum available improvement, consequently enhancing the performance.

Evaluations. Using a strategy that creates several centroids ($[30, 35]$) outperforms `Best10`, however it increases the distance from the maximum available improvement. Although, creating few centroids ($[10, 15]$) increases the performance, this strategy expands the response time. The problem is that this strategy creates less centroids, grouping more programs on the same cluster. As a result, more sequences will be evaluated. `Best10` needs to evaluate only 10 sequences. The strategy that creates about 35 centroids needs to evaluate at most $20 * 2$ sequences (20 programs in the same cluster plus 20 new sequences after changing their order), while that one that creates about 15 centroids needs to evaluate at most $30 * 2$. This means that in terms of evaluations, `Best10` is better than our hybrid approach.

5 RELATED WORK

Cavazos et al. [4] proposed a machine learning strategy to find compiler optimizations for a specific program. This work was the first to use performance counters to measure the similarity between two programs. A machine learning strategy creates a prediction model in a training stage, based on the behavior of several training programs, and the prediction model, in a deployment (or test) stage, predicts the set of optimizations that will be enabled to compile the unseen program. In a training stage, Cavazos's strategy randomly creates several compiler optimization sets for a group of training programs. After the creation of several sets, their strategy collects the performance counters of each training programs. Based on these two pieces of information, a model based on a logistic regression scheme is created, which will predict the set of optimizations. The deployment stage collects the performance counters of the test program, invokes the prediction model, and finally returns the best target code.

They demonstrated that a machine learning strategy is able to outperform the compiler optimization levels. Furthermore, they also demonstrated that the use of performance counters is a good strategy to measure the similarity between two programs. Our strategy is similar to such method, because we also use a machine learning scheme and measure the similarity between two programs in the same manner. However, while Cavazos et al. tried to find optimizations, our work is one step further due to its searches and optimization orders.

De Lima et al. [7] proposed the use of a case-based reasoning strategy to find compiler optimizations for a specific program. They argue that it is possible to find good compiler optimizations, from previous compilations, for an unseen program. This strategy creates several compiler optimization sets in a training stage. Afterwards, in the deployment stage, the strategy infers a good compiler optimization set for a new program. This step is based on the similarity between two programs. De Lima et al. proposed several models to measure similarity, also based on feature vectors, which is composed by performance counters. They demonstrated that it is possible to infer a good compiler optimization set that achieves multiple goals; for example, runtime and energy. The limitation of this work is that it does not handle the problem of ordering optimizations.

Purini and Jain [24] proposed a strategy to find good compiler optimization sets, which are able to cover several programs. This means that they do not handle this problem as program-dependent. The strategy to find several sets consists in using random and genetic algorithms to create effective sets of optimizations. After creating several sets, they eliminate the optimizations, from each set, that does not contribute to the performance. Finally, they proposed an algorithm that analyzes all sets, and extracts the best 10 sets. As a result, each test program is compiled using 10 sets, and the best target code is returned. They demonstrated that it is possible to find a small group of sets that are able to cover several programs.

Our strategy uses Purini's and Jain's strategy to provide a knowledge base of good compiler optimizations sets; however, we handle the problem of finding good optimizations as a program-dependent problem.

Tartara and Crespi [26] proposed a long-term strategy, the goal of which is to eliminate the training stage. In their strategy, the compiler is able to learn, during every compilation, how to generate good target code. In fact, they proposed the use of a genetic algorithm that creates several heuristics based on the static characteristics of the test program [20]. Basically, this strategy performs two tasks. First, it extracts the characteristics of the test program. Secondly, the genetic algorithm creates heuristics inferring which optimizations should be enabled. They demonstrated that it is possible to eliminate the training stage, using long-term learning. While Tartara's and Crespi's work does not need a training stage, our work does; however, we handle two problems concerning compiler optimizations.

Martins et al. [17] proposed a clustering strategy in order to find good compiler optimizations sets. In fact, they proposed algorithms to find good optimizations, besides algorithms to order optimizations. The strategy used by Martins et al. is similar to Purini's and Jain's, both use random and genetic algorithms. This means that their strategy can be considered as an iterative compilation, where the test program is compiled with different sets of optimizations, and the best version is chosen. Our strategy is classified as a machine learning strategy, which tries to reduce the number of times that a test program needs to be evaluated.

6 CONCLUDING REMARKS

The selection of compiler optimizations and their order of application has a significant impact on performance. In addition, we need to remember that this problem is program-specific dependent. Therefore, a good approach is to propose a design-space exploration scheme to find the program-specific optimization sequence.

In this paper we proposed a design-space exploration scheme, which aims to find good compiler optimization sequences. Our approach employs several strategies, namely: random sampling, genetic algorithm, clustering, metaheuristic, and a reduction scheme.

We implemented a hybrid approach on top of the `LLVM` compiler, and the experiment results show that it finds optimization sequences that outperform the standard optimization level `O3`, besides the approach proposed by Purini and Jain, `Best10`.

The deficiency of our approach is the system response time, due to the number of evaluations. In a future work we will investigate a strategy to decrease the system response time. In addition, as programs are composed by several subroutines and each one will probably be best-optimized by a specific sequence, another future work will be to handle each subroutine.

REFERENCES

[1] Abdelbar, A. M.—Wunsch, D. C.: Improving the Performance of MAX-MIN Ant System on the TSP Using Stubborn Ants. Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '12), New York, NY, USA, ACM, 2012, pp. 1395–1396, doi: 10.1145/2330784.2330949.

[2] Applegate, D. L.—Bixby, R. E.—Chvátal, V.—Cook, W. J.: The Traveling Salesman Problem: A Computational Study. Princeton University Press, 2007.

[3] Bertran, R.—Gonzalez, M.—Martorell, X.—Navarro, N.—Ayguade, E.: Decomposable and Responsive Power Models for Multicore Processors Using Performance Counters. Proceedings of the 24th ACM International Conference on Supercomputing (ICS '10), New York, NY, USA, ACM, 2010, pp. 147–158, doi: 10.1145/1810085.1810108.

[4] Cavazos, J.—Fursin, G.—Agakov, F.—Bonilla, E.—O'Boyle, M. F. P.—Temam, O.: Rapidly Selecting Good Compiler Optimizations Using Performance Counters. Proceedings of the International Symposium on Code Generation and Optimization (CGO '07), Washington, DC, USA, IEEE Computer Society, 2007, pp. 185–197, doi: 10.1109/CGO.2007.32.

[5] The Collective Benchmarks, 2014, `http://ctuning.org/wiki/index.php/CTools:CBench`. Access: January 20, 2016.

[6] Chabbi, M. M.—Mellor-Crummey, J. M.—Cooper, K. D.: Efficiently Exploring Compiler Optimization Sequences with Pairwise Pruning. Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, New York, NY, USA, ACM, 2011, pp. 34–45, doi: 10.1145/2000417.2000421.

[7] de Lima, E. D.—de Souza Xavier, T. C.—da Silva, A. F.—Ruiz, L. B.: Compiling for Performance and Power Efficiency. 23rd International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2013, pp. 142–149.

[8] Dongarra, J.—London, K.—Moore, S.—Mucci, P.—Terpstra, D.: Using PAPI for Hardware Performance Monitoring on Linux Systems. Proccedings of the Conference on Linux Clusters: The HPC Revolution, Linux Clusters Institute, 2001.

[9] Dorigo, M.—Stützle, T.: Ant Colony Optimization. Bradford Books, MIT Press, Cambridge, Massachusetts, 2004.

[10] Fang, S.—Xu, W.—Chen, Y.—Eeckhout, L.—Temam, O.—Chen, Y.—Wu, C.—Feng, X.: Practical Iterative Optimization for the Data Center. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 12, 2015, No. 2, pp. 15:1–15:26.

[11] Foleiss, J. H.—da Silva, A. F.—Ruiz, L. B.: An Experimental Evaluation of Compiler Optimizations on Code Size. Proceedings of the Brazilian Symposium on Programming Languages, São Paulo, Brazil, EACH USP, 2011, pp. 1–15.

[12] Kufrin, R.: PerfSuite: An Accessible, Open Source Performance Analysis Environment for Linux. Proceedings of the Linux Cluster Conference, Chapel, 2005.

[13] Lattner, C.—Adve, V.: LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation. Proceedings of the International Symposium on Code Generation and Optimization (CGO 2004), Palo Alto, California, March 2004, doi: 10.1109/CGO.2004.1281665.

[14] Lim, M. Y.—Porterfield, A.—Fowler, R.: SoftPower: Fine-Grain Power Estimations Using Performance Counters. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10), New York, NY, USA, ACM, 2010, pp. 308–311, doi: 10.1145/1851476.1851517.

[15] LLVM Team. The LLVM Compiler Infrastructure, 2016, `http://llvm.org`. Access: January 20, 2016.

[16] Malone, C.—Zahran, M.—Karri, R.: Are Hardware Performance Counters a Cost Effective Way for Integrity Checking of Programs? Proceedings of the Sixth ACM Workshop on Scalable Trusted Computing (STC '11), New York, NY, USA, ACM, 2011, pp. 71–76, doi: 10.1145/2046582.2046596.

[17] Martins, L. G. A.—Nobre, R.—Cardoso, J. A. M. P.—Delbem, A. C. B.—Marques, E.: Clustering-Based Selection for the Exploration of Compiler Optimization Sequences. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 13, 2016, No. 1, pp. 8:1–8:28, doi: 10.1145/2883614.

[18] Mucci, P. J.—Browne, S.—Deane, C.—Ho, G.: PAPI: A Portable Interface to Hardware Performance Counters. Proceedings of the Department of Defense HPCMP Users Group Conference, 1999, pp. 7–10.

[19] Muchnick, S. S.: Advanced Compiler Design and Implementation. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[20] Namolaru, M.—Cohen, A.—Fursin, G.—Zaks, A.—Freund, A.: Practical Aggregation of Semantical Program Properties for Machine Learning Based Optimization. International Conference on Compilers Architectures and Synthe-

sis for Embedded Systems (CASES '10), Scottsdale, United States, 2010, doi: 10.1145/1878921.1878951.

[21] Park, E.—Cavazos, J.—Alvarez, M. A.: Using Graph-Based Program Characterization for Predictive Modeling. Proceedings of the Tenth International Symposium on Code Generation and Optimization (CGO '12), New York, NY, USA, ACM, 2012, pp. 196–206, doi: 10.1145/2259016.2259042.

[22] Park, E.—Kulkarni, S.—Cavazos, J.: An Evaluation of Different Modeling Techniques for Iterative Compilation. Proceedings of the 14th International Conference on Compilers, Architectures and Synthesis for Embedded Systems, New York, NY, USA, ACM, 2011, pp. 65–74.

[23] Polybench. The Polyhedral Benchmark Suite. Access: March 2, 2014.

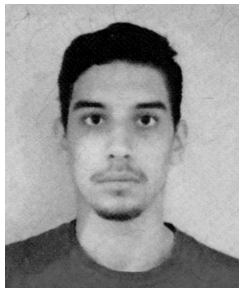
[24] Purini, S.—Jain, L.: Finding Good Optimization Sequences Covering Program Space. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 9, 2013, No. 4, pp. 56:1–56:23, doi: 10.1145/2400682.2400715.

[25] Srikant, Y. N.—Shankar, P.: The Compiler Design Handbook: Optimizations and Machine Code Generation. 2nd ed., CRC Press, Inc., Boca Raton, FL, USA, 2007.

[26] Tartara, M.—Crespi Reghizzi, S.: Continuous Learning of Compiler Heuristics. ACM Transactions on Architecture and Code Optimization (TACO), Vol. 9, 2013, No. 4, pp. 46:1–46:25, doi: 10.1145/2400682.2400705.

[27] Tavares, J.—Pereira, F. B.: Towards the Development of Self-Ant Systems. Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11), New York, NY, USA, ACM, 2011, pp. 1947–1954.

[28] Witten, I. H.—Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.



Tiago Cariolano de Souza Xavier got his B.Sc. and M.Sc. degrees in computer science from the State University of Maringá, Brazil, in 2012 and 2014, respectively. Nowadays, he is a Ph.D. student in computer science at the Federal University of Rio de Janeiro, Brazil. His research interest is in compilers and mobile systems.

Anderson Faustino da Silva is Professor in the Department of Informatics, State University of Maringá, Brazil, where he has been teaching since 2008. He received his B.Sc. degree in computer science from the State University of West Paraná, Brazil, in 2000, and his M.Sc. and Ph.D. degrees also in computer science from the Federal University of Rio de Janeiro, Brazil, in 2003 and 2006, respectively. His research interest is in compilers and parallel and distributed computing.

Computing and Informatics, Vol. 37, 2018, 186–212, doi: 10.4149/cai_2018_1_186

HYBRID DATA RACE DETECTION FOR MULTICORE SOFTWARE

Alper Sen, Onder Kalaci

Department of Computer Engineering
Bogazici University, Turkey
e-mail: {alper.sen, onder.kalaci}@boun.edu.tr

Abstract. Multithreaded programs are prone to concurrency errors such as deadlocks, race conditions and atomicity violations. These errors are notoriously difficult to detect due to the non-deterministic nature of concurrent software running on multicore hardware. Data races result from the concurrent access of shared data by multiple threads and can result in unexpected program behaviors. Main dynamic data race detection techniques in the literature are happens-before and lockset algorithms which suffer from high execution time and memory overhead, miss many data races or produce a high number of false alarms. Our goal is to improve the performance of dynamic data race detection, while at the same time improving its accuracy by generating fewer false alarms. We develop a hybrid data race detection algorithm that is a combination of the happens-before and lockset algorithms in a tool. Rather than focusing on individual memory accesses by each thread, we focus on sequence of memory accesses by each thread, called a segment. This allows us to improve the performance of data race detection. We implement several optimizations on our hybrid data race detector and compare our technique with traditional happens-before and lockset detectors. The experiments are performed with C/C++ multithreaded benchmarks using Pthreads library from PARSEC suite and large applications such as Apache web server. Our experiments showed that our hybrid detector is 15 % faster than the happens-before detector and produces 50 % less potential data races than the lockset detector. Ultimately, a hybrid data race detector can improve the performance and accuracy of data race detection, enhancing its usability in practice.

Keywords: Software testing and debugging, multithreaded programs, data race, concurrency, happens-before, lockset

Mathematics Subject Classification 2010: 68-N19

1 INTRODUCTION

Multicore processors provide high computation power, however, in order to utilize the increased power, concurrent software must be designed and written. Concurrency is achieved by multithreading in many systems. The interleaving of multiple threads can result in concurrency bugs which are hard to reproduce.

Data race is a well-known concurrency problem which is defined as two threads accessing a single memory address where at least one access is write and there is no appropriate synchronization among the accesses [19]. There have been many works in the past for detection of data races in multithreaded programs. The prior work is divided into two broad categories, which are static data race detection [28, 22, 3] and dynamic data race detection [13, 21, 2, 20, 15, 31].

Static data race detectors generate all possible thread interleavings and data races are searched among these interleavings. This approach is often infeasible due to state space explosion problem [12]. Moreover, they end up with many false positives due to the interleavings that are not possible to happen with any input data. Typically, the number of false positives is more than the number of real races [18]. *Dynamic data race detectors* observe memory accesses while an application is running. This approach is scalable to real life programs. The deficiency of dynamic analysis is that they consider an execution with a single input data, which limits the coverage of the analysis. Due to limited coverage, dynamic detectors miss some of the potential data races. In other words, dynamic detectors produce false negatives. In order to overcome this problem, dynamic analysis must be repeatedly executed with different inputs. Due to its scalability and potentially lower false positive rates, dynamic data race detection is the most commonly used method of data race detection.

There are two state-of-the-art algorithms used in dynamic data race detection, *lockset* and *happens-before* data race detection algorithms. Lockset algorithm [25] checks whether two threads access a shared variable while holding a common lock or not. For each shared memory address, the algorithm maintains a candidate lockset. Lockset detectors produce many false positives. In other words, some of the data races detected by the detectors are not real races. The main source of the false positives is that the detectors ignore all the synchronization operations except for locks. For instance, the detectors produce false positives for every shared memory access where the synchronization among accesses is generated by condition variables. Happens-before data race detection algorithm is based on Lamport's happens-before relation [13]. Happens-before relation defines a partial order among the events generated during the execution of a program in a distributed system. This relation has been extended for applications using shared memory as well. Happens-before data race detection algorithms utilize vector clocks for maintaining the happens-before relation [21, 15]. These detectors do not produce false positives, however they may miss real races (false negative). Happens-before based detectors suffer from high execution time and memory overhead. The size of each vector clock is proportional to the number of threads count in the program. On the

contrary, lockset based detectors are scalable and can be implemented with a low overhead.

There has been prior work [20, 26, 11, 21] for combining the benefits of lockset and happens-before detectors in a hybrid data race detector that has good performance and few false positives. We develop a hybrid approach in this paper as well. Almost all dynamic data race detection algorithms [2, 20, 21, 12, 15, 5] detect potential data races by tracking accesses on each memory address during the execution. This can result in memory overhead, which is crucial when working with large programs. Our algorithm instead detects potential races by tracking accesses on each segment [26], where a segment is formed by consecutive memory accesses of a single thread. No synchronization operation is allowed inside a segment, thus, all the memory accesses use the same vector clocks and the same locks. Moreover, as we later show in Table 2, for most of the applications, the number of segments is much less than the number of memory addresses accessed. This observation allows us to reduce memory overhead.

We propose four optimizations on our segment based hybrid algorithm. The first optimization is based on the observation that vector clock values of a segment does not change after it is assigned. Thus, we added a limited vector clock cache for the vector clocks of segments. The second optimization is based on exploiting the same memory accesses inside a segment. If a memory address is accessed more than once inside a segment, the second and subsequent accesses have no effect on detecting the potential data races. The third optimization is based on the active number of segments. We define a maximum number of active segments, if that is exceeded, we start discarding older segments. Although this may lead to missing some of the potential data races, performance increase is considerable in many cases. Our last optimization is based on sampling a user given percentage of memory accesses during analysis.

We implemented our hybrid detector and our optimizations using PIN Dynamic Binary Instrumentation (DBI) tool [14]. This tool allows us to work with binaries of applications rather than their source code, which may be crucial in commercial settings. In order to compare our hybrid detector we also implemented using PIN a lockset based detector (Eraser [25]) and a happens-before based detector ($DJIT+$ [21]). We performed experiments on 8 different applications from PARSEC benchmark suite [1], Apache httpd web server [9] and a parallel compression tool pbzip2 [6]. All benchmarks are written in C/C++ using Pthreads library. Our experiments showed that our hybrid detector is 15 % faster than happens-before detector and produces 50 % less potential data races than lockset detector.

The main contribution of this work can be summarized as follows:

- We develop a segment-based hybrid dynamic data race detector and present a formal treatment of the concepts and our algorithms.
- We propose four different optimizations on the hybrid algorithm. Two of these optimizations increase the performance of data race detection without sacrificing the precision. The remaining two optimizations provide a trade-off between

the number of potential data races detected and the performance of data race detection.

- We implement our techniques in a tool and compare with traditional dynamic data race algorithms on large multithreaded benchmarks.

The rest of the paper is organized as follows. Section 2 gives background on our multithreaded program model and happens-before relation. We describe dynamic data race detection algorithms in Section 3. In Section 4, we present our segment based hybrid data race detection algorithm and describe several optimizations on this algorithm in Section 5. We discuss experimental results in Section 6, which is followed by related work and conclusions.

2 BACKGROUND

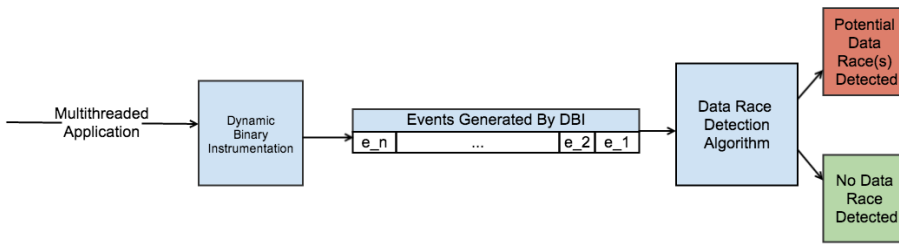


Figure 1. Overview of our dynamic data race detection

In Figure 1, an overview of our dynamic data race detection is displayed. A dynamic binary instrumentation tool instruments a multithreaded application and when this instrumented program is executed it generates events, which are input to the data race detection algorithm. Then, the algorithm decides whether the application has potential data race(s) or not.

2.1 Multithreaded Program Model

In this work, we consider multithreaded C/C++ applications that use the Pthreads library [23]. A multithreaded program consists of threads, memory addresses, and synchronization objects such as locks, condition variables, barriers, and semaphores.

During the execution of an instrumented program, a sequence of atomic operations (events), denoted by $e_x, \ldots, e_y$, are generated by each thread. We utilize the following types of events in our data race detection algorithms, similar to earlier works [20, 5, 26].

- *READ*(x, t_i): Memory address x is read by thread t_i.
- *WRITE*(x, t_i): Memory address x is written by thread t_i.

- $ACCESS(x, t_i)$: Either $READ(x, t_i)$ or $WRITE(x, t_i)$.
- $WR\_LOCK(l, t_i)$: Lock l is acquired write-held by thread t_i.
- $RD\_LOCK(l, t_i)$: Lock l is acquired read-held by thread t_i.
- $LOCK(l, t_i)$: Either $WR\_LOCK(l, t_i)$ or $RD\_LOCK(l, t_i)$.
- $UNLOCK(l, t_i)$: Lock l is released by thread t_i. This is also composed of read and write unlock operations.
- $SIGNAL(cv, t_i)$: Unblock at least one of the threads that are blocked on the condition variable cv.
- $SIGNAL\_ALL(cv, t_i)$: Unblock all threads currently blocked on the condition variable cv.
- $WAIT(cv, t_i)$: Thread t_i blocks on a condition variable cv.

We use the notion of synchronization points to generate the causality relationship among events. The following pair of corresponding events constitute the start (left) and the end (right) of synchronization points, denoted by *SYNCH_POINTS*: $(UNLOCK(l, t_i), LOCK(l, t_j))$, $(SIGNAL(cv, t_i), WAIT(cv, t_j))$, $(SIGNAL\_ALL(cv, t_i), WAIT(cv, t_j))$. Note that similar synchronization points can be defined for semaphores, barriers as well as thread creation and exit events.

2.2 Happens-Before Relation and Vector Clocks

There exist several techniques for tracking the concurrency information or the dependencies between events. Lamport's happened-before relation [13], which is a partial order relation, is used for capturing ordering between events generated during the execution of a concurrent program. More formally, the happens-before relation ($\rightarrow$) among two events e_x and e_y is denoted as $(e_x \rightarrow e_y)$ and is the smallest transitive relation that satisfies the following properties (where $x \neq y$ and $i \neq j$) [21, 5]:

- Program Order: $(e_x \in t_i \wedge e_y \in t_i) \wedge (e_x$ is executed before e_y in $t_i)$,
- Synchronization Order: $(e_x \in t_i \wedge e_y \in t_j) \wedge (i \neq j) \wedge (e_x$ and e_y is a pair of events from *SYNC_POINTS*),
- Transitivity: $(e_x \rightarrow e_z) \wedge (e_z \rightarrow e_y)$.

Two events, e_x and e_y are *concurrent* ($\|$) if neither of them happens-before the other, that is, $e_x \parallel e_y \Leftrightarrow (\neg(e_x \rightarrow e_y) \wedge \neg(e_y \rightarrow e_x))$.

Vector clocks [4, 17] are used to capture the happens-before relation among the events generated during program execution. A vector clock assigns timestamps to events such that the partial order relation between events can be determined by using the timestamps. A *vector clock*, VC, consists of a vector of n integers where n is the total number of threads in the execution. VC_i identifies the vector clock of thread t_i and $VC_i[j]$ holds the logical time of thread t_j known by thread t_i. Initially, $VC_i[j] = 0$, for $i \neq j$, and $VC_i[i] = 1$. A thread increments its own component of

the vector clock after each event. For certain events that will be described in the next section, it updates its vector clock by taking a component wise maximum with the vector clock included in the message.

Below we describe these common vector clock operations:

- *INIT*(VC_i): Initialize a Vector Clock, $VC_i[j] = 1$, for $i == j$ and $VC_i[j] = 0$, for $i \neq j$,
- *INCREMENT*(VC_i): Increment a Vector Clock, $VC_i[i] = VC_i[i] + 1$,
- *RECV*(VC_i, VC_j): Receive Vector Clock, $VC_i[k] = \max(VC_i[k], VC_j[k])$, $\forall k \in \{1, \ldots, n\}$,
- Compare Two Vector Clocks: $VC_i < VC_j$ is true, if $\forall k \in \{1, \ldots, n\} : VC_i[k] \leq VC_j[k] \wedge \exists k \in \{1, \ldots, n\} : VC_i[k] < VC_j[k]$.

We say that $e_x \rightarrow e_y$ iff $e_x.VC < e_y.VC$. Hence, vector clocks precisely capture the happens-before relation.

A sample execution of the vector clock algorithm is given in Figure 2, where the tuples in brackets represent the vector clocks. In the example, event (action) s happened-before t since $[1, 0, 0] < [2, 1, 3]$, where $v_i < v_j$ if all elements of v_i are less than or equal to the corresponding elements of v_j and at least one element of v_i is strictly less than the corresponding element of v_j. Whereas u is concurrent with t since their vector clocks are not comparable.

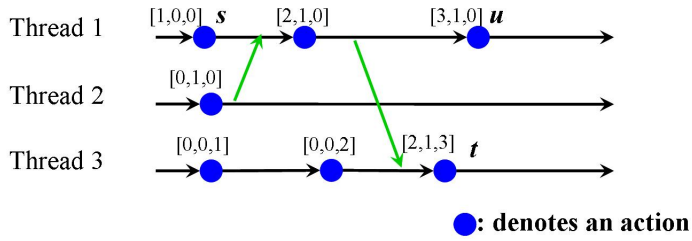


Figure 2. Happens-before relation and vector clocks

3 DATA RACE DETECTION ALGORITHMS

We briefly describe two state-of-the-art algorithms in data race detection, namely happens-before algorithm and lockset algorithm.

3.1 Happens-Before Data Race Detection

Happens-before data race detection algorithms check whether concurrent accesses by multiple threads to the same memory address are possible, if so they return a race

warning. These detectors do not produce false positives but they can produce false negatives.

We describe $DJIT^{+}$ [21] algorithm as the happens-before algorithm. The algorithm maintains a vector clock for each thread t_i, denoted by VC_i, which is initialized at startup. A vector clock is kept for each synchronization object s, denoted by $s.VC$, which is initialized to all zeros at startup. For each left event of a *SYNC_POINTS*, such as $UNLOCK(s, t_i)$ the following operations take place, $s.VC = VC_i$ and $VC_i = INCREMENT(VC_i)$. Similarly, for each right event of a *SYNC_POINTS*, such as $LOCK(s, t_i)$, the following operation takes place, $VC_i = RECV(VC_i, s.VC)$. For example, in Figure 2, in Thread 1, one can think of having an *UNLOCK* event after the second event with vector clock $[2, 1, 0]$ and the corresponding *LOCK* event occurs on Thread 3 changing the vector clock to $[2, 1, 3]$. Two vector clocks are kept for each memory address x, denoted by $x_r.VC$ and $x_w.VC$, which are used to keep track of the last read and the last write to x by each thread, respectively.

In Algorithm 1, we display the race detection portion of the happens-before based data race detection algorithm. For each read of x by t_i in line 1, the algorithm checks whether $x_w.VC$ happens-before VC_i in line 3. If not, a race is detected in line 12. Similarly, for each write of x by t_i in line 6, the algorithm checks whether each of $x_w.VC$ and $x_r.VC$ happens-before VC_i. If one of them does not, then the algorithm concludes a potential data race in line 12. Note that by exploiting the transitivity of the happens-before relation, the algorithm can decide on potential race conditions by only comparing the vector clock of the last read and write with the vector clock of the current access.

Algorithm 1 Happens-Before based data race detection algorithm

Input: Thread t_i generates a memory access event on address x, $ACCESS(x, t_i)$
Output: Potential data race detected or not

```
1:  if ACCESS(x, t_i) == READ(x, t_i) then
2:      x_r.VC[i] = VC_i[i];
3:      if x_w.VC < VC_i then
4:          return false;                          ▷ No Race Found
5:      end if
6:  else if ACCESS(x, t_i) == WRITE(x, t_i) then
7:      x_w.VC[i] = VC_i[i];
8:      if (x_w.VC < VC_i) ∧ (x_r.VC < VC_i) then
9:          return false;                          ▷ No Race Found
10:     end if
11: end if
12: return true;                                   ▷ Race Found
```

3.2 Lockset Data Race Detection

Lockset data race detection algorithms check whether accesses by multiple threads to the same memory address can occur while threads are not holding a common lock, if so they return a race warning. These detectors do not produce false negatives but they can produce false positives for a given execution.

We describe Eraser [25] algorithm as the basic lockset algorithm and show it in Algorithm 2. For each shared memory address x, the algorithm maintains a candidate lockset, $CLS(x)$. The name *candidate* is given since the algorithm cannot determine which lock is intended for which memory address. Thus, via candidate locksets, the algorithm attempts to infer whether a shared memory address is protected by a unique lock throughout the execution. When a memory address is accessed for the first time during the execution, its candidate lockset is assigned to include all possible locks. Then, on each access in line 1, its candidate lockset is updated to its intersection with the lockset of the thread that is executing the access, $LS(t_i)$. This lock refinement step aims to find the unique locks that protect the variable during the execution. If the intersection ends up with an empty set in line 2, the algorithm concludes that there is a potential data race in line 3. Eraser lockset algorithm includes optimizations and we implemented the Eraser lockset algorithm that includes optimizations in this paper.

Algorithm 2 Lockset based data race detection algorithm

Input: Thread t_i generates a memory access event on address x
Output: Potential data race detected or not

```
1: CLS(x) = CLS(x) ∩ LS(t_i)
2: if CLS(x) == {} then
3:     return true;                          ▷ Race Found
4: end if
5: return false;                             ▷ No Race Found
```

3.3 Hybrid Data Race Detection

When the above two approaches are examined in terms of preciseness, lockset detectors can produce false positives, whereas happens-before based detectors can produce false negatives. In terms of performance, lockset detectors are scalable, whereas happens-before based detectors are not because happens-before detectors require a high memory and processing overhead.

A hybrid data race detector combines the lockset and happens-before approaches. A naive hybrid algorithm maintains two vector clocks and a lockset for each memory address. Such an approach may help reduce the number of false positives compared to a lockset detector. However, performance of the detector would be worse than a happens-before based detector. First, the memory requirements would

be more than a happens-before detector since it also uses a lockset for each memory. Second, the computation overhead would increase since both vector clock comparisons and lockset calculations are needed.

4 SEGMENT BASED HYBRID DATA RACE DETECTION ALGORITHM

In order to combine the best of traditional race detectors, we developed a segment based hybrid algorithm that utilizes the concept of a *segment* to improve performance. Our segment based hybrid approach is based on Threadsanitizer [26] algorithm. We formalize this algorithm and extend it with several performance optimizations as discussed in the next chapter.

A segment seg_i of a thread t_i is a sequence of memory accesses of t_i, denoted by $\{e_i\}$, where the lockset and vector clock of the segment is the same as that of thread t_i. No function calls or synchronization operations are allowed inside a segment. The outcome of this is that all the events inside a segment are executed while the thread holds the same vector clock and the same locks. Thus, the vector clocks and the locksets could be kept on the granularity of segments instead of memory accesses. Since we observe that the total number segments is much less than the total number of memory addresses in an execution for most of the applications, this approach reduces the memory requirements and increases the performance considerably as shown in the experiments.

Our segment based hybrid algorithm maintains several data structures as shown in Table 1. Algorithm 3 shows how these data structures are updated during program execution. Note that a happens-before relation is established from a *SIGNAL* to a *WAIT* but not from an *UNLOCK* to a *LOCK* operation as in the case for happens-before algorithm. This is because the happens-before relation on lock operations ensures that the same lock is used by threads during memory access, however the lockset algorithm portion of the hybrid algorithm will consider this case.

For each memory access in line 17 of Algorithm 3, Algorithm 4 is executed. In Algorithm 4, first, the current segment of thread t_i that is executing the memory access is obtained as seg_i in line 1. Then, in line 2 for the write access, writer segment set of the memory address is updated so that it only includes segments that do not happen-before the current executing segment in line 3. Additionally, the current segment is added to the writer segment set of the memory address. Similarly, reader segment set of the memory address is updated so that it only includes segments that do not happen-before the current executing segment in line 4. It can be observed from the definition of concurrency that segments in write segment sets are pairwise concurrent, similarly for read segment sets. This is a useful performance optimization because accesses from non-concurrent segments, which are happens-before ordered, cannot cause a potential data race on a shared memory address.

Thread t_i	
Vector Clock	VC_i
Writer Lockset	$WRLS(t_i)$
Reader Lockset	$RDLS(t_i)$
Segment s_i	
Vector Clock	$seg_i.VC$
Writer Lockset	$WRLS(seg_i)$
Reader Lockset	$RDLS(seg_i)$
Condition Variable cv	
Vector Clock	$cv.VC$
Memory Address x	
Writer Segment Set	$WRST_x$.
Reader Segment Set	$RDST_x$.

Table 1. Segment-based hybrid algorithm data structures

It can be seen from Algorithm 4 that read and write accesses update segment sets differently. On write accesses, both writer and reader segment sets are updated (line 3 and 4), whereas, on read accesses, only reader segment set is updated (line 6). The reason is as follows, on write accesses, it is safe to remove any of the read accesses from $RDST_x$. Remember that, $RDST_x$ consists of concurrent segments where x is read. Since there is no read-read type of data race, removing any segment from $RDST_x$ does not lead to missing any races. On the contrary, on read accesses it is not safe to remove any segment from $WRST_x$. The reason is that, it may lead to missing a write-write data race because the removed segment might have a potential race with one of the prospective segments in the same set. The outcome of this is that all segments within any segment set are concurrent with each other. However, not all segments in $RDST_x$ are concurrent with all segments in $WRST_x$, which is handled while checking race among $WRST_x$ and $RDST_x$.

Algorithm 5 describes the data race checking between segment sets *checkRace(WRST,RDST)*, which is called at line 8 in Algorithm 4. The algorithm checks for common locks among concurrent segments such that one segment is a writer segment and the other is either a reader or a writer segment since there is no read-read data race. If the segments are a writer and a reader segment then the algorithm also makes sure that there is no happens-before relation from the writer to the reader segment as shown in line 8. We know that a happens-before relation cannot exist from the reader to the writer as these have been removed during the reader segment set update in Algorithm 4. If the algorithm could not find a common lock among concurrent segments (lines 4 and 9), it returns true indicating a data race. Note that if any two segments are ordered by the happens-before relation, they are not checked for data race, which is similar to the happens-before algorithm. Then, if two segments are concurrent then they are checked for holding a common lock, which is similar to the lockset algorithm.

Algorithm 3 Segment based hybrid data race detection algorithm instrumentation for each operation

Input: Thread t_i generates an operation *op*
Output: Instrumented program

1: **if** *op* is thread creation **then**
2: $INIT(VC_i)$; $WRLS(t_i) = RDLS(t_i) = \{\}$
3: **else if** *op* is segment creation **then**
4: $seg_i.VC = VC_i$; $WRLS(seg_i) = WRLS(t_i)$; $RDLS(seg_i) = RDLS(t_i)$
5: **else if** *op* is $WAIT(cv, t_i)$ **then**
6: $RECV(VC_i, cv.VC)$
7: **else if** *op* is $SIGNAL(cv, t_i)$ **then**
8: $RECV(cv.VC, VC_i)$; $INCREMENT(VC_i)$
9: **else if** *op* is $WR\_LOCK(l, t_i)$ **then**
10: $WRLS(t_i) = WRLS(t_i) \cup \{l\}$
11: **else if** *op* is $WR\_UNLOCK(l, t_i)$ **then**
12: $WRLS(t_i) = WRLS(t_i) \setminus \{l\}$
13: **else if** *op* is $RD\_LOCK(l, t_i)$ **then**
14: $RDLS(t_i) = RDLS(t_i) \cup \{l\}$
15: **else if** *op* is $RD\_UNLOCK(l, t_i)$ **then**
16: $RDLS(t_i) = RDLS(t_i) \setminus \{l\}$
17: **else if** *op* is $WRITE(x, t_i)$ or $READ(x, t_i)$ **then**
18: $ACCESS(x, t_i)$
19: **end if**

4.1 Segment Based Hybrid Algorithm Example

We now show an execution of our segment-based hybrid approach. A sample execution of the events belonging to two different threads and the state of data structures for each line of the execution is described in Figure 3. For simplicity, we do not display $RDST_x$ and *RDLS* of threads.

Algorithm 4 Segment-based hybrid data race detection algorithm – $ACCESS(x, t_i)$

Input: Thread t_i generates a memory access event on address x
Output: Potential data race detected or not

1: $seg_i = CurrentSegment(t_i)$
2: **if** $ACCESS(x, t_i) == WRITE(x, t_i)$ **then**
3: $WRST_x = \{seg_x \mid seg_x \in WRST_x \wedge \neg(seg_x.VC \rightarrow seg_i.VC)\} \cup seg_i$
4: $RDST_x = \{seg_x \mid seg_x \in RDST_x \wedge \neg(seg_x.VC \rightarrow seg_i.VC)\}$
5: **else if** $ACCESS(x, t_i) == READ(x, t_i)$ **then**
6: $RDST_x = \{seg_x \mid seg_x \in RDST_x \wedge \neg(seg_x.VC \rightarrow seg_i.VC)\} \cup seg_i$
7: **end if**
8: $checkRace(WRST_x, RDST_x)$

Algorithm 5 Segment-based hybrid data race detection algorithm – *checkRace*(*WRST*, *RDST*)

Input: Writer Segment Set *WRST*, Reader Segment Set *RDST*
Output: Potential data race detected or not

```
1: for wr1 ∈ WRST do
2:     for wr2 ∈ WRST do
3:         if WRLS(wr1) ∩ WRLS(wr2) = {} then
4:             return true;                          ▷ Write-Write Race Found
5:         end if
6:     end for
7:     for rd ∈ RDST do
8:         if ¬(wr1.VC → rd.VC) ∧ (WRLS(wr1) ∩ RDLS(rd) = {}) then
9:             return true;                          ▷ Read-Write Race Found
10:        end if
11:    end for
12: end for
13: return false;                                    ▷ No Race Found
```

After t_1 acquires lock l_1, a new segment s_1 is initialized. Memory address x is written in s_1, thus, s_1 is added to $WRST_x$ in ACCESS algorithm. s_1 is finalized when l_1 is released by t_1. Then, after t_2 acquires l_1, segment s_2 is initialized. When x is written in s_2, it is added to $WRST_x$ in ACCESS algorithm since s_1 and s_2 are concurrent. Since both s_1 and s_2 have a common lock, which is l_1, checkRace algorithm does not produce a data race alarm. The execution ends up with no race, which is a *True Negative*.

5 OPTIMIZATIONS ON SEGMENT-BASED HYBRID ALGORITHM

In this section, we are going to discuss four optimizations that we proposed and implemented on segment-based hybrid algorithm.

5.1 Optimization 1: Storing Vector Clock Comparison History Cache

We observed that maintaining a vector clock comparison history cache for the previously calculated vector clock comparisons can increase the performance of data race detection. This is motivated by the fact that multiple memory accesses can belong to the same segment and since all these memory accesses have the same vector clock as the segment that they belong to, there may be an excessive number of comparison operations between the same vector clocks. For each vector clock, we keep a list that holds the previous comparisons of the vector clock with other vector clocks. Since the same vector clock could be accessed concurrently, a lock is required for accessing the list. These two requirements increase the total memory requirement of data race detection but improves the performance as shown in the

Line	Thread 1	Thread 2
1	$WR\_LOCK(l_1, t_1)$	
2	$WRITE(x, t_1) \in s_1$	
3	$WR\_UNLOCK(l_1, t_1)$	
4		$WR\_LOCK(l_1, t_2)$
5		$WRITE(x, t_2) \in s_2$
6		$WR\_UNLOCK(l_1, t_2)$

Line	VC_1	$WRLS(t_1)$	VC_2	$WRLS(t_2)$	$WRST_x$
0	$\langle 1, 0\rangle$	$\{\}$	$\langle 0, 1\rangle$	$\{\}$	$\{\}$
1	$\langle 1, 0\rangle$	$\{l_1\}$	$\langle 0, 1\rangle$	$\{\}$	$\{\}$
2	$\langle 1, 0\rangle$	$\{l_1\}$	$\langle 0, 1\rangle$	$\{\}$	$\{s_1\}$
3	$\langle 1, 0\rangle$	$\{\}$	$\langle 0, 1\rangle$	$\{\}$	$\{s_1\}$
4	$\langle 1, 0\rangle$	$\{\}$	$\langle 0, 1\rangle$	$\{l_1\}$	$\{s_1\}$
5	$\langle 1, 0\rangle$	$\{\}$	$\langle 0, 1\rangle$	$\{l_1\}$	$\{s_1, s_2\}$
6	$\langle 1, 0\rangle$	$\{\}$	$\langle 0, 1\rangle$	$\{\}$	$\{s_1, s_2\}$

Figure 3. A program execution with segment-based hybrid algorithm and update of data structures during the execution

experiments. We also only cache vector clocks of segments since after initialization their values do not change, whereas vector clocks of threads and synchronization objects can change during execution, increasing computation overhead.

Our optimization is formalized in Algorithm 6. For each vector clock VC_i, a list $vc_i\_prev\_comparisons$ and a lock vc_i_lock is required. On each comparison, first the local cache $vc_i\_prev\_comparisons$ is searched. If the result is already there, there is no need to make comparison. Otherwise, the comparison is done and the result is added to $vc_i\_prev\_comparisons$.

The algorithmic complexity of this optimization is O(n), where n is the vector clock history size.

Algorithm 6 Optimization 1

Input: Vector Clocks vc_1 and vc_2
Output: Comparison of vc_1 and vc_2

1: $LOCK(vc_1\_lock)$;
2: $result = vc_1\_prev\_comparisons(vc_2)$;
3: **if** $result == None$ **then**
4: $\quad result = compare(vc_1, vc_2)$
5: $\quad vc_1\_prev\_comparisons(vc_2) = result$
6: **end if**
7: $UNLOCK(vc_1\_lock)$;
8: return $result$;

5.2 Optimization 2: Multiple Accesses of a Single Variable in a Segment

This optimization exploits the fact that repeated memory accesses of the same type and same variable belonging to the same segment do not make a difference in terms of race detection. This is because the earlier of the accesses will have already been added to the writer or reader segment set of the variable and since the later access has the same vector clock and locksets it will not have any impact. This optimization can quickly be added to the *ACCESS* algorithm as shown in Algorithm 7. There is no extra memory requirement for implementing this optimization. The only requirement is the increased CPU utilization. The algorithmic complexity of this optimization is $O(n)$, where n is the average number of segments in the segment sets.

Algorithm 7 Optimization 2 – updated Algorithm *ACCESS*

Input: Thread t_i generates a memory access event on address x
Output: Potential data race detected or not

1: $seg_i = CurrentSegment(t_i)$
2: **if** $ACCESS(x, t_i) == WRITE(x, t_i)$ **then**
3: **if** $seg_i \notin WRST_x$ **then**
4: $WRST_x = \{seg_x \mid seg_x \in WRST_x \wedge \neg(seg_x.VC \rightarrow seg_i.VC)\} \cup seg_i$
5: $RDST_x = \{seg_x \mid seg_x \in RDST_x \wedge \neg(seg_x.VC \rightarrow seg_i.VC)\}$
6: **end if**
7: **else if** $ACCESS(x, t_i) == READ(x, t_i)$ **then**
8: **if** $seg_i \notin RDST_x$ **then**
9: $RDST_x = \{seg_x \mid seg_x \in RDST_x \wedge \neg(seg_x.VC \rightarrow seg_i.VC)\} \cup seg_i$
10: **end if**
11: **end if**
12: $checkRace(WRST_x, RDST_x)$

5.3 Optimization 3: Limiting The Total Number of Segments

In segment-based hybrid approach, many of the segments do not cause potential data races. For instance, most segments do not access shared variables. Or even if some shared variables are accessed, many of the segments are happens-before ordered or protected by a common lock. Also, we make the observation that in general segments that are closer to each other in terms of execution time are more likely to cause race conditions. Therefore, discarding some of the segments may increase the performance while preserving the number of potential data races found.

In our implementation, we define a limit that identifies the maximum number of segments that can be utilized by our segment-based hybrid approach. Whenever the total number of segments exceeds the maximum number, we discard a previously created segment and remove that segment from any of the segment sets that it

is present. We utilize a FIFO queue for choosing which segment to be discarded. Although limiting the maximum number of segments increases the performance of data race detection, it may lead to missing some of the potential data races since the discarded segment can be a part of a potential data race in the execution.

It is important that the limit should be determined exclusively for each application. In our experiments, we choose the maximum number relative to the total segment count in the original execution.

In the worst case, the algorithmic complexity of this optimization is $O(n)$, where n is the total number of segment sets in the execution.

5.4 Optimization 4: Proportional Detection of Data Races

It is widely accepted that dynamic data race detection requires excessive processing power and memory. This hinders the utilization of dynamic data race detection tools in real deployed environments. PACER [2] proposes proportional detection of data races. It makes a proportionality guarantee by detecting data races at a rate equal to the sampling rate. Our sampling approach takes advantage of this observation but it is simpler than PACER with no proportionality guarantee. On each memory access operation in Algorithm 3, according to the given sampling rate, we decide whether to call *ACCESS* procedure or not. Specifically, a random integer is generated between 0 and 100. We use an equidistributed uniform pseudo-random number generator [16].

If the generated integer is smaller than the sampling rate, we execute the instrumentation for the memory access. Our experiments show that the percentage of instrumented memory accesses still converges to the sampling rate.

6 EXPERIMENTS

In this section, we describe our implementation and experiments on dynamic data race detection. We implemented three dynamic data race detection algorithms, lockset, happens-before, and segment-based hybrid algorithm. We also show the results of our optimizations described in the previous section. We performed experiments with eight multi-threaded applications from PARSEC benchmark suite [1]. We also used a parallel compression tool pbzip2 [6] and Apache httpd web server [9] as test cases. All the experiments were performed on a PC running Linux with a 4 cores CPU of 2.27 GHz and 32 GB of memory. We ran each experiment 10 times and averaged the results.

We implemented the detectors using PIN dynamic binary instrumentation (DBI) tool [14]. PIN allows us to work with binaries of applications rather than their source code, which may be crucial in commercial settings. In our implementation, we utilized the just-in-time compiler JIT mode of PIN. This allows instrumentation to be done at different granularities such as instruction, trace, image or routines. We used routine instrumentation for the synchronization function calls. For

instance, a callback is inserted after pthread mutex lock function so that our hybrid algorithm updates the caller thread's writer lockset. For memory accesses we used trace instrumentation. We know that instruction instrumentation enables to insert one analysis call for every instruction executed. In fact, instruction instrumentation could be useful for inserting an analysis call for every read and write instruction so that data race detection algorithms' memory access algorithms could be executed. However, reducing the number of analysis calls results in efficient instrumentation. Therefore, instead of instruction instrumentation we prefer trace instrumentation. A trace is composed of a sequence of Basic Block (BBL). A BBL is a single entrance and single exit sequence of instructions. Thus, by trace instrumentation we firstly iterate over BBLs that forms the trace that is instrumented. Then, for each BBL we iterate over instructions and call a function for each read or write instruction that is executed. The number of analysis calls is reduced from the executed instruction count to the number of executed traces. We developed a tool, called Dyndatarace, that incorporates our solution and can be accessed online[1].

In our implementations, we utilized the same implementation for the common parts of different detectors as much as possible. For instance, we use the same implementation of vector clocks for both the hybrid algorithm and the happens-before algorithm. Similarly, we used the same implementation of locksets for both the hybrid algorithm and the lockset algorithm.

While potential data races are being searched, it is crucial to track the dynamic allocations and deallocations to prevent false alarms. In our implementations, we overcome this problem by tracking all memory allocations and deallocations. Whenever a deallocation is executed, all the shadow memory state that is kept for the corresponding allocation call is cleaned up.

Benchmark	Base Time (sec)	# Thread	# Memory Addr.	# Segment	Execution Time				Potential Data Races		
					Empty	Lockset	Hybrid	Happens-Before	Lockset	Hybrid	Happens-Before
×264	0.18	15	4 579 993	6 225	2.92×	13.80×	20.26×	25.20×	184	27	1
freqmine	0.27	15	18 525 725	1 350	2.42×	117.93×	332.41×	261.56×	76	23	7
vips	0.22	18	22 883 398	32 044	3.67×	103.69×	250.21×	315.43×	174	64	0
swaptions	0.14	4	47 837	320	1.91×	72.43×	134.98×	202.15×	0	0	0
bodytrack	0.18	5	5 803 300	8 702	2.72×	18.67×	31.24×	50.37×	47	5	4
fluidanimate	0.25	4	1 303 047	2 163 828	1.43×	89.42×	85.10×	113.52×	281	35	0
streamcluster	0.25	8	166 555	163 641	1.73×	75.28×	133.05×	178.56×	19	14	0
canneal	1.90	4	3 012 749	1 380	4.13×	16.41×	20.88×	26.71×	1	1	0
pbzip2	1.40	16	1 050 268	157 60	2.32×	41.92×	44.27×	50.30×	29	23	0
httpd	26.11	22	251 943	229 900	2.21×	13.32×	24.11×	26.75×	1777	1151	0
average	3.09	11.10	5 762 481	262 315	2.54×	56.29×	107.65×	125.06×	258	134	1

Table 2. Results for three race detectors

[1] `https://github.com/onderkalaci/dyndatarace`

In our experiments we compare the behavior of detectors in terms of execution overhead as well as the number of potential data races. Table 2 displays our experimental results. In the table we show the overhead (slowdown) of dynamic binary instrumentation over the uninstrumented execution in the column denoted by *Empty*, which does not perform any computations or analysis related to data race detection but the overhead of instrumenting memory accesses. We also display the slowdown of each data race detector with respect to the *Empty* implementation. On average, execution slow down is 125× for happens-before detector, 107× for hybrid detector and 56× for lockset detector. Memory requirements are 2 084 M for happens-before detector, 1 258 M for hybrid detector and 916 M for lockset detector. Hence, the slow downs are arranged as *Happens-before* > *Hybrid* > *Lockset* as expected. Lockset slowdown is less than half of the happens-before and hybrid. This is acceptable since heavy memory and execution overhead of vector clocks are not present in the lockset. On the average, happens-before is about 15 % slower than hybrid.

In terms of the number of data races detected by each algorithm, the results are again expected and are as follows, *Lockset* > *Hybrid* > *Happens-before*. On average, the potential number of data races detected by happens-before, hybrid and lockset detectors are 1, 134, and 258, respectively. As discussed in the previous sections, lockset detectors produce too many false positives. On the contrary, happens-before detectors do not produce any false positives, but they can miss some of the potential data races. Hybrid algorithm poses characteristics from both approaches.

We performed experiments to measure the impact of increasing number of threads on the number of data races. For some of the applications, we were able to change the number of the threads that run concurrently without changing the inputs. Such applications include *x264*, *freqmine*, *fluidanimate*, *canneal* and *pbzip2*. For these applications, we performed three different experiments with different number of threads as shown in Table 3. Our experiments show that although the execution time depends on the thread count, the total number of potential data races detected are not affected.

Application	Original Execution	Experiment-1	Experiment-2	Experiment-3
x264	15	18	20	32
freqmine	15	18	20	32
fluidanimate	4	6	8	12
canneal	4	6	8	12
pbzip2	16	12	20	32

Table 3. Number of thread counts for different experiments

6.1 Results of Optimizations on Segment-Based Hybrid Algorithm

In this section, we explore the experimental results of the optimizations that are applied to the segment-based hybrid algorithm.

Since our goal is to find the best performance due to optimizations, we explore a combination of them rather than exploring optimizations individually. We first check the combination of Optimization 1 and 2 since these optimizations do not affect the number of data races and the best combination parameters are searched. Then we add either one of Optimization 3 and 4 to these two, since Optimization 3 and 4 cannot be combined. The reason is that both Optimization 3 and 4 lead to false negatives and affect the performance. Therefore, it would be difficult to infer the effect of each optimization on results when they are combined. The presented execution time values are relative to the execution of segment-based hybrid algorithm with no optimization applied.

Application	Execution Time				
History Size	0	1	10	50	250
×264	0.92×	0.95×	0.88×	0.89×	0.99×
freqmine	0.98×	0.85×	0.86×	0.83×	0.84×
vips	0.98×	0.9×	0.86×	0.97×	0.97×
swaptions	1.16×	1.44×	1.42×	1.33×	1.42×
bodytrack	1.09×	0.99×	0.95×	0.98×	1.03×
fluidanimate	0.96×	0.78×	0.77×	0.78×	0.79×
streamcluster	0.91×	0.68×	0.68×	0.68×	0.68×
canneal	0.99×	0.95×	0.99×	0.94×	0.97×
pbzip2	1.02×	1.0×	0.99×	1.0×	0.98×
httpd	0.71×	0.61×	0.62×	0.67×	0.64×
average	0.97×	0.91×	0.90×	0.91×	0.93×

Table 4. Optimization 1 and Optimization 2

6.2 Optimization 1 and Optimization 2

Table 4 shows the performance change by the addition of Optimization 1 when Optimization 2 is always enabled, since Optimization 2 almost always improves performance. On the average, when the vector clock history size is set to 10, the maximum performance is achieved.

For some of the applications such as swaptions, Optimization 1 reduces the performance, independent of the history size. In this case, unsuccessful searches in the vector clock history utilize CPU so much that the gain of successful searches cannot compensate the unsuccessful ones.

6.3 Optimization 1, Optimization 2 and Optimization 3

Table 5 displays the effect of Optimization 3 when Optimizations 1 and 2 are applied with a vector clock history size of 10. The table shows the effect of limiting the total segment count on both the execution time and the number of potential data races.

Since each application executes a different number of segments during execution, the parameter that is altered is the percentage of number of segments with respect to all segments in the original execution, displayed in row Limit. For instance, when the total number of segments are limited to 50 % of the original execution for that application, the performance is increased 17 % while only 4 of the potential data races are missed. We observe that even when the limit is set to 4 % the number of data races remain similar to the original and the performance improves by 31 %. However, the execution time does not decrease in a linear fashion. A potential reason for this is that the operation of destroying a segment requires a lot computation, that is the destroyed segment must be removed from all of the segment sets that it is a member of.

Limit	100 %		50 %		32 %		18 %		12 %		4 %		1 %	
Application	Time	Race	Time	Race	Time	Race	Time	Race	Time	Race	Time	Race	Time	Race
×264	0.91×	27	0.90×	27	0.86×	27	0.86×	27	0.88×	27	0.80×	26	0.66×	4
freqmine	0.88×	23	0.41×	23	0.40×	23	0.36×	23	0.31×	23	0.29×	19	0.88×	19
vips	0.87×	64	0.88×	63	0.88×	59	0.87×	55	0.79×	53	0.77×	38	0.61×	0
swaptions	1.44×	0	1.40×	0	1.34×	0	0.98×	0	0.95×	0	0.87×	0	1.32×	0
bodytrack	0.97×	5	0.83×	3	0.73×	0	0.77×	0	0.80×	0	0.71×	0	0.64×	0
fluidanimate	0.81×	35	0.67×	34	0.68×	34	0.66×	34	0.61×	34	0.59×	33	0.79×	22
streamcluster	0.72×	14	0.61×	14	0.59×	14	0.57×	14	0.55×	14	0.44×	14	0.50	14
canneal	1.0×	1	1.01×	1	0.92×	1	0.90×	1	0.95×	1	0.91×	1	0.98×	1
pbzip2	0.99×	23	0.98×	23	1.00×	23	0.98×	20	0.98×	17	0.97×	17	0.96	0
httpd	0.68×	1 151	0.62×	1 121	0.58	1 113	0.61×	1 113	0.58×	1 097	0.57×	1 097	0.68×	56
average	0.93×	134	0.83×	130	0.80×	129	0.76×	128	0.74×	126	0.69×	124	0.80×	11

Table 5. Results of Optimization 1, Optimization 2 and Optimization 3. Limit between 100 %–1 %.

6.4 Optimization 1, Optimization 2, and Optimization 4

Table 6 displays the effect of Optimization 4 when Optimizations 1 and 2 are applied with a vector clock history size of 10. The table shows that on average execution times and the number of potential data races roughly converge to the sampling rate, until 16 % sampling rate. When the sampling rate is decreased further, due to the overhead of creating and managing segments, performance does not converge to the sampling rate. Furthermore, for *pbzip*2, the execution time is not converging to the sampling rate since this application makes too many I/O operations, which is a bottleneck.

6.5 Vector Clock Operation Performance

In this section, we compare the average number of vector clock operations per memory access between the happens-before and hybrid detectors. In the happens-before Algorithm 1, the number of vector clock comparisons is one or two depending on the access type as can be seen from lines 3 and 8, respectively. As the proportion of reads increases, the average number of vector clock comparisons per memory access is expected to decrease. In the segment-based hybrid approach, the number

Sample Rate	100 %		71 %		50 %		16 %		5 %		2 %	
Application	Time	Race	Time	Race	Time	Race	Time	Race	Time	Race	Time	Race
×264	0.92×	27	0.66×	23	0.49×	19	0.17×	12	0.11×	9	0.10×	3
freqmine	0.88×	23	0.21×	23	0.13×	23	0.04×	12	0.02×	11	0.06×	4
vips	0.87×	64	0.67×	53	0.49×	35	0.12×	17	0.07×	1	0.06×	0
swaptions	1.54×	0	0.74×	0	0.67×	0	0.23×	0	0.21×	0	0.19×	0
bodytrack	1.01×	5	0.91×	0	0.52×	1	0.16×	0	0.08×	0	0.07×	0
fluidanimate	0.82×	35	0.56×	28	0.44×	27	0.08×	12	0.06×	2	0.06×	2
streamcluster	0.74×	14	0.42×	13	0.33×	12	0.12×	5	0.04×	2	0.05×	0
canneal	1.02×	1	0.82×	1	0.65×	1	0.37×	0	0.28×	0	0.31×	0
pbzip2	1.04×	23	0.97×	13	0.81×	6	0.74×	1	0.72×	0	0.75×	0
httpd	0.71×	1 151	0.41×	797	0.29×	478	0.08×	111	0.12×	23	0.04×	15
average	0.96×	134	0.64×	95	0.48×	60	0.21×	17	0.17×	4	0.17×	2

Table 6. Results of Optimization 1, Optimization 2 and Optimization 4. Sample rate between 100 %–2 %.

of vector clock operations depends on the size of the segment set for each memory access which increases as the number of concurrent accesses to a memory address increases. The comparisons are done in two different points in segment-based hybrid algorithm in lines 3, 4, 6 of Algorithm 4 and line 8 of Algorithm 5.

Application	Hybrid	Happens-Before
x264	2.12	1.07
freqmine	14.87	1.16
vips	7.99	1.11
swaptations	4.78	1.04
bodytrack	2.85	1.13
fluidanimate	1.85	1.08
streamcluster	2.58	1.07
canneal	2.54	1.34
Firefox	0.34	1.07
pbzip2	0.99	1.07
httpd	22.80	1.37
average	5.70	1.13

Table 7. Average number of vector clock comparison per memory access

The number of vector clock operations cannot be determined statically. Therefore, to calculate the average number of vector clock operations in both algorithms, we create two variables (*m_access_cnt*, *vc_compare_cnt*) and on each memory access, we increment the value of *m_access_cnt* and on each vector clock comparison, we increment the value of *vc_compare_cnt*. Then, after the execution completes, we calculate the vector clock comparison per memory access, $vc_{avg} = vc\_compare\_cnt/m\_access\_cnt$. Table 7 shows vc_{avg} for both algorithms. As expected,

vc_{avg} is 1.13 for happens-before algorithm, whereas, for segment-based hybrid algorithm, vc_{avg} is 5.70. Therefore, we can conclude that on average, happens-before algorithm performs better than the hybrid segment-based algorithm in terms of the average number of vector clock operations per memory access.

6.6 Memory Performance

We show the memory requirements of lockset, happens-before and segment-based hybrid algorithms for each application in Table 8. The results show that hybrid detector's memory requirement is almost 40 % less than happens-before detector and 30 % more than lockset detector. These results are expected since utilization of vector clocks increases memory requirements remarkably. Happens-before detectors are entirely based on vector clocks, lockset detectors do not utilize vector clocks and hybrid detectors are partially based on vector clocks.

Application	Lockset	Hybrid	Happens-Before
x264	794	884	2 785
freqmine	1 444	2 201	3 873
vips	1 502	2 220	3 564
swaptations	304	574	744
bodytrack	752	1 408	2 062
fluidanimate	674	886	1 240
streamcluster	513	590	574
canneal	750	904	1 766
pbzip2	721	910	1 104
Firefox	1 066	1 271	1 971
httpd	1 562	1 990	3 244
average	916	1 258	2 084

Table 8. Memory requirements for data race detection algorithms (MB)

7 SUMMARY OF RESULTS

We proposed four different optimizations on our algorithm to improve performance. The first two improvements do not alter the number of data races that are detected in any of the benchmark applications. When the first two optimizations are combined they reduce the execution time by 10 %, if the vector clock comparison history cache size is set to 10. The last two improvements alter both the execution time and data races detected relative to the original application. For the third optimization, when the total number of segments is limited to 50 % of the number of segments in the original application, execution time decreases by 17 % where the number of potential races almost remains the same. For the last optimization, average execution times and the number of detected data races roughly converge to a user given sampling

rate. We also performed experiments which show that the memory requirements of happens-before algorithm is more than segment-based hybrid algorithm, whereas, hybrid algorithm executes more vector clock comparisons on the average than the happens-before.

We showed that a hybrid race detector has several advantages over other types of race detectors as described above. However, due to the use of instrumentation there is a big slowdown for all detectors that can make it impractical for online execution purposes. We discuss potential improvements in future work.

8 RELATED WORK

Dynamic data race detection algorithms proposed in the literature are based on lockset approach, happens-before approach or combination of both approaches. In this section we explore these approaches and their differentiated variants. *Eraser* [25] is the pioneer of the lockset based detectors. Since lockset based detectors only recognize locks, the synchronization formed among threads by other primitives such as condition variables or barriers are ignored. Thus, false positives are inevitable with lockset detectors.

Happens-before detectors inspect data races by verifying the happens-before relation among memory accesses. The happens-before relation is represented by vector clocks in many happens-before based detectors such as *DJIT+* [21] and *LiteRace* [15]. Contrary to lockset detectors, happens-before detectors do not produce any false positives. However, they miss some potential data races. In other words, pure happens-before based detectors produce false negatives. *DJIT*$^+$ and other traditional happens-before detectors suffer from high execution time and memory overhead. The size of vector clocks are proportional to the number of threads in the system. Therefore, memory requirements and comparison complexity of vector clocks are proportional to the thread count. In order to overcome these performance issues several enhancements have been proposed. *FastTrack* [5] implements a scalar data structure, called *epoch*, consisting of two integers, and replaces vector clocks with epochs whenever possible. This replacement does not affect the precision of the happens-before algorithm. iFT [7] represents an improvement over the FastTrack method. It reduces the average runtime and memory overhead to 84 % and 37 %, respectively, of those of FastTrack.

PACER [2] proposes a sampling method on FastTrack algorithm. It makes a proportionality guarantee such that it detects potential data races at a rate equal to the sampling rate. *LiteRace* [15] is another happens-before data race detection algorithm that applies a different sampling approach than ours, and detects 70 % of data races by sampling only 2 % of memory accesses. This work is orthogonal to ours. With low sampling rates, Carisma [32] can detect race conditions also.

Hybrid data race detector algorithms combine happens-before and lockset detectors. One of the main purposes of these detectors is to solve the false positive problem of lockset approach and false negative problem of happens-before approach.

In [20], the combination of both approaches is implemented where their algorithm incurs a higher overhead than lockset detector, but the number of false positives is decreased. AccuLock [30] is a hybrid detection algorithm that combines FastTrack and a new Lockset analysis. *RaceTrack* [31] implements a hybrid adaptive algorithm that automatically pays more attention to the more suspicious code. The algorithm aims to increase the precision while decreasing the overhead. *Threadsanitizer* [26] and *Helgrind+* [11] decrease the execution overhead by inspecting data races among segments instead of memory addresses similar to ours. We implemented several other optimizations in this work that do not exist in those works and present a more formal treatment of segments, which was not done previously.

There are other approaches for race detection as well. Race detection can be considered as a safety verification problem for concurrent programs. Model checking is a formal verification technique that can find and prove the absence of races. Safety or liveness can be given in the form of temporal logics such as LTL or CTL. Model checking has been used in the context of race detection [8, 24]. However, due to the exponentiality of both the program path and the scheduling space model checking does not scale well to large programs. Runtime verification, similar to testing, deals only with observed execution of a program, whereas model checking considers all possible executions. Similar to model checking, in runtime verification temporal logic specifications can be used and one can generate trace reorderings under scheduling constraints to find bugs unseen in the observed execution [10].

In [27], the authors propose a new relation, called casually precedes, which is a more generalized relation than happens-before relation. This new relation does not sacrifice from the precision of happens-before relation, instead, it enables the detector to produce fewer false negatives. In [10], the authors present a sound predictive race detection technique based on a new foundation of maximal causal model incorporating the control flow information. There is also work on parallelizing data race detection [29] which shows that with 4 cores as the original application, they can speed up the median execution time by 4.4 for a happens-before detector and by 3.3 for a lockset race detector.

9 CONCLUSION

We presented a new segment-based hybrid data race detection algorithm with optimizations, which is a combination of happens-before and lockset algorithms. Data race detectors suffer from low performance and may produce many false alarms. We use the concept of segments as well as several other optimizations to improve its performance and reduce false alarms. We implemented our algorithms using a dynamic binary instrumentation platform. We compared our results with traditional lockset-based and happens-before based data race detection algorithm on several multithreaded applications and obtained favorable results. Our hybrid detector is 15 % faster than the happens-before detector and produces 50 % less potential data races than the lockset detector.

As a future work, the segment method can be applied to happens-before algorithm where it could improve the performance without sacrificing the precision. Although our dynamic binary instrumentation allows us to work with binaries of applications and does not require the source code, we will investigate other efficient instrumentation techniques, such as compiler based instrumentation, to improve the applicability of detectors in industrial settings.

Acknowledgment

This research was supported in part by Bogazici University Research Fund 13662.

REFERENCES

[1] Bienia, C.—Kumar, S.—Singh, J. P.—Li, K.: The PARSEC Benchmark Suite: Characterization and Architectural Implications. International Conference on Parallel Architectures and Compilation Techniques (PACT), 2008, doi: 10.1145/1454115.1454128.

[2] Bond, M. D.—Coons, K. E.—McKinley, K. S.: PACER: Proportional Detection of Data Races. Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '10), 2010, pp. 255–268, doi: 10.1145/1806596.1806626.

[3] Engler, D.—Ashcraft, K.: RacerX: Effective, Static Detection of Race Conditions and Deadlocks. ACM SIGOPS Operating Systems Review – SOSP '03, Vol. 37, 2003, No. 5, pp. 237–252, doi: 10.1145/945445.945468.

[4] Fidge, C.: Logical Time in Distributed Computing Systems. IEEE Computer, Vol. 24, 1991, No. 8, pp. 28–33, doi: 10.1109/2.84874.

[5] Flanagan, C.—Freund, N. S.: FastTrack: Efficient and Precise Dynamic Race Detection. Communications of the ACM, Vol. 53, 2010, No. 11, pp. 93–101, doi: 10.1145/1839676.1839699.

[6] Gilchrist, J.: Parallel Data Compression with bzip2. Proceedings of the International Conference on Parallel and Distributed Computing and Systems, 2004.

[7] Ha, O.-K.—Jun, Y.-K.: An Efficient Algorithm for On-the-Fly Data Race Detection Using an Epoch-Based Technique. Scientific Programming, 2015, Art. No. 205827, doi: 10.1155/2015/205827.

[8] Henzinger, A. T.—Jhala, R.—Majumdar, R.: Race Checking by Context Inference. ACM SIGPLAN Notices, Vol. 39, 2004, No. 6, pp. 1–13, doi: 10.1145/996841.996844.

[9] The Apache HTTP Server Project – 2.2.22, 2014. Accessed: May 2014.

[10] Huang, J.—O'Neil Meredith, P.—Rosu, G.: Maximal Sound Predictive Race Detection with Control Flow Abstraction. ACM SIGPLAN Notices, Vol. 49, 2014, No. 6, pp. 337–348.

[11] Jannesari, A.—Bao, K.—Pankratius, V.—Tichy, F. W.: Helgrind+: An Efficient Dynamic Race Detector. Proceedings of the IEEE International

Symposium on Parallel and Distributed Processing (IPDPS 2009), 2009, doi: 10.1109/IPDPS.2009.5160998.

[12] Kahlon, V.—Yang, Y.—Sankaranarayanan, S.—Gupta, A.: Fast and Accurate Static Data-Race Detection for Concurrent Programs. In: Damm, W., Hermanns, H. (Eds.): Computer Aided Verification (CAV '07). Lecture Notes in Computer Science, Vol. 4590, 2007, pp. 226–239.

[13] Lamport, L.: Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM (CACM), Vol. 21, 1978, No. 7, pp. 558–565, doi: 10.1145/359545.359563.

[14] Luk, C.-K.—Cohn, R.—Muth, R.—Patil, H.—Klauser, A.—Lowney, G.—Wallace, S.—Reddi, V. J.—Hazelwood, K.: Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05), 2005, pp. 190–200, doi: 10.1145/1065010.1065034.

[15] Marino, D.—Musuvathi, M.—Narayanasamy, S.: LiteRace: Effective Sampling for Lightweight Data-Race Detection. Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '09), 2009, pp. 134–143, doi: 10.1145/1542476.1542491.

[16] Matsumoto, M.—Nishimura, T.: Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. ACM Transactions on Modeling and Computer Simulation (TOMACS), Vol. 8, 1998, No. 1, pp. 3–30, doi: 10.1145/272991.272995.

[17] Mattern, F.: Virtual Time and Global States of Distributed Systems. Proceedings of the Workshop on Distributed Algorithms (WDAG), 1989, pp. 120–131.

[18] Naik, M.—Aiken, A.—Whaley, J.: Effective Static Race Detection for Java. Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '06), 2006, pp. 308–319, doi: 10.1145/1133981.1134018.

[19] Netzer, R. H. B.—Miller, B. P.: What Are Race Conditions? Some Issues and Formalizations. ACM Letters on Programming Languages and Systems (LOPLAS), Vol. 1, 1992, No. 1, pp. 74–88, doi: 10.1145/130616.130623.

[20] O'Callahan, R.—Choi, J.-D.: Hybrid Dynamic Data Race Detection. Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '03), 2003, pp. 167–178, doi: 10.1145/781498.781528.

[21] Pozniansky, E.—Schuster, A.: Efficient On-the-Fly Data Race Detection in Multithreaded C++ Programs. Proceedings of the Ninth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '03), 2003, pp. 179–190.

[22] Pratikakis, P.—Foster, S. J.—Hicks, M.: LOCKSMITH: Context-Sensitive Correlation Analysis for Race Detection. Proceedings of the 27th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '06), 2006, pp. 320–331, doi: 10.1145/1133981.1134019.

[23] POSIX Pthreads, IEEE Std 1003.1, 2013 Edition, `http://www.unix.org/version4/ieee_std.html`, 2014.

[24] Qadeer, S.—Wu, D.: KISS: Keep It Simple and Sequential. ACM SIGPLAN Notices – PLDI '04, Vol. 39, 2004, No. 6, pp. 14–24, doi: 10.1145/996841.996845.

[25] Savage, S.—Burrows, M.—Nelson, G.—Sobalvarro, P.—Anderson, T.: Eraser: A Dynamic Data Race Detector for Multithreaded Programs. ACM Transactions on Computer Systems, Vol. 15, 1997, No. 4, pp. 391–411, doi: 10.1145/265924.265927.

[26] Serebryany, K.—Iskhodzhanov, T.: ThreadSanitizer: Data Race Detection in Practice. Proceedings of the Workshop on Binary Instrumentation and Applications (WBIA '09), 2009, pp. 62–71, doi: 10.1145/1791194.1791203.

[27] Smaragdakis, Y.—Evans, J.—Sadowski, C.—Yi, J.—Flanagan, C.: Sound Predictive Race Detection in Polynomial Time. Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '12), 2012, pp. 387–400, doi: 10.1145/2103656.2103702.

[28] Voung, J. W.—Jhala, R.—Lerner, S.: RELAY: Static Race Detection on Millions of Lines of Code. Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC-FSE '07), 2007, pp. 205–214, doi: 10.1145/1287624.1287654.

[29] Wester, B.—Devecsery, D.—Chen, P. M.—Flinn, J.—Narayanasamy, S.: Parallelizing Data Race Detection. Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '13), 2013, pp. 27–38, doi: 10.1145/2451116.2451120.

[30] Xie, X.—Xue, J.: Acculock: Accurate and Efficient Detection of Data Races. Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO '11), 2011, pp. 201–212.

[31] Yu, Y.—Rodeheffer, T.—Chen, W.: Racetrack: Efficient Detection of Data Race Conditions via Adaptive Tracking. Proceedings of the Twentieth ACM Symposium on Operating Systems Principles (SOSP '05), 2005, pp. 221–234, doi: 10.1145/1095810.1095832.

[32] Zhai, K.—Xu, B.—Chan, W. K.—Tse, T. H.: CARISMA: A Context-Sensitive Approach to Race-Condition Sample-Instance Selection for Multithreaded Applications. Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA 2012), 2012, pp. 221–231, doi: 10.1145/2338965.2336780.

Alper Sen received his B.Sc. and M.Sc. degrees in electrical and electronics engineering from Middle East Technical University, Ankara, Turkey, in 1995 and 1997, respectively, and his Ph.D. degree in electrical and computer engineering from the University of Texas at Austin, Austin, in 2004. He was Technical Staff Member with Freescale Semiconductor, Austin, and an Adjunct Faculty Member with the University of Texas at Austin, until 2009. Currently he is Associate Professor with the Department of Computer Engineering, Bogazici University. His current research interests include verification of hardware and software systems, parallel programming, embedded systems, and system-level designs.

Onder Kalaci received his B.Sc. degree in computer engineering from Middle East Technical University, Ankara, Turkey, and his M.Sc. degree in computer engineering from the Bogazici University. He is currently Ph.D. student at the Bogazici University.

Computing and Informatics, Vol. 37, 2018, 213–228, doi: 10.4149/cai_2018_1_213

ELLIPSE DETECTION IN FORENSIC BLOOD STAIN IMAGES ANALYSIS

Tomasz Wójtowicz, Dariusz Bułka

CYBID Ltd.
Kuźnicy Kołłatajowskiej 15c/L2
31-234 Kraków, Poland
e-mail: tomasz.wojtowicz@ii.uj.edu.pl

Abstract. This paper presents an algorithm for ellipse detection on stains of blood, which is directly suited for the needs of forensic analysis. The algorithm is of the edge-analyzing type. It performs convexity detection and is able to split contours of overlapping ellipses by concave regions analysis. A filtering is applied to the fit ellipses to reduce the results only to those results necessary for blood drop trajectory tracing. Results for running time and fitting quality tests performed on real-life and artificial data are presented. The solution answers to both quality and running time expectations of the field of application.

Keywords: Ellipse detection, ellipse fitting, edge analysis, contour segmentation, blood stain, forensic analysis

1 INTRODUCTION

The modern forensic analysis employs computer to recreate the site of an accident or a criminal action with the use of dedicated modeling software, basically 3D, currently also 4D (with time). The on-site measurements and taken pictures allow to create a polygon mesh and orthophotomap textures for this mesh. One particular feature found in pictures that meets with high attention are the blood stains. It is possible to recreate the trajectory of a blood drop that created a given stain, producing a valuable insight into the analysed event, if the geometric characteristics of such stain are obtained. The trajectory tracing is a rather simple Newtonian/Bernoulli mechanics, the really difficult part is to produce a proper geometric description of a stain. This process is currently done manually by encircling a blood stain image

with a vector ellipse on screen. It is an exhausting and error-prone proces with scenes that sometimes may contain tens or hundreds of blood stains. For this reason, automation of this process is desired. This paper presents a working automatic blood stain detection algorithm, which uses ellipse detection, and is implemented with the aim to augment the scene modelling software.

There are substantial differences between the scientific interest in ellipse detection and the requirements of this particular field of application. The scientific papers of the topic usually boast the fitness of their algorithms in the terms of a number of detected ellipses and how complicated the input may be. On the other hand, the aspect of running time is often skipped in those papers. For the forensic analysis it is not necessary to detect every single ellipse there may happen. For instance it would be more productive to produce 20 representative ellipses rather than flood the user with 200 of them. By representative we mean such ones which will not be found as a questionable evidence in the court – only highly regular stains really matter, so the algorithm should not bother with disfigured stains. Therefore, we have introduced a filtering mechanism to reduce the output to those ellipses which would be the most relevant.

However, the running time becomes an important matter. The presented solution is supposed to work in an interactive mode, aside or within the modelling software. On the average computer found in the forensic laboratory the detection time should take some 10 seconds. Be it 10 minutes, the solution is useless, as the technician would do the job manually in that time. This restriction practically eliminates more complicated approaches that are known for having a longer running time.

Figure 1 presents a fragment of a taken picture of a real blood stain. This is the input on which the algorithm is typically working and such is the average quality. As seen in the picture, the edge of a stain is blurred, what in binarisation produces a certain level of jagginess. The stain has a tail and occasional extruding or intruding artifacts, all of which must be removed before the ellipse fitting takes place. Moreover, occasionally two or more separate splatters intersect, resulting in a cojoined stain just like the one in this picture. Cojoined stains are considered a lesser quality evidence, nevertheless the algorithm must correctly detect them just to know that status. The algorithm presented in this paper succesfully copes with the problems mentioned above.

1.1 State of the Art

Ellipse detection is an important topic in computer image understanding with numerous different applications, many of which in natural sciences, medicine and related fields. There are several different approaches to ellipse detection, employing various techniques, what is reflected in running times and output qualities of particular solutions. One such approach, probably the eldest, are voting based algorithms, such as Hough transform and its variants [3, 9]. Another approach is based on genetic algorithms, such as [10] which executes the detection as a multi-objective

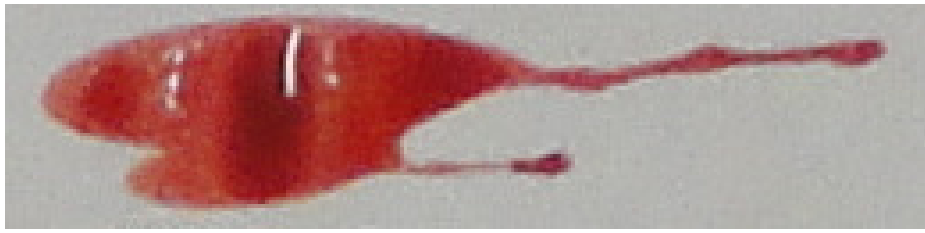


Figure 1. A stain of real blood consisting of two joined splatters, both with comet tails. Also visible camera flash reflection

optimization. Third approach is edge analysis [2, 5, 6, 7, 8], to which the algorithm presented herein belongs.

The edge-analysing designs typically implement following steps: region of interest (ROI) extraction, arc detection and joining, ellipse fitting, result improvement/filtering. For instance, the paper [5] proposes such algorithm, which extracts the ROI with the use of gradient. Then with the use of Gaussian Mixture Model the arcs are detected, then the Bayesian Ying-Yang Harmony learning algorithm supervises arc joining, ellipse fitting and result improvement. Unfortunately, the paper skips a running time discussion. It appears upon the description that it is at least $O(n \log n)$, with a large linear coefficient too. Another paper [1] proposes an ellipse fitting algorithm with great capability of detecting intersecting ellipses. The paper however skips the ROI extraction aspect, presenting only the edge analysis topic. Aside the common elements, this algorithm performs its specific Ellipse Refinement operations, which repeat certain steps until the best result is found. While offering a very high quality of the result, the running time of this algorithm is not mention but again appears too high.

The algorithm presented in this paper is similar to the algorithms such as [1, 5], but we managed to create a solution which performs the above mentioned steps in much simpler way, which results in a smaller period of running time, practically around $\Theta(n)$ with just one exception (the surface filter, Sections 4, item 4), and still maintaining the quality on the acceptable level.

2 BINARISATION

Binarisation is the process of highlighting the regions of interest, which in case of this edge-analysing algorithm would be the outlines of blood stains. There are two well known approaches: a gradient-based and a direct classification. The gradient-based method is very common in edge detection. It exploits the fact that an edge is a boundary between different colors or shades, therefore difference in pixel values will be the greatest at the edge. We have tested this method for blood stains and found it working, yet not satisfactorily. The differences among the obtained gradient values resulted in incomplete, torn outline, which would complicate the further processing.

Some of the effort of the algorithms [1, 5] is to combat the problem of gradient-based binarisation.

Because the blood stains are all in a narrow and very well specified range of color, we have discovered that direct classification works very well in this application. The direct classification method works by inspecting each pixel whether it depicts blood. In order to execute this method, the input image is converted into HSV (hue, saturation, value) color space, in which the possible colors of blood can be represented as one continous subset of each component. The decision process for each pixel is simply to check whether its HSV components are within the expected ranges.

Obviously there will be situations where the algorithm should look for stains of color outside the predefined normal blood range. For instance, with time, a blood stain may fade even to invisibility. The forensic technicians reveal such ones with luminol, after which the blood stain will be of blue color. Also liquids other than blood may be of interest. To answer these needs, our application provides a "color picker" tool, with which a user should point at least two possibly different pixels within the ROI in order to establish custom ranges for classification (an appropriate margin will be added automatically). This color picker can also be used to narrow down the ranges for normal blood stains. Note that whether the liquid would be arterial blood, venous blood, luminol treated, or perhaps coffee, all the stains in one picture would be of the same color, therefore color picking of one representative stain should result in all of them being properly detected.

The result of this process are white stains on the black background. The outline of a stain may now be easily extracted with any well known algorithm. Note that in this approach the result is always a closed loop outline without any forementioned problems related to the gradient-based method. The extracted outline is stored not as a 2D image, but as a one dimensional list of coordinates which preserves the neighborhood of edge pixels.

3 OUTLINE ANALYSIS

The purpose of this stage is to obtain out of the outline a vector ellipse described by five parameters (x, y, a, b, ϕ) where x, y are the center point coordinates, a, b are the axes lenghts and ϕ is the direction angle. The ellipse fitting algoritm will produce a worthy result only if the fitting is perfomed on pixels which belong to an elliptic curve. Figure 3 presents an outline obtained from the blood stain image in Figure 1 and it is full of unwanted features. The processing discussed in this section filters the outline to obtain only elliptic arcs and arranges them in sets representing separate ellipses if so necessary.

3.1 Convexity and Concavity Detection

In this step it is decided whether a given pixel belongs to the convex or concave fragment of the outline. First in order to do so, each pixel is described by the parameter

value of turn which marks how much the curve turns in the proximity of this pixel and in which direction. The value of turn for a given pixel B is calculated based on two auxiliary pixels A and C which are *step* away in the pixel neighbourhood sense. The *step* should be high enough to ignore noise and jagginess occuring due to binarsation of blurred edge, yet it must be low enough not to overlook the concavity we are trying to find. The best value is just above the point where the average noise no longer causes false detection. We have tried different functions calculating the *step* value from the pixel count in the outline, among which constant and linear functions, to establish that square-root-like curve works the best. On our test set, the *step* = 30 works optimally for outlines larger that 200 pixels. For the smaller ones it needs to be firmly reduced, but for the greater ones it does not need to grow equally fast, hence the use of square root curve.

The formula (1) defines the *value of turn* (B_α) as a measurement of how much $\vec{BC}$ bends from $\vec{AB}$ (Figure 2). It is 0.0 when ABC are colinear, and rises proportionally. After the (1) calculation, a plus or minus sign is assigned to specify whether it turns left or right. For this reason, the entire outline must be traversed in one direction, because ABC produces the same value but opposite sign than CBA.

$$B_\alpha = acos\left(\frac{\vec{AB}\cdot\vec{BC}}{|AB||BC|}\right). \tag{1}$$

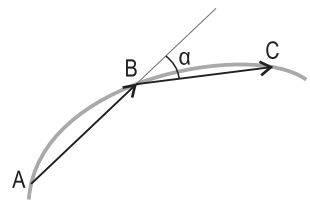


Figure 2. The method of measuring the value of turn

3.2 Outline Segmentation

In a closed convex outline every pixel's *value of turn* is of the same sign, provided the outline was traversed in one direction. Consequently, any concave feature will manifest itself as the opposite sign. In order to determine which sign represents the convexity, one must only check which sign is more frequent. In this step the convex segments are being identified on that basis (Figure 3). It is important to note, that we allow to be classified as convex the pixels whose value of turn is slightly of the opposite sign, like 0.001 radian. This small overlap neutralizes the effect of random classification of straight segments.

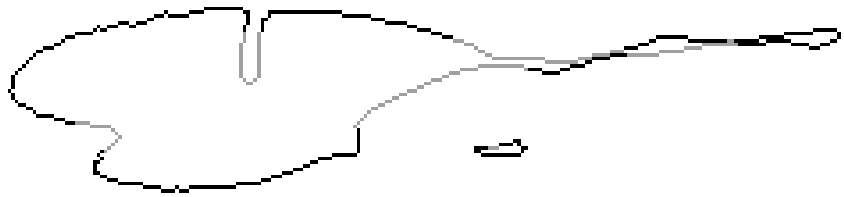


Figure 3. The outline of stain from Figure 1 with convex parts (black) identified

3.3 Segment Joining

Each segment obtained in the previous step is a convex fragment of the investigated stain's edge. If fed into the fitting algorithm right now, it would result in a separate ellipse fit for each such segment (Figure 4), which is a wrong result. The segments must now be organized into sets such that each one contains fragments of only one ellipse.

Figure 4. Useless fitting of unjoined segments

Only the neighbouring segments may be joined by their appropriate ends, thus if there are N segments out of the outline, there are exactly N joining decisions to be made. The general idea of the decision making process devised by the authors is depicted in the Figures 5 and 6. The thick gray lines are the convex segments, the thinner one is the earlier detected concave impurity. If the tangent versors are facing the same direction nearby the corresponding ends of the segments, we assume that the segments are parts of the same curve and therefore should be joined (Figure 5). Otherwise, if the versors are facing essentially different direction, like around V-shaped junction of two splatters, then we assume these are separate curves (Figure 6). The Euclidean distance limit also must be taken into consideration. Note Figure 6: the versors far enough from the V-shaped junction are facing the same direction. Without the distance limit this would be mistaken as an instance of Figure 5 case.

In our implementation, the tangent versor in each pixel is represented as a single value named *azimuth*. It is an angle between the tangent versor and one arbitrarily

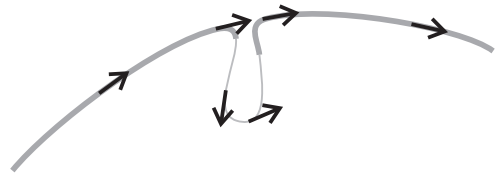


Figure 5. Tangent versors around an impurity in the stain's edge

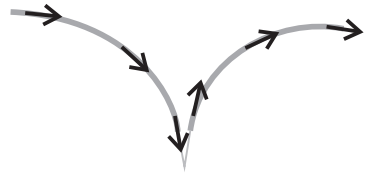


Figure 6. Tangent versors around a two splatters' junction point

chosen straight line. We are using the already obtained *value of turn* information to calculate the *azimuth*. If for the i^{th} pixel in the whole outline we denote i_α for the value of turn and i_β for the azimuth, then we calculate it as:

$$i_\beta = \frac{1}{step} \sum_{0}^{i-1} i_\alpha. \tag{2}$$

This way, the tangent versor of the 0^{th} pixel becomes the arbitrally chosen straight line. This method turns out to be very accurate – the azimuth value calculated again in the 0^{th} pixel after passing the entire outline misses 2π by less than 10^{-8} radians, up to 10^{-13} in the best noted cases.

In the decision process of joining for given two neighouring segments, each pair of pixels, one taken from the first segment, the other from the second, is inspected. This is the only $O(n^2)$ operation in this part of the algorithm, however, with the Euclidean distance limit mentioned earlier, which should be set between 0.25 and 0.5 of *step*, only very limited fraction of pixels is inspected on the corresponding ends of the segments. Considering that the number of segments is also very limited, the resulting running time impact is negligible.

For each inspected pair of pixels the absolute difference of *azimuths* is considered. The segments will be joined if the difference is below the limit which differentiates between cases of Figure 5 and Figure 6. Additionally, to help with more accurate fitting, the pixels from that pair, which has the smallest difference in azimuth, become the new endings of their respective segments.

Segments getting joined means that they are placed in one set as still separate segments. It does not mean that the gap between their respective ends is filled with artificial pixels. The ellipse fitting algorithm does not require that the input constitutes one connected component, and also there might be certain legal objections

on processing the evidence by the use of assumptions. The proposed system, as it is now, operates only on the real data.

3.4 Ellipse Fitting

The sets of segments are now fed into the ellipse fitting algorithm. The results are vector ellipses. Figure 7 shows the result in regard to the input segments, whereas Figure 8 shows the result superimposed on the input picture. More about the ellipse fitting algorithm is presented in Appendix A.

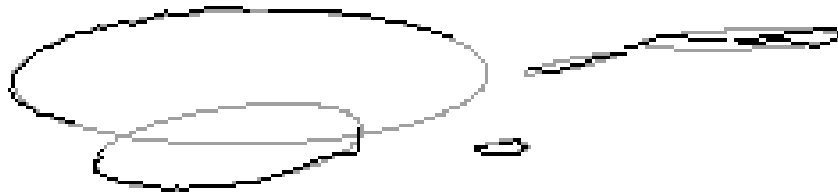


Figure 7. Remaining fragments of outline (black) being the input to the fitting algorithm and vector ellipses out of them (gray). This result is correct.

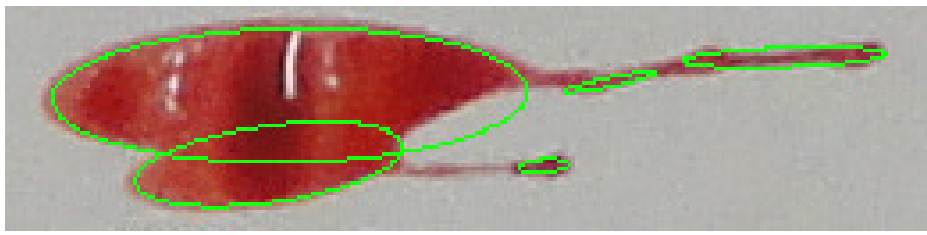


Figure 8. Vector ellipses from Figure 7 imposed on the original picture before filtering. Note the largest ellipse exceeds the back part of the splatter. Just like the "comet tail", this back part is deformed by viscosity phenomenons. What really matters is how well the vectorized ellipse follows the front part of the splatter and in the shown case it is a successfull result.

4 RESULT FILTERING

The results from previous stage are vector ellipses produced from convex sets of pixels. However, not all of them represent a worthy result. One problem, as visible in Figure 8, is that additional fittings happened on convex parts of the tails. Another one are incorrect fittings whenever the input segment was too short to produce a fitting correctly covering the edge of a stain. We employ a system of four consecutive filters:

1. The thickness filter removes all results which smaller axis is below 6 pixels. Aside the tiny ellipses which definitely are not a worthy results, occasionally there happen to be results of the fitting algorithm's instability on near-straight segments: grotesque ellipses of 0-pixel width and height exceeding the picture's dimensions.
2. The bounding-box filter compares the bounding boxes of an ellipse and the outline from which it was produced. If the ellipse protrudes over a specified margin, it is deleted, because ideally the ellipse should be exactly over the outline.
3. The area filter calculates from the bounding box the area covered by the ellipse. For all ellipses out of a single outline, removed are the ones whose area is less than 25 % of the largest one of them.
4. The surface filter inspects the surface encircled by an ellipse on the binarized image and calculates the percentage of the "white" area, effectively answering to what extent the ellipse encircles a bloody surface. If it is below 90 %, the ellipse is removed.

The filters 1 and 3 deal with the ellipses in the tail. The filters 2 and 4 deal with the fittings which do not encircle a stain correctly. In case of the filter 4 also correct fittings on damaged stains are removed, because for a court evidence only these matter which are being an unquestionable evidence and it has been assumed that 90 % of undamaged surface be the threshold.

As for running time, the filters 1, 2 and 3 require only one conditional check each, also bounding box and area calculation is not exhausting, therefore their time impact is negligible. On the other hand, the filter 4 needs to inspect every single pixel in the encircled area, therefore its running time is like $O(n^2)$. In the aim of speeding it up we had been testing solutions where only a fraction of all pixels is visited, chosen by either pattern or Monte Carlo method. However, as the running time tests have shown (Section 5.2), the solution meets the assumed time requirements with the filter visiting all the pixels. Note that, if there were no filter 2, the filter 4 would very well cover its functionality, therefore the filter 2 works as a preprocessor which speeds up the filter 4.

5 PERFORMANCE TESTING

The presented algorithm has been tested in two separate experiments, one using real, the other artificial images. The first one used 45 pictures of real blood stains provided by a forensic laboratory of the police. The purpose of this experiment is to test the quality of produced results on this kind of pictures for which it has been designed. Because of the limitations of this experiment, which are mentioned below, another experiment with artificial images has been performed to capture the algorithm performance in more quantitative terms.

5.1 Real Images

The basic problem with images depicting real blood stains is that there is no *a priori* information of what stains are there on the picture and where, therefore there is no straighforward way to tell which are the correct results. The algorithm has been designed to replace a human job, therefore we worked out the problem by performing manual encirclings, just like forensic technicians do, and then compared the results with the output of the algorithm. Whenever a detected ellipse matches the manual one within visually acceptable tollerance, we record a success event (S). If a detected ellipse intersects a manual one, yet its properties exceed the tollerance, then we record a mistake (M). If there is a manual ellipse and no detected ellipse, we call it underfit (U) and the opposite case we call overfit (O). By treating manual ellipses as one population and the detected ones as the other, we are able to calculate Dice coefficient out of the aforementioned sets (# denotes set's cardinality):

$$Dice = \frac{2 \cdot \#S}{\#(S \cup M \cup U) + \#(S \cup M \cup O)}. \tag{3}$$

One important matter discovered in this experiment is that not all provided pictures were worthy as an input. Good pictures are these which depict the stains in a proper resolution – stain's diameter of 50 pixels would be a desirable value, with no upper limit. In the bad pictures the blood stains are tiny, usually around the established lower threshold of 6 pixel diameter. With such low resolution the signal-to-noise ratio is low and so is the numerical stability, in effect giving poor results. Apparently the bad pictures have been taken only to provide a photographic overview of the scene, whereas the good one were taken with the blood tracing in mind. For the bad ones even manual encircling is not a sound option as there are uncertainties in human understanding of the picture.

On the good pictures, 154 ellipses were detected manually, the algorithm detected 222, getting Dice coefficient of 69.4 %. Expressing the classes as a fraction of the manual result we get: success $S = 83.9\,\%$, mistakes $M = 5.8\,\%$, underfit $U = 10.3\,\%$ (sums up to 100 % of manual fittings), and overfit $O = 51.83\,\%$. The best case for one file was: Dice 78 %, $S = 95.5\%$, $M = 3.0\,\%$, $U = 1.5\,\%$ and $O = 46.3\,\%$. The numbers of success, mistakes and underfit show that the algorithm has a desirable tendency to fit correctly or does not fit at all. On the other hand the relatively high number of overfits show that it also finds irrelevant objects. However, for a human operator it is much easier job to select and delete such overfits rather than perform the encirclings. An auxiliary experiment has shown that raising the size threshold from 6 to 12 pixel can reduce the number of overfits by the factor of 3, however also affecting other classes by a lesser degree.

In the case of bad pictures, there results are even incomparable between the files, as there are controversies in human interpretation of the picture. In general, the results are worse, as for instance in one file it was Dice 32.8 %, $S = 40.7\,\%$, mistakes not counted – too small stains, underfit $U = 59.3\,\%$ and overfit $O = 107.4\,\%$. In

another one it was: Dice 46.8 %, $S = 66.7\,\%$, $M = 23.3\,\%$, $U = 10\,\%$, $O = 95\,\%$. The increase in mistakes and overfit in comparison to good pictures is directly linked to the problems with numerical stability. Note that the bad file is only, if the picture contains only the too small stains. If a picture is mixture of too small and appropriate stains, then we can skip the tiny ones and treat the larger ones as the good input.

5.2 Running Time

The algorithm has been designed with the running time in concern, because it is supposed to work as a tool in an interactive process, therefore it should not overuse the operator's patience. Note that the parts of the algorithm dealing with the largest input are all $\Theta(n)$, and whenever there is a part of $O(n^2)$ (Sections 3.3, 4 item 4), the "n" is significantly smaller. It has been assumed for practical reason, that 10 seconds is target running time for an average file.

Our time testing setup consists of a laptop of the year 2013, Intel Core i5-3340M 2.70 GHz, 16 GB DDR3-1600 RAM, OS Windows 7 Professional. The application was compiled under Visual Studio 2010 in the Release mode, with the use of OpenCV library's functions (*findContours, fitEllipse*) and structures (*Mat*) as well as STL containers (*vector, list, deque, set*). The code was executed in one thread – no pararell processing.

The test consisted of the mentioned 45 pictures of real blood stains, each having between 12 and 18 million pixels. The average running time was 2.464 seconds with standard deviation of 0.431 sec., giving an average throughput of 7.16 million pixels per second (standard deviation 0.82 Mpix/sec.). The running time is mostly affected by the number of detected ellipses: correlation coefficient of 76 %. The correlation between running time and amount of bloody surface is 57 % and between running time and the number of pixels in the picture is the least of them of 54 %. Moreover, the correlation between number of ellipses and amount of bloody surface is only 44 %. As expected, the number of ellipses affects the throughput with correlation coefficient of $-80\,\%$. The target time of 10 seconds should be attainable even on older computers.

5.3 Artificial Images

The artificial image consists of a painted ellipse. Because we know exactly what were the parameters for the painting routine, we can precisely measure how well the detection results match. In this test we skip the binarisation step (the painted ellipse would either binarize fully or not at all), as well as we skip the filters, to measure just the performance of the edge-analysis, joining and fitting described in Section 3.

The most interesting question is how the algorithm performs on the damaged outline, just as it may occur with real stains. For this reason we introduced a damage mechanism – the surface of the ellipse is punctured randomly with artifacts. We

use two types of artifacts: the fine one is a 1-pixel square which imitates "pepper and salt" noise such as from high ISO sensitivity, bad pixels on camera sensor and physical impurites of a size of a grain of sand. The coarse one is 3x3 square to imitate impurities like hairs, fibers, tangles of dust, air bubbles, but also some camera flash reflections. The parameter damage factor D represents the percentage of the ellipse's surface which is lost due to the applied damage. Obviously, the isolated artifacts inside the ellipse have no contribution to the test, as the algorithm works solely on the outline, but those which coincide with the edge evidently produce an effect. Because of the stochastic nature of the damage process, each test was repeated 100 times and the average was taken to eliminate coincidence.

In each test an ellipse of sizes 180×300 (Large), 100×150 (Medium) and 40×70 (Small) was painted as if it came out of binarisation, then damaged in either Fine or Coarse process (producing 6 cases: LF, MF, SF, LC, MC, SC) to the assumed level of $D = [0.0; 0.55]$, step 0.05. Such setup resulted in 7 200 individual test instances. The parameters of the painted ellipse (x, y, a, b, ϕ) and the detected one (x', y', a', b', ϕ') were used to calculate the following scores of accuracy A:

$$A_\phi = 100\,\% - \frac{|\phi - \phi'|}{\pi}, \tag{4}$$

$$A_{ab} = 100\,\% - \frac{|\frac{a}{b} - \frac{a'}{b'}|}{\frac{a}{b}}, \tag{5}$$

$$A_{xy} = 100\,\% - \frac{\sqrt{(x - x')^2 + (y - y')^2}}{\sqrt{w^2 + h^2}}. \tag{6}$$

In the angular accuracy A_ϕ formula (4), π is the range of the returned value by the fitting algorithm. In the center point accuracy A_{xy} formula (6), w, h represent the dimensions of a bounding rectangle of the original ellipse.

As seen in Figures 9, 10, 11, the algorithm works reliably up to $D = 30\,\%$. Keeping in mind, that the stains with surface damage above 10 % would be discarded by the surface filter anyway, this leads to the conclusion that the devised algorithm is well capable of performing the entrusted task.

6 CONCLUSION

The blood drop trajectory tracing functionality of the forensic modelling software requires the blood stains to be vectorised in the form of ellipses. The solution presented herein provides automation of this process, which otherwise is done manually. The authors' experience in manual encircling for the need of the Section 5.1 test has shown that manual encircling is a highly exhausting task. It may take from 5 to 20 minutes to perform manual fitting on one picture file, however the effort required to establish carefully all the ellipse's parameters with 1 pixel precision, and so for each one ellipse individually, makes the operators wearier with each following

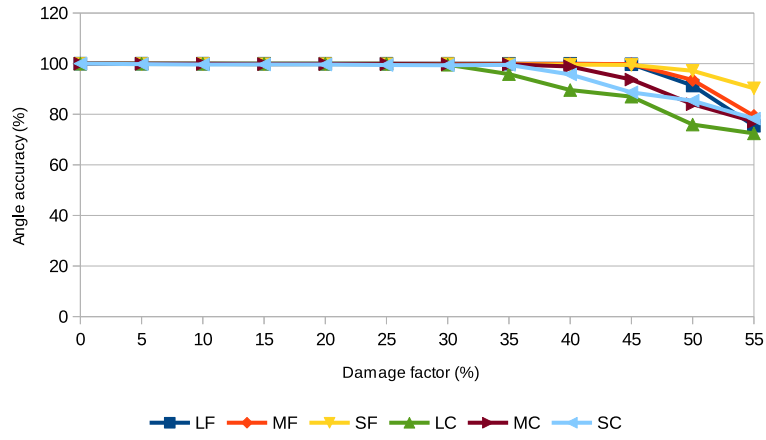


Figure 9. Angular accuracy

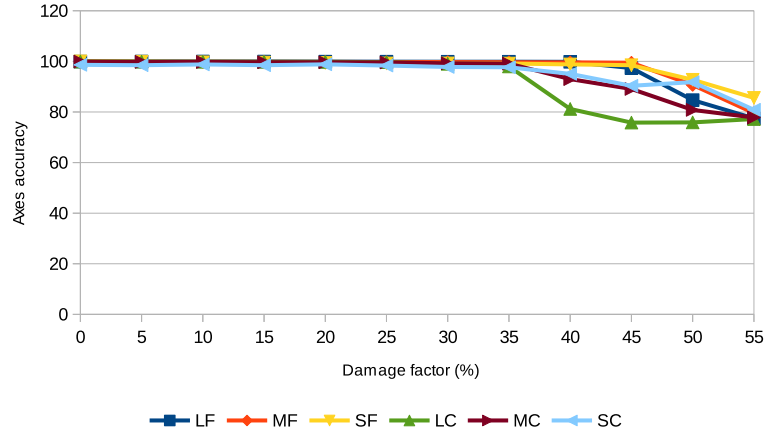


Figure 10. Axes accuracy

file, tempting them to start skipping certain stains for their comfort. Considering that for forensic technicians the manual fitting is not the essence of their job, rather a small but laborious task in the chain of many other, the automatic detection is going to spare technicians' energy for tasks where human input is more worthy.

The design of this solution took into account the qualitative requirements of what possibly may be a court evidence, so the algorithm tries to emphasize the results on the most trustworthy appearing input. The obtained test results show that the quality of the produced result meets the requirements of the field of application so does the running time.

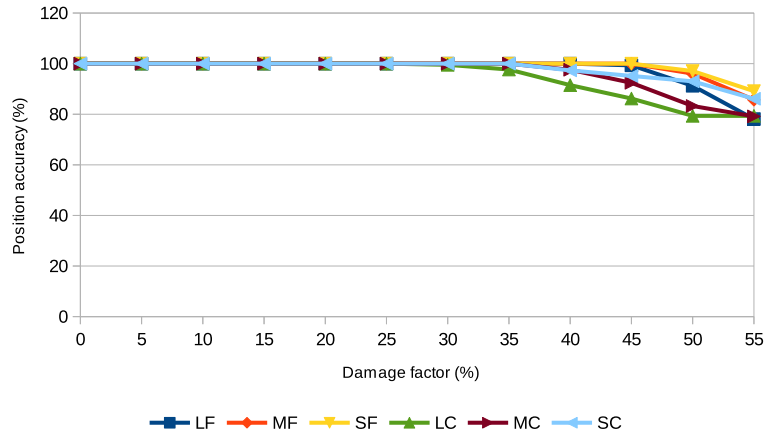


Figure 11. Center point accuracy

Acknowledgements

This work has been financially supported by the National Centre for Research and Development (`http://www.ncbr.gov.pl/`) as a part of the project "Reconstruction of the Course of Events Based on Bloodstain Pattern Analysis" (DOBR/0006/R/ID1/2012/03) realized within Defence and Security Programme (2012–2015).

Thanks to professor Zbisław Tabor, Ph.D. of Cracow University of Technology for critical feedback during the development process.

Thanks to CLKP, the Central Forensic Laboratory of the Police (`http://clk.policja.pl/`) for supplying the pictures of real blood stains.

A ELLIPSE DETECTION BY CONIC FITTING

The paper [4] investigates six approaches to the ellipse detection, four of which are of the algebraic distance type with the running time of $26n$, whereas the two other have $50n$ and $1\,300n$. The algebraic distance is the fastest approach and therefore it is widely used for the task, as in the OpenCV's function *fitEllipse* used in our implementation.

With the help of the conic section equation:

$$v_5x^2 + v_4xy + v_3y^2 + v_2x + v_1y + v_0 = 0 \tag{7}$$

the algebraic distance seeks such $v = [v_5, v_4, v_3, v_2, v_1, v_0]^T$ which satisfies the above equation with the smallest sum of errors for the given input set $S = \{(x, y)\}$. The four algebraic distance algorithms described in [4] differ in the way how it is achieved.

One way is to create the *design matrix* D of $n \times 6$ (n is the count of S) such that $D \cdot v$ implements the Equation (7). Now, the sought result can be found by solving an eigensystem

$$D^T Dv - \lambda v = 0. \tag{8}$$

The sought value v will be the eigenvector corresponding to the smallest eigenvalue λ. In the final step, the obtained v is transformed into the commonly used (x, y, a, b, ϕ).

REFERENCES

[1] Bai, X.—Sun, C.—Zhou, F.: Splitting Touching Cells Based on Concave Points and Ellipse Fitting. Pattern Recognition, Vol. 42, 2009, No. 11, pp. 2434–2446, doi: 10.1016/j.patcog.2009.04.003.

[2] Chia, A. Y.-S.—Rahardja, S.—Rajan, D.—Leung, M. K.: A Split and Merge Based Ellipse Detector with Self-Correcting Capability. IEEE Transactions on Image Processing, Vol. 20, 2011, No. 7, pp. 1991–2006.

[3] Duda, R. O.—Hart, P. E.: Use of the Hough Transformation to Detect Lines and Curves in Pictures. Communications of the ACM, Vol. 15, 1972, No. 1, pp. 11–15, doi: 10.1145/361237.361242.

[4] Fitzgibbon, A. W.—Fisher, R. B.: A Buyer's Guide to Conic Fitting. Proceedings 6$^{\text{th}}$ British Machine Vision Conference (BMVC '95), Birmingham, 1995, Part 2, pp. 513–522, doi: 10.5244/C.9.51.

[5] Huang, L.—Ma, J.: A Probabilistic Mixture Approach to Automatic Ellipse Detection. Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV 2013), 2013, pp. 573–580.

[6] Kim, E.—Haseyama, M.—Kitajima, H.: Fast and Robust Ellipse Extraction from Complicated Images. Proceedings of IEEE International Conference on Information Thechnology and Applications (ICITA 2002), 2002, pp. 357–362.

[7] Mai, F.—Hung, Y. S.—Zhong, H.—Sze, W. F.: A Hierarchical Approach for Fast and Robust Ellipse Extraction. Pattern Recognition, Vol. 41, 2008, No. 8, pp. 2512–2524, doi: 10.1016/j.patcog.2008.01.027.

[8] Prasad, D. K.—Leung, M. K. H.—Quek, C.: ElliFit: An Unconstrained, Non-Iterative, Least Squares Based Geometric Ellipse Fitting Method. Pattern Recognition, Vol. 46, 2013, No. 5, pp. 1449–1465, doi: 10.1016/j.patcog.2012.11.007.

[9] Xu, L.—Oja, E.—Kultanen, P.: A New Curve Detection Method: Randomized Hough Transform (RHT). Pattern Recognition Letters, Vol. 11, 1990, No. 5, pp. 331–338, doi: 10.1016/0167-8655(90)90042-Z.

[10] Yao, J.—Kharma, N.—Grogono, P.: A Multi-Population Generic Algorithm for Robust and Fast Ellipse Detection. Pattern Analysis and Application, Vol. 8, 2005, No. 1-2, pp. 149–162.

Tomasz Wójtowicz received his M.Sc. in computer science from Jagiellonian University and is a Ph.D. candidate therein. His research interests include pattern recognition, computer vision, artificial inteligence and also biocybernetics and evolutionary biology.

Dariusz Bułka received his M.Sc. in computer science from AGH University of Science and Technology. He is the Chief Engineer in CYBID Ltd., a company developing specialized software and systems for forensic analysis, simulations, data gathering and crime/accident event reconstruction.

Computing and Informatics, Vol. 37, 2018, 229–243, doi: 10.4149/cai_2018_1_229

PARALLELIZATION OF ANT SYSTEM FOR GPU UNDER THE PRAM MODEL

Andrej Brodnik

Department of Information Science and Technology
University of Primorska
Glagoljaška 8
6000 Koper, Slovenia
&
Faculty of Computer and Information Science
University of Ljubljana
Tržaška cesta 25
1000 Ljubljana, Slovenia
e-mail: andrej.brodnik@upr.si

Marko Grgurovič

Department of Information Science and Technology
University of Primorska
Glagoljaška 8
6000 Koper, Slovenia
e-mail: marko.grgurovic@student.upr.si

Abstract. We study the parallelized ant system algorithm solving the traveling salesman problem on n cities. First, following the series of recent results for the graphics processing unit, we show that they translate to the PRAM (parallel random access machine) model. In addition, we develop a novel pheromone matrix update method under the PRAM CREW (concurrent-read exclusive-write) model and translate it to the graphics processing unit without atomic instructions. As a consequence, we give new asymptotic bounds for the parallel ant system, resulting in step complexities $O(n \lg \lg n)$ on CRCW (concurrent-read concurrent-write) and $O(n \lg n)$ on CREW variants of PRAM using n^2 processors in both cases. Finally, we present an experimental comparison with the currently known pheromone

matrix update methods on the graphics processing unit and obtain encouraging results.

Keywords: Parallel random access machine, graphics processing unit, ant system, metaheuristics, traveling salesman problem, combinatorial optimization

Mathematics Subject Classification 2010: 68-W10

1 INTRODUCTION

In this paper, we study the parallel variants of the Ant System (AS) algorithm, which is part of the ant colony optimization (ACO) family of metaheuristics. The ACO family of algorithms simulate the behavior of real ants which find paths using pheromone trails. A number of variations on the basic idea exist, such as the Ant System [7], Ant Colony System [6], the $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System [16] and many others. In this paper we focus on the canonical Ant System algorithm, which can be adapted to solve a variety of combinatorial optimization problems such as vehicle routing [2], quadratic assignment [13], subset problems [11] and others. In this paper, we will limit ourselves to the traveling salesman problem (TSP).

The adoption of the graphics processing unit (GPU) as a computing platform in recent years has triggered a wave of papers discussing the parallelization of known algorithms. Recent papers [4, 12, 17] have focused on providing a parallel version of Ant System for the GPU. In this paper, we show that these algorithms are more general and can be studied in absence of GPU specifics. In line with this observation, we suggest a move towards more well-understood theoretical models such as the parallel random access machine (PRAM). This greatly facilitates asymptotic analysis and subsequently allows one to see where the algorithms could be improved.

The main goal of this paper is to investigate efficient AS algorithms for variants of the PRAM model of computation and to identify how these might be useful in practice. We break down the AS algorithm into two separate phases: Tour Construction and Pheromone Update. Then we show that the existing GPU algorithms for AS can be translated to the PRAM model, which permits to perform asymptotic analysis. While Tour Construction remains efficient even on PRAM, we identify bottlenecks in the Pheromone Update phase, which are caused by reliance on atomic instructions that are not readily available on most variants of PRAM (or older GPUs). We overcome this with a novel Pheromone Update algorithm that does not require such instructions. Finally, we show that these results are relevant in practice when atomic instructions are not available. We do this by implementing the resulting Pheromone Update algorithm on the GPU and we obtain significantly better results in case of no atomic instructions.

The paper is structured as follows. In Section 2 we briefly introduce the PRAM model, the traveling salesman problem, and the Ant System algorithm in its se-

quential form. In Section 3 we provide PRAM implementations of the Ant System algorithm and show how to improve them, and finally we provide results of empirical tests on the GPU. In Section 4 we provide conclusions and suggestions for future work.

2 BACKGROUND

2.1 Parallel Random Access Machine

The PRAM model is a variant of the random access machine (RAM) adapted to parallel execution. We denote the number of processors by p. In this paper, we deal with two types of synchronous PRAM: concurrent-read exclusive-write (CREW) and concurrent-read concurrent-write (CRCW). The CREW variant assumes that each memory location is tied to a specific processor, and only that processor can write to it. However, any processor can read from any memory location. In contrast, the CRCW variant has no such restriction. Since under CRCW all processors can write to the same location, it is typical to parametrize the CRCW variant by how the competing writes are handled. In this paper we consider two standard ways of doing that:

- COMMON: All processors must write the same value.
- COMBINING: All values being concurrently written are combined using some operator (e.g. addition, maximum, etc.).

In this paper we will focus on CREW, CRCW and COMBINING CRCW algorithms, where by CRCW we mean algorithms that run under the COMMON variant. An important parallel operation which we will make extensive use of is finding the largest element in an array of n elements. Throughout the paper we will use $S(n)$ to denote the step complexity of a parallel algorithm, i.e. the number of steps executed. The work complexity of an algorithm, denoted by $W(n)$, corresponds to the total number of operations executed (over all processors). It is important to note that finding the maximum among n numbers can be performed in $S(n) = O(\lg \lg n)$ time under CRCW [15] with $p = n$. However, it is only possible in $S(n) = O(\lg n)$ time under CREW with the same number of processors. The work complexity is $W(n) = O(n)$ in both cases. Under COMBINING CRCW, finding the maximum can be performed in $S(n) = O(1)$ and $W(n) = O(n)$ by making use of the combining mechanism in a trivial way (i.e. setting it to be the maximum operation).

2.2 The Traveling Salesman Problem

In the traveling salesman problem (TSP), we are given a complete, directed graph $G = (V, E)$, with V and E being the sets of vertices and edges, respectively. We are also given a function $\ell : E \to \mathbb{R}^+$ which maps each edge to its length. To simplify notation, we define $n = |V|$. The task, then, is to produce a permutation Π of V

with the least cost. Let Π_k denote the vertex at position k in the permutation Π. The cost of a permutation Π of V is then defined as:

$$\ell(\Pi_n, \Pi_1) + \sum_{2 \le k \le n} \ell(\Pi_{k-1}, \Pi_k).$$

Observe, that even though our formulation requires a complete graph, sparse graphs can be handled by inserting the missing edges with length ∞. Undirected graphs can also be handled simply by creating two directed edges for each undirected edge, with equal lengths assigned to them.

2.3 Ant System for the TSP

As in the description of the TSP problem, we assume we are given a directed, complete graph $G = (V, E)$. We then define the heuristic matrix η and the pheromone matrix τ, both of dimensions $n \times n$. The heuristic matrix is constant throughout the algorithm and represents the quality of an edge (u, v). Formally, we choose $\eta_{u,v} = 1/\ell(u, v)$. The pheromone matrix changes throughout the execution of the algorithm. Two parameters α and β regulate the importance of pheromone and heuristic information, respectively.

Algorithm 1 Sequential Ant System

```
1: procedure AntSystem(α, β, ρ, totalIterations)
2:     Allocate matrices of size n × n: η, τ, chance, π, tabu
3:     Allocate vector of size n: score
4:     for iter := 1 to totalIterations do
5:         Initialize(α, β, τ, η, score, chance, π, tabu)
6:         TourConstr(η, score, chance, π, tabu)
7:         PheromoneUpdate(τ, ρ, score)
8:     end for
9: end procedure
```

Ants then build solutions according to:

$$p(v|p, S) = \frac{\tau_{p,v}^{\alpha} \cdot \eta_{p,v}^{\beta}}{\sum_{w \in N(p,S)} \tau_{p,w}^{\alpha} \cdot \eta_{p,w}^{\beta}} \tag{1}$$

where $p(v|p, S)$ is the probability of choosing vertex v when at position p and according to the current partial solution S. The feasible neighborhood of the current incomplete solution is defined by $N(p, S)$. Since the TSP does not permit returns to previously included vertices (except for the last vertex), those vertices have probability zero of being included. This is typically accomplished by having each ant keeping the track of a tabu list.

Algorithm 2 Sequential Initialize

```
1:  procedure INITIALIZE(α, β, τ, η, score, chance, π, tabu)
2:      Allocate vector of size n: sum
3:      for i := 1 to n do
4:          sum[i] := 0
5:      end for
6:      for i := 1 to n do
7:          for j := 1 to n do
8:              sum[i] := sum[i] + τ[i, j]^α · η[i, j]^β
9:          end for
10:     end for
11:     for i := 1 to n do
12:         for j := 1 to n do
13:             chance[i, j] := τ[i, j]^α · η[i, j]^β / sum[i]
14:             tabu[i, j] := 1
15:         end for
16:     end for
17:     for i := 1 to n do
18:         π[i, 1] := i
19:         score[i] := 0
20:         tabu[i, i] := 0
21:     end for
22: end procedure
```

Once solutions are constructed, they are evaluated to obtain their respective qualities, which in most cases is simply the inverse of the cycle length. Once evaluated, the qualities are used to update the pheromone matrix. First, each cell of the pheromone matrix is decreased by a constant factor (evaporation) and then increased according to the solution score (pheromone deposit). Let $f(S)$ denote the score of solution S and let Z be the set of all solutions produced by the ants, where each ant contributes a single solution. Then, the pheromone update stage is defined by:

$$\tau_{v,w} \leftarrow (1-\rho) \cdot \tau_{v,w} + \sum_{S \in Z | (v,w) \in S} f(S). \tag{2}$$

When considering AS for TSP, the recommended number of ants equals the number of vertices [8]. Thus hereof we always assume we have n ants, each starting its solution in a different vertex. The Ant System algorithm, as we have described it, can be formalized as Algorithm 1. An initialization stage (Algorithm 2) was added where certain bookkeeping tasks can be performed. The tour construction (Algorithm 3) and pheromone update (Algorithm 4) stages correspond to what we have described. In line 7 of Algorithm 3 we call the function *rand*(), which is supposed to return a random uniformly distributed real number in the range $(0, 1)$. This is the source of randomness in the algorithm, and allows it to implement the

Algorithm 3 Sequential Tour Construction

```
1:  procedure TourConstr(η, score, chance, π, tabu)
2:      for i := 1 to n do
3:          for k := 2 to n do
4:              v := 0
5:              c := −∞
6:              for j := 1 to n do
7:                  t := chance[π[i, k − 1], j] · rand() · tabu[i, j]
8:                  if t ≥ c then
9:                      c := t
10:                     v := j
11:                 end if
12:             end for
13:             π[i, k] := v
14:             tabu[i, π[i, k]] := 0
15:             score[i] := score[i] + η[π[i, k − 1], π[i, k]]
16:         end for
17:     end for
18:     for i := 1 to n do
19:         score[i] := score[i] + η[π[i, n], π[i, 1]]
20:     end for
21: end procedure
```

probabilistic selection according to Equation (1). The algorithm also uses a number of matrices, which play the following roles: *chance* stores the visit probability values (cf. (1)), π holds the solutions, *tabu* is used to prevent infeasible solutions. The vector *score* holds the computed score for each solution.

3 PARALLEL ANT SYSTEM

It is conceptually simpler to consider Ant System as a combination of two algorithms: tour construction and pheromone update (lines 6 and 7 in Algorithm 1, respectively). Attempts at parallel AS, e.g. [5, 3], are usually not very attractive for the PRAM model, since they either employ coarse parallelization or neglect certain parts of parallel AS, typically pheromone update. However, it turns out that parallel AS algorithms for the GPU model [12, 17, 4] translate almost without any effort to the PRAM model. Thus, we focus exclusively on the translation and improvement of the GPU algorithms. It is important to note that the unit of parallelism in the GPU is a thread while on a PRAM the unit of parallelism is a processor. However since the PRAM is a theoretical model, the actual meaning of processor in this context is abstract.

Algorithm 4 Sequential Pheromone Update

```
1: procedure PheromoneUpdate(τ, ρ, score)
2:     for i := 1 to n do
3:         for j := 1 to n do
4:             τ[i, j] := (1 − ρ) · τ[i, j]
5:         end for
6:     end for
7:     for i := 1 to n do
8:         for k := 2 to n do
9:             τ[π[i, k − 1], π[i, k]] := τ[π[i, k − 1], π[i, k]] + score[i]
10:        end for
11:        τ[π[i, n], π[i, 1]] := τ[π[i, n], π[i, 1]] + score[i]
12:    end for
13: end procedure
```

Due to the decomposition of AS into two algorithms (construction and update), the complexity of AS becomes the worst of the two. We will now explore strategies for each algorithm.

3.1 Tour Construction

The simplest method (cf. [12, 17]) delegates each ant to a unique processor. Now, since each ant stochastically considers each vertex n times (cf. (1)) and has $p = n$ processors, this amounts to step complexity $S(n) = O(n^2)$ and work complexity $W(n) = O(n^3)$.

A remarkable contribution of [4] is their strategy for parallel tour construction. Their tour construction method uses $p = n^2$ processors and associates each ant with n processors. When each ant can make use of n processors, it can effectively generate multiple random numbers in parallel. Then, the maximum operation is used to choose one among n neighbouring vertices, again in parallel. In total, n maximum operations are performed per ant. When translating this result to the PRAM model, the step complexity of the algorithm depends on the model of computation. In the case of CREW, the maximum can be found with a step complexity $S(n) = O(\lg n)$ and work complexity $W(n) = O(n)$. Since there are n maximum operations per ant this brings the step complexity to $S(n) = O(n \lg n)$. There are n ants in total, each performing n maximum operations, meaning the work complexity remains $W(n) = O(n^3)$. However, under CRCW, maximum can be performed in $S(n) = O(\lg \lg n)$ step complexity (see e.g. [15]), thus the step complexity of the algorithm becomes $S(n) = O(n \lg \lg n)$, with the work complexity remaining the same as in the CREW case. Under COMBINING CRCW, this is further reduced to $S(n) = O(n)$ by simply taking the combining operation to be maximum.

It is possible to further reduce the step complexity of the CRCW algorithm to $S(n) = O(n)$ using $p = n^3$ processors and a different method to find the maximum

which takes $S(n) = O(1)$: simply compare all pairs of elements in the array in parallel. However, we will restrict ourselves to $p = n^2$, since the large amount of additional processors required hardly justifies the $\lg \lg n$ gain.

3.2 Pheromone Update

Once tour is constructed, the pheromone update must be performed. In [12, 17] the latter is accomplished sequentially rather than in parallel, i.e., one processor performs the update in $S(n) = O(n^2)$ and $W(n) = O(n^2)$ while others are waiting. This method is appropriate if we use the first construction method, which also has a step complexity of $O(n^2)$, but it becomes a bottleneck if we choose the more parallel construction method of [4].

Two pheromone update methods can be found in [4]. The first is straightforward and is based on atomic instructions for addition (cf. the summation in Equation (2)). This method corresponds to the use of COMBINING CRCW with the combining operation set to addition. Thus, we already have one parallel method for pheromone update with $p = n$ and running with a step complexity of $S(n) = O(n)$ and a work complexity of $W(n) = O(n^2)$. If we allow $p = n^2$, then the update can be performed in $S(n) = O(1)$.

The second method of [4] which they refer to as "scatter to gather" is more computationally intensive, but does not use atomic instructions. In this case each cell of the pheromone matrix is represented by a distinct processor, so we require $p = n^2$. Each processor loops through all solutions, summing only the relevant qualities. Solutions are of size $O(n)$ and there are n solutions, meaning each processor performs $S(n) = O(n^2)$ operations. Since there are n^2 processors, this yields a $W(n) = O(n^4)$ work complexity. This method works under both CREW and CRCW models, but in terms of computational complexity, it is uninteresting. Better bounds are accomplished by performing pheromone update sequentially, i.e. by a single processor while others wait. Nonetheless, we mention this method because we will show how to improve its complexity.

3.3 Improvements

In this subsection, we propose a novel method for pheromone update, which improves the currently known bounds under the CREW and CRCW models. Tour construction in our algorithm is performed as in [4], which translates effortlessly to the PRAM. However, instead of using their "scatter to gather" pheromone update, we develop a new technique.

Theorem 1. Pheromone update using $p = n^2$ processors can be performed in $S(n) = O(n)$ and $W(n) = O(n^3)$ under a CREW PRAM.

Proof. Each ant already stores a list of n entries, which correspond to vertices in the order it visited them. In addition to this list, we require that each ant also

stores an array *edge* of length n, implicitly storing which edge was used to reach a particular vertex. For example, if the edge (u, v) was used to visit vertex v, then we would set $edge[v] := u$. During pheromone update, we can now check whether a given solution S contains the desired edge in constant time. Without this array, we would have to inspect every element of the solution, which would take $O(n)$ time. There are n solutions, so the step complexity of pheromone update becomes $S(n) = O(n)$ and the work complexity becomes $W(n) = O(n^3)$. □

The pseudocode for the parallel algorithm is shown in Algorithms 5, 6, 7 and 8. PRAM algorithms use a scalar processor identifier. To improve readability we use a two-dimensional processor identifier $(x, y) \in [n] \times [n]$, where $[n] = \{1, 2, \ldots, n\}$. Remember that each ant is using n processors, so the x component of the identifier denotes an ant and the y component denotes an ant's processor. The algorithm consists of an initialization phase, where we compute the probability (*chance* matrix) of choosing certain edges and reset structures after each iteration. We explicitly denote variables that are local to each processor by prefixing them with a *local* identifier in their initialization. All matrices in the algorithm are of size $n \times n$. The matrices η, τ, *chance*, π, *tabu* and vector *score* were already described in Section 2. Additional matrices exist for the parallel algorithm which have the following roles: R holds the results from parallel random number generation and *edge* is used as described in the proof of Theorem 1.

Theorem 2. Algorithm 5 executes on a CREW PRAM.

Proof. It is easy to see that writes to R (line 3 in Algorithm 7) and τ (lines 2 and 5 in Algorithm 8) preserve write exclusivity since only processor (x, y) writes to $R[x, y]$ and $\tau[x, y]$. We lump together the proof of write exclusivity for *chance* (line 8 in Algorithm 6), *tabu* (lines 9 and 13 in Algorithm 6 and line 7 in Algorithm 7), *score* (line 12 in Algorithm 6 and lines 8 and 13 in Algorithm 7) and *edge* (lines 6 and 12 in Algorithm 7). Observe that in each case the processor's y index is set to one. For *score*, which only has one dimension, this avoids conflicts. The rest are matrices and all writes from processor $(x, 1)$ are to cells (x, k) where $k \in [n]$, which does not lead to any conflicts. Note that the proof for the write exclusivity of π (line 11 in Algorithm 6 and line 4 in Algorithm 7) is the same. Naturally, we require that the parallel implementation of arg max observes the write exclusivity of π (which leads to different implementations on CREW and CRCW). □

Corollary 1. Algorithm 5 executes under a CRCW PRAM.

It is easy to see that the suggested pheromone update method can be sped up if more processors are provided. For example, given $p = n^3$ processors, each cell in the pheromone matrix can be represented by n processors, allowing pheromone summation to be performed using reduction. However, there seems little incentive to do so, since the complexity of parallel Ant System algorithm becomes dominated by the tour construction step.

Algorithm 5 Parallel Ant System

```
1: procedure PAntSystem(α, β, ρ, totalIterations)
2:     Allocate matrices of size n × n: R, η, τ, chance, π, tabu, edge
3:     Allocate vector of size n: score
4:     for i := 1 to totalIterations do
5:         for (x, y) ∈ [n] × [n] in parallel do
6:             PInitialize(x, y, α, β, τ, η, score, chance, π, tabu)
7:             PTourConstr(x, y, R, η, score, chance, π, tabu, edge)
8:             PPheromoneUpdate(x, y, τ, ρ, edge, score)
9:         end for
10:    end for
11: end procedure
```

Algorithm 6 Parallel Initialize

```
1: procedure PInitialize(x, y, α, β, τ, η, score, chance, π, tabu)
2:     if y = 1 then
3:         local float sum := 0
4:         for i := 1 to n do
5:             sum := sum + τ[x, i]^α · η[x, i]^β
6:         end for
7:         for i := 1 to n do
8:             chance[x, i] := τ[x, i]^α · η[x, i]^β / sum
9:             tabu[x, i] := 1
10:        end for
11:        π[x, 1] := x
12:        score[x] := 0
13:        tabu[x, x] := 0
14:    end if
15: end procedure
```

Table 1 summarizes complexity bounds derived from the previous work as well as new bounds resulting from the improvements presented in this paper. Since a single iteration of the parallel Ant System algorithm requires both tour construction and pheromone update, the bound becomes the worse of the two.

3.4 Empirical Comparison

We implemented different pheromone update methods on the GPU. We used Nvidia CUDA, which was also used in recent papers [4, 12, 17] studying the parallel GPU implementation of the Ant System algorithm. Compared to the GPU, the PRAM model is much simpler. While programs on the PRAM execute in SIMD (single instruction, multiple data) lock-step fashion, the GPU model of execution is the

Algorithm 7 Parallel Tour Construction

```
1: procedure PTourConstr(x, y, R, η, score, chance, π, tabu, edge)
2:     for k := 2 to n do
3:         R[x, y] := chance[π[x, k − 1], y] · rand() · tabu[x, y]
4:         Compute (arg max_{i∈{1...n}} R[x, i]) and store result in π[x, k]
5:         if y = 1 then
6:             edge[x, π[x, k]] := π[x, k − 1]
7:             tabu[x, π[x, k]] := 0
8:             score[x] := score[x] + η[π[x, k − 1], π[x, k]]
9:         end if
10:    end for
11:    if y = 1 then
12:        edge[x, π[x, 1]] := π[x, n]
13:        score[x] := score[x] + η[π[x, n], π[x, 1]]
14:    end if
15: end procedure
```

Algorithm 8 Parallel Pheromone Update

```
1: procedure PPheromoneUpdate(x, y, τ, ρ, edge, score)
2:     τ[x, y] := (1 − ρ) · τ[x, y]
3:     for k := 1 to n do
4:         if edge[k, y] = x then
5:             τ[x, y] := τ[x, y] + score[k]
6:         end if
7:     end for
8: end procedure
```

significantly more ambiguous SIMT (single instruction, multiple threads), where such lock-step guarantees are lost. Together with details like different levels of memory with different speeds and capacities, writing programs becomes a matter of mixing theoretical and practical considerations. With this paper we mainly focus on the theoretical aspects of such programs by studying them in the cleaner PRAM model, then transferring them over to the "messier" GPU.

The tests were run on an Nvidia GeForce GTX 560Ti using stock Nvidia frequencies. Test instances were taken from TSPLIB [14], which are standard test cases. We included some of the instances that have also been used by [4] to facilitate comparisons. We compared only the pheromone update stage, since our tour construction step is identical to the one presented in [4], thus we refer readers interested in comparisons between various tour construction methods or comparisons between the parallel and sequential code to that paper.

We tested three methods: atomic, scatter-gather and non-atomic fast. The atomic method updates the pheromone matrix using atomic instructions for addition. The scatter-gather method is the non-atomic method proposed by [4]. Finally,

Previous Work				
		CREW	**CRCW**	**CMB. CRCW**
Tour [4]	**S(n)**	$O(n \lg n)$	$O(n \lg \lg n)$	$O(n)$
	W(n)	$O(n^3)$	$O(n^3)$	$O(n^3)$
PH [4]	**S(n)**	$O(n^2)$	$O(n^2)$	$O(1)$
	W(n)	$O(n^2)$	$O(n^2)$	$O(n^2)$
Total	**S(n)**	$O(n^2)$	$O(n^2)$	$O(n)$
	W(n)	$O(n^3)$	$O(n^3)$	$O(n^3)$
This Paper				
		CREW	**CRCW**	
PH	**S(n)**	$O(n)$	$O(n)$	
	W(n)	$O(n^3)$	$O(n^3)$	
Total	**S(n)**	$O(n \lg n)$	$O(n \lg \lg n)$	
	W(n)	$O(n^3)$	$O(n^3)$	

Table 1. Previous and new bounds for the parallel Ant System, which is comprised of two sub-algorithms: tour construction and pheromone (PH) update. We denote the COMBINING CRCW model by CMB. CRCW. Step and work complexities are denoted by $S(n)$ and $W(n)$, respectively. All bounds assume n^2 processors.

	Method		
Instance	**Atomic** [4]	**Scatter-Gather** [4]	**Non-Atomic Fast**
att48	0.06	1.29	0.19
kroC100	0.11	17.35	0.51
a280	0.47	759.14	3.61
pcb442	0.82	4 681	11.5
d657	1.74	$22 \cdot 10^3$	34.7
pr1002	3.48	$118 \cdot 10^3$	114.8
pr2392	16.39	$3\,800 \cdot 10^3$	1 525.4

Table 2. Running time (milliseconds) of pheromone update methods on TSPLIB instances

the non-atomic fast method is the one suggested in this paper. We also remark that, in our case, the atomic update method made full use of $p = n^2$ threads, since we found its performance to be significantly better compared to $p = n$ threads, as used in [4]. The results are shown in Table 2 and are also shown as a plot in Figure 1.

It is reassuring to see that the theoretical improvements also translate into practice. While the atomic variant is significantly faster, many older GPUs still in use today do not have access to the appropriate atomic instructions. Thus, these results are practically relevant for GPU implementations if code is expected to work on all GPUs currently in use.

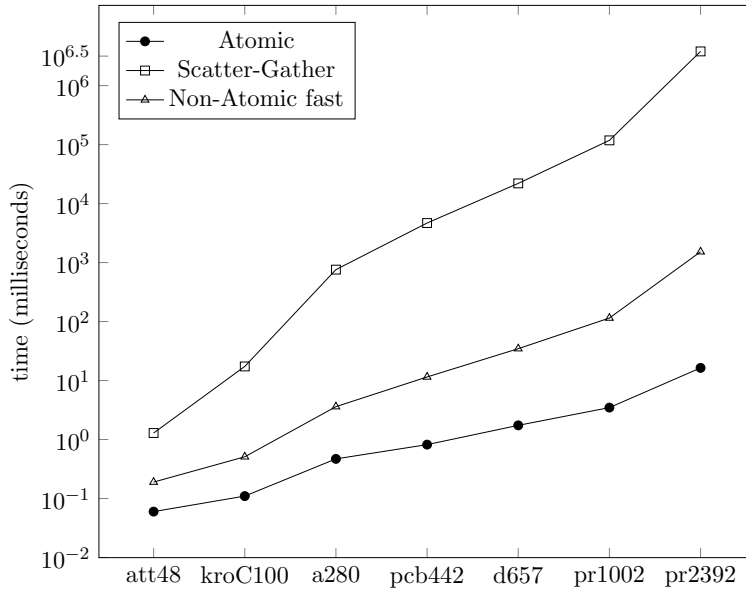


Figure 1. Plotted running times of pheromone update methods on TSPLIB instances

4 CONCLUSION

In this paper we have shown that recent parallel variants of the Ant System algorithm for the GPU systems can be easily modeled by the more general PRAM model. This makes them both simpler to understand and to analyze. The facilitation in a theoretical analysis allowed us to determine which parts of the algorithm needed improvement. It turned out that in two out of three variants of PRAM models studied, the parallel Ant System algorithm was dominated by the pheromone update. We proposed a new pheromone update method that improves the asymptotic bound of the parallel Ant System algorithm to such an extent, that the entire algorithm becomes dominated by the tour construction phase.

Future research directs us to study the possibility of application of the proposed pheromone update method to other algorithms in the ACO family. Moreover, optimization problems other than the TSP could be parallelized in a similar fashion. The algorithms could be studied under various other parallel computation models. Last but not least, we are also interested in other algorithms that could be more efficiently parallelized if they are split into two phases or more phases.

REFERENCES

[1] Bilchev, G.—Parmee, I. C.: The Ant Colony Metaphor for Searching Continuous Design Spaces. In: Fogarty, T. C. (Ed.): Evolutionary Computing (AISB EC 1995). Springer, Lecture Notes in Computer Science, Vol. 993, 1995, pp. 25–39.

[2] Bullnheimer, B.—Hartl, R. F.—Strauss, C.: An Improved Ant System Algorithm for the Vehicle Routing Problem. Annals of Operations Research, Vol. 89, 1999, pp. 319–328, doi: 10.1023/A:1018940026670.

[3] Bullnheimer, B.—Kotsis, G.—Strauss, C.: Parallelization Strategies for the Ant System. In: De Leone, R. et al. (Eds.): High Performance Algorithms and Software in Nonlinear Optimization. Springer, Boston, Applied Optimization, Vol. 24, 1998, pp. 87–100.

[4] Cecilia, J. M.—García, J. M.—Nisbet, A.—Amos, M.—Ujaldón, M.: Enhancing Data Parallelism for Ant Colony Optimization on GPUs. Journal of Parallel and Distributed Computing, Vol. 73, 2013, No. 1, pp. 42–51, doi: 10.1016/j.jpdc.2012.01.002.

[5] Delisle, P.— Krajecki, M.—Gravel, M.—Gagné, C.: Parallel Implementation of an Ant Colony Optimization Metaheuristic with OpenMP. Proceedings of the Third European Workshop on OpenMP, International Conference on Parallel Architectures and Compilation Techniques, 2001, pp. 8–12.

[6] Dorigo, M.—Gambardella, L. M.: Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. IEEE Transactions on Evolutionary Computation, Vol. 1, 1997, No. 1, pp. 53–66, doi: 10.1109/4235.585892.

[7] Dorigo, M.—Maniezzo, V.—Colorni, A.: Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics), Vol. 26, 1996, No. 1, pp. 29–41, doi: 10.1109/3477.484436.

[8] Dorigo, M.—Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, MA, 2004.

[9] Korošec, P.—Šilc, J.: Using Stigmergy to Solve Numerical Optimization Problems. Computing and Informatics, Vol. 27, 2008, No. 3, pp. 377–402.

[10] Korošec, P.—Šilc, J.—Filipič, B.: The Differential Ant-Stigmergy Algorithm. Information Sciences, Vol. 192, 2012, pp. 82–97, doi: 10.1016/j.ins.2010.05.002.

[11] Leguizamon, G.—Michalewicz, Z.: A New Version of Ant System for Subset Problems. In: Angeline, P. J. et al. (Eds.): Proceedings of the Congress on Evolutionary Computation (CEC '99), Washington, D.C., July 1999, pp. 1459–1464, doi: 10.1109/CEC.1999.782655.

[12] Li, J.—Hu, X.—Pang, Z.—Qian, K.: A Parallel Ant Colony Optimization Algorithm Based on Fine-Grained Model with GPU-Acceleration. International Journal of Innovative Computing, Information, and Control, Vol. 5, 2009, No. 11 (A), pp. 3707–3716.

[13] Maniezzo, V.—Colorni, A.: The Ant System Applied to the Quadratic Assignment Problem. IEEE Transactions on Knowledge and Data Engineering, Vol. 11, 1999, No. 5, pp. 769–778, doi: 10.1109/69.806935.

[14] Reinelt, G.: TSPLIB – A Traveling Salesman Problem Library. INFORMS Journal on Computing, Vol. 3, 1991, No. 4, pp. 376–384, doi: 10.1287/ijoc.3.4.376.

[15] Shiloach, Y.—Vishkin, U.: Finding the Maximum, Merging, and Sorting in a Parallel Computation Model. Journal of Algorithms, Vol. 2, 1981, No. 1, pp. 88–102, doi: 10.1007/BFb0105127.

[16] Stützle, T.—Holger, H. H.: MAX-MIN Ant System. Future Generation Computer Systems, Vol. 16, 2000, No. 8, pp. 889–914.

[17] You, Y.-S.: Parallel Ant System for Traveling Salesman Problem on GPUs. In: Raidl, G. et al. (Eds.): Genetic and Evolutionary Computation Conference (GECCO 2009), New York, July 2009, pp. 1–2.

Andrej Brodnik received his Ph.D. from the University of Waterloo, Ontario, Canada. In 2002 he moved to University of Primorska. During the same time he also worked as Researcher and Adjoined Professor with the University of Technology in Luleå, Sweden. He authored several tens of various scientific papers. He is also author and co-author of patents in Sweden and USA. The CiteSeer and ACM Digital Library lists over 200 citations of his works. Currently he holds positions with the University of Ljubljana and the University of Primorska.

Marko Grgurovič is a Ph.D. student in computer science at the University of Primorska. He received his B.Sc. (2010) and M.Sc. (2012) degrees in computer science from the University of Primorska. His research interests lie in theoretical computer science, particularly in the design and analysis of algorithms.

Computing and Informatics, Vol. 37, 2018, 244–268, doi: 10.4149/cai_2018_1_244

REGULARIZED SURFACE AND POINT LANDMARKS BASED EFFICIENT NON-RIGID MEDICAL IMAGE REGISTRATION

Said Khalid Shah

Department of Computer Science
University of Science and Technology, Bannu
Khyber Pukhtoun Khawa, Pakistan
e-mail: skhalids2000@yahoo.com

Abstract. Medical image registration is one of the fundamental tasks in medical image processing. It has various applications in field of image guided surgery (IGS) and computer assisted diagnosis (CAD). A set of non-linear methods have been already developed for inter-subject and intra-subject 3D medical image registration. However, efficient registration in terms of accuracy and speed is one of the most demanded of today surgical navigation (SN) systems. This paper is a result of a series of experiments which utilizes Fast Radial Basis Function (RBF) technique to register one or more medical images non-rigidly. Initially, a set of curves are extracted using a combined watershed and active contours algorithm and then tiled and converted to a regular surface using a global parameterization algorithm. It is shown that the registration accuracy improves when higher number of salient features (i.e. anatomical point landmarks and surfaces) are used and it also has no impact on the speed of the algorithm. The results show that the target registration error is less than 2 mm and has sub-second performance on intra-subject registration of MR image real datasets. It is observed that the Fast RBF algorithm is relatively insensitive to the increasing number of point landmarks used as compared with the competing feature based algorithms.

Keywords: Medical image registration, deformation, radial basis functions, image guided surgery, radiotherapy

1 BACKGROUND

Literature study showed that non-rigid medical image registration methods perform better and produce good results in case of deformable soft tissue than rigid registration. Rueckert et al. [13] have demonstrated the superiority of free-form deformations based on B-splines when compared to rigid and affine transformations applied to MRI breast images. Non-rigid registration techniques are usually divided into two broad categories: intensity based and feature based. Intensity based techniques directly operate on gray values but require optimization criteria like mutual information (MI) to find the best possible mapping. Such methods are accurate but computationally expensive. On the other hand, feature based techniques require to identify (manually or semi-automatically) the corresponding feature points, contours or even surfaces between images, to map one image onto the other.

Chui et al. [1] formulated feature-based non-rigid registration as a non-rigid point matching problem using points, contours and curves. They developed an algorithm called TPS-RPM (Thin Plate Spline – Robust Point Matching) along with thin-plate splines (TPS's) as the parameterisation of the non-rigid spatial mapping.

Levin et al. [6, 7] developed a technique for improving the speed of the point landmarks based non-rigid registration using a standard PC based on the built-in fast tri-linear interpolation feature of off-the-shelf graphics cards available in market. They execute a thin-plate spline (TPS) based warp at discrete positions on a grid that overlays each slice of the image data and has a configurable size. Built-in interpolation capability of the underlying card is used to calculate the intensity values of the voxels of each grid cell. With a data set of $512 \times 512 \times 173$ voxels and 92 manual point landmarks, they reduced the registration time from 148.2 seconds to 1.63 seconds using a brute force implementation of the grid based warp. However, the accuracy of the grid based approach was less than the corresponding brute force approach.

Another more recent example is the work by Lapeer et al. [5]. They adapted the method of Livne and Wright [8] and developed a point-based algorithm for fast medical registration using RBFs. They showed that the warp speed reduced to 0.54 s for a size 256^3 dataset (CT/MRI) of the Vanderbilt Database using 8–44 manually defined point landmarks. They also concluded that the most optimum and theoretically correct RBF function for 3D is the bi-harmonic spline instead of the 'popular' thin-plate spline (which is only optimal in 2D).

Usually point-based methods are often used in SN systems for head and neck surgery by medical experts due to ease of identification of corresponding landmark features. Further, they use it as a similarity metric such as the Target Registration Error (TRE) to measure the registration accuracy. However, if more point landmarks are needed then it is more time consuming and becomes impractical. On the other hand, surface based registration uses more or sufficient number of points, but involves a pre-processing step (called segmentation) to extract corresponding surfaces and again results in consuming more processing time. Though, few point landmarks are required to run the registration algorithm, but the use of more ac-

curately placed landmarks improves the registration accuracy much more and thus a hybrid approach of point-based and surface-based registration will be a good option to register two corresponding 3D surfaces or 3D volumes. This can be obtained by applying a point-based registration technique to a set of corresponding landmarks of two images or volumes obtained in the form of a few corresponding manual landmarks and a corresponding parameterized surface with the same number of points.

This paper presents a non-rigid feature based registration method aimed at pre- or intra-operative registration of medical images during surgical guidance. Therefore, the method needs to be fast enough for SG and having an acceptable accuracy i.e. less than 2 mm. The technique uses radial basis functions (RBFs), and more particularly the biharmonic spline (BHS), to define a non-linear mapping function between 3D images to be registered.

The presented registration method is a non-rigid point-based method [5] where the corresponding features are anatomical point landmarks and surfaces, the latter being generated by extracting corresponding curves from images to be registered using a semi-automatic method based on active contours and watersheds [4]. The extracted curves are then converted to 3D parameterized surfaces using a 3D surface generation [9] algorithm and a proper surface parameterization technique of Yoshizawa et al. [19]. Furthermore, the method is largely insensitive to the number of point landmarks used and has no effects on algorithm speed during the evaluation stage, i.e. execution time, as compared to similar methods. Accuracy of the registration should improve by using higher number of point landmarks during registration which is subject to accurate landmarks placement. Practically manual landmarks placement is not only time consuming during the registration process but also prone to errors. Therefore, we use two corresponding parameterized surfaces for the two volumes to be registered. Parameterized surfaces will not only increase the number of points but also minimize user involvement in landmarks placement and preparation. The rest of this paper describes the methods we used to obtain 3D curves, 3D surface generation and parameterization, the fast RBF technique and a comparative experiment of the fast RBF method with other feature-based non-rigid registration methods.

2 METHODS AND ALGORITHMS

2.1 Initial Contours (Curves) Extraction

Previously [5], we use manual way to place single anatomical point landmarks in matching slices of both images to be registered. We found that increasing landmarks increased the accuracy of the registration. But this manual way is time consuming, prone to errors and also requires the knowledge of expert such as radiologist. To increase the number of point landmarks for the corresponding images to be registered, we extend our point based method to curve-based method; the latter uses the boundary curves. The corresponding curves (point sets) with sufficient number of points were extracted from the both images using two popular techniques:

active contours; and watersheds. Both methods have certain limitations, i.e., the watershed technique is sensitive to image noise, causes over-segmentation and active contours suffer from initialization problems. The problem of over-segmentation in watersheds [4] was removed by using internal and external markers into regions of interest while the output of watersheds solves the active contour initialization. Both techniques overcome each other's limitations resulting into a smooth and accurate contour. The immersion-based watershed technique presented by Vincent and Soille [17] and extended by Lapeer et al. [4] to convert watershed boundaries into valid active contours using a boundary following algorithm was used. Once a boundary contour, which is piece-wise linear, is obtained, we resample each curve with a fine set of points (at pixel level) into a fixed and coarser set of points (at edgel level) by continuously reducing a given set of points into a two point set based on the computation of a mid point value. The resampling algorithm actually *parameterises* the input curves so they have the same number of points. This process is repeated to get sufficient number of curves for the surface generation of both datasets. The resultant set of corresponding boundary landmarks (curve) along each slice are triangulated to form a 3D surface for each dataset.

2.2 From Curves to 3D Surface Generation

First we build an initial 3D surface using the curves obtained from the slices and then *parameterized* to be used for registration. To get a surface mesh from two adjacent and parallel curves we apply the advancing-front algorithm [9]. It uses tiling to generate a mesh from a 3D set of points (curves). In order to get the internal mesh for the last curve which covers the ROI for the corresponding slice, we do calculate the mean position based on the boundary points of the last curve and every boundary point is connected simply to the mean position. Though this may give a *fan* triangulation (see Figure 1), a *remeshing* algorithm (coming later) can automatically fill the boundary region with well-shaped triangles. Figure 1 shows an example of two triangulated MRI data sets of human heads downloaded from Vanderbilt database, which we think to register. *Fan* triangulation contains obtuse triangles and that is not good for the application like registration. Therefore, we need a proper *reparameterization* method for correspondence creation and also a suitable *optimisation* technique to minimize the aspect ratio of obtuse triangles.

2.3 3D Surface Parameterization

Surface Parameterisation (SP) is the process of dividing a 3D surface into subsurfaces (patches), followed by finding one-to-one mapping between each subsurface and a planar domain. It can also be referred to as surface flattening, because it maps a 3D surface into a flat (2D) surface. Most of the time surfaces of arbitrary and complex shapes such as a human head or brain are represented by a collection of

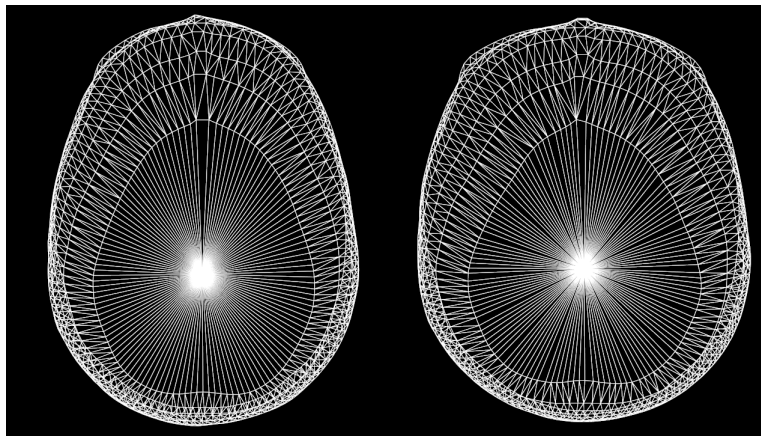


Figure 1. Corresponding mesh models after applying the advancing-front algorithm [9] to two adjacent and parallel curves obtained from MR-T1 dataset of Vanderbilt database. The mean point position of the last curve points is calculated and its triangulation with every point on the last curve is the point of maximum curvature of the ROI.

triangles and their mapping is piecewise linear. SP has various applications in computer graphics and geometric modeling, for example texture mapping, remeshing, surface repairing, and creation of regular and structured surfaces.

The concept of a SP has been extensively discussed in differential geometry, and Floater et al. [3] have presented a detailed survey about recent advances in it. SP is the process of finding a mapping function which converts a 3D surface to an equivalent 2D planar surface, i.e., it deforms a 3D surface in a continuous fashion to a planar domain. However, a better SP is the one which creates a smooth one-to-one mapping with a minimum deformation (metric distortion). Floater et al. [3] have further classified this mapping (SP) which can either be:

- *conformal*, i.e., has no distortion in angles, or
- *equi-areal*, i.e., has no distortion in areas, or
- *isometric*, i.e., minimises some combination of angle distortion and area distortion.

The literature shows that a lot of work has been carried out on each of the above mapping methods used in SP.

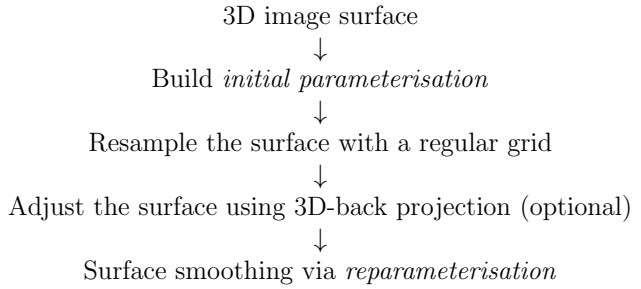
If a mapping is conformal and equi-areal then the mapping is *isometric*, i.e., it preserves distances, areas, and angles. *Isometry* is one of the most desirable properties of any parameterisation to be achieved during flattening, which means that a 3D surface after parameterisation should have all the features in their corresponding parameter domain as well. Moreover, *isometric* parameterisation exists only if a surface is locally 'flatable' (developable). But for our work with particular (targeted) applications, we found that minimising some combination of angle and area

distortion [14, 19] (*isometric* parameterisation) which has the feature of *reproduction*, i.e., a surface will be reproducible (developable) if and only if the mapping used is bijective [3], would be a better choice.

The work in this section is for targeted and specific registration applications where large deformations are involved, e.g. brain MR images. Surface parameterisation will develop a global surface based correspondence measure for two or more such images to be registered. Such a correspondence is produced through flattening (initial parameterisation) each surface to a common parametric domain, followed by distortion measurement in the *initial parameterisation* and an *optimisation* procedure that further improves the two surface's alignment in a spatial or parametric domain. We use a 2D square as a parametric domain in this work, but this can be extended to any other domain (sphere, cylinder, torus) based on the target application and shape of the object. The technique, we use for flattening the 3D surface is based on the efficient low-stretch parameterisation method of Yoshizawa et al. [19], which is a global parameterisation method and has both features of minimising angles and areas distortion. We then optimise the *initial parameterisation* (correspondence) obtained through Yoshizawa et al. [19] by using our proposed *parameterisation* technique given below.

2.3.1 Our Parameterisation Approach

Our proposed parameterisation method consists of three steps. First, it parameterises the 3D surface using the algorithm by Yoshizawa et al. [19] and creates an *initial parameterisation*. Second, it re-meshes the flattened mesh using a regular triangular grid followed by an optional surface adjustment. The third step takes the *initial parameterisation* [19] as an input and gradually improves it using *reparameterisation*. A logical flow diagram of our proposed surface parameterisation approach is given as follows:



The three steps of the proposed *parameterisation* approach are explained below:

i) Initial parameterisation. Our *initial parameterisation* approach is based on the flattening [2] and stretch minimisation [19] method. During research experiments on meshes it was observed that the method of Yoshizawa et al. [19] is preferred for global parameterisation of large and complex meshes such as

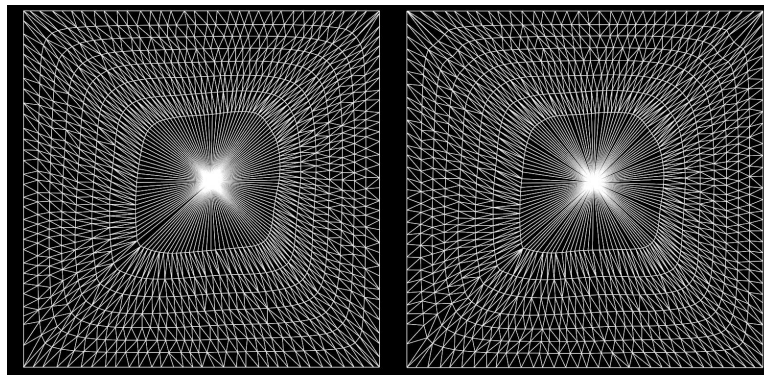


Figure 2. Left (original MR image) and right (deformed MR image); both are the corresponding 2D flattened meshes after applying the *initial parameterisation* algorithm [19] to the 3D meshes given in Figure 1. The deformation is visible from the curves structure in the mesh.

the one shown in Figure 1. It is not only a fast and efficient technique but it also produces a parameterised mesh while trying to reduce production of obtuse triangles. Obtuse triangles will impact the registration results (TRE) directly, if not avoid or reduce them to a certain extent. Figure 2 shows the 2D plane meshes on a unit square generated after applying the *initial parameterisation* method [19]. The connectivity of the flattened 2D mesh is the same as that of the original 3D mesh. All the coordinates (i.e. both x and y) of the flattened mesh are normalized and in the range $[0, 1]$.

ii) ***Resampling* (*remeshing*) using a structured 2D grid.** After flattening of the 3D mesh, we need to rebuild the original 3D surface from the flattened mesh coordinates. The *resampling* will generate a new 3D surface with proper coordinates to be used to the established correspondence representation of two or more surfaces. As we want to increase the number of points and distribute them uniformly to the original surface, we create a regular and triangular 2D grid G^r of an arbitrary $n \times n$ size instead of the flattened one. For example a 2D regular triangular grid of size 30×30 (see Figure 4 first row with two grids of size 30×30) is used to represent and rebuild the original 3D meshes of Figure 1. The size of the grid is configurable and set by the user. This particular grid of size 30×30 will create exactly 900 landmarks and will be used during the registration process.

In order to get the corresponding 3D point for each vertex of the 2D grid, we use the barycentric coordinates of the corresponding parametric triangle in the parametric domain to calculate the corresponding 3D positions for each vertex of G^r. For example, to get the 3D coordinates of a point $p'_{(x',y',z')} \in \Re^3$ having a corresponding point $u \in \Re^2$ in a parametric triangle (u_1, u_2, u_3) with

barycentric coordinates $(\lambda_1, \lambda_2, \lambda_3)$, this can be mapped (ϕ) as:

$$\phi(u_1)\lambda_1 + \phi(u_2)\lambda_2 + \phi(u_3)\lambda_3,$$

i.e.

$$p'_{(x',y',z')} = x\lambda_1 + y\lambda_2 + z\lambda_3 \tag{1}$$

where $\lambda_1, \lambda_2, \lambda_3 \geq 0$, and $\lambda_1 + \lambda_2 + \lambda_3 = 1$. x, y, and z represent the 3D coordinates of the corresponding 3D triangle to which the point p' belongs. It is a piecewise linear map and its inverse is represented as $u_i : \Re^3 \rightarrow \Re^2$. The mapping used during resampling is visualised in Figure 3.

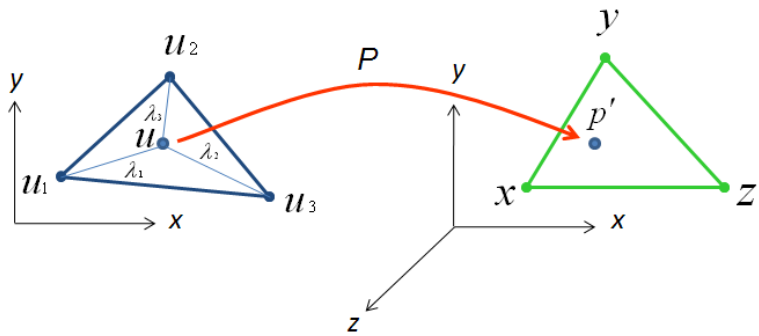
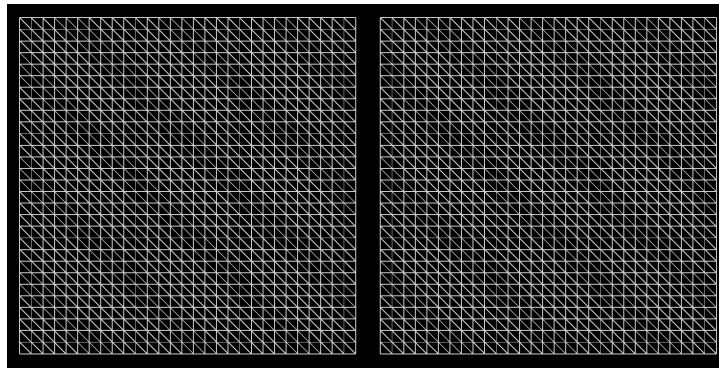


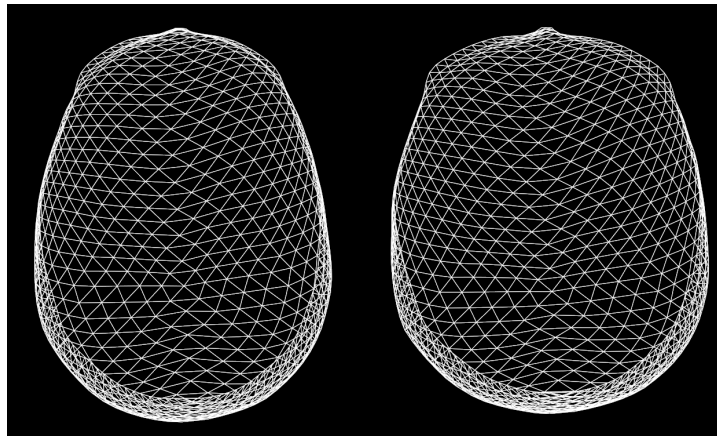
Figure 3. A barycentric mapping of given point u in a certain parametric triangle (u_1, u_2, u_3) with barycentric coordinates $(\lambda_1, \lambda_2, \lambda_3)$, which corresponds to a point $p'_{(x',y',z')} \in \Re^3$ through a parameterisation P

After replacing each vertex position of G^r by a corresponding 3D position, we get a new 3D mesh with a uniform and equivalent number of points for each of the two corresponding meshes of Figure 1. Figure 4 shows the corresponding a) regular meshes, b) resampled and c) meshes flattened back to 2D using *reparameterisation*. The discrepancies in c) show that the parameterisation itself introduces errors which are visually apparent from the differences in comparing the original a) 2D grid with the one c) flattened back to 2D after *reparameterisation*. Images in row b) show the corresponding resampled meshes of the meshes shown in Figure 1 after sampling by a regular grid. Now, we have two new meshes (row b) with the same number of points and with a known and established one-to-one correspondence between their points. The corresponding points can be used onward for *training* and *test* purposes during registration.

iii) *Reparameterisation*: Reducing triangle *obtuseness*. We start with an *initial parameterisation* U^0 and then improve it further to $U^1 \ldots U^{opt}$ by *reparameterisation*. We stop the optimisation procedure when an optimum value in



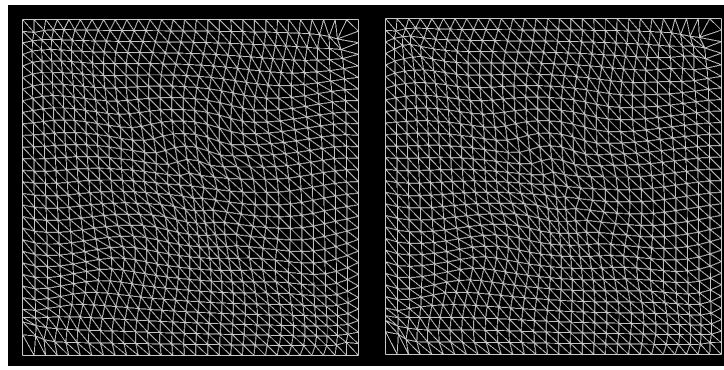
a)



b)

terms of minimum distortion (stretching error) over the whole mesh is obtained. We observed that the first optimisation step already improves the quality of mesh parameterisation. Figure 5 shows the parameterisation result of the mannequin head model and an MR-T1 dataset surface. It is clear from the third image in first and second row of Figure 5 that the first step (U^1) has better results as compared to U^0 due to *reparameterisation*, and also the triangles have better aspect ratios. Similarly, Figure 6 further demonstrates that the *reparameterisations* (U^1, U^2, U^3) produce triangles of better aspect ratios as compared to *initial parameterisation* [19] (U^0).

The total stretching error measured quantitatively during *initial parameterisation* (U^0) and *reparameterisations* (U^1, U^2, U^3, U^4) is shown in Table 1 for each model. It shows that after the first step (U^0) the stretch converges to a constant value, hence we stop the optimisation process.



c)

Figure 4. The first row a) shows the two regular grids of size 30×30 each, the second row b) corresponds to the resampled meshes of Figure 1 using the grids in the previous row, whereas the third row c) shows the corresponding 2D meshes of row b) flattened back to 2D using the same algorithm. The discrepancies in c) in comparison to a) show that the parameterisation itself introduces errors.

Technique	U^0	U^1	U^2	U^3	U^4
Mannequin head model	1.36	1.42	1.35	1.34	1.35
MR-T1 surface	1.16	1.01	1.01	1.02	1.02

Table 1. Row one and two show the total stretch for mannequin head and MR-T1 surface models during the *initial parameterisation* U^0 and *reparameterisation* U^1, U^2, U^3, and U^4, respectively

2.4 The Fast Radial Basis Functions Method

The Radial Basis Function (RBF) method is one of the most widely used technique to approximate or interpolate data scattered in more than one dimensions. The purpose of interpolation is to approximate a real-valued function $f(\mathbf{x})$ over a finite set of values $f = (f_1, \ldots, f_N)$ at the distinct points $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbf{R}^d$. In similar situation, one chooses an RBF, $s(\mathbf{x})$, for representing such approximations, normally of the following general form:

$$s(\mathbf{x}) = p(\mathbf{x}) + \sum_{i=1}^{N} \lambda_i \phi \left(\|\mathbf{x} - \mathbf{x}_i\| \right), \qquad \mathbf{x} \in \mathbf{R}^d \tag{2}$$

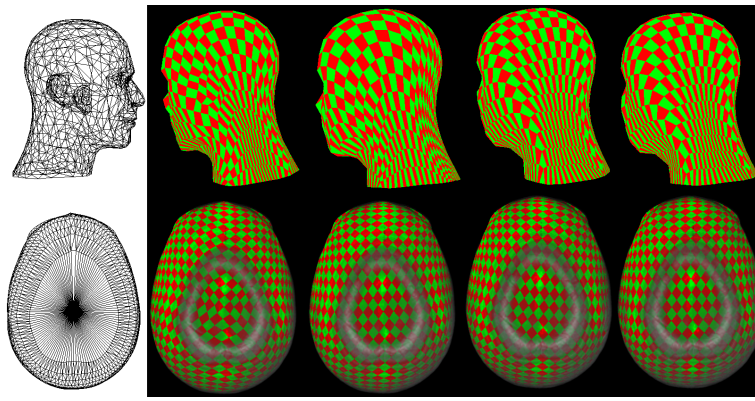


Figure 5. The images from left to right of both rows show the corresponding meshes of a mannequin head model and an MR-T1 dataset surface before and after (first and second image in both rows) the *initial parameterisation* [19] (U^0) while the last three images (U^1, U^2, U^3) represent the meshes after applying the proposed *reparameterisation* technique three times to the *initial parameterisation* U^0. For the MR-T1 dataset, the last four surfaces show the image data and mesh together.

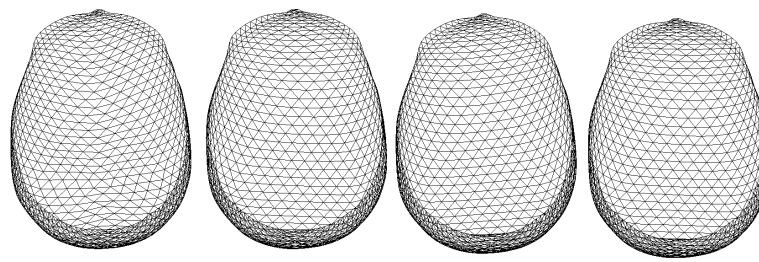


Figure 6. The same sequence as in the bottom row of Figure 5 but displayed in mesh format only

where $p(\mathbf{x})$ is a polynomial, λ_i is a real-valued weight[1], ϕ is the (radial) basis function and $\|\mathbf{x} - \mathbf{x}_i\| = r$ is the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}_i$. So, an RBF might be defined as a weighted sum of a radially symmetric basis function, added together with a polynomial term.

The basis function ϕ can take several forms, but three of them have a common property of minimizing specific quantities of energy [10], which makes them suitable

[1] The λ weights are determined in the 'calculation' step using a least mean squares approach. This step is followed by the 'evaluation' step which applies the RBF to (usually) all voxels. The latter step is much more time-consuming than the former one.

for use in 2D and 3D non-rigid medical image registration techniques. Rohr [10] further shows that the biharmonic spline (BHS): $\phi(r) = r$ and the thin-plate spline (TPS): $\phi(r) = r^2 \log r$, both minimize a bending energy potential of order two in 3D and 2D space, respectively. Thus to warp 3D image data, the BHS is therefore the choice to be preferred. Lapeer et al. [5] confirmed its theoretical optimality in 3D as shown by Rohr experimentally.

Lapeer et al. [5] rewrite Equation (2) without the linear polynomial part for sake of clarity, and extend it to 3D for evaluation of $i = 1 \ldots m$ evaluation points/voxels (targets) represented by the target vector $\mathbf{x}_i$, after having found the spline parameters λ_j for $j = 1 \ldots n$ landmarks represented by the source (landmark) vector $\mathbf{y}_j$:

$$s(\mathbf{x}_i) = \sum_{j=0}^{n} \lambda(\mathbf{y}_j)\phi(\|\mathbf{x}_i - \mathbf{y}_j\|), \qquad i = 0, 1, \ldots, m. \tag{3}$$

Livne and Wright [8] describe a new technique for fast multilevel evaluation of RBF expansions. The main idea of the fast RBF technique is to accurately represent a smooth RBF, ϕ, on a regular and coarser grid with few nodes as compared to the full voxel set of the image data, and thus the expensive summation in Equation (3) needs to be applied to these few nodes only. The rest of the voxel values can finally be computed using a less expensive formulation based on the values determined for the surrounding nodes. Unlike the grid based approach by Levin et al. [7], those are the RBF coefficients that are interpolated within the grid and not the intensity values of the voxels.

The main principle behind the fast RBF method is to encapsulate the source (landmarks) and target (voxel) points in two new separate and corresponding uniform grids of size H. The new uniform grids overlap the old landmark and voxel sets respectively, which results into a two stage conversion process of the RBF in Equation (3). The first stage is to calculate the level H expansion coefficients replacing the original **source** points (landmarks) with their corresponding grid points by using a centered p^{th} order tensor product interpolation:

$$\phi(\|\mathbf{x}_i - \mathbf{y}_j\|) = \sum_{j:J_k \epsilon \sigma_j^{(k)}} \omega_{jJ_3}\omega_{jJ_2}\omega_{jJ_1}\phi(\|\mathbf{x}_i - \mathbf{Y}_{(J_1,J_2,J_3)}\|) \tag{4}$$

where $j = 0, 1, \ldots, n$ and for dimension $k = 1, 2, 3$: $\sigma_j^{(k)} := \left\{ J_k : \left| Y_{J_k}^{(k)} - y_j^{(k)} \right| < pH/2 \right\}$, where ω_{jJ_k} are the new centered p^{th}-order interpolation weights from the coarse centres $Y_{J_k}^{(k)}$ to the landmark positions $y_j^{(k)}$. The second stage replaces the original **target** points (i.e. voxels) with their corresponding grid points using the same approach:

$$\phi(\|\mathbf{x}_i - \mathbf{Y}_\mathbf{J}\|) = \sum_{I_k \epsilon \bar{\sigma}_i^{(k)}} \bar{\omega}_{iI_3}\bar{\omega}_{iI_2}\bar{\omega}_{iI_1}\phi(\|\mathbf{X}_{(I_1,I_2,I_3)} - \mathbf{Y}_\mathbf{J}\|) \tag{5}$$

where $i = 0, 1, \ldots, m$, $\mathbf{J} = (J_1, J_2, J_3)$, and for dimension $k = 1, 2, 3$: $\bar{\sigma}_i^{(k)} := \left\{ I_k : \left| X_{I_k}^{(k)} - x_i^{(k)} \right| < pH/2 \right\}$, where $\bar{\omega}_{iI_k}$ are the centered p^{th}-order interpolation weights from the coarse evaluation point $X_{I_k}^{(k)}$ to the level h (original image grid size) evaluation point $x_i^{(k)}$.

The so called *anterpolation* method is used to properly distribute the known RBF coefficients $\lambda(\mathbf{y}_j)$ at each landmark position to the surrounding nodes of grid $\mathbf{Y}$. More detail of the fast RBF method in 1D and 2D, and 3D can be found in [8] and [11], respectively.

2.5 Performance Metric

The following two performance metrics were used to access the accuracy of our method:

Target Registration Error (TRE): It is the RMS error between the homologous validation landmarks after registration. The distance between every corresponding pair of points of the two meshes (surfaces) is calculated to determine how close and well registered the surfaces are. The closer the registered surfaces, the better the registration. This distance is calculated as an RMS error between the corresponding *test* landmarks after the registration process.

Normalized Mutual Information (NMI): As the NMI metric (Studholme et al. [16]) is suited to both mono-modal and multi-modal scenarios, we use this metric for image similarity measurement. It is the overlap invariant, and has an optimal and minimum value of 2.0 and 1.0, respectively.

3 EXPERIMENTAL RESULTS

After getting the corresponding surfaces (set of 3D points) of a pair of images, we fit the spline to the corresponding *training* landmarks to get the transformation matrix and spline parameters (weights) for final registration and validation (using the *test* landmarks). A quantitative experiment has performed, to show that the fast RBF method is insensitive in terms of speed to an increasing number of accurately placed landmarks in the form of corresponding surface points. Further, it is also shown that the increasing number of accurately placed landmarks improves the registration accuracy as well. For this purpose, six different competing methods are tested:

1. Brute force (non-optimized) RBF – applying a standard software based method which applies the spline model to each voxel in the data set without any optimisation. This method is considered to be the gold standard in terms of accuracy.
2. Brute force (non-optimized) RBF with hardware acceleration – the same algorithm as before but implemented on the GPU (Graphics Processing Unit) which enables a significant speedup due to its parallel processing capabilities.

3. Fast RBF method – the software-based optimised algorithm as described in Section 2.4. In our previous work [5], it is observed that optimal value for the H parameter in the model is 0.025.

4. Fast RBF method with hardware acceleration – the previous method implemented on the GPU.

5. The grid approach by Levin et al. [7] with two different grid sizes.

6. The FFD (free form deformation) based non-rigid registration algorithm of Rueckert et al. [13, 15] implemented in IRTK (The Image Registration Toolkit) [12], is used to compare the results of our proposed Fast RBF method with the state-of-art technique IRTK using NMI (for multi-model image registration), *warp time* and as well as visual assessment in the form of difference images. In FFD based registration, we set the initial control point spacing to 25.6 mm and run up to three levels in a coarse-to-fine fashion, where level 1 represents the coarsest level and level 3 represents the finest (optimised) level.

To evaluate the speed-optimized algorithms which use hardware acceleration, i.e. 2., 4. and 5., in terms of accuracy, the brute force algorithm 1. is considered as the gold standard. This is because current GPU's, despite being significantly faster than CPU's, only have 32 bits for floating point representation, whereas CPU's have 64 bits, what affects the accuracy of the warp. The experiments were run on computer under the Windows XP operation system. The hardware in the computer included: Intel Core 2 Quad 6600+ CPU; 3 GB of DDR2 RAM; and an NVIDIA GeForce 8800GTX Graphics Card with 768 MB memory.

3.1 Non Rigid Image Registration Using Real Datasets

The MR datasets of three subjects of the ADNI database (adni.loni.ucla.edu) were used and resampled to 256^3 with 1 mm slice thicknesses. These datasets were used to test intra-patient point based non-linear registration from the original dataset to its natural deformed version (see Figures 11 to 13, columns 1 and 2 of all rows). We ran our experiment with an increased number of landmarks by using first a few anatomical point landmarks followed by a combination of surface points with few manual anatomical point landmarks. This way the increase in point landmarks represents more and more deformations in corresponding images and will eventually improve the registration results. During the experiment, we used the BHS spline rather than TPS, due to its suitability for 3D non-rigid medical image registration in terms of *speed* and *accuracy*, as shown in [5].

Results of the different registration algorithms with ADNI datasets are given in Table 2.

BHS ($\phi(r) = r$)

25 Landmarks	**Warp Time in sec.**	**NMI**	**%NMI**	**TRE in mm**
Brute force S/W	28.55 (1.54)	1.202 (0.043)	100.0	1.63 (0.49)
Brute force H/W	0.51 (0.05)	1.192 (0.036)	99.1	1.63 (0.49)
Fast RBF S/W 0.025	15.27 (0.47)	1.202 (0.043)	100.0	1.63 (0.49)
Fast RBF H/W 0.025	0.53 (0.04)	1.199 (0.041)	99.6	1.63 (0.49)
Grid 13	0.43 (0.01)	1.144 (0.066)	95.1	1.63 (0.49)
Grid 138	16.31 (1.08)	1.144 (0.066)	95.1	1.63 (0.49)
450 + 25 Landmarks	**Warp Time in sec.**	**NMI**	**%NMI**	**TRE in mm**
Brute force S/W	486.48 (4.02)	1.227 (0.013)	100.00	1.81 (0.20)
Brute force H/W	2.42 (0.47)	1.214 (0.027)	98.9	1.81 (0.20)
Fast RBF S/W 0.025	31.15 (1.04)	1.227 (0.024)	100.0	1.81 (0.20)
Fast RBF H/W 0.025	0.62 (0.02)	1.222 (0.021)	99.6	1.81 (0.20)
Grid 13	2.69 (0.06)	1.146 (0.051)	93.4	1.81 (0.20)
Grid 138	282.25 (2.78)	1.143 (0.052)	93.1	1.81 (0.20)
800 + 25 Landmarks	**Warp Time in sec.**	**NMI**	**%NMI**	**TRE in mm**
Brute force S/W	843.04 (5.30)	1.222 (0.025)	100.00	1.48 (0.13)
Brute force H/W	3.91 (0.20)	1.210 (0.020)	99.1	1.48 (0.13)
Fast RBF S/W 0.025	35.65 (1.56)	1.223 (0.027)	100.0	1.48 (0.13)
Fast RBF H/W 0.025	0.80 (0.01)	1.219 (0.024)	99.8	1.48 (0.13)
Grid 13	4.53 (0.02)	1.137 (0.045)	93.1	1.48 (0.13)
Grid 138	485.71 (3.10)	1.136 (0.044)	93.0	1.48 (0.13)
IRTK (FFD)	**Warp Time in sec.**	**NMI**	**%NMI**	**TRE in mm**
Level **1**	$\approx$ 240.00	1.217 (0.040)	99.1	N/A
Level **2**	$\approx$ 888.00	1.251 (0.039)	101.9	N/A
Level **3**	$\approx$ 3 700.00	1.268 (0.039)	103.3	N/A

Table 2. Results after applying the BHS basis function for non-rigid registration of the MR-T1 ADNI datasets of the same subject taken at different time points. 25 point landmarks were used for training and 25 for validation in the first part of the table. In the second part of the table, 450 surface based landmarks were used for training, while another 450 surface based point landmarks were used for validation, plus an additional 25 manually placed point landmarks were used. The third part is similar to the second part but the number of training and validation landmarks are both increased to 800. The last part (last three rows) of the table shows the results after applying the multilevel free form deformation (FFD) non-rigid registration algorithm of Rueckert et al. [13, 15] implemented in IRTK (The Image Registration Toolkit) [12]. The FFD based results are calculated after different levels of registration, i.e. level 1, 2, and 3. All tests were run over 5 subjects. All the values show the averages along with standard deviation in brackets. Second column shows the evaluation, i.e. warp time of the RBF in seconds. The third column shows the NMI. The next column shows the %NMI as compared to the Brute-Force Software and used as the golden standard. The fifth and final column shows the TRE in mm which is evaluated on the validation landmarks – note that the latter is the same for all methods as its calculation is based on the same BHS model.

4 RESULTS AND DISCUSSION

4.1 Quantitative Results: ADNI Datasets

Let us first have a look at the *evaluation time* of different algorithms. The second column in Table 2 shows that the *evaluation time* of the Fast RBF method (both software and hardware versions) is only marginally affected by increasing the number of landmarks with a factor of almost 20, unlike all other methods which are proportionally more affected. In comparison to IRTK, the evaluation time of the Fast RBF hardware accelerated method is in subseconds, while IRTK takes approximately 240, 888, and 3 700 seconds during level 1, 2 and 3, respectively, using each dataset. Thus, the evaluation time of the Fast RBF method in hardware is significantly less dependent on the number of landmarks used than for competing methods, and it is substantially *faster* than the IRTK method as well as the other competing algorithms.

The final column in Table 2 shows the TRE in mm which is evaluated on the validation landmarks. It should be noted that the TRE is the same for all methods as its calculation is based on the same BHS model. The average TRE with 475 landmarks is slightly worse, however the standard deviation is substantially smaller despite being measured over a much larger set of validation points illustrating a statistically more significant result. Similarly, the average TRE with 825 landmarks is decreasing with the increase in number of landmarks from 475 to 825. The table shows that the average TRE for 825 landmarks is 1.48 mm, which is better than when using 25 (1.63 mm) and 475 (1.81 mm) landmarks.

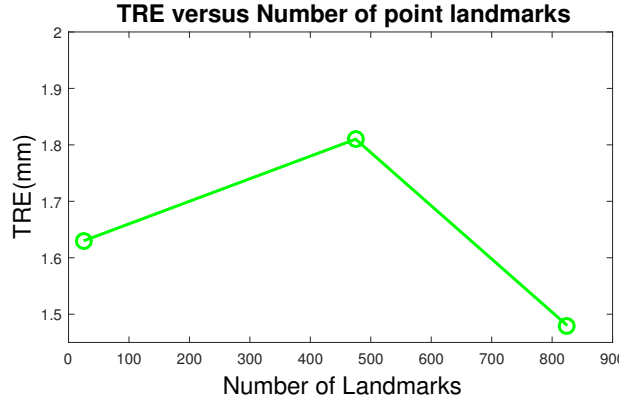


Figure 7. Effect of the number of point landmarks on TRE of each warping algorithm

It can also be seen from Figure 7, that the TRE (with error bars) goes up (from 1.63 mm to 1.81 mm) initially due to manual error in landmarks placement but then going down (from 1.81 mm to 1.48 mm) with the increase (from 475 to 825) in number of surface point landmarks. The error bars further indicate that point

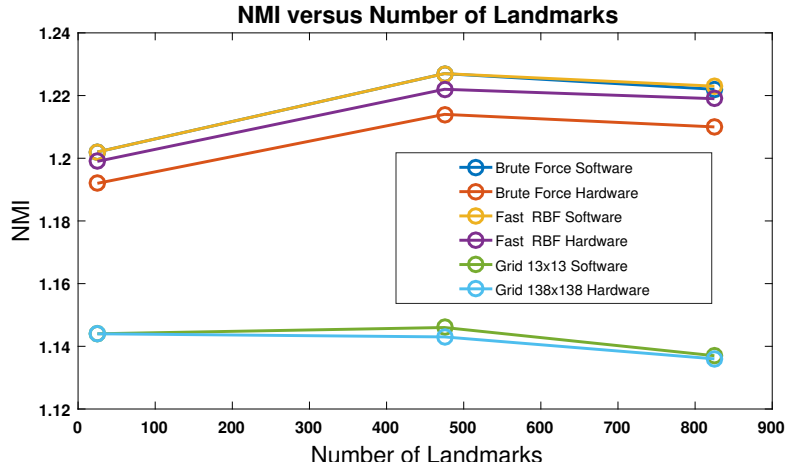


Figure 8. Effect of the number of landmarks on NMI of each warping algorithm

landmarks created using *parameterisation* are more consistent than the manually placed anatomical point landmarks. Furthermore, there are no statistically significant differences ($p >> 0.10$) between TRE's of the upper and middle part of the Table 2 ($t = -0.643$, $dof = 8$, $p = 0.54$), as well as between the upper and lower part of the Table 2 ($t = -0.471$, $dof = 8$, $p = 0.65$) but the TRE's of the middle and lower part of the Table 2 ($t = 2.471$, $dof = 8$, $p = 0.038$) has significant difference based on a two-tailed *t-test* p value (i.e. $p < 0.05$).

Looking at the *accuracy* using the NMI metric (third column of Table 2), we see virtually no loss in accuracy for the Fast RBF method in software as compared to the gold standard (brute force software) and when implemented in hardware, its accuracy is better than the brute force hardware implementation. The NMI of the larger landmark sets (475 and 825) is better than when using just 25 landmarks. The IRTK method shows slightly better results in term of NMI with increment of deformation levels (level 1 to level 3), but this is due to local support features of FFD based registration. The difference images in Figures 11 to 13 (see last column in all rows) also justify this. As can be seen from the difference images, where the local regions are aligned slightly better using the IRTK algorithm than the proposed Fast RBF method which is based on global support RBFs (the BHS function). This problem of Fast RBF could likely be removed by using locally constrained RBFs [18].

Figure 8 indicates that the NMI of the Fast RBF hardware and software based algorithms is almost the same with both sets of landmarks, i.e. 475 and 825, which is better than using a few (25) manual point landmarks only[2].

[2] The result of the FFD method are not displayed as this method does not depend on the number of landmarks.

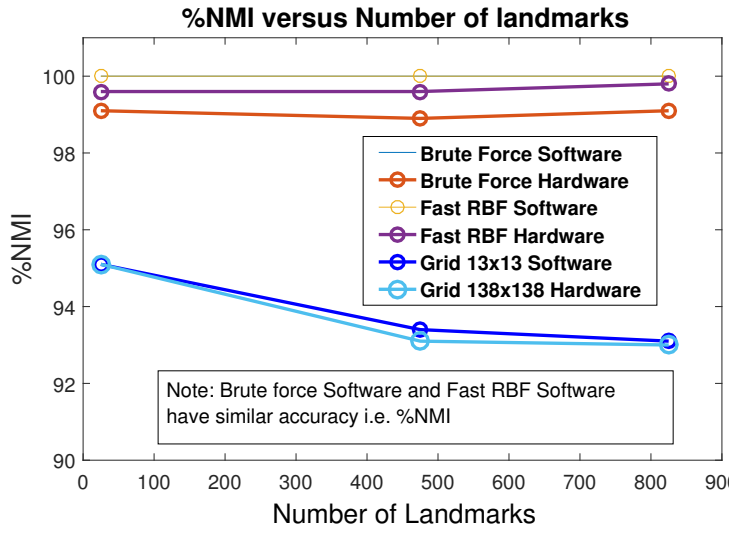


Figure 9. Effect of the number of landmarks on %NMI of each warping algorithm, compared against the non-optimised 'Brute force' software based method (gold standard)

The %NMI metric shows the performance of the optimised techniques in comparison to the non-optimised 'Brute force' software based method (gold standard). Figure 9 shows tha %NMI drawn against the number of landmarks. This suggests that the fast RBF method implemented in hardware exhibits the highest correspondence (99 % +) as compared to the brute force implementation. The main reason of the loss in accuracy is due to the single floating point precision capability of the GPU used.

4.2 Visual Results: ADNI Datasets

The following figures show the results for visual assessment and comparison with IRTK. Figure 10 represents the corresponding triangulated parameterised (red and green) meshes before (separately) and after (fused) the registration experiment. Images in the last row of Figure 10 show the deformation vectors between the corresponding vertices of the two meshes before (leftmost) and after (center and rightmost) the registration experiment. These vectors are visualised through color coding from blue to red in ascending order of deformation. The vectors in the center and right most image show the TRE error (as a displacement) between the validation landmarks before and after the registration process.

Figures 11 to 13 show arbitrarily selected transverse slices from the full resolution MR ADNI datasets. The first two images of the first row of each figure show the original dataset and its natural deformed version before registration, whereas the last two images show corresponding registered and absolute difference images after the

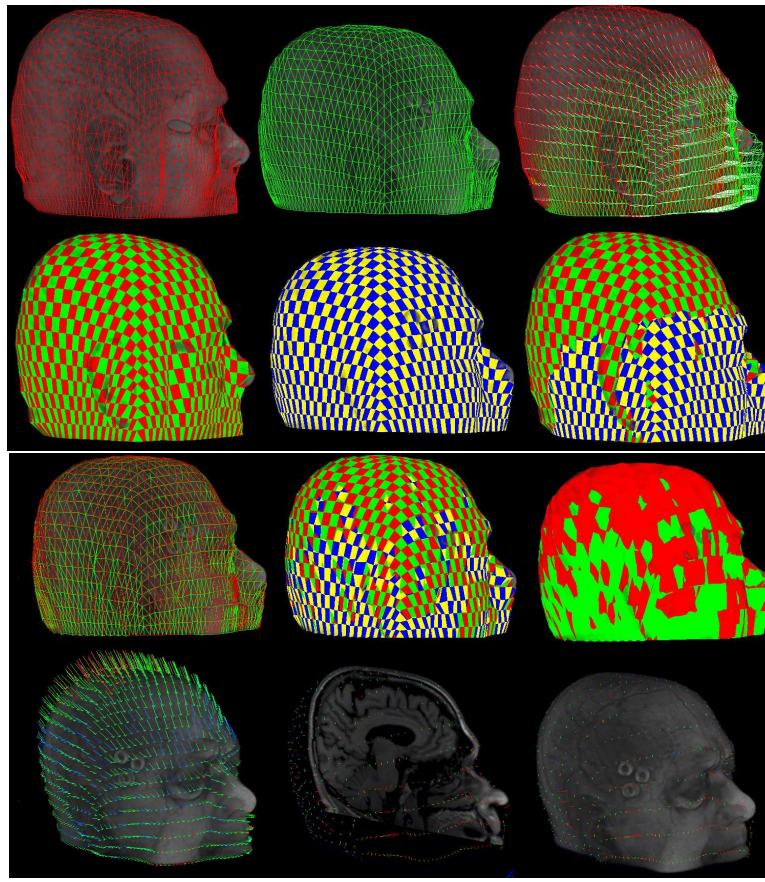


Figure 10. The first two images in the first row show the corresponding triangulated parameterised (red and green) meshes and the last image shows their combination before the registration experiment. In the last image, a white displacement line is drawn between every two corresponding vertices of the two meshes. The second row from left to right shows the images of the above row by using colored surfaces. The third row from left to right shows the fused meshes and surfaces after the registration experiment. Images in the last row show the deformation vectors between the corresponding vertices of the two meshes (leftmost) before and (center and rightmost) after the registration experiment. These vectors are visualised through color coding from blue to red in ascending order of deformation. The vectors in the center and rightmost image shows the TRE error (as a displacement) between the validation landmarks before and after the registration process.

registration experiment performed with our method (Fast RBF). Row 2 corresponds to the registered and absolute difference images after the registration experiment performed with IRTK at level 1 and 3, respectively. The first three images in some rows show the corresponding *training* and *validation* (red and blue) landmarks before registration, and after (red, blue and green, respectively) registration.

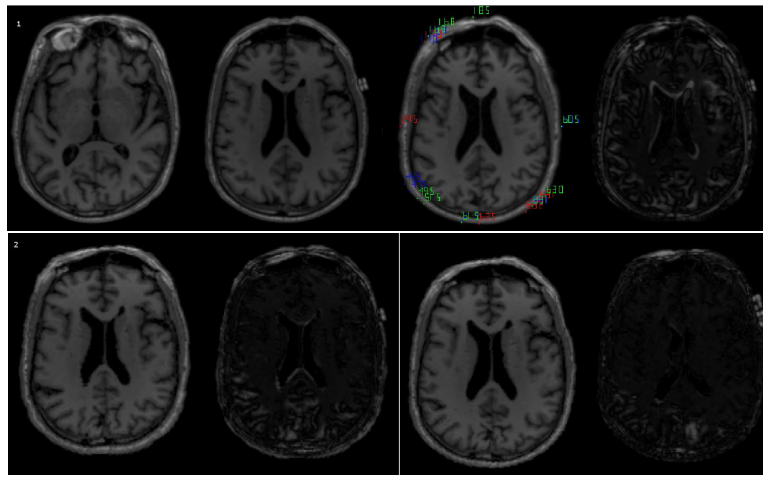


Figure 11. All the rows from left to right show arbitrarily selected transverse slices from the full resolution MR datasets (ADNI database). The first two images of row 1 illustrate the original and naturally deformed MR image before registration, while the last two images show corresponding registered and absolute difference images after the registration experiment performed with our method (Fast RBF). Row 2 corresponds to the registered and absolute difference images after the registration experiment performed with IRTK at level 1 and 3, respectively. The first three images in some rows show the corresponding *training* and *validation* (red and blue) landmarks before registration, and after (red, blue and green, respectively) registration.

5 CONCLUSION

In this article, we have presented the Fast RBF non-rigid registration method for medical imaging data using anatomical point landmarks and optimised parameterised surfaces, respectively. We have seen that the increase in the number of landmarks affects both accuracy and evaluation time. The number of point landmarks increased from anatomical point landmarks to a surface which represents an image deformation field better than the standalone anatomical point landmarks. The hardware implementation of the algorithm to run the evaluation part of the algorithm in less than a second using standard computer with a latest graphic card.

The evaluation (warp) time of both the hardware and software implementation of the Fast RBF algorithm is clearly less susceptible to the number of point land-

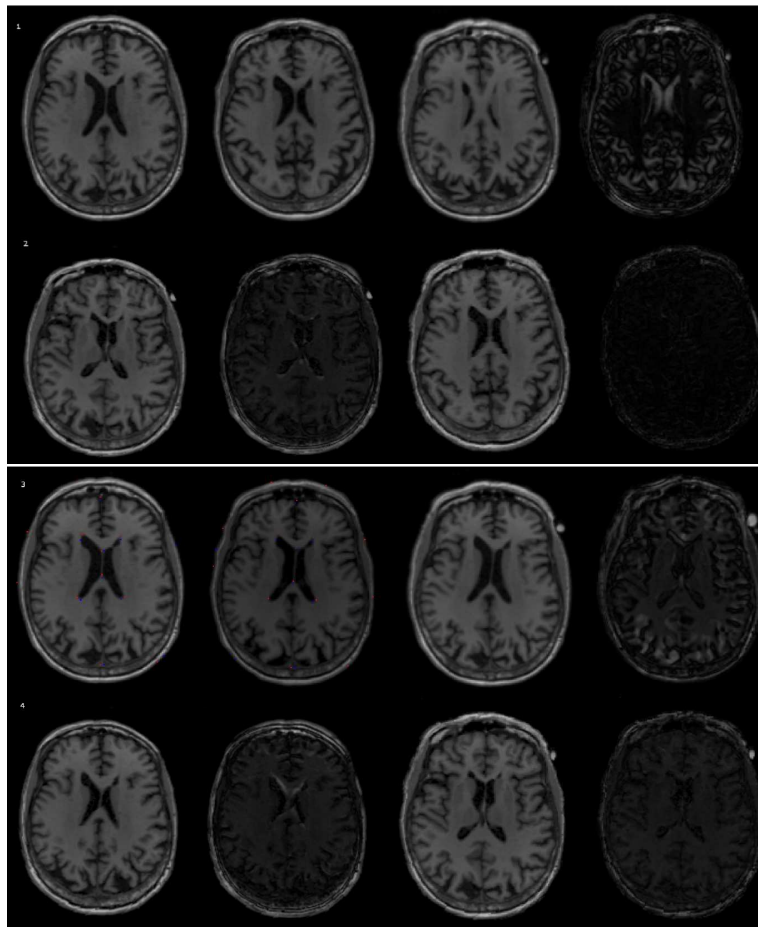


Figure 12. For each image block: Both rows show arbitrarily selected transverse slices from the full resolution MR datasets (ADNI database). The first two images of row 1 illustrate the original and naturally deformed MR image before registration, while the last two images show corresponding registered and absolute difference images after the registration experiment performed with our method (Fast RBF). Row 2 corresponds to the registered and absolute difference images after the registration experiment performed with IRTK at level 1 and 3, respectively.

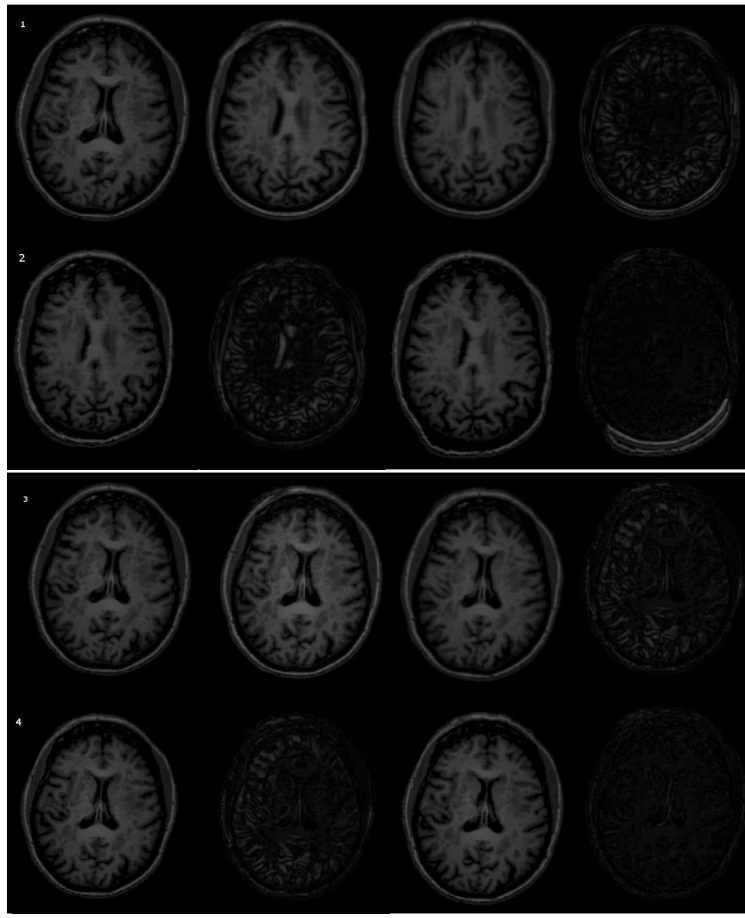


Figure 13. For each image block: Both rows show arbitrarily selected transverse slices from the full resolution MR datasets (ADNI database). The first two images of row 1 illustrate the original and naturally deformed MR image before registration, while the last two images show corresponding registered and absolute difference images after the registration experiment performed with our method (Fast RBF). Row 2 corresponds to the registered and absolute difference images after the registration experiment performed with IRTK at level 1 and 3, respectively.

marks used as compared to the other tested competing methods. It is considered that the use of more accurately placed point landmarks using *surface parameterisation* and *reparameterisation* improves its accuracy and makes the algorithm more favourable for IGS applications where both speed and accuracy are critical. We presented experiments on real medical datasets with a larger non-rigid deformation, for example MR images of the brain. It was observed that Fast RBF in software and hardware outperforms the feature-based methods both in terms of speed and accuracy, whilst performs a little bit less well in accuracy (when expressed in terms of NMI) than the FFD based method, which is due to iterative nature of the FFD.

REFERENCES

[1] Chui, H.—Rangarajan, A.: A New Point Matching Algorithm for Non-Rigid Registration. Computer Vision and Image Understanding, Vol. 89, 2003, No. 2-3, pp. 114–141, doi: 10.1016/S1077-3142(03)00009-2.

[2] Floater, M. S.: Parametrization and Smooth Approximation of Surface Triangulations. Computer Aided Geometric Design, Vol. 14, 1997, No. 3, pp. 231–250, doi: 10.1016/S0167-8396(96)00031-3.

[3] Floater, M. S.—Hormann, K.: Recent Advances in Surface Parameterization. Proceedings Multiresolution in Geometric Modelling 2003, 2003, pp. 259–284.

[4] Lapeer, A. C.—Tan, R. J.—Aldridge, R. V.: Active Watersheds: Combining 3D Watershed Segmentation and Active Contours to Extract Abdominal Organs from MR Images. In: Dohi, T., Kikinis, R. (Eds.): Medical Image Computing and Computer-Assisted Intervention (MICCAI 2002). Springer, Berlin, Lecture Notes in Computer Science, Vol. 2488, 2002, pp. 596-603.

[5] Lapeer, R. J.—Shah, S. K.—Rowland, R. S.: An Optimised Radial Basis Function Algorithm for Fast Non-Rigid Registration of Medical Images. Computers in Biology and Medicine, Vol. 40, 2010, No. 1, pp. 1–7.

[6] Levin, D.—Dey, D.—Slomka, P. J.: Acceleration of 3D, Nonlinear Warping Using Standard Video Graphics Hardware: Implementation and Initial Validation. Computerized Medical Imaging and Graphics, Vol. 28, 2004, No. 8, pp. 471–483, doi: 10.1016/j.compmedimag.2004.07.005.

[7] Levin, D.—Dey, D.—Slomka, P.: Efficient 3D Nonlinear Warping of Computed Tomography: Two High-Performance Implementations Using OpenGL. In: Galloway, R.L. Jr., Cleary, K. R. (Eds.): Medical Imaging 2005: Visualization, Image-Guided Procedures, and Display. Proceedings of the SPIE, Vol. 5744, 2005, pp. 34–42, doi: 10.1117/12.595935.

[8] Livne, O. E.—Wright, G. B.: Fast Multilevel Evaluation of Smooth Radial Basis Function Expansions. Electronic Transactions on Numerical Analysis, Vol. 23, 2006, pp. 263–287.

[9] Lo, S. H.: A New Mesh Generation Scheme for Arbitrary Planar Domains. International Journal for Numerical Methods in Engineering, Vol. 21, 1985, No. 8, pp. 1403–1426.

[10] Rohr, K.: Landmark-Based Image Analysis Using Geometric and Intensity Models. Kluwer Academic Publishers, 2001, doi: 10.1007/978-94-015-9787-6.

[11] Rowland, R. S.: Fast Registration of Medical Imaging Data Using Optimised Radial Basis Functions. Ph.D. thesis, University of East Anglia, 2007.

[12] Rueckert, D.: Image Registration Toolkit (IRTK). `http://www.doc.ic.ac.uk/~dr/software/index.html`, 2006.

[13] Rueckert, D.—Sonoda, L. I.—Hayes, C.—Hill, D. L. G.—Leach, M. O.—Hawkes, D. J.: Nonrigid Registration Using Free-Form Deformations: Application to Breast MR Images. IEEE Transactions on Medical Imaging, Vol. 18, 1999, No. 8, pp. 712–721, doi: 10.1109/42.796284.

[14] Sander, P. V.—Snyder, J.—Gortler, S. J.—Hoppe, H.: Texture Mapping Progressive Meshes. Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01), 2001, pp. 409–416, doi: 10.1145/383259.383307.

[15] Schnabel, J. A.—Rueckert, D.—Quist, M.—Blackall, J. M.—Castellano-Smith, A. D.—Hartkens, T.—Penney, G. P.—Hall, W. A.—Liu, H.—Truwit, C. L.—Gerritsen, F. A.—Hill, D. L. G.—Hawkes, D. J.: A Generic Framework for Non-Rigid Registration Based on Non-Uniform Multi-Level Free-Form Deformations. Medical Image Computing and Computer-Assisted Intervention (MICCAI 2001). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2208, 2001, pp. 573–581, doi: 10.1007/3-540-45468-3_69.

[16] Studholme, C.—Hill, D. L. G.—Hawkes, D. J.: An Overlap Invariant Entropy Measure of 3D Medical Image Alignment. Pattern Recognition, Vol. 32, 1999, No. 1, pp. 71–86, doi: 10.1016/S0031-3203(98)00091-0.

[17] Vincent, L.—Soille, P.: Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 13, 1991, No. 6, pp. 583–598. ISSN 0162-8828, doi: 10.1109/34.87344.

[18] Wendland, H.: Piecewise Polynomial, Positive Definite and Compactly Supported Radial Functions of Minimal Degree. Advances in Computational Mathematics, Vol. 4, 1995, pp. 389–396, doi: 10.1007/BF02123482.

[19] Yoshizawa, S.—Belyaev, A.—Seidel, H. P.: A Fast and Simple Stretch-Minimizing Mesh Parameterization. Proceedings of the Shape Modeling Applications, 2004, pp. 200–208.

Said Khalid Shah received his Master's degree in computer science (1999) from Department of Computer Science, University of Peshawar. From 2000 to 2004, he worked as Lecturer at the Department of Computer Science, University of Peshawar, Pakistan. In 2004, he joined the University of Science and Technology, Bannu, KPK, Pakistan as Lecturer and then he was promoted to Assistant Professor in 2012. He received his Ph.D. degree from the University of East Anglia, UK, with a thesis on non-rigid medical image registration in 2011. After completing his Ph.D., he joined the University of Science and Technology, Bannu where he was responsible for teaching various computer science subjects and also supervising academic/industrial research projects in the area of medical image processing and analysis such as segmentation, visualization, and registration.