Computing and Informatics, Vol. 38, 2019, 765–789, doi: 10.31577/cai_2019_4_765

GROUP-BASED ASYNCHRONOUS DISTRIBUTED ALTERNATING DIRECTION METHOD OF MULTIPLIERS IN MULTICORE CLUSTER

Dongxia Wang, Yongmei Lei*, Shenghong Jiang

School of Computer Engineering and Science
Shanghai University
Nanchen Road No. 333, Baoshan District, Shanghai, 200436, China
e-mail: wangdongxia1983@126.com, lei@shu.edu.cn, jsh79@i.shu.edu.cn

Abstract. The distributed alternating direction method of multipliers (ADMM) algorithm is one of the effective methods to solve the global consensus optimization problem. Considering the differences between the communication of intra-nodes and inter-nodes in multicore cluster, we propose a group-based asynchronous distributed ADMM (GAD-ADMM) algorithm: based on the traditional star topology network, the grouping layer is added. The workers are grouped according to the process allocation in nodes and model similarity of datasets, and the group local variables are used to replace the local variables to compute the global variable. The algorithm improves the communication efficiency of the system by reducing communication between nodes and accelerates the convergence speed by relaxing the global consistency constraint. Finally, the algorithm is used to solve the logistic regression problem in a multicore cluster. The experiments on the Ziqiang 4000 showed that the GAD-ADMM reduces the system time cost by 35 % compared with the AD-ADMM.

Keywords: ADMM, global consensus optimization, multicore cluster, logistic regression, GAD-ADMM

Mathematics Subject Classification 2010: 68W15

* Corresponding author

1 INTRODUCTION

Machine learning has become an important method to extract structured information from raw data and transform it into different automatic predictions and applied hypotheses [1]. In the era of big data, sometimes not only is the number of samples large, but also the dimension of samples is high. Therefore, in the traditional machine learning algorithm, it is difficult to implement the relevant processing and calculation of the big data in a reasonable amount of time. It is necessary to consider how to transform the traditional machine learning algorithms into distributed ones using high-performance distributed computing. Most supervised machine learning algorithms can be viewed as cost-function optimization methods [1], which can be expressed as the following mathematical form:

$$\min f_x(x, D) \tag{1}$$

where $D \in R^{m*n}$ is the sample dataset, $x \in R^n$ represents the model parameter, m is the number of samples and n is the dimension of the samples.

The alternating direction method of multipliers (ADMM) is an effective method suitable for separable convex optimization, which has been used in distributed optimization and statistical machine learning [2]. The ADMM algorithm can transform the large global problem into several smaller, local sub-problems, and can derive the solution of the global problem by coordinating the solutions of the sub-problems [2]. We can transform the original problem (1) into a global consensus optimization problem, suitable for distributed environments, as shown below:

$$\min \sum_{i=1}^{N} f_i(x_i, D_i) + g(z), \quad \text{s.t. } x_i = z, i = 1, 2, \ldots, N \tag{2}$$

where $D_i \in R^{m_i*n}$, $\sum_{i=1}^{N} m_i = m$, $x_i \in R^n$ is the local variable, $z \in R^n$ is the global consensus variable, $f_i : R^n \to R$ is the cost function, and $g : R^n \to R \bigcap \{\infty\}$ is the regularization function.

In the formula (2), the objective function $f(x, D)$ is divided into N sub-problems $f_i(x_i, D_i)$, and the local variable x_i is required to be consistent with the global variable z, so it is very suitable for a parallel solution in the distributed environment. The distributed ADMM algorithm is implemented by MapReduce in a study by Lubell-Doughtie et al. [3] and by MPI in a study by Taylor et al. [4]. The distributed ADMM algorithms implemented in [3] and [4] are synchronous. Due to the difference in computing and communication performance between different nodes, the synchronization overhead becomes the bottleneck for shortening the running time of the algorithm, and the asynchronous distributed ADMM algorithm becomes a new research hotspot [5, 6, 7]. Compared with the synchronous ADMM algorithm, the asynchronous ADMM algorithm can better solve the "slow node" problem caused by network delay and the difference between nodes in the synchronous algorithm [7], as well as improve the convergence speed of the algorithm. How-

ever, the asynchronous ADMM algorithm often needs more iterations to make it converge. As the size of the distributed system increases, the algorithm makes convergence more difficult to achieve. Moreover, the increase in the number of nodes also makes the central node overloaded, affecting the overall performance of the system. However, with the advances of computer hardware technology, it is very common for a single machine to have multicore and multiprocessor in the modern high-performance distributed system. Message Passing Interface (MPI), which is one of the important means of parallel program development in multicore systems, has different communication mechanisms between the intra-nodes and inter-nodes: the processes in the same node transmit data through shared memory or cache, while processes in different nodes transmit data through the network interface [8, 9]. Moreover, the different communication mechanisms cause the unbalanced arrival problem [8].

In order to solve the problems such as the distributed ADMM algorithm has slow convergence speed, the center node is overloaded, the unbalanced arrival problem in the multicore cluster, and to improve the system communication efficiency and speed up algorithm convergence, we propose a group-based asynchronous distributed alternating direction method of multipliers (GAD-ADMM). In our proposal, every process is viewed as a unit, and all the workers are grouped according to the process allocation in the multicore cluster and the model similarity of datasets. One worker is selected as the group leader in each group. Each worker is responsible for the update of local variable and dual variable, and then sends the local variable to the group leader to update the group variable. Finally, the master collects the group variables to update the global variable. The master only communicates with the group leaders, and the worker communicates with the master indirectly through the group leader, thus reducing the communication between nodes and the load of the master. In order to solve the unbalanced arrival problem, the GAD-ADMM algorithm adopts an asynchronous protocol between groups.

The main contributions of this paper can be summarized as follows:

1. We propose a group-based asynchronous distributed ADMM, which improves the communication efficiency by reducing the communication between nodes in a multicore cluster, and accelerates the convergence of the algorithm by relaxing the constraint conditions on global consistency.
2. The asynchronous communication protocol is used between groups to further improve the convergence speed of the algorithm on the premise of ensuring the convergence of the algorithm.
3. Instead of directly transmitting the group local variable and dual variable, the group leader first performs related operations on the group local variable and dual variable before transmitting the result to the master, further reducing the communication between nodes and reducing the calculation load of the master.
4. The GAD-ADMM algorithm is implemented in a high performance multicore distributed cluster. The benchmark experiments show that the GAD-ADMM

algorithm can reduce the number of external iterations, improve communication efficiency, and reduce the total time cost of the system by 35 % compared with the AD-ADMM algorithm.

The remainder of the paper is organized as follows. The background and related work of ADMM are introduced in Section 2. Section 3 introduces the GAD-ADMM. The theoretical analysis of the GAD-ADMM is presented in Section 4. In Section 5, experiments on logistic regression (LR) solved by the GAD-ADMM are presented. Finally, we present the conclusion of this paper in Section 6.

2 BACKGROUND AND RELATED WORK

Many machine learning problems can be transformed into global consensus optimization problems. The distributed ADMM algorithm based on global consistency is used to solve the SVM problem by Zhang et al. [10] and solve the logistic regression problem by Lubell-Doughtie et al. [3]. The ADMM algorithm is introduced by Boyd et al. [2] to solve the global consensus optimization problem. The iterative formula is shown as follows:

$$x_i^{k+1} := \underset{x_i}{\operatorname{argmin}} \left(f_i(x_i, D_i) + \frac{\rho}{2} \left\| x_i + \frac{1}{\rho} y_i^k - z^k \right\|_2^2 \right), \tag{3}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \left(g(z) + \frac{\rho}{2} \sum_{i=1}^{N} \left\| x_i^{k+1} + \frac{1}{\rho} y_i^k - z \right\|_2^2 \right), \tag{4}$$

$$y_i^{k+1} := y_i^k + \rho \left(x_i^{k+1} - z^{k+1} \right) \tag{5}$$

where $y_i \in R^N$ is the dual variable, ρ is the penalty parameter, and $\left\| x + \frac{1}{\rho} y - z \right\|$ is the penalty term. It is shown in Equations (3), (4) and (5) that the updates of local variables and dual variables can be executed parallel in different nodes, while the aggregation of all local variables and dual variables is desired to solve the global variable. Generally, one master can be used to update the global variable, and N workers can be used to update local variables and dual variables independently. Due to the fault tolerance of the machine learning algorithm and the decomposable characteristics of the algorithm, moderate relaxation of the accuracy requirements of the iterative process can make the algorithm converge faster.

According to different communication topologies, distributed ADMM algorithms can be classified into point-to-point mode [11] and master-slave mode [3, 12]. The global consensus optimization problems are usually solved using a master-slave mode. According to different communication protocols, distributed ADMM algorithms can be classified into synchronous and asynchronous distributed ADMM. In the synchronous distributed ADMM algorithm, the master must wait to receive the parameters of all workers before updating the global variable [2, 3, 4, 10], so the speed of the algorithm is limited by the slowest node. To solve this problem, Zhang

et al. [7] proposed an asynchronous distributed ADMM (async-ADMM). A new asynchronous distributed ADMM (AD-ADMM) proposed by Chang et al. [12], which does not need the loss function, must be a convex function. Chang et al. [13] further proved that the AD-ADMM had linear convergence. Unlike the synchronous distributed ADMM algorithm, the AD-ADMM master only needs to receive the local variables from a partial list of workers to update the global variable z. As the number of workers increases, the convergence of the distributed ADMM algorithm becomes more difficult. To solve this problem, Wang et al. [14] proposed a group-based distributed ADMM (GADMM): all the workers are grouped into several groups by model similarity in the GADMM algorithm, and then the group variables are used to replace local variables and update the global variable. The convergence speed of the algorithm is improved by relaxing the constraint on global consistency.

The distributed ADMM algorithms introduced above optimize the iterative format and transmission process of the distributed ADMM algorithm from different angles, but do not consider the difference of data communication between intra-nodes and inter-nodes. This kind of communication variability often leads to the unbalanced arrival problem, which slows down the convergence speed of the algorithm. However, as the number of distributed system nodes increases, the distributed ADMM algorithm converges slowly. The GAD-ADMM algorithm proposed in this paper adds a layer of grouping on the basis of the AD-ADMM algorithm framework, where the workers are grouped according to the distribution of processes in the distributed system and the model similarity of datasets. In addition, it uses group variables instead of local variables to update the global variable. To improve the algorithm efficiency and solve the unbalanced arrival problem in the multicore cluster, a reasonable grouping of the workers was adopted for the GAD-ADMM. Besides, the convergence speed of the GAD-ADMM algorithm was accelerated by relaxing the constraints of global consistency. This paper bridges the structural characteristics of distributed systems with the characteristics of distributed algorithms and proposes an efficient grouping asynchronous distributed ADMM (GAD-ADMM) algorithm. The GAD-ADMM is also different from the GADMM in two ways:

1. the GAD-ADMM fully considers the communication differences between processes in a multicore cluster, and takes these differences as the main factor of process grouping;
2. instead of a synchronous protocol, the asynchronous protocol is used between groups.

3 THE GROUP-BASED ASYNCHRONOUS DISTRIBUTED ADMM (GAD-ADMM)

We define the notations included in the rest of this paper as follows.

Definition 1. P represents the number of nodes in the system and Q_i represents the number of working processes in node P_i. P_{ij} represents the j^{th} process in the i^{th} node. All the processes in the node P_i are divided into M_i groups, and M is the total number of groups. M_1 is the number of groups of the processes of the node where the master is located. N is the total number of workers. W_{ij} represents the j^{th} worker in the G_i^{th} group. D_{ij} represents the dataset processed by P_{ij} or W_{ij}, and C_{ij} is the number of samples of D_{ij}. The data transfer rate in the node is V_{in} B/s while between nodes is V_{out} B/s. The number of bytes occupied by the parameter x is F_{dim} (the number of bytes occupied by the parameter y or z is also F_{dim}).

As the size of the data increases, the algorithm becomes more and more difficult to converge. The main purpose of the GAD-ADMM is to improve communication efficiency and speed up the convergence of the algorithm by grouping the processes. Based on the traditional master-slave mode, the GAD-ADMM adds a grouping layer to form a two-layer master-slave mode. After grouping, there are three different types of processes in the system: the master, the group leader and the worker. The entire algorithm framework of GAD-ADMM consists of four steps. First, the processes of workers are classified into several groups according to the process allocation of the system and model similarity of datasets. Each group selects one process as the group leader. Second, the worker updates local variable and dual variable, then sends the local variable to the group leader, and finally waits to receive the group variable and global variable from the group leader for the next update. Third, the group leader gathers all the local variables of the group to produce the group local variable and dual variable, then the group leader must send the group variable to the master and wait to receive the global variable from the master, and finally broadcast the group local variable and global variable to the workers. Finally, the master gathers the group variables from group leaders to update the global variable and then sends the updated global variable to the group leaders. Due to the differences between groups and network delay, we adopt an asynchronous communication protocol between groups. The master does not wait for the parameters of all groups to arrive, but only waits for the parameters of some groups (the number of groups can be set by the user) to arrive to update the global variable. All these steps, with the exception of the first one, are iterated until the algorithm converges.

3.1 The Decomposition of the Calculation and the Grouping Method of the Process

In a multicore cluster, each node may be assigned multiple processes, and each process processes an independent dataset, and updates the local variable and dual variable in parallel. In the iterative process, formulae (3), (4) and (5) can be further modified as follows:

$$x_{ij}^{k+1} := \underset{x_{ij}}{\operatorname{argmin}} \left(f_i(x_{ij}, D_{ij}) + \frac{\rho}{2} \left\| x_{ij} + \frac{1}{\rho} y_{ij}^k - z^k \right\|_2^2 \right), \tag{6}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \left(g(z) + \frac{\rho}{2} \sum_{i=1}^{P} \sum_{j=1}^{Q_i} \left\| x_{ij}^{k+1} + \frac{1}{\rho} y_{ij}^k - z \right\|_2^2 \right), \tag{7}$$

$$y_{ij}^{k+1} := y_{ij}^k + \rho \left(x_{ij}^{k+1} - z^{k+1} \right) \tag{8}$$

where D_{ij} represents the dataset processed by the process P_{ij}, and x_{ij} and y_{ij} represent the local variable and dual variable updated by P_{ij}, respectively. According to formula (7), the solution of the global variable needs to aggregate all local variables and dual variables. As the number of processes increases, the load on the master increases. This algorithm first performs group aggregation on local variables, and then uses the aggregate value of the group to update the global variable in order to reduce the load on the master. So how to divide the working processes into groups appropriately is considered in this phase. Considering the high cost of data communication between nodes, we first assign processes in the same node to the same group by default. Then, we analyze the impact of different groups on system time cost.

In each iteration, the master needs to aggregate the group variables of all groups, and then broadcast the global variable to each group. Therefore, the total system time (T_{total}) of the master includes the waiting time (T_{wait}), the calculation time (T_{cal}) and the sending time (T_{send}). The waiting time is determined by the update time of each group and the transfer time of model parameters. The calculation time is the time when the master calculates the global variable. The sending time is the time required for the master to transfer the global variable to the group leaders. The waiting time of each iteration can be expressed as follows:

$$T_{wait} = \overline{T_{update}} + T_{trans} \tag{9}$$

where $\overline{T_{update}}$ represents the average update time of all groups, and $\overline{T_{update}}$ is mainly determined by the update time of local variables in the workers, so the change of the number of groups does not have a great influence on it. T_{trans} is the transfer time, which can be computed as formula (10), and includes the time of the model parameters transferred between the master and the group leaders, and between the group leader and the workers:

$$T_{trans} = 3N \frac{F_{dim}}{V_{in}} + 3 \sum_{i=2}^{P} M_i \left(\frac{F_{dim}}{V_{out}} - \frac{F_{dim}}{V_{in}} \right). \tag{10}$$

The sending time can be computed as formula (11) and the calculation time can be computed as formula (12) in each iteration, in which T_{add} represents the time of

an accumulation operation:

$$T_{send} = (M - M_1)\frac{F_{dim}}{V_{out}} + M_1\frac{F_{dim}}{V_{in}}, \tag{11}$$

$$T_{cal} = \sum_{i=1}^{M} T_{add}. \tag{12}$$

Usually, $V_{in} > V_{out}$. It can be induced from (9), (10), (11) and (12) that when the dataset and the number of processes are constant, as the number of groups M increases, T_{trans}, T_{cal} and T_{send} increase, and $\overline{T_{update}}$ is basically unchanged. So the total time of the system in one iteration increases. That is to say, in each iteration, the fewer the number of groups, the higher the system communication efficiency. So we first divide the processes in the same node into a group by default.

After grouping processes, the group variables are used instead of local variables to update the global variable to speed up the convergence of the algorithm. Therefore, we use the similarity of the dataset as another criterion for grouping, that is, the model similarity of the dataset of the processes in the same group after grouping is higher. To achieve this purpose, the processes in the same node are further grouped according to model similarity. In this algorithm, we use Euclidean distance as the measure of similarity: the smaller the Euclidean distance, the larger the similarity of datasets. Measuring model similarity of datasets also requires expensive computing and communication overhead, so we use the similarity of the local variable x to replace the model similarity of the dataset as the measurement indicator. The Euclidean distance (Ed) between the process P_{ij} and the process P_{ik} can be calculated by formula (13):

$$Ed(P_{ij}, P_{ik}) = \|x_{ij}^1 - x_{ik}^1\|^2 \tag{13}$$

where $x_{ij}^1, x_{ik}^1 \in R^n$ represent the first updated values of the local variables of P_{ij} and P_{ik}, respectively. After calculating the similarity between processes, the L algorithm [15] is used to determine the number of internal groups, and the DIANA algorithm [18] is used to perform regrouping operations on each default group. The DIANA algorithm is a top-down hierarchical clustering algorithm, which needs to determine the number of groups in advance, while the L algorithm can automatically determine the number of groups in the hierarchical clustering algorithm [15]. Other clustering algorithms can also be used to regroup the processes. Local variables are used instead of datasets as indicators for similarity measurement. Although it is necessary to perform an update operation on the local variables in advance, the updated values can be used as the initial value of subsequent iteration updates, so it will not increase the total system time cost.

3.2 Group-Based Parallel Iterative Update Strategy

The workers are responsible for updating local variables and dual variables, which can be executed in parallel. Each worker first updates local variable and dual variable, then sends the local variable to the group leader, waits to receive the group variable and global variable from the group leader, and then updates the local variable again with the group variable. x_{ij} represents the local variable and y_{ij} represents dual variable of the j^{th} worker in the group G_i. Using formulae (14) and (15), each worker updates local variable and dual variable, respectively,

$$x_{ij}^{k_i+1} := \underset{x_{ij}}{\operatorname{argmin}}(f_i\left(x_{ij}, D_{ij}\right) + \frac{\rho}{2}\left\|x_{ij} + \frac{1}{\rho}y_{ij}^{k_i} - \widetilde{z}_{G_i}\right\|_2^2\Bigg), \tag{14}$$

$$y_{ij}^{k_i+1} := y_{ij}^{k_i} + \rho\left(x_{ij}^{k_i+1} - \widetilde{z}_{G_i}\right) \tag{15}$$

where $D_{ij} \in R^{m_{ij}*n}$, $x_{ij} \in R^n$, $y_{ij} \in R^n$, $\sum_{i=1}^{P}\sum_{j=1}^{Q_i} m_{ij} = m$, and $\widetilde{z}_{G_i}$ represents the latest z-value received by the group G_i. In the GAD-ADMM, the workers in the same group are synchronous, while in different groups they are asynchronous, so the workers in the same group have the same $\widetilde{z}_{G_i}$ while workers in different group may have different $\widetilde{z}_{G_i}$. The procedure for the worker is described in Algorithm 1.

Because the datasets are grouped according to model similarity, the local variable is set to the same as the group local variable. It is worth noting that since the local variable is updated at the time of grouping, the first update of the local variable is directly set equal to the initial value, which is the updated value of the local variable in the process grouping. Update of the local variable is the optimal value for solving the sub-problem (14). In this paper, we use the Trust Region Newton method (TRON) [16] to solve the sub-problem. Of course, other optimization methods are also applicable to this algorithm.

Algorithm 1: Group-based Asynchronous Distributed ADMM (GAD-ADMM): Processing by worker j in the group G_i

Initialize x_{ij}^0, y_{ij}^0 and set k_i=0.
set $x_{ij}^1 = x_{ij}^0$ and update $y_{ij}^{k_i+1}$ using (15)
send x_{ij}^1 to the group leader
repeat
  wait until receiving $\widetilde{z}_{G_i}$ and x_{G_i} from the group leader.
  set $k_i = k_i + 1$

$$x_{ij} = x_{G_i} \tag{16}$$

  update $x_{ij}^{k_i+1}, y_{ij}^{k_i+1}$ using (14), (15)
  send $x_{ij}^{k_i+1}$ to the group leader.
until *the stopping conditions are satisfied*;

3.3 Group-Based Parameter Aggregation Communication

After grouping the processes, the process with the minimum process number is selected as the group leader of the group. The group leader gathers all local variables from the workers in the group, then updates the group variable x_{G_i} and the group dual variable y_{G_i}. After the group leader updates the values of $x_{G_i}^{k_i+1}$ and $y_{G_i}^{k_i+1}$, the updated values can be directly sent to master. It can be seen from Equation (7) that, when solving the global variable, we need to sum up $y_i/\rho + x_i$. Therefore, the calculation can be performed prior to the transmission, leading the calculation result w_{G_i} (w_{G_i} should be computed as formula (17)) transferred to the master directly instead of x_{G_i} and y_{G_i}. In this case, the transfer time can be calculated by formula (18), it can be induced by formulae (10) and (18) that $T_{trans}^{'} < T_{trans}$:

$$w_{G_i}^{k_i+1} = y_{G_i}^{k_i+1}/\rho + x_{G_i}^{k_i+1}, \tag{17}$$

$$T_{trans}^{'} = 3N\frac{F_{dim}}{V_{in}} + 2\sum_{i=2}^{P} M_i \left(\frac{F_{dim}}{V_{out}} - \frac{F_{dim}}{V_{in}}\right). \tag{18}$$

According to the grouping method, the model similarity of a dataset between workers in the same group is relatively high, and workers in the same group are in the same node, so the synchronous communication protocol is adopted between workers in the group. Considering that the number of samples on each worker may be different, the group variable is calculated from the weighted average of local variables of all the workers in each group. We define η_{ij} as the ratio of dataset D_{ij} in the group G_i, and η_{ij} can be computed by formula (19). Then, the group variable x_{G_i} can be updated as shown in formula (20), and the group dual variable y_{G_i} can be updated using formula (21):

$$\eta_{ij} = C_{ij} / \sum_{W_{ij} \in G_i} C_{ij}, \tag{19}$$

$$x_{G_i}^{k_i+1} := \sum_{W_{ij} \in G_i} \eta_{ij} x_{ij}^{k_i+1}, \tag{20}$$

$$y_{G_i}^{k_i+1} := y_{G_i}^{k_i} + \rho\left(x_{G_i}^{k_i+1} - \widetilde{z}_{G_i}\right) \tag{21}$$

where $\widetilde{z}_{G_i}$ represents the latest z-value received by the group leader of group G_i. After this, the group leader sends w_{G_i} to the master, then waits to receive the global variable from the master, and finally broadcasts the group local variable and global variable to the workers. The whole procedure for the group leader is shown in Algorithm 2.

Algorithm 2: Group-based Asynchronous Distributed ADMM (GAD-ADMM): Processing by the group leader of the group G_i

Initialize $x_{G_i}, y_{G_i}, z_{G_i}$ and set k_i=0.
repeat
  wait x_{ij} from all workers in the group G_i
  update $\{x_{G_i}^{k_i+1}, y_{G_i}^{k_i+1}, w_{G_i}^{k_i+1}\}$ using (20), (21), (17)
  send $\{w_{G_i}^{k_i+1}\}$ to the master
  wait until receiving $\widetilde{z}_{G_i}$from the master.
  broadcast $\widetilde{z}_{G_i}$ and x_{G_i} to all workers in the group G_i.
  set $k_i = k_i + 1$
until *the stopping conditions are satisfied*;

3.4 Global Variable Data Synchronization and Update Based on Bounded Delay

The master waits to receive the group variables from the group leaders, updates the global variable z with the group variables, and then sends the new global variable to the corresponding group leaders. Only a partial synchronization is required for each iteration of the master, not full synchronization for all groups. The master only broadcasts the global variable to the corresponding groups in each iteration. However, since the number of processes in each group may be different, we use the weighted value of the grouped variables to solve the global variable. Assuming that there are N_{G_i} processes in the group G_i, the global variable can be updated using formulae (22) and (23).

$$w_{G_i}^{k+1} = \begin{cases} \widetilde{w}_{G_i}, & \forall G_i \in A_k, \\ w_{G_i}^{k}, & \forall G_i \in A_k^c, \end{cases} \tag{22}$$

$$z^{k+1} := \underset{z}{\operatorname{argmin}} \left(g(z) + \frac{\rho}{2} \sum_{i=1}^{M} \frac{MN_{G_i}}{N} \left\| w_{G_i}^{k+1} - z \right\|_2^2 + \frac{\theta}{2} \left\| z - z^k \right\|_2^2 \right) \tag{23}$$

where A_k represents the index subset of group leaders received by the master in iteration k, A_k^c represents the complementary set of A_k, and ρ and θ are the penalty parameters. The penalty term $\left\| z - z^k \right\|_2^2$ is added to further ensure the convergence of the algorithm. The whole procedure for the master is shown in Algorithm 3.

In the GAD-ADMM, the bulk synchronous parallel (BSP) mode is adopted between workers in the same group, while the stale synchronous parallel (SSP) [17] mode is adopted between groups. The workers in the same group have the same iteration cycles and between groups may have different iteration cycles. The SSP mode achievement involves several steps. First, the minimum synchronized block size is set to A ($M \geq A \geq 1$), indicating that the global variable z will not be updated until the master has successfully received the group variables from A group

Algorithm 3: Group-based Asynchronous Distributed ADMM (GAD-ADMM): Processing by the master

Initialize z and set $k = 0$, $d_{G_i} = \ldots = d_{G_M} = 0$.

repeat

wait until receiving a minimum of A updated variables from the group leaders and $d_{G_i} \leq d$ for all $i \in \{1, \ldots, M\}$

update

$$d_{G_i} = \begin{cases} 0 : \forall G_i \in A_k \\ d_{G_i+1} : \forall G_i \in A_k^c \end{cases} \tag{24}$$

update z^{k+1} using (23).

broadcast z^{k+1} to the group leaders in A_k.

set $k = k + 1$

until *the stopping conditions are satisfied*;

leaders. Second, the maximum delay cycle is set to d ($d \geq 1$), and every group must update at least once in this cycle. Every group leader has its own delay counter d_{G_i}, which is stored on the master. When the w_{G_i} from the group G_i arrives at the master, the corresponding d_{G_i} is set to 0, otherwise, d_{G_i} is increased by one as the master's counter k increments. The d_{G_i} for each group must be less than d. The GAD-ADMM reduces to synchronous when A is equal to M or d is set to 1. Figure 1 shows the timing diagram of the GAD-ADMM when the threshold A is set to 2 and cycle d is set to 3.

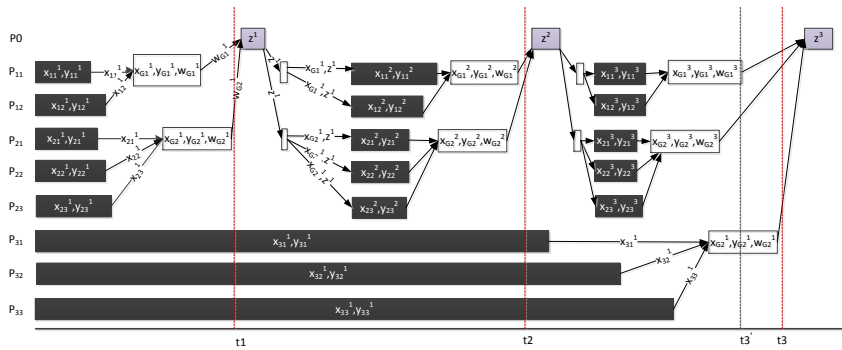


Figure 1. The timing diagram of the GAD-ADMM ($A = 2$, $d = 3$): P_0 represents the master, and P_{ij} represents the j^{th} worker in the group G_i. In this figure, there are 8 workers grouped into 3 groups, in which P_{11}, P_{21} and P_{31} are the group leaders of the corresponding groups. The number in the upper right corner of the variable represents the number of iterations.

4 THEORETICAL ANALYSIS

In this section, we analyze the convergence of the GAD-ADMM and further compare the convergence speed of the GAD-ADMM with the AD-ADMM.

4.1 Convergence Analysis

In this section, we analyze the convergence of the GAD-ADMM, in which the augmented Lagrangian for (2) can be modified to the following formula:

$$L_\rho(x,y,z)=\sum_{i=1}^{M}f_i(x_{G_i})+g(z)+\sum_{i=1}^{M}\frac{MN_{G_i}}{N}\left(\langle y_{G_i},x_{G_i}-z\rangle+\frac{\rho}{2}\|x_{G_i}-z\|_2^2\right). \quad (25)$$

First, we make the assumption as follows to simplify analysis.

Assumption 1. The function g is proper, closed and convex; Each function f_i is a convex function, and there is a constant $\mathrm{L}\geq 0$ such that the gradient of f_i satisfies the Lipschitz continuous condition; Moreover, problem (2) has an optimal solution $f^\star$, which is bounded; Set $d\geq 1$ as the maximum bounded delay; For all $i\in\{1,\ldots,M\}$ and $k\geq 0$, it must be that $i\in A_k\bigcup A_{k-1}\bigcup\ldots\bigcup A_{max(k-d+1,-1)}$, and there exists a constant $B\in[1,M]$ such that $|A_k|\leq B$ for all k ($|A_k|$ represents the number of A_k).

Theorem 1. If Assumption 1 is true, then the sequence of $\left(\{x_i^k\}_{i=1}^N,\{y_i^k\}_{i=1}^N,z^k\right)$ generated by the GAD-ADMM is bounded and has limit points, which satisfy the KKT conditions of problem (2) under the condition that formulae (26), (27), (28) are established:

$$\infty > L_\rho\left(x^0,z^0,y^0\right)-f^\star\geq 0, \quad (26)$$

$$\rho\geq\left(\left(1+L^2\right)+\sqrt{\left(1+L^2\right)^2+8L^2}\right)/2, \quad (27)$$

$$\theta > \left(B\left(1+\rho^2\right)(d-1)^2-M\rho\right)/2. \quad (28)$$

The proof of Theorem 1 is similar to that of Corollary 1 in [12]. It is implied by Theorem 1 that the GAD-ADMM is guaranteed to converge to the set of KKT points so long as ρ and θ are large enough. The reciprocal of θ can be considered as the step size of z.

4.2 Convergence Speed Analysis

We further analyze the convergence speed of the GAD-ADMM and the AD-ADMM in this section. First, we analyze the update speed of the local variable for each worker. We define the objective function of the sub-problem as follows:

$$F(x_i)=f(x_i)+h(x_i) \quad (29)$$

where $h(x_i) = \frac{\rho}{2}\|x_i - z + \frac{1}{\rho}y_i\|^2$, $f(x_i)$ is the loss function and $f(x_i)$ is convex. We also denote $x_i^{k_i}$ as the optimal value of the worker i in iteration k_i.

Lemma 1. Assuming that the AD-ADMM and the GAD-ADMM have the same method for updating the global variable z, all N workers in the GAD-ADMM are classified into M groups. For any $k_i \geq 1$, the local variable $x_i^{k_i}$ sequence satisfies the formula (30):

$$\sum_{i=1}^{N} F(x_i^{k_i}) \geq \sum_{i=1}^{M} N_i F\left(\sum_{W_{ij}\in G_i} \eta_{i,j} x_{ij}^{k_i}\right) \tag{30}$$

where N_i represents the number of the workers in the group G_i, and $\eta_{i,j}$ is the weight of W_{ij} in the group G_i.

Proof. We first analyze the situation that there are only two workers (W_a and W_b) in the group, and we define x_c is the average of the local variables x_a and x_b. In the GAD-ADMM, the workers in the same group are synchronous, so the workers in the same group have the same k_i and $\widetilde{z}_{G_i}$. Because $f(x)$ is convex, we can induce that

$$f(x_a) + f(x_b) \geq 2f(x_c) \tag{31}$$

when $k_i = 2$, it can be deduced from (14), (15) and (31) that

$$\begin{aligned} &F(x_a^2) + f(x_b^2) - 2f(x_c^2) \geq (\rho/2)\left\|x_a^2 - \widetilde{z}_{G_i}^1 + (1/\rho)y_a^1\right\|^2 \\ &+ (\rho/2)\left\|x_b^2 - \widetilde{z}_{G_i}^1 + (1/\rho)y_b^1\right\|^2 - \rho\left\|x_c^2 - \widetilde{z}_{G_i}^1 + (1/\rho)y_c^1\right\|^2 \geq (\rho/2)\left\|x_a^2 + x_a^1 - \widetilde{z}_{G_i}^1\right\|^2 \\ &+ (\rho/2)\left\|x_b^2 + x_b^1 - \widetilde{z}_{G_i}^1\right\|^2 - (\rho/4)\left\|x_a^2 + x_a^1 + x_b^2 + x_b^1 - 2\widetilde{z}_{G_i}^1\right\|^2. \end{aligned} \tag{32}$$

Similar, when $k_i > 2$,

$$\begin{aligned} F(x_a^{k_i}) + f\left(x_b^{k_i}\right) - 2f\left(x_c^{k_i}\right) &\geq (\rho/2)\left\|\sum_{i=1}^{k^i} x_a^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 \\ &+ (\rho/2)\left\|\sum_{i=1}^{k^i} x_b^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 - \rho\left\|\sum_{i=1}^{k^i} x_c^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 \\ &\geq (\rho/2)\left\|\sum_{i=1}^{k^i} x_a^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 + (\rho/2)\left\|\sum_{i=1}^{k^i} x_b^i - \sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 \\ &- (\rho/4)\left\|\sum_{i=1}^{k^i} x_b^i + \sum_{i=1}^{k^i} x_b^i - 2\sum_{i=1}^{k^i-1} \widetilde{z}_{G_i}^i\right\|^2 \geq 0. \end{aligned} \tag{33}$$

If there are p workers in a group, then in the k_i^{th} iteration, it can be deduced that

$$\sum_{i=1}^{p} F\left(x_i^{k_i}\right) - pF\left(\frac{1}{p}\sum_{i=1}^{p} x_i^{k_i}\right) \geq (\rho/2)\sum_{j=1}^{p}\left\|\sum_{i=1}^{k_i} x_j^i - \sum_{i=1}^{k_i-1}\widetilde{z}_{G_i}^i\right\|^2 - \frac{\rho}{2p}\left\|\sum_{i=1}^{k_i}\sum_{j=1}^{p} x_j^i - p\sum_{i=1}^{k_i-1}\widetilde{z}_{G_i}^i\right\|^2 \geq 0. \quad (34)$$

When all workers are divided into M groups, and the number of workers in the G_i group is N_i, the difference between the original function $F(x_i)$ and the function $F(x_i)$ after grouping is:

$$\sum_{i=1}^{N} F\left(x_i^{k_i}\right) - \sum_{i=1}^{M} N_i F\left(\sum_{W_{ij}\in G_i}\eta_{i,j}x_{ij}^{k_i}\right) = \sum_{i=1}^{N} F\left(x_i^{k_i}\right) - \sum_{i=1}^{M} N_i F\left(\frac{1}{N_i}\sum_{j=1}^{N_i} x_{ij}^{k_i}\right) \geq \sum_{i=1}^{M}\left\{\sum_{j=1}^{N_i}\left\|\sum_{i=1}^{k_i-1}\widetilde{z}_{G_i}^i\right\|^2 - \frac{\rho}{2N_i}\left\|\sum_{i=1}^{k_i}\sum_{j=1}^{N_i} x_{ij}^i - N_i\sum_{i=1}^{k_i-1}\widetilde{z}_{G_i}^i\right\|^2\right\} \geq 0. \quad (35)$$

□

The main difference between the GAD-ADMM and the AD-ADMM is based on the sum of the local variable $x_i^{k_i}$ sequence, so $F(\eta x_i)$ is going down faster than $F(x_i)$.

Moreover, we define the objective function of the AD-ADMM and the GAD-ADMM as follows:

$$G(x^k, z^k) = \sum_{i=1}^{N} f_i(x_i^k) + g(z^k), \quad (36)$$

$$\bar{G}\left(\bar{x}^k, \bar{z}^k\right) = \sum_{i=1}^{N} f_i\left(\bar{x}_i^k\right) + g\left(\bar{z}^k\right). \quad (37)$$

Theorem 2. If $g(z) = \beta\|z\|^2$and the synchronous communication protocol is used both in the GAD-ADMM and the AD-ADMM, then

$$G(x^k, z^k) - \bar{G}(\bar{x}^k, \bar{z}^k) \geq \frac{\rho^2 M(N-M)}{N^2(2\beta + M\rho)^2(2\beta + N\rho)}\left\|v_i^{k+1}\right\|^2 \quad (38)$$

where $v_i^{k+1} = \sum_{i=1}^{N}\left(x_i^{k+1} + \frac{1}{\rho}y_i^{k+1}\right)$.

Proof. Since $g(z) = \beta\|z\|^2$, then it can be induced by (16),(20) and (23) that

$$\bar{x_i}^k = \sum_{W_{ij} \in G_i} \eta_{i,j} x_{ij}^k, \tag{39}$$

$$\bar{z}^k = \frac{M\rho}{N(2\beta + M\rho)} \sum_{i=1}^{M} N_i \left(x_{G_i}^k + (1/\rho) y_{G_i}^k \right). \tag{40}$$

Since $f(x)$ is a convex function and non-negative, when the N workers are grouped into M groups, the loss function of sub-problem satisfies:

$$\sum_{i=1}^{N} f_i(x_i^k) \geq \sum_{i=1}^{M} N_i f_i \left(\frac{1}{N_i} \sum_{W_{ij} \in G_i} x_{ij}^k \right) \geq \sum_{i=1}^{M} N_i f_i \left(\sum_{W_{ij} \in G_i} \eta_{i,j} x_{ij}^k \right). \tag{41}$$

It can be induced that

$$z^{k+1} = \frac{\rho}{2\beta + N\rho} \sum_{i=1}^{N} \left(x_i^{k+1} + (1/\rho) y_i^{k+1} \right), \tag{42}$$

$$\begin{aligned} \bar{z}^k &= \frac{M\rho}{N(2\beta + M\rho)} \sum_{i=1}^{M} N_i \left(x_{G_i}^k + (1/\rho) y_{G_i}^k \right) \\ &\leq \frac{M\rho}{N(2\beta + M\rho)} \sum_{i=1}^{M} \sum_{W_{ij} \in G_i} \left(x_{ij}^{k+1} + (1/\rho) y_{ij}^{k+1} \right) \\ &\leq \frac{M\rho}{N(2\beta + M\rho)} \sum_{i=1}^{N} \left(x_i^{k+1} + (1/\rho) y_i^{k+1} \right) \leq z^{k+1}, \end{aligned} \tag{43}$$

$$\begin{aligned} g\left(z^{k+1}\right) - g\left(\bar{z}^{k+1}\right) &\geq g\left(\frac{\rho v_i^{k+1}}{2\beta + N\rho} \right) - g\left(\frac{M\rho v_i^{k+1}}{N(2\beta + M\rho)} \right) \\ &\geq \frac{M(\rho v_i^{k+1})^T}{N(2\beta + M\rho)} \left(\frac{\rho v_i^{k+1}}{2\beta + N\rho} - \frac{M\rho v_i^{k+1}}{N(2\beta + M\rho)} \right) \\ &= \frac{\rho^2 M(N-M)}{N^2(2\beta + M\rho)^2(2\beta + N\rho)} \left\| v_i^{k+1} \right\|^2. \end{aligned} \tag{44}$$

Therefore, it can be induced by (41) and (44) that

$$G\left(x^k, z^k\right) - \bar{G}\left(\bar{x}^k, \bar{z}^k\right) \geq g\left(z^{k+1}\right) - g\left(\bar{z}^{k+1}\right) \geq \frac{\rho^2 M(N-M)}{N^2(2\beta + M\rho)^2(2\beta + N\rho)} \left\| v_i^{k+1} \right\|^2. \tag{45}$$

□

Theorem 2 indicates that when $M < N$, the difference between the object function of the AD-ADMM and the GAD-ADMM is greater than 0, and the smaller the M, the greater the difference. It also indicates that when M is equal to N, the GAD-ADMM is equal to the AD-ADMM.

5 EXPERIMENTS AND DISCUSSION

In this section, we solve the logistic regression problem with L2 regularization by the GAD-ADMM and the AD-ADMM [12]. The problem is described as follows:

$$\min \frac{1}{N} \sum_{i=1}^{N} l_i \left(D_i^T x_i - b_i \right) + \beta \|z\|^2, \quad \text{s.t. } x_i = z, i = 1, 2, \ldots, N. \tag{46}$$

where $D \in R^{m*n}$ is the sample dataset, m represents the number of samples, n represents the number of features of samples, $b_i \in \{0, 1\}$ is the label of the sample, and $\beta > 0$ is the scalar regularization parameter.

5.1 Experimental Environment and Parallel Implementation

The experimental platform of this paper is the Ziqiang 4000 high-performance cluster of Shanghai University. Each node of the cluster has an Intel E5-2690 CPU (2.9 GHz/8-core) processor and 64 GB memory (RDIMM DDR3 1 600 MHz), the network is gigabit Ethernet. We use the kdd2010 (bridge to algebra)[1] as the datasets, which contains 19 264 097 training samples and 1 163 024 features. All datasets are divided into training/testing datasets at a ratio of 3:1 for experimental testing. We implement the algorithm using the MPI implementation MPICH v3.2.1[2] as the inter-processor communication and C++ as the programming language.

The parallel implementation of GAD-ADMM and AD-ADMM uses eight compute nodes, and each node in the system is allocated with eight processes. We select one process as the master and other processes as the workers. The scalar regularization parameter β is set to 2. In the GAD-ADMM algorithm, we use the L algorithm to obtain the total number of groups M, which is eight.

The TRON [16] method is used to solve the local variables. The parameter ρ is set to 6 and the super parameter C is set to 1. We use the primal residual r and dual residual s as the stop conditions, which should satisfy the conditions shown in (47) and (48) to stop the iteration.

$$\|r^k\|_2 \leq \text{ABS} * \sqrt{Mn} + \text{REL} * \max\left\{ \|x_{G_i}^k\|_2, \|z^k\|_2 \right\}, \tag{47}$$

$$\|s^k\|_2 \leq \text{ABS} * \sqrt{Mn} + \text{REL} * \left\| \rho y_{G_i}^k \right\|_2 \tag{48}$$

[1] https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

[2] http://www.mpich.org/

where $\left\|r^k\right\|_2^2 = \sum_{i=1}^{M} \left\|x_{G_i}^k - z^k\right\|_2^2$, $\left\|s^k\right\|_2^2 = M\rho^2 \left\|z^k - z^{k-1}\right\|_2^2$. The absolute error ABS and relative error REL are both set to 0.001.

5.2 Experiment Test and Analysis

In this section, the performance of the GAD-ADMM is compared with the AD-ADMM in terms of the convergence speed, the system cost and the accuracy. Furthermore, the influences caused by the different parameters and different groups are also analyzed for the GAD-ADMM.

5.2.1 Convergence and Convergence Speed Test

In this section, the convergence of the GAD-ADMM and the AD-ADMM will be tested. The convergence of these two algorithms is compared when different values of threshold A are taken. We set $d = 5$, $\theta = 0$ in both of the algorithms. The results are shown in Figures 2 and 3.

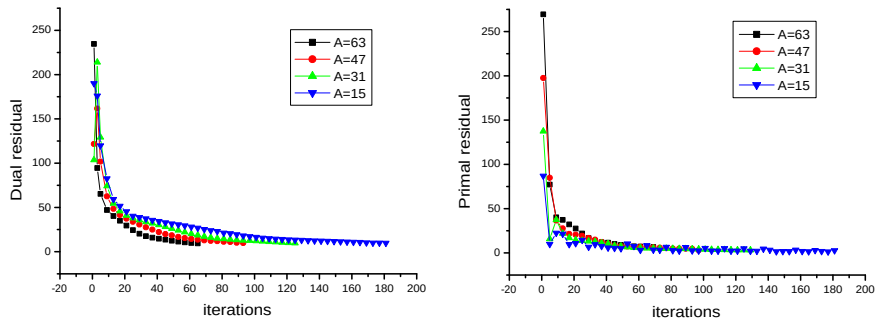


Figure 2. The convergence of the AD-ADMM

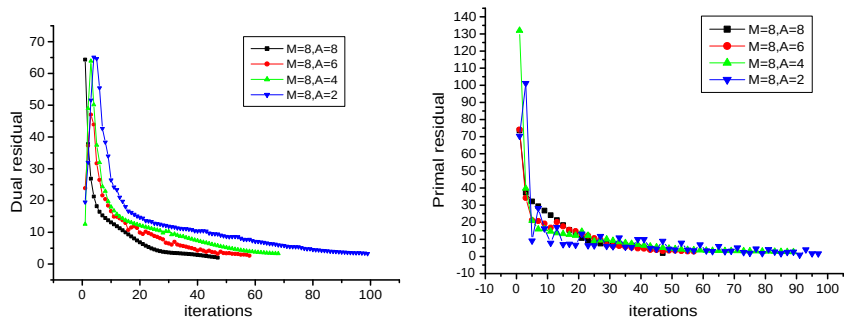


Figure 3. The convergence of the GAD-ADMM

Figures 2 and 3 show that the GAD-ADMM algorithm requires fewer iterations than that of the AD-ADMM to converge in a peer-to-peer situation. This is because in the GAD-ADMM, we relax the constraint conditions on global consensus, and the objective function is faster to converge as analyzed in Section 4. When M is fixed, the bigger the value of threshold A, the fewer the iterations required to converge. Because the more information the master receives from the leaders in each iteration, the less the "old values" used to update z, and the faster the algorithm converges. When A is larger, it may take longer for the master to wait for receiving, which shown in the next section.

5.2.2 System Time Cost and Accuracy Test

This section tests the system time cost and accuracy of the AD-ADMM and the GAD-ADMM. The system time cost includes the waiting time, the calculation time and the sending time. The accuracy of the system (Accur) is computed as follows:

$$\text{Accur} = (N_{tp} + N_{tz})/N_{total} \tag{49}$$

where N_{tp} represents the number of "true positive", N_{tz} represents the number of "true zero" and N_{total} represents the total number of testing datasets. The parameters of this section are set in accordance with Section 5.2.1. Figure 4 shows the system time cost of the master of the AD-ADMM and the GAD-ADMM in different thresholds. Figure 5 shows the accuracy of these two algorithms.

Figure 4 shows that the GAD-ADMM algorithm requires less system time than that of the AD-ADMM in a peer-to-peer situation. This is because only the group leader communicates directly with the master in the GAD-ADMM, which reduces the traffic between nodes. Thus, the waiting time and sending time for each iteration are reduced. At the same time, according to Equation (12), the smaller the number of groups, the less time each calculation takes. Furthermore, the smaller the number of groups, the less the number of external iterations, so the total system time cost is reduced. If M is constant, when the threshold value A decreases, the total number of iterations increases, and the calculation time increases, but the waiting time and sending time for each iteration decreases. So selecting appropriate A can minimize the system total time.

Figure 5 shows that compared with the AD-ADMM, the accuracy of the GAD-ADMM decreases, that is because the convergence conditions are relaxed. It also shows that the accuracy decreases less than 0.2 % when the total time is reduced by at least 35 % under peer conditions.

5.2.3 System Time Cost Test with Different Parameters

In Section 3.3, we mentioned that the leader can send parameters x and y to the master respectively, or send parameter w to the master. Figure 6 shows the system time allocation of the GAD-ADMM and the AD-ADMM in these two cases when the threshold A is equal to the group number M.

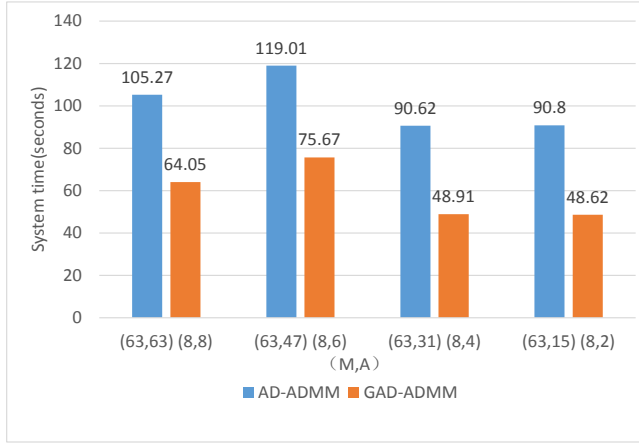


Figure 4. The system time cost of the AD-ADMM and the GAD-ADMM: A is the threshold and M is the number of groups

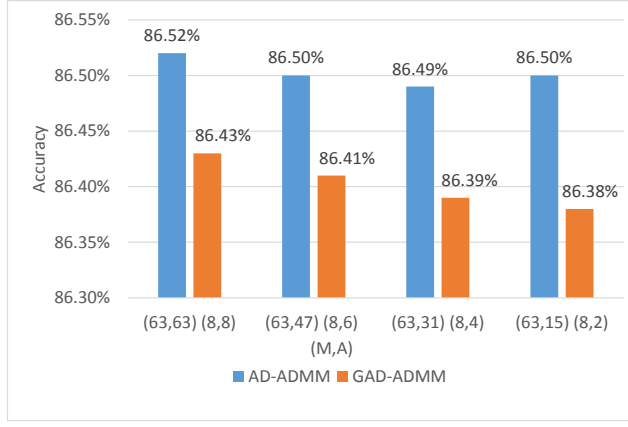


Figure 5. The accuracy of the AD-ADMM and the GAD-ADMM: A is the threshold and M is the number of groups

Figure 6 shows that when the group leader sends w to the master, the waiting time is reduced compared to sending x and y. This is because the parameters that the group leader transmits to the master each time are reduced by nearly half. Compared with the AD-ADMM, the GAD-ADMM does not have obvious advantages when transferring w. That is because the number of processes communicating directly with the master decreases after grouping, and the performance improvement is not obvious when the system bandwidth is high.

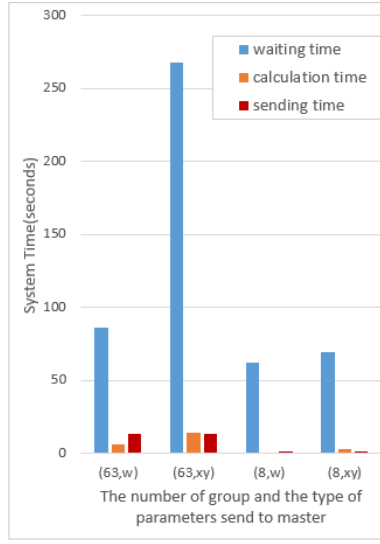


Figure 6. The system time allocation of the AD-ADMM and the GAD-ADMM when sending different types of parameters

5.2.4 System Time Cost Test of GAD-ADMM with Different Groups

In order to analyze the effects of different groups on the system time cost of the GAD-ADMM, the processes within the nodes are divided into groups 1, 2, 4 and 8 (the number of groups of the node where the master is located is 7), that is, M is taken as 8, 16, 32, and 63. Figure 7 shows the allocation of system time cost of the master under different groups and different thresholds. It shows that the system time cost decreases as the number of groups decreases in peer to peer conditions. The main reason is that the reduction of the number of groups makes the system converge faster. Another reason is that the system communication efficiency is improved by reducing the communication between nodes.

6 CONCLUSION

The GAD-GADMM algorithm is proposed in this paper for global consensus optimization problem. A grouping layer is added to the star topology, and the processes are grouped according to the process allocation in the multicore cluster and the model similarity of datasets. The convergence speed of the algorithm is improved by relaxing the global consistency constraint and reducing the communication between nodes. In the GAD-ADMM, the workers in the same group are synchronous while in different groups they are asynchronous, and we use partial barrier and bounded delay to guarantee the convergence of the asynchronous algorithm. In order to re-

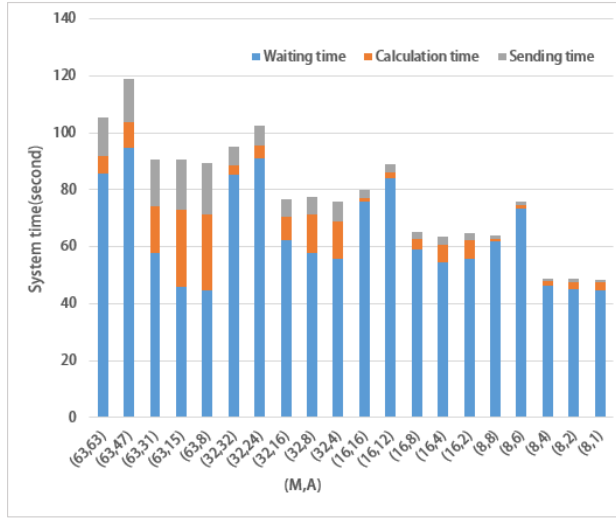


Figure 7. The system time allocation of the GAD-ADMM with different groups (M) and thresholds (A)

duce the master load, the parameter that the group leaders send to the master is w, not x and y, thus further reducing the system time cost. The convergence of the GAD-ADMM is proved both in theory and experiments. We also explained why the GAD-ADMM has a faster convergence speed than the AD-ADMM. The experiments on LR problem show that compared with the AD-ADMM, the GAD-ADMM can reduce the total system time by at least 35 % when the accuracy is reduced less than 0.2 % under peer conditions. Finally, the effects of different parameters and different number of groups on the performance of the GAD-ADMM algorithm were analyzed. The grouping method proposed in this paper first divides the processes in the same node into a group by default, and then further groups the default groups. The minimum number of groups is the number of nodes. It is not considered that the number of groups is less than the number of nodes, which will be studied in the future.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under grant No. U1811461.

REFERENCES

[1] Xing, E. P.—Ho, Q.—Xie, P.—Wei, D.: Strategies and Principles of Distributed Machine Learning on Big Data. Engineering, Vol. 2, 2016, No. 2, pp. 179–195, doi: 10.1016/j.eng.2016.02.008.

[2] Boyd, S.—Parikh, N.—Chu, E.—Peleato, B.—Eckstein, J.: Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. Foundations and Trends in Machine Learning, Vol. 3, 2011, No. 1, pp. 1–122, doi: 10.1561/2200000016.

[3] Lubell-Doughtie, P.—Sondag, J.: Practical Distributed Classification Using the Alternating Direction Method of Multipliers Algorithm. IEEE International Conferences on Big Data, Vol. 1, 2013, No. 1, pp. 773–776, doi: 10.1109/bigdata.2013.6691651.

[4] Taylor, G.—Burmeister, R.—Xu, Z.—Singh, B.—Patel, A.—Goldstein, T.: Training Neural Networks Without Gradients: A Scalable ADMM Approach. Proceedings of the 33rd International Conference on Machine Learning (ICML 2016). The Journal of Machine Learning Research: Workshop and Conference Proceedings, Vol. 48, 2016, pp. 2722–2731.

[5] Wei, E.—Ozdaglar, A.: On the $O(1/k)$ Convergence of Asynchronous Distributed Alternating Direction Method of Multipliers. IEEE Global Conference on Signal and Information Processing, 2013, pp. 551–554, doi: 10.1109/globalsip.2013.6736937.

[6] Shi, W.—Ling, Q.—Yuan, K.—Wu, G.—Yin, W.: On the Linear Convergence of the ADMM in Decentralized Consensus Optimization. IEEE Transactions on Signal Processing, Vol. 62, 2014, No. 7, pp. 1750–1761, doi: 10.1109/tsp.2014.2304432.

[7] Zhang, R.—Kwok, J. T.: Asynchronous Distributed ADMM for Consensus Optimization. Proceedings of the 31th International Conference on Machine Learning (ICML 2014). The Journal of Machine Learning Research: Workshop and Conference Proceedings, Vol. 32, 2014, pp. II-1701-II-1709.

[8] Patarasuk, P.—Yuan, X.: Efficient MPI Bcast across Different Process Arrival Patterns. IEEE International Symposium on Parallel and Distributed Processing, 2008, pp. 1–11, doi: 10.1109/ipdps.2008.4536308.

[9] Tipparaju, V.—Nieplocha, J.—Panda, D.: Fast Collective Operations Using Shared and Remote Memory Access Protocols on Clusters. Proceedings of the 17th International Symposium on Parallel and Distributed Processing (IPDPS '03), 2003, Art. No. 84, doi: 10.1109/ipdps.2003.1213188.

[10] Zhang, C.—Lee, H.—Shin, K. G.: Efficient Distributed Linear Classification Algorithms via the Alternating Direction Method of Multipliers. Proceedings of the 15th International Conference on Artificial Intelligence and Statistics, La Palma, Spain, 2012, pp. 1398–1406.

[11] Mota, J. F. C.—Xavier, J. M. F.—Aguiar, P. M. Q.—Püschel, M.: D-ADMM: A Communication-Efficient Distributed Algorithm for Separable Optimization. IEEE Transactions on Signal Processing, Vol. 61, 2013, No. 10, pp. 2718–2723, doi: 10.1109/tsp.2013.2254478.

[12] Chang, T. H.—Hong, M.—Liao, W. C.—Wang, X.: Asynchronous Distributed ADMM for Large-Scale Optimization – Part I: Algorithm and Convergence Analysis. IEEE Transactions on Signal Processing, Vol. 64, 2016, No. 12, pp. 3118–3130, doi: 10.1109/TSP.2016.2537271.

[13] Chang, T. H.—Hong, M.—Liao, W. C.—Wang, X.: Asynchronous Distributed ADMM for Large-Scale Optimization – Part II: Linear Convergence Analysis and Numerical Performance. IEEE Transactions on Signal Processing, Vol. 64, 2016, No. 12, pp. 3131–3144, doi: 10.1109/TSP.2016.2537261.

[14] Wang, H.—Gao, Y.—Shi, Y.—Wang, R.: Group-Based Alternating Direction Method of Multipliers for Distributed Linear Classification. IEEE Transactions on Cybernetics, Vol. 47, 2017, No. 11, pp. 3568–3582, doi: 10.1109/tcyb.2016.2570808.

[15] Salvador, S.—Chan, P.: Determining the Number of Clusters/Segments in Hierarchical Clustering/Segmentation Algorithms. Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), 2004, pp. 576–584, doi: 10.1109/ictai.2004.50.

[16] Lin, C. J.—Weng, R. C.—Keerthi, S. S.: Trust Region Newton Methods for Large-Scale Logistic Regression. Proceedings of the 24th International Conference on Machine Learning (ICML 2007), Corvalis, Oregon, USA, 2007, pp. 561–568, doi: 10.1145/1273496.1273567.

[17] Ho, Q.—Cipar, J.—Cui, H.—Kim, J. K.—Lee, S.—Gibbons, P. B.—Gibson, G. A.—Ganger, G. R.—Xing, E. P.: More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server. Proceedings of the 26th International Conference on Neural Information Processing Systems (NIPS '13), Vol. 1, 2013, pp. 1123–1231.

[18] Kaufman, L.—Rousseeuw, P. J.: Finding Groups in Data: An Introduction to Cluster Analysis. New York, NY, USA, Wiley, 1990.

Dongxia Wang received her M.Eng. degree in circuit and system from the East China University of Technology at Fuzhou in 2009. She is currently a Ph.D. candidate at Shanghai University, Shanghai, China. Her current research interests include distributed machine learning, parallel and distributed computing.

Yongmei Lei received her Sc.D. degree in 1999 from Xi'an Jiaotong University at Xi'an, and her M.Eng. degree in 1989 from Xidian University at Xi'an. She is Professor at School of Computer Engineering and Science of Shanghai University. She has published over 30 technical papers. Her main research interests include parallel and distributed computing, big data analysis, etc.

Shenghong Jiang received her B.Sc. degree in electronics and information engineering from Anhui Science and Technology University, Anhui in 2016. She is currently a postgraduate student at Shanghai University, Shanghai, China. Her current research interests include parallel computing, distributed machine learning and high performance computing.

Computing and Informatics, Vol. 38, 2019, 790–816, doi: 10.31577/cai_2019_4_790

AGENT-BASED SYSTEM FOR MOBILE SERVICE ADAPTATION USING ONLINE MACHINE LEARNING AND MOBILE CLOUD COMPUTING PARADIGM

Piotr Nawrocki, Bartłomiej Śnieżyński

AGH University of Science and Technology
Faculty of Computer Science, Electronics and Telecommunications
Department of Computer Science
al. A. Mickiewicza 30
30-059 Krakow, Poland
e-mail: {piotr.nawrocki, bartlomiej.sniezynski}@agh.edu.pl

Jakub Kołodziej

CRIF Sp. z o.o.
Lublańska 38
31-476 Krakow, Poland
e-mail: jakub.lukasz.kolodziej@gmail.com

Abstract. An important aspect of modern computer systems is their ability to adapt. This is particularly important in the context of the use of mobile devices, which have limited resources and are able to work longer and more efficiently through adaptation. One possibility for the adaptation of mobile service execution is the use of the Mobile Cloud Computing (MCC) paradigm, which allows such services to run in computational clouds and only return the result to the mobile device. At the same time, the importance of machine learning used to optimize various computer systems is increasing. The novel concept proposed by the authors extends the MCC paradigm to add the ability to run services on a PC (e.g. at home). The solution proposed utilizes agent-based concepts in order to create a system that operates in a heterogeneous environment. Machine learning algorithms are used to optimize the performance of mobile services online on mobile devices. This guarantees scalability and privacy. As a result, the solution makes it possible to reduce service execution time and power consumption by mobile devices.

In order to evaluate the proposed concept, an agent-based system for mobile service adaptation was implemented and experiments were performed. The solution developed demonstrates that extending the MCC paradigm with the simultaneous use of machine learning and agent-based concepts allows for the effective adaptation and optimization of mobile services.

Keywords: Agent-based system, machine learning, adaptation, mobile service, mobile cloud computing

1 INTRODUCTION

The immense popularity of mobile devices, mainly smartphones, has brought about a rapid development of mobile services. At the same time, this development of mobile devices, including the availability of various sensors, has caused that the services offered are becoming still smarter. They can adapt to the context in which they are located and are able to learn some of their users' behavioral patterns. The user can obtain certain hints and context-based information from the device (location, time, network communication method, light level, pulse, etc.) and from the knowledge acquired in the learning process. The user may use this information for a great variety of purposes, e.g. determining the best route or evaluating his/her health. However, some mobile services overload the mobile device, especially in the context of energy consumption. In such a situation, the Mobile Cloud Computing (MCC) paradigm can be used. This paradigm makes it possible to execute these services (or elements) in the cloud. Utilizing cloud resources allows service-oriented computations (which would otherwise be resource-intensive) to be performed faster using cloud infrastructure, and the results are sent to the mobile device. The ability to change the location where the service is run makes it possible to optimize the execution time of the service, also taking into account the energy consumption of the mobile device. In MCC, in order to decide whether the mobile service should run on the mobile device or in the cloud in the case in question, machine learning algorithms can be used. An analysis of existing MCC solutions has shown that the concept of using machine learning to optimize such systems has not yet been thoroughly investigated. There are a few articles dealing with this topic such as [1] and [2], also including the work conducted by the authors of this article: [3, 4] and [5]. In addition, there are no comprehensive studies discussing the possibility of extending the MCC paradigm to service-oriented computations on local PCs, which has been demonstrated in a novel way by the authors of this article. Such an extension allows the operating costs of a system to be decreased, but the system becomes more complex. In this case, the use of machine learning is even more justified.

A distributed system, like the one considered in this paper, is a natural environment for agent-based solutions. If the environment is complex, it is very difficult

to design all system details *a priori*. To overcome this problem, one can apply a learning algorithm which makes it possible to adapt agents to the environment. In multi-agent systems, most applications use reinforcement learning [6, 7, 8]. However, in a complex environment (where the state-space is large), reinforcement learning needs time to reach satisfactory performance and the knowledge learned is very difficult to analyze. These problems suggest that other solutions, like supervised learning, should be sought. It appears that this method can be also applied to multi-agent systems and it yields results faster and in a human-readable form [9].

The main novelty of the solution proposed in this paper is the application of the agent-based framework with agents which learn autonomously on a mobile device. This makes it possible to achieve scalable learning, because no computations are performed on the data collected from mobile devices. It also protects privacy, because the information collected about task execution is processed locally instead of being sent to a server. An agent adapts the mobile service execution strategy on-line. Mobile services may be executed on a computational cloud, a local PC (personal computer) or a mobile device. The experiments are performed in a real environment.

The structure of the article is as follows: Section 2 presents the analysis of research in the field of agent-based systems in the context of mobile devices, Section 3 presents the concept of agent-based system for mobile service adaptation, Section 4 describes the implementation of this system, Section 5 introduces the results of the experiments conducted and Section 6 contains conclusions.

2 RELATED WORK

Nowadays, an increase in the importance of agent-based systems can be noticed, especially when they are used as parts of heterogeneous environments. This approach to developing systems facilitates communication between their elements located in different places. Benefits of agent-based systems are that they are composed of intelligent elements which are capable of learning and adapting. These elements are able to communicate with one another and share experiences, which further helps the evolution of the system to adapt to the current state of the environment. For this reason, agent-based systems are among the best platforms for building intelligent adaptive systems.

2.1 Learning Agents

The development of a complex, heterogeneous system is very difficult. It is practically impossible to foresee and design optimal behavior for all situations that may arise. Therefore the agents must adapt to the situations they encounter. One of the most commonly used adaptation mechanisms that may be applied for this purpose is agents learning optimal behaviors.

The most common technique used for learning strategies in agent-based systems is reinforcement learning [6, 7, 8]. The learning agent model assumes that the agent

interacts with the environment in discrete steps, sets its state s by observing the environment, executes actions and receives a reward $r \in R$. The reward is high if its actions are good, and low if they are bad. The agent has to learn which action should be executed in a given state. The formal model of learning is based on a Markov process. Reinforcement learning algorithms are simple and computationally inexpensive. However, the process of learning requires a lot of trials and without complex extensions it is inefficient in large state spaces [10]. This is a result of the *curse of dimensionality*, a well-known problem in dynamic programming [11], on which reinforcement learning is based. Another disadvantage of reinforcement learning is the unreadability of the knowledge generated because the knowledge learned is represented in a low-level manner (e.g. Q-value tables). Limitations of reinforcement learning can be overcome by adding extensions to the basic algorithm. One of the most impressive results has been achieved by combining reinforcement learning and neural networks [12]. The Deep Q-Network developed as a result of this approach allows the processing of large-dimensional spaces and even image processing. However, it should be noted that such learning exhibits high computational complexity. Therefore, currently it cannot be implemented on mobile devices.

Evolutionary computation, which is a second popular method of adaptation used in agent-based systems, relies on using multiple agent generations to improve performance [6]. This approach is computationally expensive, because many populations of agents have to be maintained. As a result, it is not an appropriate choice for mobile devices. Therefore this type of adaptation is not considered here.

There are few works on supervised learning applications in multi-agent systems. In [13], rule induction is used in a multi-agent system for vehicle routing problems. However, in that work the learning is done offline. First, rules are generated by the AQ (algorithm quasi-optimal) algorithm (the same as used in this work) from global traffic data. Next, agents use these rules to predict traffic. An extension of that approach is [14]. Agents use a hybrid learning algorithm, which is executed online. Rule induction is used to decrease the size of the search space for reinforcement learning. Another approach is applied in [15, 16], where Airiau et al. add learning capabilities to the Belief-Desire-Intention model. Decision tree learning is used to support plan applicability testing. In [17], the C4.5 algorithm is used by agent to build a model of teammates.

2.2 Mobile Cloud Computing

The development of technology, including the increase in processing power, the ongoing miniaturization, and improving availability of various sensors, means that mobile devices, including smartphones, have better technical parameters and more computing power each year, which results in a broader range of applications. The development of mobile devices has improved our ability to determine the context of the device and its user. Information on the context has enabled the development of solutions that are capable of learning how they should operate in order to optimize the use of mobile device resources and also meet user expectations as best as possible.

A large number of existing solutions only utilize the resources of mobile devices and information about context [18]. Devices learn how to best perform services based on the experience acquired. However, these solutions do not enable a permanent reduction in mobile device resource consumption or a significant increase in the execution speed of mobile services. This is why another idea – the Mobile Cloud Computing paradigm – is becoming increasingly popular. In this concept, cloud computing is used for the purpose of offloading mobile services to the cloud in order to increase operating speed and reduce the load on mobile devices. There are many solutions using the MCC paradigm such as Adaptive Code Offloading for Mobile Cloud Applications, AIOLOS, AlfredO (An Architecture for Flexible Interaction with Electronic Devices), CACTSE (Cloudlet Aided Cooperative Terminal Service Environment), COMET (Code Offload by Migrating Execution Transparently), COSMOS (Clouddb for Seamless Mobile Services), Cuckoo, Elijah, EMCO, IC-Cloud (Computation Offloading to an Intermittently-Connected Cloud), MALMOS (Machine Learning-based Mobile Offloading Scheduler), MAUI (Mobile Assistance Using Infrastructure), Mirroring Mobile Device, Mobile Cloud Execution Framework, Mobility Prediction Based on Machine Learning, MOCHA (Mobile Cloud Hybrid Architecture), Replicated Application Framework, ThinkAir, VMCC (Virtualization in Mobile Cloud Computing), MpOS (Multiplatform Offloading System), VCLA (Virtual Cloud Learning Automata), Service-Oriented Context-Aware Recommender System, Mobile Multimedia Processing System and Service-Oriented Mobile Processing System (SMPS). However, only a few solutions (MALMOS, IC-Cloud, VCLA, Service-Oriented Context-Aware Recommender System, Mobile Multimedia Processing System and SMPS) utilize machine learning in order to determine the optimal location for executing the service (the mobile device or the cloud)

The architecture proposed in the MALMOS solution [1] enables decisions to be made, with the application of machine learning technologies, when to offload applications from the mobile device to the cloud. For the offloading, the solution uses the DPartner environment (Java-based on-demand offloading framework). In order to properly teach the system where it should run applications so that they launch faster, the solution uses an online training mechanism. This solution only determines the optimal location for executing the application/service and completely fails to address the important aspect of energy consumption during the launching of the application and transferring the data to the cloud and also the amount of energy used by the online learning mechanism.

Another solution, IC-Cloud [2], uses machine learning to estimate task execution times for the mobile device and for the cloud. These are used to determine the location where the task is to be executed. The time estimation system uses two components simultaneously: the offline component, which prepares the execution model for each task for a specific device prior to the launch of the application, and the online component, which utilizes the online training mechanism with machine learning algorithms on an ongoing basis. In addition, the solution also estimates the quality of network connection based on historical data and on radio signal strength. This solution requires the prior offline setup of the task execution model for each mo-

bile device separately, which may make broader application of the solution difficult. Neither of the solutions (MALMOS, IC-Cloud) are being developed any longer.

The idea of using MCC to optimize the operation of mobile devices is still valid and important [19]. In [20], the authors propose the MpOS system (Multi-platform Offloading System), which allows offloading for mobile applications (Android/Windows Phone) using the MCC and cloudlet concepts. The decision when to offload tasks is made by the Dynamic Decision System (DDS) on the basis of predefined simple metrics such as RTT (round-trip time) or the type of network connection. At the same time, the authors investigate the impact of various types of serialization methods on the performance of the offloading process itself. The DDS system does not take into account the specificity of individual mobile applications nor their requirements related to the use of the processor or energy consumption.

Another article [21] presents an extension of the MCC concept using the cloud learning automata algorithm (VCLA). Some mobile devices are selected using Learning Automata (LA) as ad hoc virtual cloud elements and used to perform calculations. This complements the MCC concept when there is no connection to the cloud. The results of tests conducted using the QualNet 4.5 simulator have shown that the use of VCLA makes it possible to optimize the choice of the number of remote devices (in an ad hoc virtual cloud) on which the calculations are carried out. However, no experiments were conducted in a real-life testing environment which would take into account, among other things, the actual energy consumption of mobile devices.

Recently, the Deep Neural Network (DNN) mechanism has been commonly used in mobile applications, for example for speech recognition purposes (Apple Siri). The use of DNNs in the MCC often involves transferring large amounts of data between the mobile device and the cloud. In [22], the authors study the possibility of offloading only part of the DNN calculation to the cloud. The decision what calculations to send to the cloud takes into account the energy consumption of mobile devices and limited cloud resources. Test results demonstrated an increase in calculation speed and a reduction in the energy consumption of the mobile device. However, the authors only conducted simulation experiments without testing their solution under real-life conditions.

Another article dealing with certain aspects of energy consumption by mobile devices in the context of MCC is [23]. The authors propose an agent-based MCC framework using the Dynamic Programming After Filtering (DPAF) algorithm to enable the optimization of the offloading strategy, taking into account the energy consumption of mobile devices. The experiments conducted demonstrated the usefulness of the framework developed in reducing energy consumption. However, they were only performed in a simulation environment without verifying the solution developed in real-life mobile devices.

An important aspect of using the MCC concept to optimize the operation of mobile devices is security. In [24], the authors propose a secure and efficient offloading scheme which employs a combination of regular rekeying and random padding.

However, in the research conducted, the aspect of optimizing the operation of services/applications on mobile devices was not addressed and the experiments were only performed in a simulation environment.

The other solutions developed by the authors of this article such as the Service-Oriented Context-Aware Recommender System – SoCaRS [25], Mobile Multimedia Processing System – MMPS [4] and SMPS [3] make it possible to optimize (in terms of execution time and the energy used) the location where services are executed (locally or in the cloud) using MCC and machine learning algorithms.

MCC studies have been further developed in work related to Mobile Edge Computing (MEC) [26, 27] in which cloud resources are complemented by edge devices (servers) located close to the infrastructure which enables wireless transmission. This concept is used primarily in cellular networks (including 5G). Most often, systems using MEC implement optimizations on the infrastructure side and use MCC on the device and mobile application side. In MEC research, machine learning algorithms are sometimes used to optimize the use of resources [28, 29].

In [28], a novel post-decision state (PDS) based learning algorithm was used to optimize the operation of MEC. This enabled a significant improvement in edge computing performance with regard to energy aspects. The research conducted only concerned the optimization of operation and energy consumption by edge devices and did not take into account aspects related to the optimization of mobile device operation. Moreover, the authors' experiments were only conducted in a simulation environment.

In [29], the authors propose solutions for offloading in the MEC environment in the context of IoT applications (IoT-Q-L). For this purpose, they use learning agents and the Q-learning algorithm which improves the offloading of computing tasks and reduces energy consumption. Experiments confirming the possibility of optimizing task offloading were only performed in a simulation environment. Another article [30] proposes a multi-agent based flexible IoT edge computing system (F-IoT-EC) which makes it possible to optimize operation and reduce the amount of energy consumed. The system uses a rule-based engine with a fixed rule set. The tests were carried out in a simulation environment.

2.3 Agent-Based Platforms for Mobile Devices

In order to choose a framework for our system, we have analyzed the agent-based platforms available on mobile devices.

In [31], authors describe the JADE-LEAP platform as an agent-based technology for use in connection with mobile devices. JADE-LEAP is a modified version of the JADE platform that can be run on both PCs and servers, but also on mobile devices using the Java environment (e.g., on Android mobile devices). JADE is a Java framework designed for developing agent-based applications that are compliant with FIPA (Foundation of Intelligent Physical Agents) specifications [32]. JADE-LEAP message exchanges are ACL standard compliant. The exchange of messages is done asynchronously. Each agent has its own message queue, to which data are sent from

the other agents. The main limitation of JADE-LEAP is the inability to run more than one agent in a separate container on the mobile device and the fact that the main container cannot be created on the mobile device. The JADE-LEAP design requires it always to be located on a PC.

In [33], authors present an agent-based software development platform called JaCaMo. JaCaMo's operating environment can be defined as a designed and programmed set of computing units called artifacts, collected in workspaces that can be distributed across the network. Agents can communicate with each other and use artifacts. In order for the agents to use artifacts, each of them should provide an appropriate interface composed of a set of operations and properties, where operations are actions that allow agents to interact with the environment. Properties define the state of a given artifact, which can be read and modified by the agent through corresponding operations.

In [34], the author describe the μ^2 platform, which is an environment used to build applications that use an effective communicating μ-agent. The platform provides a comprehensive network support. Applications developed for desktop can be (in most cases) launched on mobile devices. The main component of the μ^2 platform are agents and their roles. In contrast to conventional agent-based solutions, the μ^2 platform is strongly focused on efficiency and the minimization of performance problems. Application development is therefore primarily about implementing roles and modeling agent organizations. Java provides the basis for working in the μ^2 platform environment (the alternative is using Clojure). Jetlang is used as an internal messaging mechanism between agents. XStream is used to serialize objects over the network in XML. Netty is used for establishing connections, agent discovery and agent communication over the network.

2.4 Mobile Service Adaptation

In Sections 2.2 and 2.3, the authors primarily analyzed various existing solutions which enable service adaptation using, among others, machine learning and the MCC paradigm. The result of this analysis is the comparison of existing solutions, which is presented in Table 1.

As we can see, some systems are rule-based with manually created rules and mostly apply machine learning to train the model that is used for service adaptation. The latter solution is better because it makes it possible to adapt the system to changing conditions by executing the machine learning algorithm again. Considering all possible mobile devices, tasks and conditions would result in a very complex set of rules.

Analyzing this table, it can also be observed that many state-of-the-art solutions are tested in simulation environments. It should be noted that simulations of wireless networks are simplified and do not account for all the technical problems which occur in the real world [35].

Only a few systems use agents. However, agents are often used on the modeling level rather than at the implementation stage (SoCaRS, MMPS, SMPS, IoT-Q-L,

DPAF). Moreover, there is one type of the agent defined in these systems, and that is an abstract entity which encapsulates learning and decision algorithms. It interacts with environment rather than with other agents. Also importantly, these systems are developed in a standard way, without using an agent-based framework, and certain advantages of applying an agent-based methodology (like interoperability or the ability to operate in heterogeneous environments) are not exploited. The only solution known to the authors which employs a multi-agent system in this domain is F-IoT-EC. However, it is implemented in the cloud and on the edge rather than on mobile devices and no machine learning algorithm is used.

Solutions	**Env**	**Energy**	**Time**	**ML/Rules**	**Agents**	**Tests**
MALMOS	MCC	no	yes	ML	no	real
IC-Cloud	MCC	no	yes	ML	no	real
MpOS	MCC	no	yes	Rules	no	real
VCLA	MCC	no	yes	ML	no	simulation
DNN	MCC	yes	yes	ML	no	simulation
DPAF	MCC	yes	yes	Rules	one	simulation
SoCaRS	MCC	yes	yes	ML	one	real
MMPS	MCC	no	yes	ML	one	real
SMPS	MCC	yes	yes	ML	no	real
PDS	MEC	yes	yes	ML	no	simulation
IoT-Q-L	MEC	yes	yes	ML	one	simulation
F-IoT-EC	MEC	yes	yes	Rules	multi	simulation

Table 1. Comparison of existing service adaptation solutions

Several important elements of our research constitute contributions to the area of MCC. The important aspect is that solution performance has been tested in a real-life environment. Additionally, the experience gained by the authors in developing their solutions (SoCaRS, MMPS, SMPS) has allowed them to develop the concept of an agent-based system for mobile service adaptation. However, in contrast to previous research, the authors used an agent-based platform for mobile devices (μ^2) to optimize the performance of mobile services. As a result, a genuine multi-agent system with learning agents was built. The final contribution is that the solution developed extends (in comparison to previous work) the ability to perform services on local PCs, which makes the environment more complex and heterogeneous. Currently, it consists of mobile devices, the mobile cloud and the PCs accessible via a local Wi-Fi network.

In summary, when compared with the solutions analyzed, the solution developed by the authors is the only one that uses a multi-agent system, takes into account time and energy consumption, applies ML (Machine Learning) online and has been tested in a real-life environment.

3 AGENT-BASED SYSTEM FOR MOBILE SERVICE ADAPTATION

The purpose of the research was to develop, implement and test the concept of an agent-based system for mobile service adaptation using online machine learning and the MCC paradigm. The system developed should allow adaptation in order to select the optimal place of service execution from the point of view of execution time and power consumption by a mobile device. The system keeps track of the service being performed on a current basis and selects the place of its execution using the machine learning concept on the basis of defined parameters.

The concept of system architecture (Figure 1) assumes that agents perform the services commissioned on a mobile device, on a PC and in the cloud. The operation of the system is based on interaction between agents in a heterogeneous environment. The management agent containing the service adaptation module is located on the mobile device. At the same time, there are agents responsible for launching the mobile service on the mobile device (locally), on the PC and in the cloud. By using this approach, it is possible to define a common way of exchanging messages and data, which allows communication between different service locations and the mobile device itself.

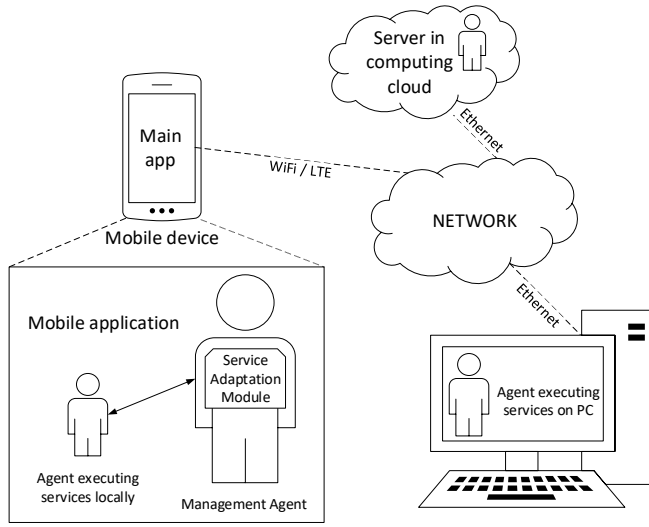


Figure 1. Concept of system architecture

The Service Adaptation Module (Figure 2) consists of four main elements:

- The Manager whose task is to receive a service request, collect training data, cooperate with the Learning Module and return a response that represents the location selected.

- The Learning Module, which builds the knowledge model K from training data. K is used to select the service execution location.
- Training data – examples representing requests, locations of their execution and results of their execution as described by attributes *Attr* collected during service execution.
- Knowledge (K) – models allowing the prediction of request execution results in a given location.

The Manager receives the service request and possible locations from the Management Agent, describes them with *Attr*, creating a *problem*, and passes it to the Learning Module. The Learning Module applies K and returns a *response* representing the predicted outcome. The Manager's task is to select the best location based on such predictions. It also collects data from the execution of services and stores them in T (Training Data). It is also responsible for passing T to the Learning Module when necessary to build new knowledge K. The structure of knowledge is closely related to the type of machine learning algorithm used. Model examples include a set of regression parameters, a set of neural network parameters or a decision tree.

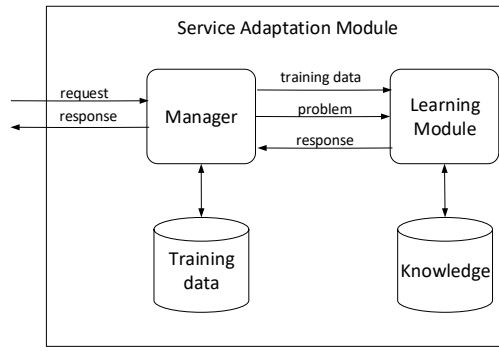


Figure 2. Service Adaptation Module architecture

More formally, the Service Adaptation Module may be defined as the following tuple:

$$SAM = (A, K, T, L) \tag{1}$$

where A is a set of attributes that are used to describe tasks and the context, K is *generated knowledge*, T is *training data*, which is a set of *examples*, and L is a set of possible execution locations.

Input data for the module is a pair $x = (t, c)$ representing the task t which should be executed in the context c. The aim of the module is to return a location $l \in L = \{l_1, l_2, \ldots l_{ns}\}$, which corresponds to engaging one of ns services. The processing module describes x with observation attributes $O = \{o_1, o_2, \ldots o_n\} \subset A$,

which yields $x^O = (o_1(x), o_2(x), \ldots o_n(x))$, i.e. a description of the *problem*. Next, using the knowledge stored in K, it solves the *problem* by selecting $l \in L$, which has the minimum predicted cost. If K is empty, l is randomized.

The decision about execution location l is then applied and the task is run using the corresponding service (i.e. locally or in the mobile cloud). After the execution, the module obtains the execution result $r(x, l)$, which is described by $Res = \{r_1, r_2, \ldots r_m\} \subset A$ attributes. For example, the four following attributes may be used: whether execution was successful $es(x, l)$, power consumption $b(x, l)$, calculation time $ct(x, l)$ and user's dissatisfaction $dis(x, l)$, which may be measured by observing if the user overrode the module's decision. Therefore, the set of all attributes used to describe the input data is the sum of O and Res

$$A = O \cup Res. \tag{2}$$

The module stores these results together with x^O and location l in T. Therefore the complete training example t stored in T has the form

$$t = (o_1(x), o_2(x), \ldots o_n(x), r_1(x, l), r_2(x, l), \ldots r_m(x, l), l). \tag{3}$$

The models $\{M_{r_i}\}$ to predict Res values are constructed using supervised learning algorithms and stored in K. These models influence the decision selected. There are $|Res|$ models stored in K. Every model $M_{r_i} : X, L \to [0, 1]$ calculates the normalized r_i value for given x and l.

Using predictions of the r_i value returned by the learned model: $M_{r_i}(x, l)$, the module may rate its decisions about all locations $l \in L$ by calculating predicted expenses $e(x, l)$:

$$e(x, l) = \sum_{i=1}^{m} w_i * M_{r_i}(x, l) \tag{4}$$

where w_i are weights of the result r_i. The weights represent user's preferences and are also related to the specificity of application which executes services. In case of an interactive application, the execution time may be more important and its weight may be higher. If mobile services are executed in a background process, the weight of battery usage may have higher value. In the production version of the system, weights can be modified by the user in the settings.

The module selects the location $l^* \in L$ for which execution is predicted to be successful and the expense is predicted to be the lowest:

$$l^* = \begin{cases} \arg\min_{l \in L}\{e(x, l) | M_{es}(x, l) = 1\}, & M_{es} \in K, \\ \arg\min_{l \in L}\{e(x, l)\}, & M_{es} \notin K. \end{cases} \tag{5}$$

The full algorithm of the *Service Adaptation Module* is presented in Figure 3. At the beginning, K and T are set to empty (lines 1–2). Next, algorithm waits for the task (line 4). If there is no learned knowledge, the decision is randomized (lines 5–6). Else there is some knowledge, therefore expenses are calculated for all

possible locations (line 8). Next, the best location is selected (line 9). To provide exploration, l^* is replaced by a random location with ε probability (line 10). The task is executed in the selected service (line 12). Results of the execution are observed (line 13) and the example is stored in T (line 14). After processing a given number of tasks (line 15), *Learning Module* is called to generate new knowledge from T and the learned knowledge is stored in K (line 16).

```
1:  K := ∅;
2:  T := ∅;
3:  while module is working do
4:     wait for x = (t, c);
5:     if K = ∅ then
6:        l := random decision;
7:     else
8:        calculate e(x, l) according to Equation (4) for all l ∈ L applying
          models from K to predict r_i;
9:        l* := best location according to Equation (5);
10:       replace l* by a random location with ε probability;
11:    end if
12:    execute task at the service determined by l*;
13:    observe execution results;
14:    store example t (see Equation (3)) in the T;
15:    if it is learning time (e.g. every 100 steps) then
16:       learn K from T;
17:    end if
18: end while
```

Figure 3. Algorithm of the *Service Adaptation Module* allowing the adaptation of mobile service using machine learning

4 IMPLEMENTATION

The μ^2 platform was used for implementation because it is lightweight and allows the easy building of agents working in a heterogeneous environment. Each agent operating within the μ^2 platform must be on the same network (e.g. the same Wi-Fi network). For this reason, it is not possible to establish direct communication between an agent operating on a mobile device and an agent operating in the public cloud. Exchanges of messages between agents are accomplished using the TCP (Transmission Control Protocol).

The most important agent in the system is the ManagementAgent, which receives requests for service execution. The ManagementAgent selects the optimal location for the service and then passes the service's startup parameters to one of the service agents:

- AndroidProviderAgent – the agent responsible for the execution of the service commissioned on the mobile device;
- AWSProviderAgent – the agent responsible for the execution of the service commissioned in the computing cloud;
- PCProviderAgent – the agent responsible for the execution of the service commissioned on the desktop device.

The communication between agents is based on exchanging MicroMessage messages. The message includes information about the sender and *Intent* (data). In the Agent-based System for MCC Service Adaptation, the *Intent* object is composed of:

- the data needed for service execution;
- information about the time when service execution started;
- battery status information at the start of service execution;
- information about the type of service executed;
- the result of service execution.

The detection of active agents is accomplished through a polling mechanism – Heartbeat. The XML configuration file defines the frequency with which an agent should query the presence of other agents.

At the time of receiving the service request, the ManagementAgent selects where to execute the service, then creates a MicroMessage message with the data necessary to execute it and the information described above. The Management-Agent sends the message to the recipient who should execute the service via the *send*(*MicroMessage message*) function. If the service should be executed by all agents available, then the *sendGlobalBroadcast*(*MicroMessage message*) function is used.

The Mobile application was written for Android mobile devices in accordance with API 17. During the implementation work, the following technologies and tools were used:

- Java language version 1.7;
- Gradle for building and managing application dependencies;
- Power Tutor for measuring the energy consumption of mobile device;
- WEKA (Waikato Environment for Knowledge Analysis) – a library providing machine learning algorithms;
- Tess4J – Java interface for the Tesseract library for text recognition (Optical Character Recognition – OCR);
- iText – a library for creating PDF documents;
- AWS (Amazon Web Services) Android SDK – a library for communicating between a mobile device and the AWS computing cloud;

- μ^2 – a platform for multi-agent systems.

WEKA was selected as the machine learning library because it has been ported to the Android system used in the development and it provides a broad range of algorithms, thus enabling comparisons between different solutions. In the production version of the system, it may be replaced by a more modern machine learning library like TensorFlow.

The Amazon AWS cloud has been used in the Agent-based System for MCC Service Adaptation. For service execution, the AWS Lambda solution is used, which is based on the IaaS (Infrastructure as a Service) model. AWS Lambda enables the implementation of features that can be remotely invoked by web applications, desktop applications and mobile applications. Due to the limitations related to μ^2 platform operation in a local area network, it was not possible to run the agent in the AWS Lambda cloud computing environment.

Two AWS Lambda functions have been created for MCC Service Adaptation: the OCR and $convertingPNGToPDF$ functions. The OCR service uses the Tess4J API for Tesseract library operations. The same mechanism was used to implement the service on a desktop. For the OCR service using the Tesseract library, it was not possible to use the same source code as for the mobile app where the dedicated tess-two library was used, which is a modified version of the Tesseract library that can run on Android mobile devices. The same source code (using the iText library) was used for the conversion of the image file to PDF. The AWS Lambda cloud computing environment in which the functions are executed is a Java-based one. The maximum memory for the functions was set at 512 MB and the timeout was set to five minutes.

The desktop application within the Agent-based System for MCC Service Adaptation uses the μ^2 environment and Java. The most important part of the application is the agent that works on the desktop device. The agent communicates with the Management Agent on the mobile device through the network. As with other agents operating within the framework of the system, it receives the service request, executes it on the desktop device and then sends the result of its execution to the Management Agent.

In order to objectively compare the services offered by cloud computing and the desktop application, we have decided to run the desktop application under conditions that are as close as possible to those in the cloud computing environment. For this purpose, the Docker tool is used to place the program and all its dependencies, such as additional libraries, in a lightweight, portable and virtual container that can be run on a Linux server. In order to run a container on a Docker, an image must be created with boot parameters that install the appropriate libraries and other dependencies.

In order to map the AWS Lambda environment on a PC, a desktop application is started via Docker using a docker-lambda image. It has the same software and libraries installed, the same file structure and permissions, and the same environmental variables as the AWS Lambda environment. The image makes it possible to

run AWS Lambda functions in the Node.JS and Python environments. Currently, the Java environment is not yet fully supported, but it is possible to launch Java programs manually.

5 PERFORMANCE EVALUATION

A series of experiments were conducted to verify the operation of the Agent-based System for MCC Service Adaptation. As test services, text recognition (OCR) and PNG transformation to PDF format were selected due to the complexity of the service and its demand for mobile device hardware resources.

Eight graphic files with varying sizes, resolutions, and amounts of text contained in them were selected to test the service with different input parameters. Each of the experiments was designed to perform a defined set of services under certain conditions. The tests included service execution, time measurements, energy consumption measurements by the mobile device while the service was being executed, and the recording of results. Observed attributes were $O = \{tt, s, res, con, sig\}$. They correspond to task type, file size, number of pixels in the picture, connection type and network connection signal strength, respectively. Execution results observed were $Res = \{ct, b\}$ (calculation time and battery usage). Result expenses were calculated according to the following formula:

$$e(x, l) = w_{ct} ct(x, l) + w_b b(x, l) \tag{6}$$

where:

- w_{ct} – weight of service execution time;
- w_b – weight of device energy consumption.

In experiments, sets of service requests were executed as consecutive rounds. During the first round, the location is selected in a random way. During each subsequent round, the Service Adaptation Module is trained on the data collected from previous rounds $1 \ldots r - 1$. To ensure exploration, the location is randomly selected with a probability of 10% even if knowledge is not empty.

After each service execution, the results are recorded in Training Data. After each round, the Service Adaptation Module builds new knowledge using one of the following algorithms provided by the WEKA library: J48, RandomForest, KStar, MultilayerPerceptron and SimpleLogistic. For the J48, RandomForest and KStar algorithms, numerical values of parameters are discretized into 6 compartments of equal frequency.

Every experiment consists of 8 rounds, repeated 10 times. The measurement involves total service execution time and total energy consumption of the mobile device during the round in question. The results are presented as charts representing average values and standard deviations of these measures over ten repetitions for each of the rounds.

The experiment was carried out using a mobile device – LG G2 (CPU[1] – 2.26 GHz, memory – 2 GB), router – Wi-Fi TP-Link TL-WR842N (802.11bgn), cloud – AWS Lambda platform and PC (CPU – Intel Core i5-3320M 2.6 GHz, memory – 8 GB).

5.1 Initial Tests

Initially, we measured values for cases when all tasks were executed on the mobile device. These values are 364 806 ms for average execution time and 399 556 mJ for average battery power consumption. Similarly, for executing all requests in the cloud we get an average execution time of 340 714 ms and an average battery power consumption of 408 212 mJ. It means that execution in the cloud is faster, but requires somewhat more energy because of data transmission. There are no results related to the execution of all tasks on the desktop device since the solution implemented does not enable the execution of tasks on a PC when the mobile device uses HSDPA (High-Speed Downlink Packet Access). In the scenario simulated, the user can only use the desktop device at home.

5.2 Energy Optimization

The experiment was aimed at comparing the effectiveness of machine learning algorithms in optimizing the power consumption of the mobile device. For comparison, we selected five classification algorithms: J48, RandomForest, MultilayerPerceptron, logistic regression and k-NN (k-Nearest Neighbors). The set of service requests consists of 16 individual OCR service requests with various input data. Eight of them are executed using the Wi-Fi connection, and eight using HSDPA.

Coefficient values are: the cost of service execution time w_{ct} at 0.1 and the cost of device energy consumption w_b at 0.9. Figure 4 shows the average results for each round together with standard deviations.

The logistic regression algorithm was the worst from the point of view of optimization. The results obtained by this algorithm in the rounds where location was selected were worse than in the first round where the choice of execution location was random.

The k-NN algorithm enabled a significant optimization of energy consumption, but it was susceptible to overfitting. Initially, the algorithm was getting better with each round until the fifth one, after which it started to oscillate. The statistical significance of the improvement between the first and last rounds (d) was checked using t-Student test. The p-value equals to 0.1196, which means that d is not statistically significant.

Three other algorithms (J48, Random Forest and MultilayerPerceptron) enabled similar levels of optimization. For each of them, a significant reduction in energy consumption could be observed in the early rounds. After the fifth round had been

[1] Central Processing Unit

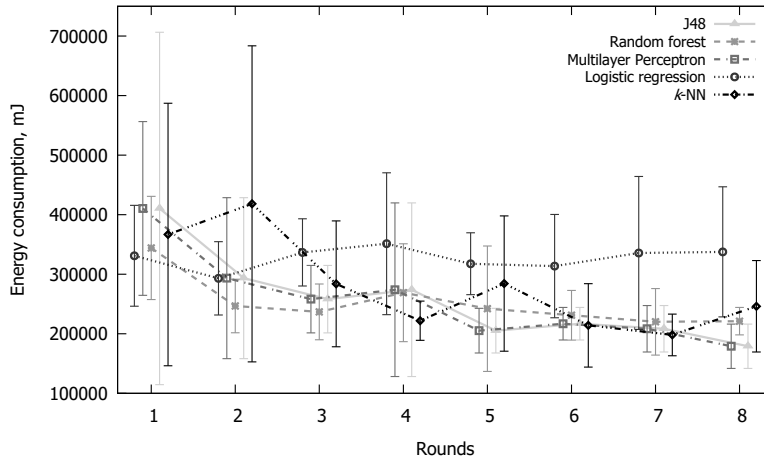


Figure 4. Optimizing energy consumption for coefficients $w_{ct} = 0.1$ and $w_b = 0.9$

completed, the process of reducing energy consumption slowed down, but oscillations were smaller. Results of the t-Student test for these classifiers (p-value) are as follows:

- for J48: 0.0245, which means that d is statistically significant;
- for RandomForest: 0.0004, which means that d is statistically significant;
- for MultilayerPerceptron: ≤ 0.0001, which means that d is statistically significant.

The experiment also examined the level of optimization in execution time with the same cost factors. The results are shown in Figure 5.

As in the case of energy consumption optimization, the logistic regression algorithm exhibited the worst results because no improvement could be observed. The k-NN and MultilayerPerceptron algorithms made it possible to decrease execution time. However, it should be noted that oscillations appeared after several rounds. The best results were obtained by the J48 and RandomForest algorithms, which reduced execution time with small standard deviations in each round and oscillations were small. The results of the t-Student test (p-value) are as follows:

- for logistic regression algorithm: 0.3501, which means that d is not statistically significant;
- for k-NN: 0.0190, which means that d is statistically significant;
- for MultilayerPerceptron: 0.0024, which means that d is statistically significant;
- for J48: ≤ 0.0001, which means that d is statistically significant;
- for RandomForest: ≤ 0.0001, which means that d is statistically significant.

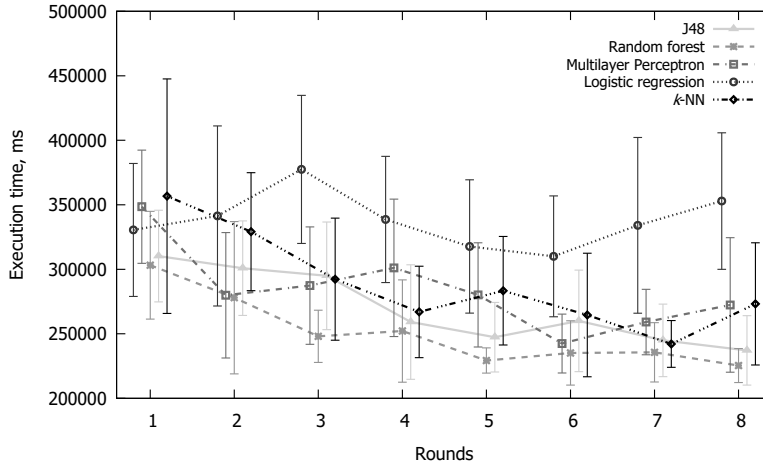


Figure 5. Optimizing execution time for coefficients $w_{ct} = 0.1$ and $w_b = 0.9$

5.3 Time Execution Optimization

The experiment methodology was analogous to the one described above. The experiment was designed to compare machine learning algorithms for selecting the service execution location in order to optimize execution time. For comparison, the three best algorithms were selected: J48, RandomForest and MultilayerPerceptron. The set of service requests is defined in the same way as in the previous experiment. The coefficients are: the cost of service execution time w_{ct} at 0.9 and the energy consumption of the device w_b at 0.1. Average results together with standard deviations are shown in Figure 6.

The J48 and RandomForest algorithms exhibited similar results. Between the second and fourth rounds, the improvement was almost linear. From round four onwards, the reduction in execution time was no longer present, and then in rounds from five to eight the execution time increased slightly. Standard deviations for these two algorithms were significantly higher than for the MultilayerPerceptron algorithm. The MultilayerPerceptron algorithm behaved similarly to the other two, with the difference that from the second to fourth rounds it enabled better optimization of execution time, and from the fifth round onwards the service execution time for the MultilayerPerceptron algorithm oscillated around the same level and standard deviations were small. The results of the t-Student test (p-value) are as follows:

- for J48: 0.0002, which means that d is statistically significant;
- for RandomForest: ≤ 0.0001, which means that d is statistically significant;
- for MultilayerPerceptron: 0.0033, which means that d is statistically significant.

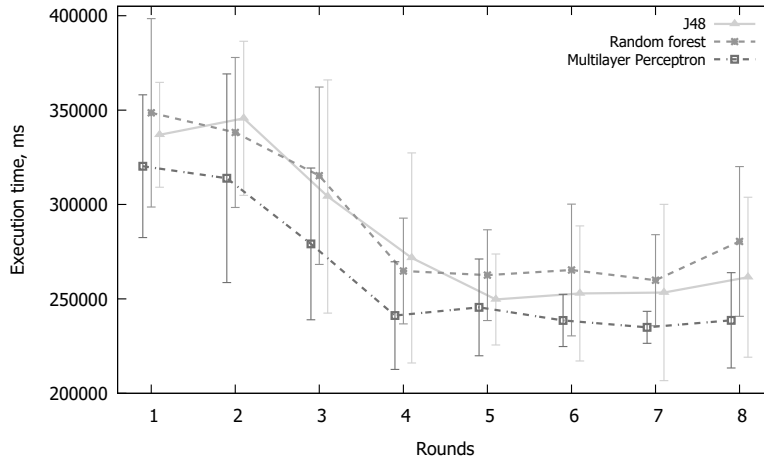


Figure 6. Optimizing execution time for coefficients $w_{ct} = 0.9$ and $w_b = 0.1$

The experiment also examines the level of energy consumption optimization on the mobile device with the same cost factors. The results are shown in Figure 7.

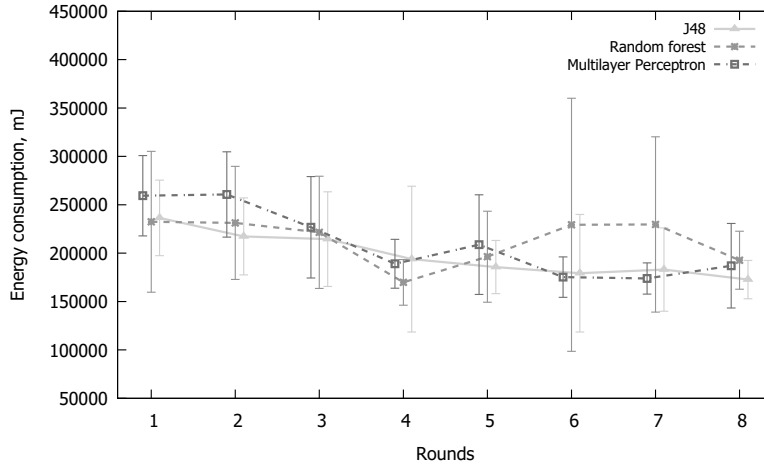


Figure 7. Optimizing energy consumption for coefficients $w_{ct} = 0.9$ and $w_b = 0.1$

In the fourth round, energy consumption by the mobile device was the lowest for the Random Forest algorithm. However, from the fourth round onwards there was an increase in energy consumption by the device and a significant increase in standard deviation could be observed in rounds six and seven. The significant increase in standard deviation was related, among others, to the specificity of the

HSDPA wireless transmission technology used in the tests, which does not allow to guarantee the quality of the network connection (including delay and bandwidth). The J48 and MultilayerPerceptron algorithms proved better in the end and they also had lower standard deviations. The results of the t-Student test (p-value) are as follows:

- for J48: 0.0002, which means that d is statistically significant;
- for RandomForest: 0.1757, which means that d is not statistically significant;
- for MultilayerPerceptron: 0.0013, which means that d is statistically significant.

5.4 Time and Energy Optimization

The experiment methodology was analogous to the previous experiments. The experiment aimed at comparing machine learning algorithms for selecting service execution location in order to optimize both mobile device execution time and energy consumption. The same algorithms were compared as in the previous experiments: J48, Random Forest and MultilayerPerceptron. The values of coefficients were as follows: service execution time cost w_{ct} at 0.5 and device energy consumption cost w_b at 0.5. Average results together with standard deviations are shown in Figures 8 and 9.

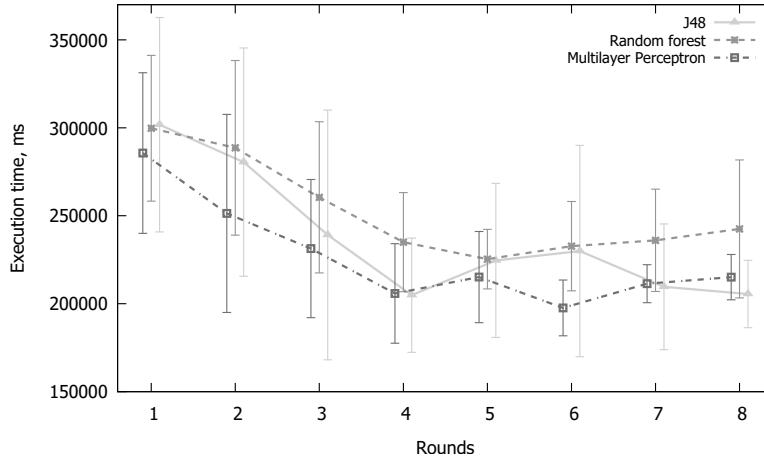


Figure 8. Optimizing execution time for coefficients $w_{ct} = 0.5$ and $w_b = 0.5$

The J48 algorithm enabled the optimization of both mobile device execution time and energy consumption. In the case of energy consumption, until the fourth round of the algorithm a significant reduction in energy consumption was noted. From the fifth round onwards, the process slowed down, but continued. A similar situation occurred with the optimization of execution time, with the difference that

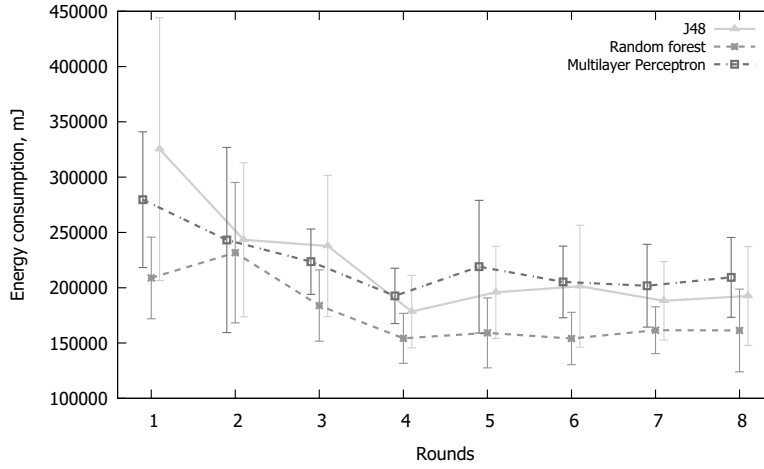


Figure 9. Optimizing energy consumption for coefficients $w_{ct} = 0.5$ and $w_b = 0.5$

from round four to six there was an increase in execution time but from round six to eight the execution time decreased again. The Random Forest algorithm behaved in the same manner with respect to both measurements. Until the fourth round, there was a significant improvement in both parameters, and from the fourth to the eighth rounds the values of the parameters oscillated around the same level. In addition, the algorithm was characterized by the fact that the standard deviation value over all rounds remained at a similar, low level. The MultilayerPerceptron algorithm also optimized both the execution time and energy consumption of the mobile device well. In both cases, the reduction was considerable until the fourth round. From round four to round eight, the values of the metrics oscillated. The feature that distinguished this algorithm from the others was also the low value of standard deviation for all rounds. The results of the t-Student test for energy consumption optimization (p-value) are as follows:

- for J48: 0.0039, which means that d is statistically significant;
- for RandomForest: 0.0059, which means that d is statistically significant;
- for MultilayerPerceptron: 0.0105, which means that d is statistically significant.

The results of the t-Student test for execution time optimization (p-value) are as follows:

- for J48: 0.0002, which means that d is statistically significant;
- for RandomForest: 0.0002, which means that d is statistically significant;
- for MultilayerPerceptron: 0.0053, which means that d is statistically significant.

5.5 Analysis of Results

As we can see, the J48, RandomForest and MultilayerPerceptron algorithms are useful for service adaptation, while k-NN and SimpleLogistic yield poor results. It should be noted that a careful choice of parameters with values different from default ones might improve their results, but the first three algorithms listed work well without such tuning.

The best optimization results were achieved by the J48 algorithm. It exhibited the lowest energy consumption or execution time in the last round in three out of six cases. This learning algorithm is relatively simple and has a low computational complexity. What is also important, the knowledge learned has a form of a decision tree, so it may potentially be analyzed by a human. Thus it appears to be the best choice for mobile cloud computing and service adaptation.

To determine which algorithms yielded the fastest improvements, the results in round three were compared. The fastest learning algorithm was Random Forest, since it won in three out of six cases. Unfortunately, this learning algorithm is more complex and needs more resources. The models learned are also difficult to analyze in this case.

All algorithms yielded rapid improvements in performance in the first three rounds. The reason for this is that during these rounds, various new examples were added to the training set. During later rounds, the examples added were more frequently similar to the previous ones and as a result, progress was much slower: results were sometimes even worse than in earlier rounds or they oscillated.

6 CONCLUSIONS

In this paper, we have proposed an agent-based solution for mobile service adaptation using machine learning. This approach enables online, autonomous adaptation on a mobile device. Through the use of machine learning algorithms as well as the Mobile Cloud Computing concept, various mobile services and applications may become smarter, faster and more energy efficient. The local execution of the learning algorithm allows for scalability because each device learns on its own. This solution also ensures privacy protection as no usage data are being sent.

We have also shown that it is possible to add one more location to execute computation, a PC computer that may by located at home or at work and used instead of the cloud. Such solution makes task-related communication much faster, because only local area network may be used. Increased complexity of the system did not influence the adaptation capability. Application of the multi-agent platform helped to design and implement PC-mobile devices cooperation.

Experimental results obtained in a real-life environment demonstrate that the application of this approach brings improvements in energy consumption and execution time that are statistically significant. The best results are obtained by the J48 algorithm. This algorithm creates models in a form of decision tree. Therefore these models can be analyzed what may be important in some applications.

In future research we would like to address, among other things, the use of localization data and the building of user models. The first improvement would make it possible to take into account user needs specific to his/her location, e.g. the ability to recharge the device while at home. On the other hand, building a user model would enable a faster start. Instead of an empty knowledge set, an initial knowledge base matching the owner profile could be used. We would also like to incorporate other metrics such as the quality of the result and its cost in the cost function (Equation (6)). We will be also investigating possibilities to use agent-based platform on a cloud by extending the functionality of the μ^2 platform with the possibility of unicast communication.

Acknowledgements

The research presented in this paper was supported by funds from the Polish Ministry of Science and Higher Education assigned to the AGH University of Science and Technology.

REFERENCES

[1] Eom, H.—Figueiredo, R.—Cai, H.—Zhang, Y.—Huang, G.: MALMOS: Machine Learning-Based Mobile Offloading Scheduler with Online Training. Proceedings of the 2015 3rd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MOBILECLOUD '15), 2015, pp. 51–60, doi: 10.1109/mobilecloud.2015.19.

[2] Shi, C.—Pandurangan, P.—Ni, K.—Yang, J.—Ammar, M.—Naik, M.—Zegura, E.: IC-Cloud: Computation Offloading to an Intermittently-Connected Cloud. Technical report, School of Computer Science, Georgia Institute of Technology, 2013.

[3] Nawrocki, P.—Sniezynski, B.: Autonomous Context-Based Service Optimization in Mobile Cloud Computing. Journal of Grid Computing, Vol. 15, 2017, No. 3, pp. 343–356, doi: 10.1007/s10723-017-9406-2.

[4] Nawrocki, P.—Sniezynski, B.: Adaptive Service Management in Mobile Cloud Computing by Means of Supervised and Reinforcement Learning. Journal of Network and Systems Management, Vol. 26, 2018, No. 1, pp. 1–22, doi: 10.1007/s10922-017-9405-4.

[5] Nawrocki, P.—Sniezynski, B.—Slojewski, H.: Adaptable Mobile Cloud Computing Environment with Code Transfer Based on Machine Learning. Pervasive and Mobile Computing, Vol. 57, 2019, pp. 49–63, doi: 10.1016/j.pmcj.2019.05.001.

[6] Panait, L.—Luke, S.: Cooperative Multi-Agent Learning: The State of the Art. Autonomous Agents and Multi-Agent Systems, Vol. 11, 2005, No. 3, pp. 387–434, doi: 10.1007/s10458-005-2631-2.

[7] Sen, S.—Weiss, G.: Learning in Multiagent Systems. In: Weiss, G. (Ed.): Multiagent Systems. Chapter 6. MIT Press, Cambridge, MA, USA, 1999, pp. 259–298.

[8] Tuyls, K.—Weiss, G.: Multiagent Learning: Basics, Challenges, and Prospects. AI Magazine, Vol. 33, 2012, No. 3, pp. 41–52, doi: 10.1609/aimag.v33i3.2426.

[9] Sniezynski, B.: A Strategy Learning Model for Autonomous Agents Based on Classification. International Journal of Applied Mathematics and Computer Science, Vol. 25, 2015, No. 3, pp. 471–482.

[10] Kaelbling, L. P.—Littman, M. L.—Moore, A. W.: Reinforcement Learning: A Survey. Journal of Artificial Intelligence Research, Vol. 4, 1996, pp. 237–285, doi: 10.1613/jair.301.

[11] Bellman, R. E.: Dynamic Programming. A Rand Corporation Research Study. Princeton University Press, 1957.

[12] Mnih, V.—Kavukcuoglu, K.—Silver, D.—Rusu, A. et al.: Human-Level Control Through Deep Reinforcement Learning. Nature, Vol. 518, 2015, No. 7540, pp. 529–533, doi: 10.1038/nature14236.

[13] Gehrke, J. D.—Wojtusiak, J.: Traffic Prediction for Agent Route Planning. In: Bubak, M., Dick van Albada, G. D., Dongarra, J., Sloot, P. M. A. (Eds.): Computational Science – ICCS 2008. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5103, 2008, pp. 692–701, doi: 10.1007/978-3-540-69389-5_77.

[14] Sniezynski, B.—Wojcik, W.—Gehrke, J. D.—Wojtusiak, J.: Combining Rule Induction and Reinforcement Learning: An Agent-Based Vehicle Routing. Proceedings of the 2010 Ninth International Conference on Machine Learning and Applications (ICMLA 2010), 2010, pp. 851–856, doi: 10.1109/icmla.2010.132.

[15] Airiau, S.—Padgham, L.—Sardina, S.—Sen, S.: Incorporating Learning in BDI Agents. Proceedings of the ALAMAS + ALAg Workshop at AAMAS, Estoril, Portugal, May 2008.

[16] Singh, D.—Sardina, S.—Padgham, L.—Airiau, S.: Learning Context Conditions for BDI Plan Selection. Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS'10), Toronto, Canada, 2010, Vol. 1, pp. 325–332.

[17] Barrett, S.—Stone, P.—Kraus, S.—Rosenfeld, A.: Learning Teammate Models for Ad Hoc Teamwork. International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Adaptive Learning Agents (ALA) Workshop, Valencia, Spain, June 2012.

[18] Korpipaa, P.—Mantyjarvi, J.—Kela, J.—Keranen, H.—Malm, E. J.: Managing Context Information in Mobile Devices. IEEE Pervasive Computing, Vol. 2, 2003, No. 3, pp. 42–51, doi: 10.1109/mprv.2003.1228526.

[19] Zhou, B.—Buyya, R.: Augmentation Techniques for Mobile Cloud Computing: A Taxonomy, Survey, and Future Directions. ACM Computing Surveys (CSUR), Vol. 51, 2018, No. 1, pp. 13:1–13:38, doi: 10.1145/3152397.

[20] Rego, P. A. L.—Costa, P. B.—Coutinho, E. F.—Rocha, L. S.—Trinta, F. A. M.—de Souza, J. N.: Performing Computation Offloading on Multiple Platforms. Computer Communications, Vol. 105, 2017, pp. 1–13, doi: 10.1016/j.comcom.2016.07.017.

[21] Krishna, P. V.—Misra, S.—Saritha, V.—Raju, D. N.—Obaidat, M. S.: An Efficient Learning Automata Based Task Offloading in Mobile Cloud Comput-

ing Environments. 2017 IEEE International Conference on Communications (ICC), 2017, pp. 1–6, doi: 10.1109/icc.2017.7997139.

[22] Eshratifar, A. E.—Pedram, M.: Energy and Performance Efficient Computation Offloading for Deep Neural Networks in a Mobile Cloud Computing Environment. Proceedings of the 2018 Great Lakes Symposium on VLSI (GLSVLSI '18), ACM, 2018, pp. 111–116, doi: 10.1145/3194554.3194565.

[23] Kuang, Z.—Guo, S.—Liu, J.—Yang, Y.: A Quick-Response Framework for Multi-User Computation Offloading in Mobile Cloud Computing. Future Generation Computer Systems, Vol. 81, 2018, pp. 166–176, doi: 10.1016/j.future.2017.10.034.

[24] Meng, T.—Wolter, K.—Wu, H.—Wang, Q.: A Secure and Cost-Efficient Offloading Policy for Mobile Cloud Computing Against Timing Attacks. Pervasive and Mobile Computing, Vol. 45, 2018, pp. 4–18, doi: 10.1016/j.pmcj.2018.01.007.

[25] Nawrocki, P.—Śnieżyński, B.—Czyżewski, J.: Learning Agent for a Service-Oriented Context-Aware Recommender System in a Heterogeneous Environment. Computing and Informatics, Vol. 35, 2016, No. 5, pp. 1005–1026.

[26] Dolui, K.—Datta, S. K.: Comparison of Edge Computing Implementations: Fog Computing, Cloudlet and Mobile Edge Computing. 2017 Global Internet of Things Summit (GIoTS), June 2017, pp. 1–6, doi: 10.1109/giots.2017.8016213.

[27] Mao, Y.—You, Ch.—Zhang, J.—Huang, K.—Letaief, K. B.: A Survey on Mobile Edge Computing: The Communication Perspective. IEEE Communications Surveys and Tutorials, Vol. 19, 2017, No. 4, pp. 2322–2358, doi: 10.1109/comst.2017.2745201.

[28] Xu, J.—Chen, L.—Ren, S.: Online Learning for Offloading and Autoscaling in Energy Harvesting Mobile Edge Computing. IEEE Transactions on Cognitive Communications and Networking, Vol. 3, 2017, No. 3, pp. 361–373, doi: 10.1109/tccn.2017.2725277.

[29] Alam, M. G. R.—Hassan, M. M.—Uddin, M. Z.—Almogren, A.—Fortino, G.: Autonomic Computation Offloading in Mobile Edge for IoT Applications. Future Generation Computer Systems, Vol. 90, 2019, pp. 149–157, doi: 10.1016/j.future.2018.07.050.

[30] Ogino, T.—Kitagami, S.—Suganuma, T.—Shiratori, N.: A Multi-Agent Based Flexible IoT Edge Computing Architecture Harmonizing Its Control with Cloud Computing. International Journal of Networking and Computing, Vol. 8, 2018, No. 2, pp. 218–239.

[31] Bergenti, F.—Caire, G.—Gotta, D.: Agents on the Move: JADE for Android Devices. In: Santoro, C., Bergenti, F. (Eds.): WOA 2014. CEUR Workshop Proceedings, Vol. 1260, 2014, Art. No. 9.

[32] Bellifemine, F.—Poggi, A.—Rimassa, G.: JADE: A FIPA2000 Compliant Agent Development Environment. Proceedings of the Fifth International Conference on Autonomous Agents (AGENTS '01), ACM, 2001, pp. 216–217, doi: 10.1145/375735.376120.

[33] Boissier, O.—Bordini, R.—Hubner, J.—Ricci, A.—Santi, A.: Multi-Agent Oriented Programming with JaCaMo. Science of Computer Programming, Vol. 78, 2013, No. 6, pp. 747–761, doi: 10.1016/j.scico.2011.10.004.

[34] Frantz, Ch.: Micro-Agents Revisited: A Modern Reimplementation of the Micro-Agent Layer of the Otago Agent Platform (OPAL). Master thesis, University of Koblenz-Landau, Germany, 2010.

[35] Haq, F.—Kunz, T.: Simulation vs. Emulation: Evaluating Mobile Ad Hoc Network Routing Protocols. Proceedings of the International Workshop on Wireless Ad-Hoc Networks (IWWAN 2005), London, 2005.

Piotr Nawrocki is Assistant Professor in the Department of Computer Science at the AGH University of Science and Technology, Poland. His research interests include distributed systems, computer networks, mobile systems, service-oriented architectures and Mobile Cloud Computing. He obtained his Ph.D. in computer science from AGH Krakow.

Bartłomiej Śnieżyński received his Ph.D. degree in computer science in 2004 from the AGH University of Science and Technology in Krakow, Poland. In 2004 he worked as Postdoctoral Fellow under the supervision of the Professor R. S. Michalski at the Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, USA. Currently, he is Assistant Professor in the Department of Computer Science at the AGH. His research interests include machine learning, multi-agent systems, and knowledge engineering.

Jakub Kołodziej is a computer science graduate student at the AGH University of Science and Technology in Krakow, Poland.

Computing and Informatics, Vol. 38, 2019, 817–850, doi: 10.31577/cai_2019_4_817

MULTILEVEL ALGEBRAIC APPROACH FOR PERFORMANCE ANALYSIS OF PARALLEL ALGORITHMS

Luisa D'Amore, Valeria Mele

University of Naples Federico II
Naples, Italy
e-mail: {luisa.damore, valeria.mele}@unina.it

Diego Romano

Institute of High Performance Computing and Networking (ICAR), CNR
Naples, Italy
e-mail: diego.romano@cnr.it

Giuliano Laccetti

University of Naples Federico II
Naples, Italy
e-mail: giuliano.laccetti@unina.it

Abstract. In order to solve a problem in parallel we need to undertake the fundamental step of splitting the computational tasks into parts, i.e. decomposing the problem solving. A whatever decomposition does not necessarily lead to a parallel algorithm with the highest performance. This topic is even more important when complex parallel algorithms must be developed for hybrid or heterogeneous architectures. We present an innovative approach which starts from a problem decomposition into parts (sub-problems). These parts will be regarded as elements of an algebraic structure and will be related to each other according to a suitably defined dependency relationship. The main outcome of such framework is to define a set of block matrices (dependency, decomposition, memory accesses and execution) which simply highlight fundamental characteristics of the correspond-

ing algorithm, such as inherent parallelism and sources of overheads. We provide a mathematical formulation of this approach, and we perform a feasibility analysis for the performance of a parallel algorithm in terms of its time complexity and scalability. We compare our results with standard expressions of speed up, efficiency, overhead, and so on. Finally, we show how the multilevel structure of this framework eases the choice of the abstraction level (both for the problem decomposition and for the algorithm description) in order to determine the granularity of the tasks within the performance analysis. This feature is helpful to better understand the mapping of parallel algorithms on novel hybrid and heterogeneous architectures.

Keywords: Complexity and performance of numerical algorithms, performance metrics, data decomposition, concurrency, parallel algorithms

Mathematics Subject Classification 2010: 65Y05, 65Y20, 68R01

1 INTRODUCTION AND MOTIVATION

Numerical algorithms are at the heart of the software that enable scientific discoveries. The development of effective algorithms has a tremendous impact on harnessing emerging computer architectures to achieve new science. The mapping problem, first considered in 1980s [8], refers to the implementation of algorithms on a given target architecture which is capable to maximize some performance metrics [5, 6, 31, 26, 32]. Due to the multidimensional heterogeneity of modern architectures, it is becoming increasingly clear that using the performance metrics in a one-size-fits-all approach fails to discover sources of performance degradation that hamper to deliver the desired performance level. The present article attempts to collect our efforts towards the development of a performance model, based on mathematical tools, guiding the understanding of computational tasks within an algorithm. We briefly summarize main novelties we provide in this work.

- We address the study of data dependencies in an algorithm, through the *dependency matrix*.
- We introduce the *decomposition matrix* describing a decomposition of the problem.
- We introduce the *execution matrix* describing the mapping of the algorithm on the computing machine.
- We define the *memory accesses matrix*, that helps us to define the software execution time.
- The block structure of the above matrices corresponds to the multiple levels (of the performance analysis) for the proposed approach. This feature is helpful for understanding the mapping of complex parallel algorithms solving real problems on novel hybrid and heterogeneous architectures.

- A set of parameters characterizing the matrices structure, namely their number of rows and columns, and a set of computing environment parameters, such as the execution time for one floating point operation, are used both to describe the problem and to compute speed up, efficiency, cost, overhead, scale up and operating point of the algorithm, *starting from the problem decomposition.*
- Even though for simplicity of notations we assume that at each level of parallelism the computing architecture is homogeneous, it is possible to extend the proposed framework considering – at the same level of decomposition – features of distributed algorithms on heterogeneous computing architectures. In this case, dependency matrix should firstly be employed to analyze problem and data decomposition; then, execution matrix, whose rows depend on the execution time of the machine operations at a given level of granularity, highlights the corresponding workload distribution at this level (cfr. Section 6).

The article is organized as follows. Section 2 will review basic concepts and definitions useful for setting up the mathematical framework. We define the decomposition matrix; following [33], we describe a parallel algorithm as an ordered set of operators, moreover we give the definition of complexity of the algorithm depending on the number of such operators; finally, we define the execution matrix describing the mapping of the algorithm on the target computing resource. Section 4 focuses on two metrics characterizing the algorithm performance, such as the scale up factor and the speed up. In Section 5, we analyse the performance of parallel algorithms arising from the same problem decomposition. We derive the Generalized Amdhal's Law and some important upper and lower bounds of the performance metrics. In Section 6, we consider the particular case where the operators of an algorithm have the same execution time (namely, the operators are the usual floating point operations); in other words, we are assuming to get a decomposition at the lowest level of granularity and we derive the standard expressions for the performance metrics. Section 7 introduces the access memory matrix and some useful performance metrics to evaluate specific aspects of the software implementation. In Section 8 conclusions are drawn.

1.1 Related Works and Criticism

The appropriate mapping depends upon both the specification of the algorithm and the underlying architecture. Firstly, it implies a transformation of the algorithm into an equivalent, but in a more appropriate form. Works on the mapping problem can be classified according to the used representation.

Graph based approaches perform transformations on the algorithm and the architecture, both represented as graphs. In this approach the algorithm is modeled in terms of graphs structures and the mapping in terms of graphs partitions [8]. Linear algebra approaches represent the graph and its data dependencies by a matrix, then they transform the graph by performing matrix operations. Language based approaches transform one form of program text into another form, where the target

form textually incorporates information about the architecture [25]. Characteristic based approaches represent the algorithm in terms of a set of characteristics which determines the transformations. Included in this category is the work of [28], where a technique which abstracts a computation in terms of its data dependencies is described. The method is based on a mathematical transformation of the index sets and of the data-dependency vectors associated with the given algorithm. Finally, we underline that there are a lot of scientific groups that are working on similar issue from the years. In particular we mention the Heterogeneous Computing Laboratory at University College in Dublin focusing on efficient use of heterogeneous architectures. They mainly focus the attention on workload distribution, data distribution, communication performance models and optimization of communication operations of parallel or distributed algorithm on the network, by analyzing partitioning workload in proportion to the speed of the processing elements [29, 35].

One common issue of the aforementioned approaches is that very often the model used for the representation and analysis of the algorithm cannot be explicitly employed for deriving the expression of performance metrics of the software. On the contrary, performance analysis is often accomplished with automatic tools on a combination of the algorithm and the parallel architecture on which it is implemented (the so-called parallel system), exploiting automating mappings, automatic translations, re-targeting mappings tracing, auto-tuning tools (such as: the PaRSEC runtime system [11], that provides a portable way to automatically adapt algorithms to new hardware trend). Nevertheless, these approaches ignore dependency among sub problems within the problem decomposition. Instead, our model, through the choice of the computing operators of the algorithm, allows to set a level of abstraction for the algorithm description; each level determines the granularity/detail of the performance analysis, and could be used to better understand the subsequent mapping on the computing architecture. This topic is mainly important to analyse performance of complex algorithm solving real problems.

2 PRELIMINARY CONCEPTS AND DEFINITIONS

We introduce a dependency relationship among the parts of a computational problem, among operators of the algorithm that solves the problem and, finally, among memory accesses of the algorithm. In this way we are able to define the matrices which are the foundations of the mathematical model we are going to introduce.

To this aim we first give some definitions[1].

Definition 1 (Computational Problem)**.** A computational problem $\mathcal{B}_{N_r}$ is the mathematical problem specified by an input/output functional relation $\mapsto$:

$$\mathcal{B}_{N_r} : In_{\mathcal{B}_{N_r}} \mapsto Out_{\mathcal{B}_{N_r}}$$

[1] It is worth to note that these definitions do not claim to be general. Their aim is to establish the mathematical setting on which we will restrict our attention.

where N_r is the input data size and $r \in \mathbb{N}$, between the data and the solution of $\mathcal{B}_{N_r}$.

In the following the computational problem $\mathcal{B}_{N_r}$ is identified by the triple:

$$\mathcal{B}_{N_r} \equiv \left(N_r, In_{\mathcal{B}_{N_r}}, Out_{\mathcal{B}_{N_r}}\right).$$

Definition 2 (Similar Computational Problem)**.** Two computational problems, $\mathcal{B}_{N_r}$ and $\mathcal{B}_{N_q}$, are said similar if they are specified by the same functional relation $\mapsto$ and they only differ in the input/output data size. If $\mathcal{B}_{N_r}$ and $\mathcal{B}_{N_q}$ are similar we write $\mathcal{B}_{N_r} \mathscr{S} \mathcal{B}_{N_q}$.

Dividing a computation into smaller computations, some or all of which may potentially be executed in parallel, is the key step in designing parallel algorithms. These parts often share input, output, or intermediate data, such that the output of one part is the input of another. In our mathematical framework these relationships will be described by the so called decomposition matrix. In order to define this matrix we need to introduce the following algebraic structure

Definition 3 (Dependency Group)**.** Let $(\mathcal{E}, \pi)$ be a group and let $\pi_{\mathcal{E}}$ be a strict partial order relation on $\mathcal{E}$, which is compatible with π. We say that any element of $\mathcal{E}$, let us say A, depends on an element of $\mathcal{E}$, let us say B, if $A \pi_{\mathcal{E}} B$, and we write $A \leftarrow B$. If A and B do not depend on each other we write $A \nleftarrow B$. The group $(\mathcal{E}, \pi)$ equipped with $\pi_{\mathcal{E}}$ is called dependency group and it is denoted as $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$.

Remark 1. Since $\pi_{\mathcal{E}}$ is transitive, from Definition 2 it follows that any two elements of $\mathcal{E}$, let us say A and B, are independent if there is no any relationship between them. In this case we write $A \nleftarrow B$ and $B \nleftarrow A$, or even $A \nleftrightarrow B$.

Now we are able to define the dependency matrix on $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$.

Definition 4 (Dependency Matrix)**.** Given $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$, the matrix $\mathcal{F}$, of size $r_D \cdot c_D$, whose elements $d_{i,j} \in (\mathcal{E}, \pi)$, are such that $\forall i \in [0, r_D - 1]$

$$d_{i,j} \nleftrightarrow d_{i,s}, \quad \forall s, j \in [0, c_D - 1] \tag{1}$$

and $\forall i \in [1, r_D - 1], \quad \exists q \in [0, c_D - 1]$ s.t.

$$d_{i,j} \leftarrow d_{i-1,q}, \quad \forall j \in [0, c_D - 1], \tag{2}$$

while the others elements are set equal to zero, is said the dependency matrix.

Remark 2. Matrix $\mathcal{F}$ is unique (through its construction), up to a permutation of elements on the same row. $c_{\mathcal{F}}$ is said the concurrency degree[2] of $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$ and $r_{\mathcal{F}}$ is the said the dependency degree of $\mathcal{E}$. Concurrency degree measures the intrinsic concurrency among sub-problems of $(\mathcal{E}, \pi, \pi_{\mathcal{E}})$. It is obtained as the number of columns of $\mathcal{F}$.

[2] A similar concept has already been highlighted in [22]

2.1 The Problem Decomposition

Let $S(\mathcal{B}_{N_r})$ denote the solution[3] of $\mathcal{B}_{N_r}$.

Definition 5 (Decomposition of a Computational Problem)**.** Given $\mathcal{B}_{N_r}$, any finite set of computational problems $\{\mathcal{B}_{N_i}\}_{i=0,\ldots,k-1}$, where $k \in \mathbb{N}$, such that $\mathcal{B}_{N_r} \leftarrow \mathcal{B}_{N_i}$, where $N_i < N_r$, and

$$\sum_{i=0}^{k-1} N_i \geq N_r$$

is called a decomposition of $\mathcal{B}_{N_r}$. $\mathcal{B}_{N_i}$ denotes a sub-problem of $\mathcal{B}_{N_r}$. A decomposition of $\mathcal{B}_{N_r}$, which is denoted as

$$D_k(\mathcal{B}_{N_r}) := \{\mathcal{B}_{N_0}, \ldots, \mathcal{B}_{N_{k-1}}\} \tag{3}$$

defines the computational problem

$$D_k(\mathcal{B}_{N_r}) \equiv \left(\sum_{i=0}^{k-1} N_i, In_{\mathcal{B}_{N_r}}, Out_{\mathcal{B}_{N_r}}\right).$$

The set of all the decompositions of $\mathcal{B}_{N_r}$ is denoted as $\mathcal{DB}_{N_r}$.

Definition 6 (Similar Decompositions)**.** Given $\mathcal{B}_{N_r} \mathscr{S} \mathcal{B}_{N_q}$, two decompositions $D_{k_i}(\mathcal{B}_{N_r})$ and $D_{k_j}(\mathcal{B}_{N_q})$ are called similar if

$$k_i = card(D_{k_i}(\mathcal{B}_{N_r})) = card(D_{k_j}(\mathcal{B}_{N_q})) = k_j$$

and

$$\forall \mathcal{B}_{N_s} \in D_{k_i}(\mathcal{B}_{N_r}) \exists! \mathcal{B}_{N_t} \in D_{k_j}(\mathcal{B}_{N_q}) : \mathcal{B}_{N_s} \mathscr{S} \mathcal{B}_{N_t}$$

and we write

$$D_{k_i}(\mathcal{B}_{N_r}) \mathscr{S} D_{k_j}(\mathcal{B}_{N_q}).$$

Remark 3 (Decomposition Matrix)**.** In order to capture interactions among parts (or sub-problems) of $\mathcal{B}_{N_r}$, we use the dependency matrix on $D_k(\mathcal{B}_{N_r})$. More precisely, by using Definition 2 we introduce the group $(D_k(\mathcal{B}_{N_r}), g_{sol})$ where g_{sol} is any application between any two elements $\mathcal{B}_{N_i}$ and $\mathcal{B}_{N_j}$ of $D_k(\mathcal{B}_{N_r})$, equipped with the strict partial order relation $\pi_{D_k(\mathcal{B}_{N_r})}$. Then, we construct the (unique) dependency matrix $\mathcal{F}$ corresponding to the decomposition $D_k(\mathcal{B}_{N_r})$. In the following we denote this matrix as $M_D(D_k(\mathcal{B}_{N_r}))$, or M_{D_k} for simplicity, and we refer to it as the **decomposition matrix**. Given $D_k(\mathcal{B}_{N_r})$, let c_{D_k} denote the number of columns. This is the (unique) concurrency degree of $\mathcal{B}_{N_r}$. Let r_{D_k} denote the row number of rows. This is the (unique) dependency degree of $\mathcal{B}_{N_r}$. Concurrency degree measures the intrinsic concurrency among sub-problems of $\mathcal{B}_{N_r}$.

[3] Here, for the sake of simplicity, we assume that $S(\mathcal{B}_{N_r})$ exists and it is unique.

If there are not empty elements, the problem $\mathcal{B}_{N_r}$ has the highest intrinsic concurrency, hence we give the following

Definition 7 (Perfectly Decomposed Problems)**.** $\mathcal{B}_{N_r}$ is said perfectly decomposed if $\exists D_k(\mathcal{B}_{N_r})$ and M_D such that

- $c_D > 1$,
- $\forall i, j,\ d_{i,j} \neq \emptyset$.

The next step is to take these parts and assign them (i.e., the mapping step) onto the computing machine. In the next section we introduce the computing environment characterized by the set of logical-operational operators/operations that it is able to apply/execute and by the memory system.

2.2 The Computing Architecture

Let $\mathcal{M}_P$ denote the computing architecture equipped with $P \geq 1$ processing elements with specific logical-operational capabilities such as: basic operations (arithmetic, ...), special functions evaluations (sin, cos, ...), solvers (integrals, equations system, non linear equations, ...). These are the computing operators of $\mathcal{M}_P$. In particular, we will use the following characterization of operators of $\mathcal{M}_P$.

Definition 8 (Computing Operators)**.** The operator I^j of $\mathcal{M}_P$ is a correspondence between $\mathbb{R}^s$ and $\mathbb{R}^t$, where $s, t \in \mathbb{N}$ are positive integers.

Given $\mathcal{M}_P$, the set

$$Cop_{\mathcal{M}_P} := \{I^j\}_{j\in[0,q-1]}, \quad q \in \mathbb{N}$$

where operators I^j are taken without repetitions, characterizes logical-operational capabilities of $\mathcal{M}_P$.

Operators, properly organized, provide the solution to $\mathcal{B}_{N_r}$, as stated in the following definition.

Definition 9 (Solvable Problems)**.** $\mathcal{B}_{N_r}$ is solvable in $\mathcal{M}_P$ if

$$\exists D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r} : \forall \mathcal{B}_{N_j} \in D_k(\mathcal{B}_{N_r}) \qquad \exists I^j \in Cop_{\mathcal{M}_P} : I^j[\mathcal{B}_{N_j}] = S(\mathcal{B}_{N_j})$$

that is, if there exists any relation

$$\theta : \mathcal{B}_{N_j} \in D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r} \longmapsto I^j \in Cop_{\mathcal{M}_P}. \tag{4}$$

In particular, we say that a decomposition is suited for $\mathcal{M}_P$ if θ is a function. From now on, we assume any problem $\mathcal{B}_{N_r}$ to be solvable, and we assume a decomposition $D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r}$ suited for $\mathcal{M}_P$ to be fixed[4].

We associate to each $I^i \in Cop_{\mathcal{M}_P}$ in $\mathcal{M}_P$ the parameter t_i, denoting the execution time measured, for instance, in seconds. If $I^i \equiv \varnothing$, we set $t_\varnothing = 0$.

[4] Note that there is no loss of generality.

2.3 Memory Hierarchy and Communications

A computing architecture is not only characterized by the set of operations that is able to apply, but also by the memory system. Indeed, the effective performance of an algorithm relies not only on the processor speed for arithmetic operations but also on the ability of the memory system to feed data to the processor. At the logical level, a memory system, possibly consisting of multiple levels of caches, let us say L, takes in a request for a memory word and returns a block of data containing the requested word after $tmem_l$ nanoseconds. Here, $tmem_l$ consists of memory latency time, measuring the time between the of a read request and the release of its corresponding data, plus the data transfer time. Memory latency time depends on the latency of the memory, which is typically bridged by a hierarchy of successively faster memory that rely on locality of data reference to deliver higher performance. The rate at which data can be moved from the memory to the processor determines the bandwidth of the memory system. It is determined by the memory bus bandwidth as well as by the memory unit.

So, we consider a computing machine $\mathcal{M}_P$ such that

- its memory has $L \geq 2$ levels,
- for each level l, where $l \in [1, L]$, nd_l denotes the bandwidth, i.e. the rate for transferring (read/write) data of the same type. It is $nd_l \geq 1$,
- memory access time is
$$tmem_l := t_l^{acc} + t_l^{trans}$$
where t_l^{acc} measures the memory latency time while t_l^{trans} measures the transfer time. Moreover, let $tcalc$ be the execution time of one floating point operation. We assume that
$$tcalc < tmem_1 \leq tmem_2 \leq \ldots \leq tmem_L$$

and

$$tmem_l = \alpha_l^{mem} \cdot tmem, \quad \forall l \in [1, L], \quad \alpha_l \in \Re - \{\infty\} \tag{5}$$

where $tmem$ denotes the execution time needed for moving a memory word.

Remark 4. In case of latency bound algorithms (i.e., t_l^{acc} prevails over t_l^{trans}) or bandwidth bound algorithms (i.e., t_l^{trans} prevails over t_l^{acc}) the model could be properly refined by specifying $tmem_l$.

Communication means moving data, either between levels of a memory hierarchy or between processors of the reference machine. Hence, this mathematical framework includes the communication level within the memory accesses, and the last (that is the slowest) memory level (L^{th}) refers to it. Let $tcom := tmem_L$ denote the unitary communication time, we assume that $tcom \gg tmem_i, i \in [1, L-1]$.

Such computing machine is denoted as $\mathcal{M}_{P,L,nd_L}$ or simply as $\mathcal{M}_P$ if there is no ambiguity.

3 THE ALGORITHM

In literature, an algorithm is any procedure consisting of a finite number of unambiguous rules that specify a finite sequence of operations bringing to a solution of a problem or of a specific class of problems [24]. Analogously, we define an algorithm as a finite set of operators solving $\mathcal{B}_{N_r}$.

Definition 10 (Algorithm)**.** Given $D_k(\mathcal{B}_{N_r})$, an algorithm solving $\mathcal{B}_{N_r}$, indicated as

$$A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P} = \{I^{i_0}, I^{i_1}, \ldots, I^{i_k}\}$$

is a sequence of elements (not necessarily distinct) of $Cop_{\mathcal{M}_P}$, such that[5]

$$I^{i_k} \circ I^{i_{k-1}} \circ \ldots \circ I^{i_0}[\mathcal{B}_{N_r}] = S(\mathcal{B}_{N_r})$$

where $j \in [0, card(Cop_{\mathcal{M}_P}) - 1]$, and such that there is a bijective correspondence

$$\gamma : \mathcal{B}_{N_\nu} \in D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r} \longleftrightarrow I^{i_j} \in A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}. \tag{6}$$

Every ordered subset of $A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}$ is a sub-algorithm of $A_{D_k(\mathcal{B}_{N_r}),\mathcal{M}_P}$.

For simplicity of notations and when there is no ambiguity, we indicate algorithms briefly as $A_{k,P}$.

Definition 11 (Equal Algorithms)**.** Two algorithms

$$A^i_{k,P} = \{I^{i_0}, I^{i_1}, \ldots, I^{i_k}\}, \quad A^j_{k,P} = \{I^{j_0}, I^{j_1}, \ldots, I^{j_k}\}$$

are said equal if $\forall s \in [0, k], I^{i_s} \equiv I^{j_s}$.

We note that in this mathematical framework, *two equal algorithms have the same cardinality.*

Definition 12 (Granularity Set of an Algorithm)**.** Given $A_{k,P}$, the subset $\mathcal{G}(A_{k,P})$ of $A_{k,P}$ made of distinct operators of $A_{k,P}$ defines the granularity set of $A_{k,P}$. Two algorithms

$$A^i_{k,P} = \{I^{i_0}, I^{i_1}, \ldots, I^{i_k}\}, \quad A^j_{k,P} = \{I^{j_0}, I^{j_1}, \ldots, I^{j_k}\}$$

have the same granularity if $\mathcal{G}(A^i_{k,P}) \equiv \mathcal{G}(A^j_{k,P})$.

Let $AL_{\mathcal{B}_{N_r}}$ (or simply AL) be the set of algorithms that solve $\mathcal{B}_{N_r}$, obtained by varying $\mathcal{M}_P$, the number of processing units P and $D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r}$. Even if one can easily formulate infinite variations of an algorithm that do the same thing, for simplicity in the following we assume AL to be finite.

[5] In the following we use the symbol $\circ$ to denote correspondence composition.

Definition 13 (The Quotient Set $\frac{AL}{\varrho}$)**.** Let

$$\varphi : A_{k,P} \in AL \longrightarrow D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r} \tag{7}$$

be the surjective correspondence which induces on AL an equivalence relationship ϱ of AL in itself, such that

$$\varrho(A_{k,P}) = \{\widetilde{A_{k,P}} \in AL : \varphi(\widetilde{A_{k,P}}) = \varphi(A_{k,P})\}. \tag{8}$$

The set $\varrho(A_{k,P})$ consists of algorithms of AL associated with the same decomposition $D_k(\mathcal{B}_{N_r}) \in \mathcal{DB}_{N_r}$. ϱ induces the quotient set $\frac{AL}{\varrho}$, whose elements are disjoints and finite subsets of AL determined by ϱ, that is they are equivalence classes under ϱ.

In the following we assume $A_{k,P}$ to represent its equivalence class in AL.

Definition 14 (Complexity)**.** The cardinality of $A_{k,P}$, denoted as $C(A_{k,P})$, is said complexity of $A_{k,P}$. It is $C(A_{k,P}) := card(A_{k,P}) = k$.

Remark 5. $C(A_{k,P}) = k$ equals to the number of non empty elements of M_{D_k}, i.e. the decomposition matrix defined on $D_k(\mathcal{B}_{N_r})$. By virtue of the bijective correspondence γ in (6), it holds that

$$card(A_{k,P}) = card(D_k(\mathcal{B}_{N_r})) = k, \quad \forall A_{k,P} \in \varrho(A_{k,P}). \tag{9}$$

Each algorithm belonging to the same equivalence class according to ϱ has the same complexity. An integer (the complexity) is therefore associated with each element $\varrho(A_{k,P})$ of quotient set $\frac{AL}{\varrho}$ which induces an ordering relation between the equivalence classes in $\frac{AL}{\varrho}$: therefore there is a minimum complexity for algorithms that solve the problem $\mathcal{B}_{N_r}$.

Remark 6 (Similar Algorithms)**.** Given $\mathcal{B}_{N_r}\mathscr{S}\mathcal{B}_{N_q}$ and their relative similar decompositions $D'_k(\mathcal{B}_{N_r})\mathscr{S}D''_k(\mathcal{B}_{N_q})$ (see Definition 6), algorithms belonging to $\varrho(A_{k,P}) = \varphi^{-1}(D'_k(\mathcal{B}_{N_r}))$ (see (7)) are similar to algorithms belonging to $\varrho(A_{k,P}) = \varphi^{-1}(D''_k(\mathcal{B}_{N_q}))$. From Definition 6 and 14 and (9), it follows that

$$A_{k,P}\mathscr{S}A_{k,P} \Longrightarrow C(A_{k,P}) = C(A_{k,P}) = k,$$

that is, *similar algorithms have the same complexity.*

Remark 7. As we can associate $I^{i_k} \in A_{k,P}$ to each subproblem according to γ, then the operators of $A_{k,P}$ inherit the dependencies existing between subproblems of $\mathcal{B}_{N_r}$, but they do not inherit independency, because for instance, two operators may depend on the availability of computing units of $\mathcal{M}_P$ during their execution [33].

Remark 8 (Execution Matrix)**.** According to Definition 3, we introduce the group $\left(\mathcal{P}\left(A_{k,P}\right), \circ, \pi_{A_{k,P}}\right)$ where $\mathcal{P}\left(A_{k,P}\right)$ is the set of all the sub-algorithms of $A_{k,P}$, and

$\pi_{A_{k,P}}$ is the strict partial order relation between any two elements of $\mathcal{P}(A_{k,P})$ that guarantees that two elements cannot be performed in any arbitrary order and simultaneously[6]. We construct matrix $\mathcal{F}$ of order $r_E \cdot c_E$, where $c_E = P$[7] as a dependency matrix (see Definition 4). The number of columns of this matrix will represent the maximum number of sub-algorithms that can be performed simultaneously on $\mathcal{M}_P$. In the following, we denote this matrix as **execution matrix** and we refer to it by using the symbol $M_E(A_{k,P}) = (e_{i,j})$ or simply $M_{E_{k,P}}$ if there is no ambiguity. Matrix $M_{E_{k,P}}$ is unique up to a permutation of elements on the same row. This matrix can be placed in analogy with the execution graphs (see [7, 10, 12, 30]) that are often used to describe the sequence of steps of an algorithm on a given machine for a particular input or a particular configuration.

Remark 9. As it is $card(A_{k,P}) = card(D_k(\mathcal{B}_{N_r}))$, then M_{D_k} and $M_{E_{k,P}}$ have the same number of non empty elements (k), whichever is $P \geq 1$. If $c_E = P = c_{D_k}$, there exists $A_{k,P}$ whose matrix $M_{E_{k,P}}$ has exactly the same structure of the matrix M_{D_k}.

Definition 15. $A_{k,P}$ is said perfectly parallel if: $c_E > 1$ and $\forall i, j : e_{i,j} \neq \emptyset$. $A_{k,P}$ is said sequential if: $c_E = 1$ and $\nexists j > 1 : e_{i,j} \neq \emptyset$. $A_{k,P}$ is said (simply) parallel if: $c_E > 1$ and $\exists i, j : e_{i,j} = \emptyset$. Moreover, every row of matrix $M_{E_{k,P}}$ such that $\exists e_{i,j} \neq \emptyset$, where $j > 1$, is a parallel sub-algorithm of $A_{k,P}$. Every row of matrix $M_{E_{k,P}}$ such that $\exists! e_{i,j} \neq \emptyset$ is a sequential sub-algorithm of $A_{k,P}$.

Remark 10. Observe that the concurrency degree of $\mathcal{B}_{N_r}$ in a given decomposition provides an upper limit to the maximum number of independent sub-algorithms executable simultaneously on the machine. The dependency degree provides a lower limit to the execution time of the algorithm.

Finally, from correspondence γ (see (6)), we say that $\mathcal{B}_{N_r}$ is solvable in $\mathcal{M}_P \Leftrightarrow \exists D_k(\mathcal{B}_{N_r}) \in \mathcal{D}\mathcal{B}_{N_r} : \exists A_{k,P}$ that solves $\mathcal{B}_{N_r}$.

Theorem 1. If $\mathcal{B}_{N_r}$ is perfectly decomposed according to D_k, $\exists \mathcal{M}_P$, where $P > 1$, such that $\exists A_{k,P}$ perfectly parallel that solves $\mathcal{B}_{N_r}$.

Proof. If $\mathcal{B}_{N_r}$ is perfectly decomposed then the matrix M_{D_k} has not empty elements and has order greater than 1. Since $card(A_{k,P}) = card(D_k(\mathcal{B}_{N_r})) = k$, there exists $A_{k,P}$ with execution matrix $M_{E_{k,P}}$ of order $r_E \cdot c_E$, with only non zero elements,

[6] The condition that two elements cannot be performed in any arbitrary order induces the inheritance of dependencies between decomposition subproblems and algorithm operators, while the condition that two elements cannot be performed simultaneously – relating to availability of resources – adds possible reasons for dependency between operators, which depend on the machine on which algorithm A is intended to run [33].

[7] In general $c_E \leq P$, but we can exclude cases where dependencies existing between subproblems do not allow to use all the computing units available, i.e., in which $c_E < P$, because they can be easily taken back to the case where $c_E = P$.

such that $r_E = r_{D_k}$ and $c_E = P = c_{D_k}$ or[8] $r_E = n \cdot r_{D_k}$ and $c_E = P = c_{D_k}/n$ with the integer n is such that $n < c_{D_k}$ and $c_{D_k} \bmod n = 0$. In conclusion, $M_{E_{k,P}}$ has $c_E = P > 1$ columns, and no rows have an empty element; so $A_{k,P}$ is perfectly parallel. □

4 PERFORMANCE METRICS

In this section we employ the mathematical settings we introduced in Section 2, in order to define two quantities to measure the performance of an algorithm: the scale up and the speed up.

Let us consider two decompositions $D_{k_i}(\mathcal{B}_N)$ and $D_{k_j}(\mathcal{B}_N)$ in $\mathcal{DB}_N$. Let us consider $A_{k_i,P}$ and $A_{k_j,P}$ representing their equivalence class in AL. We introduce the following quantity:

Definition 16 (Scale Up Factor)**.** If $A_{k_i,P}$ and $A_{k_j,P}$ have the same granularity set (see Definition 12), the ratio

$$Sc_{up}(A_{k_i,P}, A_{k_j,P}) := \frac{k_i}{k_j} \tag{10}$$

is said scale up factor of $\varrho(A_{k_j,P})$ measured with respect to $\varrho(A_{k_i,P})$.

Note that by using Definition 14, we get

$$Sc_{up}(A_{k_i,P}, A_{k_j,P}) = \frac{C(A_{k_i,P})}{C(A_{k_j,P})}. \tag{11}$$

Next proposition quantifies the scale up when we solve the same problem with an algorithm that is the concatenation of several algorithms which are similar to the first one, with polynomial complexity of degree d.

Proposition 1. Given $\mathcal{B}_{N_r}$, $D_k(\mathcal{B}_{N_r})$ and $D_{k'}(\mathcal{B}_{N_r}) = \{D_{k'_i}(\mathcal{B}_{N_q})\}_{i=1,\mu}$ where

- $N_q = N_r/\mu$ with $\mu \in N$, $\mu \leq N_r$, and $N_r \bmod \mu = 0$,
- $\mathcal{B}_{N_q} \mathscr{S} \mathcal{B}_{N_r}$,
- $D_k \mathscr{S} D_{k'_i} \mathscr{S} D_{k'_j}$, $\forall i \neq j$.

Consider $A_{k,P} \in \varphi^{-1}(D_k(\mathcal{B}_{N_r}))$ and $A_{k'_i,P} \in \varphi^{-1}(D_{k'_i}(\mathcal{B}_{N_q}))$ and assume that $C(A_{k,P}) = k = \mathcal{P}^d(N_r)$ and $C(A_{k'_i,P}) = k'_i = \mathcal{P}^d(N_q)$ where

$$\mathcal{P}^d(x) = a_d x^d + a_{d-1} x^{d-1} + \ldots + a_0, \quad a_d \neq 0 \in \Pi_d, x \in \Re,$$

[8] If the concurrency degree c_{D_k} is so great that we cannot imagine a real machine with so many units, we can always use a number of computing units $P = c_{D_k}/n$ with $c_{D_k} \bmod (n) = 0$. This will mean that the execution matrix of $A_{k,P}$ will have n times more rows and n times less columns than the dependency matrix.

then $Sc_{up}(A_{k,P}, A_{k',P}) = \xi(N_r, \mu) \cdot \mu^{d-1}$ where

$$\xi(N_r, \mu) := \frac{a_d + \frac{a_{d-1}}{N_r} + \ldots + \frac{a_0}{N_r^d}}{a_d + a_{d-1}\frac{\mu}{N_r} + \ldots + a_0\frac{\mu^d}{N_r^d}}. \tag{12}$$

Proof. We have that

$$C(A_{k',P}) = \sum_{i=1}^{\mu} C(A_{k'_i,P}) = \mu \cdot \mathcal{P}^d(N_q), \tag{13}$$

then from the (11) it follows that

$$Sc_{up}(A_{k,P}, A_{k',P}) = \frac{C(A_{k,P})}{C(A_{k',P})} = \frac{\mathcal{P}^d(N_r)}{\mu \cdot \mathcal{P}^d(N_q)}, \tag{14}$$

that is

$$Sc_{up}(A_{k,P}, A_{k',P}) = \frac{a_d N_r^d + a_{d-1} N_r^{d-1} + \ldots + a_0}{\mu \cdot \left(a_d N_q^d + a_{d-1} N_q^{d-1} + \ldots + a_0\right)}. \tag{15}$$

Since $N_q = N_r/\mu$, then it is

$$Sc_{up}(A_{k,P}, A_{k',P}) = \frac{a_d (\mu N_q)^d + a_{d-1} (\mu N_q)^{d-1} + \ldots + a_0}{\mu \cdot \left(a_d N_q^d + a_{d-1} N_q^{d-1} + \ldots + a_0\right)}, \tag{16}$$

then thesis follows from the (12). □

It comes out the following result:

Corollary 1. If N_r is fixed, and $\mu \simeq N_r$, it is $\xi(N_r, \mu) = const$ where $const \in (0, 1]$ and $Sc_{up}(A_{k,P}, A_{k',P}) \leq N_r^{d-1}$. If μ is fixed, it is $\lim_{N_r \to \infty} \xi(N_r, \mu) = const$ where $const \in (0, 1]$ and $\lim_{N_r \to \infty} Sc_{up}(A_{k,P}, A_{k',P}) \leq \mu^{d-1}$. If $a_i = 0$, $\forall i < d$, then $\xi(N_r, \mu) = 1$ and $S_{up}(A_{k,P}, A_{k',P}) = \mu^{d-1}$, $\forall \mu$.

We observe that in the following, when we need to refer to the execution time of computing operators of $A_{k,P}$, we will use the notation $\beta^{calc}_{\ldots, M_{E_{k,P}}}$ for the parameters highlighting the execution matrix $M_{E_{k,P}}$ and characterizing the mapping of the algorithm on the machine $\mathcal{M}_P$. We assume that

$$\forall I^{i_j} \in Cop_{\mathcal{M}_P}, \quad t_{i_j} = \beta^{calc}_{i_j, M_{E_{k,1}}} \cdot tcalc, \beta^{calc}_{i_j, M_{E_{k,1}}} \in \Re, \quad \beta^{calc}_{i_j, M_{E_{k,1}}} \geq 1. \tag{17}$$

Definition 17 (Row Execution Time)**.** The quantity

$$T_r(A_{k,P}) := \max_{j \in [0, c_E - 1]} t_{r_j} \tag{18}$$

is said execution time of the row r of $M_{E_{k,P}}$ (which is a sub-algorithm of $A_{k,P}$).

Remark 11. Let $\beta^{calc}_{r,M_{E_{k,P}}} := \max_{j\in[0,c_E-1]} \beta^{calc}_{r_j,M_{E_{k,P}}}$, then

$$T_r(A_{k,P}) = \max_{j\in[0,c_E-1]} \beta^{calc}_{r_j,M_{E_{k,P}}} \cdot tcalc = \beta^{calc}_{r,M_{E_{k,P}}} \cdot tcalc.$$

Note that $\beta^{calc}_{i_j,M_{E_{k,1}}} \geq 1$, then $\beta^{calc}_{r,M_{E_{k,1}}} \geq 1$.

Definition 18 (Execution Time)**.** The quantity

$$T(A_{k,P}) := \sum_{r=0}^{r_E-1} T_r(A_{k,P}) \tag{19}$$

is said execution time of $A_{k,P}$.

Remark 12. Let $\beta^{calc}_{M_{E_{k,P}}} := \sum_{r=0}^{r_E-1} \beta^{calc}_{r,M_{E_{k,P}}}$, then $\beta^{calc}_{M_{E_{k,P}}} \geq r_E$.

$$T(A_{k,P}) = \beta^{calc}_{M_{E_{k,P}}} \cdot tcalc. \tag{20}$$

Remark 13. Let

$$\beta^{calc}_{sum,M_{E_{k,P}}} := \sum_{i=0}^{r_E-1} \sum_{j=0}^{c_E-1} \beta^{calc}_{i_j,M_{E_{k,P}}}. \tag{21}$$

Then, if $P = 1$, then $\beta^{calc}_{M_{E_{k,P}}} := \beta^{calc}_{sum,M_{E_{k,P}}}$.

Remark 14. Let

- $r_{seq} \leq r_E$ denote the number of rows of $M_{E_{k,P}}$ with only one non-empty element (sequential sub-algorithms of $A_{k,P}$);
- $r_{par} = r_E - r_{seq}$, with $r_{par} \leq r_E$, denote the number of rows of $M_{E_{k,P}}$ with more than one non empty element.

From the sequence $i = 0, \ldots, r_E - 1$, numbering the r_E rows of $M_{E_{k,P}}$, two sub-sequences of indices originate $\{i_q\}_{q\in[0,r_{seq}-1]}$, and $\{i_r\}_{r\in[0,r_{par}-1]}$, and the following definition follows.

Definition 19 (Parallel Execution Time)**.** The quantity

$$T_{par}(A_{k,P}) := \sum_{r=0}^{r_{par}-1} T_{i_r}(A_{k,P}) \tag{22}$$

is said parallel execution time of $A_{k,P}$.

Definition 20 (Sequential Execution Time)**.** The quantity

$$T_{seq}(A_{k,P}) := \sum_{q=0}^{r_{seq}-1} T_{i_q}(A_{k,P}) \tag{23}$$

is said sequential execution time of $A_{k,P}$.

The (19) can be written as

$$T(A_{k,P}) = T_{seq}(A_{k,P}) + T_{par}(A_{k,P}). \tag{24}$$

This states that, by looking at matrix $M_{E_{k,P}}$, the model expresses the size of the parallel and the sequential parts composing the execution time $A_{k,P}$.

Let

$$R^{calc}(A_{k,P}) := \frac{\beta^{calc}_{M_{E_{k,P}}}}{r_E}. \tag{25}$$

R^{calc} is the parameter of the algorithm $A_{k,P}$ depending on the most computationally intensive sub-algorithms of A.

It holds

$$T(A_{k,P}) = R^{calc}(A_{k,P}) \cdot r_E \cdot tcalc = \sum_{r=0}^{r_E-1} \beta^{calc}_{r,M_{E_{k,P}}} \cdot tcalc. \tag{26}$$

Remark 15. If $P = 1$, since $r_E = C(A_{k,1}) = k$ from (25), it is

$$R^{calc}(A_{k,1}) := \frac{\beta^{calc}_{all,M_{E_{k,P}}}}{k}. \tag{27}$$

Corollary 2. From the (25) it follows

$$T(A_{k,1}) = k \cdot R^{calc}(A_{k,1}) \cdot tcalc, \tag{28}$$

$$T(A_{k,P}) \geq r_D \cdot R^{calc}(A_{k,P}) tcalc \tag{29}$$

and it assumes its minimum value when $r_E = r_D$.

$$T(A_{k,P}) = (r_{seq} + r_{par}) \cdot R^{calc}(A_{k,P}) \cdot tcalc. \tag{30}$$

Definition 21 (Speed Up in $\frac{AL}{\rho}$)**.** Given

- $\mathcal{B}_{N_r}$,
- $A_{k,P} \in \varphi^{-1}(D_k(\mathcal{B}_{N_r}))$ where $P > 1$,
- two different decompositions $D_k(\mathcal{B}_{N_r})$ and $D_{k'}(\mathcal{B}_{N_r})$,
- $A_{k',1} \in \varphi^{-1}(D_{k'}(\mathcal{B}_{N_r}))$

where $\mathcal{M}_1$ and $\mathcal{M}_P$ differ only in the number of processing elements, if $\mathcal{G}(A_{k,P}) = \mathcal{G}(A_{k',P})$, then the speed up of $A_{k,P}$ with respect to $A_{k',1}$ is

$$Sp(A_{k,P}, A_{k',1}) := Sc_{up}(A_{k,P}, A_{k',1}) \cdot \frac{T(A_{k,1})}{T(A_{k,P})} = \frac{k'}{k} \cdot \frac{\beta^{calc}_{sum,M_E(A_{k,P})}}{\beta^{calc}_{M_E(A_{k,P})}}. \tag{31}$$

Remark 16 (Ideal Speed Up)**.** Since it is always[9]

$$\beta^{calc}_{sum,M_E(A_{k,P})} \leq P \cdot \beta^{calc}_{M_E(A_{k,P})},$$

then it holds that

$$Sp(A_{k,P}, A_{k',1}) \leq Sc_{up}(A_{k,P}, A_{k',1}) \cdot P. \tag{32}$$

Definition 22 (Speed Up in $\rho(A_{k,P})$)**.** The speed up of $A_{k,P}$ with respect to $A_{k,1}$ is

$$Sp(A_{k,P}) = \frac{T(A_{k,1})}{T(A_{k,P})} = \frac{\beta^{calc}_{sum,M_E(A_{k,P})}}{\beta^{calc}_{M_E(A_{k,P})}}. \tag{33}$$

Example 1. Preliminary results on speed up and scale up validating this approach appear in [27, 3, 14]. In [27], the authors addressed the development of a modular implementation of MGRIT (MultiGrid-In-Time), a parallel iterative algorithm to solve linear and nonlinear systems that arise from the discretization of evolutionary models with a parallel in time approach in the context of the PETSc (the Portable, Extensible Toolkit for Scientific computing) library. The algorithm speed up has been analyzed a priori to provide the best number of processing elements and grid levels needed to address the scaling of MGRIT. In [3, 14], the performance analysis carried out by the authors using the scale up factor suggests the introduction of a highly scalable problem decomposition.

5 ALGORITHMS IN THE SAME EQUIVALENCE CLASS

We consider algorithms that are in the same equivalence class, i.e. those corresponding to the same decomposition of the problem

Theorem 2. $\forall \mathcal{B}_{N_r}$ perfectly decomposed according to the decomposition $D_k(\mathcal{B}_{N_r})$, and $\forall A_{k,P}$ perfectly parallel algorithm that solves it on $\mathcal{M}_P$ with $P > 1$, if

$$Cop_{\mathcal{M}_1} \equiv Cop_{\mathcal{M}_p},$$

it follows that:

$$T(A_{k,P}) = \frac{T(A_{k,1})}{P} \cdot \frac{R^{calc}(A_{k,P})}{R^{calc}(A_{k,1})}. \tag{34}$$

Proof. If $A_{k,P}$ is perfectly parallel, then $M_{E_{k,P}}$ has no empty elements so

$$r_E = \frac{k}{c_E} = \frac{k}{P}.$$

[9] $\beta^{calc}_{M_E(A_{k,P})}$ is the sum of the maximum operator time on each row, so $\beta^{calc}_{sum,M_E(A_{k,P})}$ can be equal to $P \cdot \beta^{calc}_{M_E(A_{k,P})}$ only if the operators have all the same time.

Therefore, from the (26) and (28), it is

$$T(A_{k,P}) = r_E \cdot R^{calc}(A_{k,P}) \cdot tcalc,$$
$$= \frac{k}{c_E} \cdot R^{calc}(A_{k,P}) \cdot tcalc = \frac{T(A_{k,1})}{P} \cdot \frac{R^{calc}(A_{k,P})}{R^{calc}(A_{k,1})}. \quad (35)$$

□

Theorem 3. For all the matrices $M_{E_{k,P}}$ of algorithms in $\varrho(A_{k,P})$, it holds

$$c_E \leq c_{D_k} \quad (36)$$

and

$$r_E \geq r_{D_k}. \quad (37)$$

Moreover, let us consider $A^i_{k,P}$ and $A^j_{k,P}$ two algorithms belonging to $\varrho(A_{k,P})$, and their matrices $M^i_{E_{k,P}}$ and $M^j_{E_{k,P}}$. We have:

- $c^i_E < c^j_E \Rightarrow r^i_E \geq r^j_E$;
- $c^i_E > c^j_E \Rightarrow r^i_E \leq r^j_E$.

Proof. From inheritance on $A_{k,P}$ of dependencies defined on $D_k(\mathcal{B}_{N_r})$, it is not possible that $c_E > c_D$, therefore $c_E \leq c_{D_k}$. Then there is at least one row of M_{D_k} with c_{D_k} non-empty elements. Let d be the difference between c_{D_k} and c_E. Therefore, since M_{D_k} and M_E have the same number of non-empty elements, it is $r_E \geq r_D + \lceil (d/c_{E\cdot}) \rceil$.

Similarly, it can be proved that if $c^i_E < c^j_E$ then $r^i_E \geq r^j_E$, and if $c^i_E > c^j_E$ then $r^i_E \leq r^j_E$. □

Remark 17. The minimum execution time is proportional to the dependency degree of $\mathcal{B}_{N_r}$, that is when the number of computing units is equal to the concurrency degree of $\mathcal{B}_{N_r}$.

We now define a subset of the equivalence class of $\varrho(A_{k,P})$. Let $\simeq$ be the equivalence relation identifying two algorithms with the same P. Then

$$\hat{\varrho}(A_{k,P}) := \varrho(A_{k,P}) / \simeq \quad (38)$$

i.e. consisting of the representatives of the equivalence classes of $\simeq$[10].

Let us now consider matrices $M_{E_{k,P}}$ associated to algorithms belonging to $\hat{\varrho}(A_{k,P})$, varying P.

The following result defines the speed up of a parallel algorithm with respect to the sequential algorithm belonging to its class.

[10] For example, we can take the algorithm in $\hat{\varrho}(A_{k,P})$, $P \geq 1$, whose execution matrix has the fewest number of rows.

Theorem 4. Consider $A_{k,1} \overset{\varrho}{\equiv} A_{k,P}$ with

$$M_{E_1}, \text{ of order } N_1^E = r_{E_1} \cdot 1 \text{ and } M_{E_P} \text{ of order } N_P^E = r_{E_P} \cdot P.$$

It holds

$$Sp(A_{k,P}) = \frac{\beta^{calc}_{sum,M_{E_{k,1}}}}{r_{E_P} \cdot R^{calc}(A_{k,P})}. \tag{39}$$

Proof. From the (26), (27) and (33), it follows

$$\begin{aligned} Sp(A_{k,P}) &= \frac{r_{E_1} \cdot R^{calc}(A_{k,1}) \cdot tcalc}{r_{E_P} \cdot R^{calc}(A_{k,P}) \cdot tcalc} = \frac{C(A_{k,P})}{r_{E_P}} \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})} \\ &= \frac{\beta^{calc}_{sum,M_{E_{k,1}}}}{r_{E_P} \cdot R^{calc}(A_{k,P})}. \end{aligned} \tag{40}$$

□

Corollary 3. Since $(r_{E_P} \cdot c_{E_P}) \geq C(A_{k,P})$, from the (40) it follows that

$$Sp(A_{k,P}) \leq c_{E_P} \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})} = P \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}.$$

Definition 23 (Ideal Speed Up in $\hat{\varrho}(A_{k,P})$)**.** We let

$$Sp_{Ideal}(A_{k,P}) = P \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})} \tag{41}$$

be the ideal speed up.

Let r_{par_i} denote the number of rows having $i > 1$ not empty elements, and $r_{par_1} = r_{seq}$, then it is

$$r_{E_P} = \sum_{i=1}^{P} r_{par_i}.$$

Definition 24 (Total Time of A with i Non Empty Elements)**.** Let T_{j_i} the time of a row with $i \geq 1$ not empty elements. The quantity

$$T_{par_i}(A_{k,P}) = \sum_{j=0}^{r_{par_i}-1} T_{i_j} \tag{42}$$

is the execution time of the part of A with i non empty elements on each row.

Remark 18. It holds that $r_{par} = r_{E_P} - r_{seq} = \sum_{i=2}^{P} r_{par_i}$ then $T_{par_1}(A_{k,P}) = T_{seq}(A_{k,P})$.

Next result shows how the generalized Amdhal's Law can be derived by using the rows of the execution matrix $M_{E_{k,P}}$ having at least one non empty element.

Theorem 5 (Generalized Amdhal's Law)**.** It is

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\sum_{i=1}^{P} \alpha_i} \tag{43}$$

where

$$\alpha_i = \frac{r_{par_i}}{C(A_{k,P})}.$$

Proof. From (40) it is

$$Sp(A_{k,P}) = \frac{\frac{C(A_{k,P}) \cdot R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{r_{seq} + \sum_{i=2}^{P} r_{par_i}}. \tag{44}$$

By dividing for $C(A_{k,P})$ it follows that

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\frac{r_{seq}}{C(A_{k,P})} + \sum_{i=2}^{P} \frac{r_{par_i}}{C(A_{k,P})}}, \tag{45}$$

that is

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\alpha_1 + \sum_{i=2}^{P} \alpha_i}. \tag{46}$$

□

Then, the Amdhal's Law [1] comes out as a particular case of the previous theorem.

Corollary 4 (Amdhal's Law)**.** If we assume that $M_{E_{k,1}}$ only has rows with 1 element or P elements, we have

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\alpha + \frac{1-\alpha}{P}} \tag{47}$$

where

$$\alpha := \frac{r_{seq}}{C(A_{k,P})}.$$

Proof. From (43) it follows that

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\alpha_1 + \sum_{i=2}^{P} \alpha_i} \tag{48}$$

where

$$\alpha_i := \frac{r_{par_i}}{C(A_{k,P})}$$

and

$$\frac{r_{par}}{C(A_{k,P})} = \sum_{i=2}^{P} \alpha_i.$$

If the rows with more than one non empty element have P elements, it is

$$r_{par} = \frac{C(A_{k,P}) - r_{seq}}{P},$$

therefore, if we let $\alpha_1 = \alpha = \frac{r_{seq}}{C(A_{k,P})}$, we get

$$Sp(A_{k,P}) = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\frac{r_{seq}}{C(A)} + \frac{r_{par}}{C(A_{k,P})}} = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\frac{r_{seq}}{C(A_{k,P})} + \frac{C(A_{k,P}) - r_{seq}}{C(A_{k,P}) \cdot P}} = \frac{\frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}}{\alpha + \frac{1-\alpha}{P}}. \tag{49}$$

□

Let Q denote the cost of $A_{k,P}$. The cost is defined as the product of the execution time and the number of processors utilized [19]. In this mathematical settings it holds that the cost Q can be written as

$$Q(A_{k,P}) = c_E \cdot r_E \cdot R^{calc}(A_{k,P}) \cdot tcalc. \tag{50}$$

If $c_E = 1$, from the (28) it holds

$$\begin{aligned} Q(A_{k,1}) &= r_E \cdot R^{calc}(A_{k,P}) \cdot tcalc = T(A_{k,1}) = C(A_{k,P}) \cdot R^{calc}(A_{k,1}) \cdot tcalc \\ &= \beta^{calc}_{sum,M_{E_{k,1}}} \cdot tcalc. \end{aligned} \tag{51}$$

The overhead of $A_{k,P}$ is the total time spent by all the processing elements over and above that spent in useful computation.

Definition 25 (Algorithm Overhead)**.** The quantity

$$Oh(A_{k,P}) := Q(A_{k,P}) - Q(A_{k,1}) = \left(c_E \cdot \beta^{calc}_{M_{E_{k,P}}} - \beta^{calc}_{sum,M_{E_{k,1}}} \right) \cdot tcalc \tag{52}$$

is said overhead of $A_{k,P}$.

Theorem 6. It holds

$$C(A_{k,P}) \cdot (R^{calc}(A_{k,P}) - R^{calc}(A_{k,1})) \cdot tcalc \begin{cases} = 0, & \text{if } R^{calc}(A_{k,P}) = R^{calc}(A_{k,1}), \\ > 0, & \text{otherwise.} \end{cases} \tag{53}$$

Proof. It holds

$$\begin{aligned} Q(A_{k,P}) &\geq card(A_{k,P}) \cdot R^{calc}(A_{k,P}) \cdot tcalc \\ &= C(A_{k,P}) \cdot R^{calc}(A_{k,P}) \cdot tcalc = k \cdot R^{calc}(A_{k,P}) \cdot tcalc. \end{aligned} \tag{54}$$

Moreover,

$$Q(A_{k,1}) = C(A_{k,1}) \cdot R^{calc}(A_{k,1}) \cdot tcalc = k \cdot R^{calc}(A_{k,1}) \cdot tcalc,$$

therefore it follows from (52)

$$Oh(A_{k,P}) \geq \big(k \cdot (R^{calc}(A_{k,P}) - R^{calc}(A_{k,1}))\big) \cdot tcalc$$

and (53) follows. □

Definition 26 (Ideal Overhead in $\hat{\varrho}(A_{k,P})$)**.** From the (53) it follows

$$Oh_{Ideal}(A_{k,P}) = \big(k \cdot (R^{calc}(A_{k,P}) - R^{calc}(A_{k,1})\big) \cdot tcalc. \tag{55}$$

Let $Ef(A_{k,P}) := \frac{Sp(A_{k,P})}{P}$ be the efficiency of A where $P \geq 1$.

Theorem 7. Let $N_P^E = c_{E_P} \cdot r_{E_P}$ denote the dimension of the execution matrix of $A_{k,P}$, it holds that

$$Ef(A_{k,P}) = \frac{\beta^{calc}_{sum,M_{E_{k,1}}}}{N_P^E \cdot R^{calc}(A_{k,P})}. \tag{56}$$

Proof. Since $c_E = P$, it follows that

$$Ef(A_{k,P}) = \frac{Sp(A_{k,P})}{P} = \frac{\beta^{calc}_{sum,M_{E_{k,1}}}}{c_{E_P} \cdot r_{E_P} \cdot R^{calc}(A_{k,P})}. \tag{57}$$

□

Definition 27 (Ideal Efficiency in $\hat{\varrho}(A_{k,P})$)**.** Since $Sp_{(k,P)} \leq P \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}$, it always is $Ef(A_{k,P}) \leq \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}$. So let

$$Ef_{Ideal}(A_{k,P}) = \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})} \tag{58}$$

be the ideal efficiency of $A_{k,P}$.

Remark 19. It is worth to note the role of parameters $R^{calc}(A_{k,P})$ and $R^{calc}(A_{k,1})$ in (47), (55) and (56). If in $A_{k,P}$ there are few operators which are much more time consuming than the others, and $k >> r_E$ then $\beta^{calc}_{M_{E_{k,P}}} \simeq \beta^{calc}_{sum,M_{E_{k,1}}}$ and $R^{calc}(A_{k,P}) >> R^{calc}(A_{k,1})$. The more the operators are and the greater the difference is in (55), or the lower the ratio is in (47) and (56). Hence, the greater the

overhead is, the lower the speed up and the efficiency are. This is a consequence of a problem decomposition, associated to $A_{k,P}$ not well balanced.

Let us now suppose that the algorithm $A_{k,P}$ is perfectly parallel, that is its execution matrix M_{E_P} has not any empty element. Since $r_{E_P} \cdot c_{E_P} = C(A_{k,P})$ it follows from Corollary 3 that

$$Sp(A_{k,P}) = Sp_{Ideal}(A_{k,P}) = P \cdot \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})},$$

from (53) that

$$Oh(A_{k,P}) = Oh_{Ideal}(A_{k,P}) = \big(C(A_{k,P}) \cdot (R^{calc}(A_{k,P}) - R^{calc}(A_{k,1}))\big) \cdot tcalc,$$

from (58)

$$Ef(A_{k,P}) = Ef_{Ideal}(A_{k,P}) = \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}.$$

Remark 20. If $P = c_D$, $r_E = r_D$ and $c_E = c_D$, if $P = c_D$ then the following results hold on:

1. $Q(A_{k,P}) = c_D \cdot r_D \cdot R^{calc}(A_{k,P}) \cdot tcalc = N_D \cdot R^{calc}(A_{k,P}) \cdot tcalc$;
2. $Sp(A_{k,P}) = \frac{C(A_{k,P})}{r_D} \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}$;
3. $Oh(A_{k,P}) = (c_D \cdot r_D - C(A_{k,P})) \cdot R^{calc}(A_{k,P}) \cdot tcalc$;
4. $Ef(A_{k,P}) = \frac{C(A_{k,P})}{r_D \cdot c_D} \frac{R^{calc}(A_{k,1})}{R^{calc}(A_{k,P})}$.

Example 2. The well known reduction problem is interesting to expose the nature of Algorithm Overhead Oh and its impact in some classical metrics.

Let $\mathcal{B}_{27}$ denote the computational problem of the sum of 27 real numbers and $D_{13}(\mathcal{B}_{27}) = \{\mathcal{B}_3^i\}_{0 \le i < 13} \in \mathcal{DB}_{27}$ one of its decompositions, where $\mathcal{B}_3^i$ represents the sum of 3 real numbers.

The decomposition matrix is

$$M_D(D_{13}(\mathcal{B}_{27}) = \begin{bmatrix} \mathcal{B}_3^0 & \mathcal{B}_3^1 & \mathcal{B}_3^2 & \mathcal{B}_3^3 & \mathcal{B}_3^4 & \mathcal{B}_3^5 & \mathcal{B}_3^6 & \mathcal{B}_3^7 & \mathcal{B}_3^8 \\ \mathcal{B}_3^9 & \mathcal{B}_3^{10} & \mathcal{B}_3^{11} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \\ \mathcal{B}_3^{12} & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing & \varnothing \end{bmatrix}. \tag{59}$$

Therefore, the concurrency degree is $c_{D_{13}} = 9$, the dependency degree is $r_{D_{13}} = 3$, and the problem is not perfectly decomposed. Let us suppose $\mathcal{B}_{27}$ is solvable on $\mathcal{M}_P$ with $P = 3$. Let $Cop_{\mathcal{M}_3} = \{++, \ldots\}$, be the computing operators of $\mathcal{M}_3$, and let $A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_3} = \{++_0, \ldots, ++_{12}\}$ be the algorithm that we choose to solve $\mathcal{B}_{27}$, given

$D_{13}(\mathcal{B}_{27})$. Then the execution matrix is

$$M_{E_{13,3}} = \begin{bmatrix} ++_0 & ++_1 & ++_2 \\ ++_3 & ++_4 & ++_5 \\ ++_6 & ++_7 & ++_8 \\ ++_9 & ++_{10} & ++_{11} \\ ++_{12} & \varnothing & \varnothing \end{bmatrix} \tag{60}$$

and the corresponding algorithm is simply parallel. Moreover, $T(A_{13,3}) = 5 \cdot tcalc$, where $tcalc$ is the execution time for the sum $++$ of three real numbers.

If we take another $\mathcal{M}_P$ with $P = 4$ where $\mathcal{B}_{27}$ is solvable, let us suppose that $Cop_{\mathcal{M}_4} = Cop_{\mathcal{M}_3}$ and $A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_4} = A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_3}$. Then, the execution matrix can be written as

$$M_{E_{13,4}} = \begin{bmatrix} ++_0 & ++_1 & ++_2 & ++_3 \\ ++_4 & ++_5 & ++_6 & ++_7 \\ ++_8 & \varnothing & \varnothing & \varnothing \\ ++_9 & ++_{10} & ++_{11} & \varnothing \\ ++_{12} & \varnothing & \varnothing & \varnothing \end{bmatrix}. \tag{61}$$

The corresponding algorithm is still simply parallel, and $T(A_{13,4}) = 5 \cdot tcalc$.

Considering the classical metrics of speed-up and efficiency, evaluated together with cost and algorithm overhead, we have:

- for $A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_3}$

$$\begin{aligned} Sp(A_{13,3}) &= \frac{r_{E_{A_{13,1}}}}{r_{E_{A_{13,3}}}} = 2.6, \\ Q(A_{13,3}) &= c_{E_{A_{13,3}}} \cdot r_{E_{A_{13,3}}} \cdot tcalc = 15 \cdot tcalc, \\ Oh(A_{13,3}) &= Q(A_{13,3}) - Q(A_{13,1}) = 2 \cdot tcalc, \\ Ef(A_{13,3}) &= \frac{r_{E_{A_{13,1}}}}{r_{E_{A_{13,3}}} \cdot c_{E_{A_{13,3}}}} = 0.87, \end{aligned} \tag{62}$$

- for $A_{D_{13}(\mathcal{B}_{27}),\mathcal{M}_4}$

$$\begin{aligned} Sp(A_{13,4}) &= \frac{r_{E_{A_{13,1}}}}{r_{E_{A_{13,4}}}} = 2.6, \\ Q(A_{13,4}) &= c_{E_{A_{13,4}}} \cdot r_{E_{A_{13,4}}} \cdot tcalc = 20 \cdot tcalc, \\ Oh(A_{13,4}) &= Q(A_{13,3}) - Q(A_{13,1}) = 7 \cdot tcalc, \\ Ef(A_{13,4}) &= \frac{r_{E_{A_{13,1}}}}{r_{E_{A_{13,4}}} \cdot c_{E_{A_{13,4}}}} = 0.65. \end{aligned} \tag{63}$$

Observe that while the speed up is the same, the overhead reveals that the mapping on M_4 is not the optimal one, showing the performance bottleneck of the algorithm.

6 ALGORITHMS WHOSE OPERATORS HAVE THE SAME EXECUTION TIME

We assume that all the operators of the algorithm have the same execution time. For example they are the elementary floating point operations. The execution time is $\beta^{calc} \cdot tcalc$, and without loss of generality we assume that $\beta^{calc} = 1$.

Hence, it follows that, $\forall P, \quad \beta^{calc}_{r,M_{E_{k,P}}} = 1, \beta^{calc}_{M_{E_{k,P}}} = r_E, \beta^{calc}_{sum,M_{E_{k,P}}} = k$. Finally, from (25) it follows that $\forall P, R^{calc}(A_{k,P}) = 1$. Hence, we get

- $Sp(A_{k,P}, A_{k',1}) := \frac{k'}{k} \cdot \frac{k}{r_E}$,
- if $Q = 1$, then $Sp(A_{k,P}) := \frac{k}{r_E}$,
- $Sp_{Ideal}(A_{k,P}, A_{k',1}) = Sc_{up}(A_{k,P}, A_{k',1}) \cdot P = \frac{k'}{k} \cdot P$,
- $Sp_{Ideal}(A_{k,P}) = c_{E_P} = P$.

Finally, if $\mathcal{B}_{N_r}$ is perfectly decomposed, then

$$T(A_{k,P}) = \frac{T(A_{k,1})}{P}, \tag{64}$$

i.e., $A_{k,P}$ has the ideal speed up in the classical definition.

Let us now consider matrices $M_{E_{k,P}}$ associated with algorithms in $\hat{\varrho}(A_{k,P})$, varying P. The following results hold: $Q(A_{k,P}) = c_E \cdot r_E \cdot tcalc$ and, if $c_E = 1$, then $Q(A_{k,1}) = k \cdot tcalc$; $Oh_{Ideal}(A_{k,P}) = 0$; $Ef_{Ideal}(A_{k,P}) = 1$.

Theorem 8. Let us suppose that

$$\forall I^{i_j} \in Cop_{\mathcal{M}_P}, \quad t_{i_j} = \beta^{calc}_{i_j,M_{E_{k,1}}} \cdot tcalc = tcalc, \quad \forall i, j. \tag{65}$$

Given $A_{k,P}$, $P > 1$, $M_{E_{k,P}}$ of order $N_P^E = r_E \cdot P$, let V_r be the number of empty elements of the row r of $M_{E_{k,P}}$; it is

$$Oh(A_{k,P}) = \sum_{r=0}^{r_E-1} V_r \cdot tcalc. \tag{66}$$

Proof. It holds that $c_E \cdot r_E = card(A_{k,P}) + \sum_{r=0}^{r_E-1} V_r = C(A_{k,P}) + \sum_{r=0}^{r_E-1} V_r = k + \sum_{r=0}^{r_E-1} V_r$ then from (52)

$$Oh(A_{k,P}) = \left(k + \sum_{r=0}^{r_E-1} V_r - k \right) \cdot tcalc = \sum_{r=0}^{r_E-1} V_r \cdot tcalc. \tag{67}$$

□

Remark 21. Note that $\sum_{r=0}^{r_E-1} V_r$ is the sparsity degree of the execution matrix.

Among the decomposition approaches, recursive decomposition very often is the most suitable approach for employing a performance analysis, especially in the presence of complex algorithms solving real-world applications/simulations. In this case, as described in the toy example below, the problem is solved by firstly decomposing it into a set of independent sub-problems. Furthermore, each one of these subproblems is solved by applying a similar decomposition into smaller subproblems followed by a combination of their results, and so on. In this way we get a decomposition matrix whose elements are problems which could be subsequently decomposed and analyzed until the desired level of detail is reached.

Example 3. Let $\mathcal{B}_{16}$ denote the computational problem of the sum of 16 real numbers and $D_3(\mathcal{B}_{16}) = \{B_8, B_8, B_2\} \in \mathcal{DB}_{16}$. The decomposition matrix is

$$M_{D_3}(\mathcal{B}_{16}) = \begin{bmatrix} \mathcal{B}_8 & \mathcal{B}_8 \\ \mathcal{B}_2 & \varnothing \end{bmatrix}. \tag{68}$$

If $\mathcal{B}_8$ can be decomposed as $D_3^1(\mathcal{B}_8) = \{\mathcal{B}_4, \mathcal{B}_4, \mathcal{B}_2\} \in \mathcal{DB}_8$ then

$$M_{D_3^1}(\mathcal{B}_8) = \begin{bmatrix} \mathcal{B}_4 & \mathcal{B}_4 \\ \mathcal{B}_2 & \varnothing \end{bmatrix}. \tag{69}$$

In the same way, if $\mathcal{B}_4$ can be decomposed as $D_3^2(\mathcal{B}_4) = \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \in \mathcal{DB}_8$ and

$$M_{D_3^2}(\mathcal{B}_4) = \begin{bmatrix} \mathcal{B}_2 & \mathcal{B}_2 \\ \mathcal{B}_2 & \varnothing \end{bmatrix}. \tag{70}$$

We have three decompositions for $\mathcal{B}_{16}$:

$$\begin{aligned}
D_3 \in \mathcal{D}(\mathcal{B}_{16}) &= \{\mathcal{B}_8, \mathcal{B}_8, \mathcal{B}_2\}, \\
D_7 \in \mathcal{D}(\mathcal{B}_{16}) &\equiv D_3^1(\mathcal{B}_8) \cup D_3^1(\mathcal{B}_8) \cup \{\mathcal{B}_2\} \\
&\equiv \{\mathcal{B}_4, \mathcal{B}_4, \mathcal{B}_2\} \cup \{\mathcal{B}_4, \mathcal{B}_4, \mathcal{B}_2\} \cup \{\mathcal{B}_2\}, \\
D_{15} \in \mathcal{D}(\mathcal{B}_{16}) &\equiv D_3^2(\mathcal{B}_4) \cup D_3^2(\mathcal{B}_4) \cup \{\mathcal{B}_2\} \cup D_3^2(\mathcal{B}_4) \cup D_3^2(\mathcal{B}_4) \cup \{\mathcal{B}_2\} \cup \{\mathcal{B}_2\} \\
&\equiv \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \cup \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \cup \{\mathcal{B}_2\} \cup \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \\
&\quad \cup \{\mathcal{B}_2, \mathcal{B}_2, \mathcal{B}_2\} \cup \{\mathcal{B}_2\} \cup \{\mathcal{B}_2\} \\
&\equiv \{\mathcal{B}_2^i\}_{0 \le i < 15} \in \mathcal{DB}_{16} \qquad (71)
\end{aligned}$$

with the following characteristics, according to the corresponding decomposition matrices:

- D_3: cardinality 3, concurrency degree 2 and dependence degree 2,
- D_7: cardinality 7, concurrency degree 4 and dependence degree 3,
- D_{15}: cardinality 15, concurrency degree 8 and dependence degree 4,

meaning that the intrinsic concurrency of a problem heavily depends on the decomposition chosen for that problem. Each decomposition has a level of detail depending on the type of subproblems that are considered.

7 SOFTWARE

From now on, we consider memory accesses performed by an algorithm and we assume, for simplicity, that to each access corresponds one read/write of a single data. Moreover, we assume that computations and memory accesses are not performed simultaneously, instead they depend on each other.

Definition 28. Given the set of elementary operators of $\mathcal{M}_P$, we introduce memory access operators corresponding to the memory access (read/write) of processing elements of $\mathcal{M}_P$. The set $OA_{\mathcal{M}_P} = \{r(\cdot), w(\cdot)\}$ where $r(a)$, which reads a, and $w(a)$, which writes a, contains memory access operators of $\mathcal{M}_P$.

Note that $\mathcal{M}_P$ is now the union of the set of elementary operators and the set of memory accesses operators, $\mathcal{M}_P = Op_{\mathcal{M}_P} \bigcup OA_{\mathcal{M}_P}$.

Definition 29. We introduce the ordered set (whose elements should not be different) of accesses operators of $\mathcal{M}_P$ $AC = \{oa_0(\cdot), oa_1(\cdot), \ldots, oa_k(\cdot)\}$ where $oa_i(\cdot) \in OA_{\mathcal{M}_P}$. Moreover, we consider the surjective correspondence

$$\bar{\gamma} : oa_i(\cdot) \in OA_{\mathcal{M}_P} \longleftrightarrow op_i \in A_{k,P}. \qquad (72)$$

Note that $card(AC) \geq card(A_{k,P})$.

The set $AC_{\mathcal{M}_{P,L,nd_L}}(l) := \{oa^l_{i_0}(\cdot), oa^l_{i_1}(\cdot), \ldots, oa^l_{i_k}(\cdot)\} \subset Xop_{\mathcal{M}_P}$ denotes an ordered set of accesses operators of $\mathcal{M}_{P,L,nd_L}$ at level l. Let $AC_{\mathcal{M}_{P,L,nd_L}} := \bigcup_{l=1}^{L} AC_{\mathcal{M}_{P,L,nd_L}}(l)$ denote the set of the memory accesses of $\mathcal{M}_{P,L,nd_L}$. For simplicity of notations, if there is no ambiguity, the set $AC_{\mathcal{M}_{P,L,nd_L}}$ of memory accesses of $\mathcal{M}_{P,L,nd_L}$ is briefly denoted as $AC(L)$.

Definition 30 (Software)**.** The set $SW(A_{k,P}) := A_{k,P} \cup AC(L)$ where the order relation on $AC(L)$ is induced by the ordering on $A_{k,P}$ is said the Software corresponding to algorithm $A_{k,P}$. More simply, in the sequel we denote the Software as $SW(A_{k,P})$.

Definition 31. Given M_E and AC, matrix $AM_{A_{D_k},\mathcal{M}_{P,L,nd_L}}(l)$, defined in AC of order $r_{AM_l} \times c_{AM_l}$, with $c_{AM} = nd_l$[11] is said the l^{th} access matrix of SW.

Let $r_{AM} := \sum_{l=1}^{L} r_{AM_l}$ and let $r_{COM} := r_{AM_L}$ $(r_{COM} \leq r_{AM})$ be the parameter counting the rows of the L^{th} matrix $AM(L)$ related to L^{th} level. If $P = 1$ then $r_{COM} = 0$.

Definition 32 (Memory Access Time)**.** The quantity

$$T_M(SW(A_{k,P})) := \sum_{l=1}^{L-1} (r_{AM_l} \cdot tmem_l) \tag{73}$$

is said memory access time of $SW(A)$.

Computational intensity is defined as the number of operations per memory accesses [23]. More precisely, it measures how intensely A computes with data, once it has been received.

Definition 33 (Computational Intensity)**.** The quantity

$$C_I(SW(A_{k,P})) := \frac{r_E}{r_{AM}} \in [1, \infty[\tag{74}$$

is said software computational intensity.

Remark 22. If instead of r_{AM} we only consider the number of rows of the L^{th} memory access matrix, which is related to the software communications, i.e., we only consider r_{COM} and take the reciprocal of $C_I(SW(A_{k,P}))$, we get the so called software communication intensity. It measures how much communications dominate with respect to the operations. This quantity is usually called surface-to-volume ratio [14].

[11] In general $c_{AM_l} \leq nd_l$, but with no loss of generality we assume that $c_{AM_l} = nd_l$.

Definition 34 (Communication Intensity)**.** The quantity

$$Com_I(SW(A_{k,P})) := \frac{r_{COM}}{r_E} \tag{75}$$

is said software communication intensity.

Definition 35 (Communication Time)**.** The quantity

$$T_{COM}(SW(A_{k,P})) := r_{COM} \cdot tcom \tag{76}$$

is said the software communication time.

We now assume that $\mathcal{M}_{P,nd_L}$ is such that $P \geq 1$ and $L \geq 3$, that is, it includes the level L of the communications among processing elements. Moreover, since overlapping communication with computation comes at the expense of increased memory requirements, we assume that memory accesses (including communications) and computations cannot be executed simultaneously, but they are dependent on each other.

Definition 36 (Execution Time)**.** The quantity

$$T(SW(A_{k,P})) := T(A_{k,P}) + T_M(SW(A_{k,P})) + T_{COM}(SW(A_{k,P})) \tag{77}$$

is said software execution time of $SW(A_{k,P})$.

Definition 37 (Machine Communication Overhead)**.** The ratio

$$UCom_{oh}(\mathcal{M}_P) := \frac{tcom}{tcalc} \tag{78}$$

is said unitary machine communication overhead.

Observe that at present time it is $UCom_{oh}(\mathcal{M}_{P,L,nd_L}) \gg 1$. Machine communication overhead, also known as *machine balance*, is one of the parameters depending on the machine [17].

Definition 38 (Software Communication Overhead)**.** The quantity

$$Com_{oh}(SW(A_{k,P})) := \frac{T_{COM}(SW(A_{k,P}))}{T(A_{k,P})}, \tag{79}$$

which is the software communication overhead, is expressed as follows

$$\begin{aligned} Com_{oh}(SW(A_{k,P})) &:= \frac{r_{COM}}{r_E} \cdot \frac{tcom}{tcalc} \\ &\equiv Com_I(SW(A_{k,P})) \cdot UCom_{oh}(\mathcal{M}_P). \end{aligned} \tag{80}$$

Definition 39 (Memory Traffic)**.** The quantity

$$M_T(A_{k,P}) := \frac{T_M(SW(A_{k,P}))}{T(A_{k,P})}, \tag{81}$$

which is the memory traffic, is expressed as follows

$$M_T(A_{k,P}) := \frac{\sum_{l=1}^{L-1}(r_{AM_l} \cdot tmem_l)}{\sum_{r=0}^{r_E-1} T_r(A_{k,P})}. \tag{82}$$

Remark 23. If memory traffic grows, then the computational intensity $C_I(SW(A_{k,P}))$ decreases (see Definition 33).

Definition 40 (l^{th} Software Memory Traffic)**.** The ratio

$$M_T(A_{k,P})_l := \frac{r_{AM_l} \cdot tmem_l}{\sum_{r=0}^{r_E-1} T_r(A_{k,P})} \tag{83}$$

is said level the l^{th} memory traffic of $SW(A)$.

Definition 41 (Software Speed Up)**.** Given $SW(A_{k,P_1})$ and $SW(A_{k,P_2})$, where $P_2 > P_1$, the ratio

$$Sp(SW(A_{k,P_2})) := \frac{T(SW(A_{k,P_1}))}{T(SW(A_{k,P_2}))} \tag{84}$$

is said software speed up of $SW(A_{k,P_2})$.

Proposition 2.

$$\begin{aligned} Sp(SW_{A_{k,P},\mathcal{M}_{P,nd_{L1}}}) &= \frac{T(SW_{A_{k,1},\mathcal{M}_{1,nd_{L^1}}})}{T(SW_{A_{k,P},\mathcal{M}_{P,nd_{L^P}}})} \\ &= \frac{r_{E^1} \cdot tcalc + (r_{AM^1} - r_{COM^1}) \cdot tmem + r_{COM^1} \cdot tcom}{r_{E^P} \cdot tcalc + (r_{AM^P} - r_{COM^P}) \cdot tmem + r_{COM^P} \cdot tcom}. \end{aligned}$$

In the same way as we have previously done for algorithm A, the Software efficiency $Ep(SW)$ and all the other performance metrics can be defined in terms of the Software execution time $T(SW)$.

8 CONCLUSIONS

This paper targets an important topic of current importance in High Performance Computing community, which is the performance analysis of parallel algorithms; it should be re-evaluated to find out the best-practice algorithm on novel architectures [4, 15, 16, 18, 20, 21, 26, 34]. In this paper we presented a mathematical framework which can be used to get a multilevel description of a parallel algorithm,

and we proved that it can be suitable for analysing the mapping of an algorithm on a given machine. The model is multilevel, in the sense that it allows the choice of a level of abstraction of both the problem decomposition and of the operations in the algorithm, which determines the level of granularity of the performance analysis. This feature can be very useful in practice to analyze performance of complex algorithms solving real problems and to indicate performance bottlenecks within the algorithm. Furthermore, the model allows to take into account the initial decomposition of the problem into subproblems, and so their mutual dependencies. In order to show how to use the performance model, we validated this approach in practice using real problems on real architectures. In [27], a preliminary performance analysis, carried out considering the speed up of the algorithm before its mapping on the computing architectures, provided the best number of processing elements and grid levels to address the scaling of a multigrid in time algorithm. According to the time-stepping procedure, the performance analysis was carried out choosing matrix-vector products or linear system solutions as the elements of the dependency matrix, and therefore as computing operators of the algorithm. In [3, 14], the performance analysis of the algorithm, carried out in terms of scale up, suggested the introduction of a highly scalable decomposition of a variational data assimilation problem. This approach completely redesigned the mapping of the numerical algorithm on high performance computing architectures.

We remark that energy consumption on computer systems has emerged as an important concern, and the energy consumed in executing an algorithm cannot be inferred from its performance alone. We will employ the proposed framework to also model energy consumption of parallel algorithms. As example, according to [13], we performed the energy analysis of a variational data assimilation algorithm running on an ARM-based HPC systems assuming that total energy depends on the energy consumed in all steps executed by the parallel algorithm; the energy consumed at each step is measured in terms of parameters depending on both the algorithm and the computing architecture [2]. This approach can be used to extend the performance metrics to address the analysis of software energy consumption.

We conclude that we have assumed abstract models for both algorithms and architectures, and we have made numerous simplifying assumptions. Indeed, we believe that a simplified parameterized model gives a useful generalization for a better understanding of algorithms that can run really fast, no matter how complicated the underlying computer architecture [17].

REFERENCES

[1] Amdahl, G. M.: Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. AFIPS Conference Proceedings (AFIPS '67), Vol. 30, 1967, pp. 483–485, doi: 10.1145/1465482.1465560.

[2] Arcucci, R.—Basciano, D.—Cilardo, A.—D'Amore, L.—Mantovani, F.: Energy Analysis of a 4D Variational Data Assimilation Algorithm and Evaluation

on ARM-Based HPC Systems. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (Eds.): Parallel Processing and Applied Mathematics (PPAM 2017). Springer, Cham, Lecture Notes in Computer Science, Vol. 10778, 2018, pp. 37–47, doi: 10.1007/978-3-319-78054-2_4.

[3] Arcucci, R.—D'Amore, L.—Celestino, S.—Laccetti, G.—Murli, A.: A Scalable Numerical Algorithm for Solving Tikhonov Regularization Problems. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (Eds.): Parallel Processing and Applied Mathematics. Springer, Cham, Lecture Notes in Computer Science, Vol. 9574, 2016, pp. 45–54, doi: 10.1007/978-3-319-32152-3_5.

[4] Ballard, G.—Demmel, J.—Holtz, O.—Schwartz, O.: Minimizing Communication in Numerical Linear Algebra. SIAM Journal on Matrix Analysis and Applications, Vol. 32, 2011, No. 3, pp. 866–901, doi: 10.1137/090769156.

[5] Berman, F.—Snyder, L.: Mapping Parallel Algorithms into Parallel Architectures. Journal of Parallel and Distributed Computing, Vol. 4, 1987, No. 5, pp. 439–458, doi: 10.1016/0743-7315(87)90018-9.

[6] Berman, F.: The Mapping Problem in Parallel Computation. In: Rice, J. R. (Ed.): Mathematical Aspects of Scientific Software. Springer, New York, NY, The IMA Volumes in Mathematics and Its Applications, Vol. 14, 1988, pp. 41–57, doi: 10.1007/978-1-4684-7074-1_2.

[7] Bernstein, A. J.: Analysis of Programs for Parallel Processing. IEEE Transactions on Electronic Computers, Vol. EC-15, 1966, No. 5, pp. 757–763, doi: 10.1109/pgec.1966.264565.

[8] Bokhari, S. H.: On the Mapping Problem. IEEE Transactions on Computers, Vol. C-30, 1981, No. 3, pp. 207–214, doi: 10.1109/tc.1981.1675756.

[9] Browne, S.—Dongarra, J.—Garner, N.—Ho, G.—Mucci, P.: A Portable Programming Interface for Performance Evaluation on Modern Processors. The International Journal of High Performance Computing Applications, Vol. 14, 2000, No. 3, pp. 189–204, doi: 10.1177/109434200001400303.

[10] Bosilca, G.—Bouteiller, A.—Danalis, A.—Herault, T.—Lemarinier, P.—Dongarra, J.: DAGuE: A Generic Distributed DAG Engine for High Performance Computing. Parallel Computing, Vol. 38, 2012, No. 1–2, pp. 37–51, doi: 10.1016/j.parco.2011.10.003.

[11] Bosilca, G.—Bouteiller, A.—Danalis, A.—Faverge, M.—Herault, T.—Dongarra, J.: PaRSEC: Exploiting Heterogeneity to Enhance Scalability. Computing in Science and Engineering, Vol. 15, 2013, No. 6, pp. 36–45, doi: 10.1109/mcse.2013.98.

[12] Coffman, E. G. Jr.—Denning, P. J.: Operating Systems Theory. Prentice Hall, 1973.

[13] Korthikanti, V. A.—Agha, G.: Energy-Performance Trade-Off Analysis of Parallel Algorithms for Shared Memory Architectures. Sustainable Computing: Informatics and Systems, Vol. 1, 2011, No. 3, pp. 167–176, doi: 10.1016/j.suscom.2011.05.004.

[14] D'Amore, L.—Arcucci, R.—Carracciuolo, L.—Murli, A.: A Scalable Approach for Variational Data Assimilation. Journal of Scientific Computing, Vol. 61, 2014, No. 2, pp. 239–257, doi: 10.1007/s10915-014-9824-2.

[15] D'Amore, L.—Laccetti, G.—Romano, D.—Scotti, G.—Murli, A.: Towards a Parallel Component in a GPU-CUDA Environment: A Case Study with the L-BFGS Harwell Routine. International Journal of Computer Mathematics, Vol. 92, 2014, No. 1, pp. 59–76, doi: 10.1080/00207160.2014.899589.

[16] D'Amore, L.—Mele, V.—Laccetti, G.—Murli, A.: Mathematical Approach to the Performance Evaluation of Matrix Multiply Algorithm. In: Wyrzykowski, R., Deelman, E., Dongarra, J., Karczewski, K., Kitowski, J., Wiatr, K. (Eds.): Parallel Processing and Applied Mathematics. Springer, Cham, Lecture Notes in Computer Science, Vol. 9574, 2016, pp. 25–34, doi: 10.1007/978-3-319-32152-3_3.

[17] Demmel, J.: Applications of Parallel Computers. U.C. Berkeley CS267, `http://www.cs.berkeley.edu/~demmel/cs267_Spr12`, 2012.

[18] Demmel, J.—Eliahu, D.—Fox, A.—Kamil, S.—Lipshitz, B.—Schwartz, O.—Spillinger, O.: Communication-Optimal Parallel Recursive Rectangular Matrix Multiplication. Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing (IPDPS '13), 2013, pp. 261–272, doi: 10.1109/ipdps.2013.80.

[19] Flatt, H. P.—Kennedy, K.: Performance of Parallel Processors. Parallel Computing, Vol. 12, 1989, No. 1, pp. 1–20, doi: 10.1016/0167-8191(89)90003-3.

[20] Gunnels, J. A.—Henry, G. M.—van de Geijn, R. A.: A Family of High-Performance Matrix Multiplication Algorithms. In: Alexandrov, V. N., Dongarra, J. J., Juliano, B. A., Renner, R. S., Tan, C. J. K. (Eds.): Computational Science – ICCS 2001, Part I. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2073, 2001, pp. 51–60, doi: 10.1007/3-540-45545-0_15.

[21] Gunnels, J. A.—Gustavson, F. G.—Henry, G. M.—van de Geijn, R. A.: FLAME: Formal Linear Algebra Methods Environment. ACM Transactions on Mathematical Software, Vol. 27, 2001, No. 4, pp. 422–455, doi: 10.1145/504210.504213.

[22] Gupta, A.—Kumar, V.: Performance Properties of Large Scale Parallel Systems. Journal of Parallel and Distributed Computing, Vol. 19, 1993, No. 3, pp. 234–244, doi: 10.1006/jpdc.1993.1107.

[23] Hockney, R. W.: The Science of Computer Benchmarking. SIAM, Software, Environments and Tools Series, 1996, doi: 10.1137/1.9780898719666.

[24] Kronsjö, L. I.: Algorithms, Their Complexity and Efficiency. John Wiley and Sons, New York, NY, USA, 1979.

[25] Kuck, D. J.—Kuhn, R. H.—Padua, D. A.—Leasure, B.—Wolfe, M.: Dependence Graphs and Compiler Optimization. Proceeding of the 8th ACM SIGPLAN – SIGACT Symposium on Principles on Programming Languages (POPL '81), 1981, pp. 207–218, doi: 10.1145/567532.567555.

[26] Miller, B.—Vahid, F.—Givargis, T.—Brisk, P.: Graph-Based Approaches to Placement of Processing Element Networks on FPGAs for Physical Model Simulation. ACM Transaction on Reconfigurable Technology and Systems, Vol. 7, 2015, No. 4, Art. No. 37, doi: 10.1145/2629521.

[27] Mele, V.—Constantinescu, E. M.—Carracciuolo, L.—D'Amore, L.: A PETSc Parallel-in-Time Solver Based on MGRIT Algorithm. Concur-

rency and Computation: Practice and Experience, Vol. 30, 2018, No. 24, doi: 10.1002/cpe.4928.

[28] Moldovan, D. I.: On the Analysis and Synthesis of VLSI Algorithms. IEEE Transactions on Computers, Vol. C-31, 1982, No. 11, pp. 1121–1126, doi: 10.1109/tc.1982.1675929.

[29] Rico-Gallego, J. A.—Díaz-Martín, J. C.—Manumachu, R. R.—Lastovetsky, A. L.: A Survey of Communications Performance Models for High-Performance Computing. ACM Computing Surveys, Vol. 51, 2019, No. 6, Art. No. 126, doi: 10.1145/3284358.

[30] Sharp, J. A. (Ed.): Data Flow Computing: Theory and Practice. Ablex Publishing Corporation, Norwood, New Jersey, 1992.

[31] Lee, S.-Y.—Aggarwal, J. K.: A Mapping Strategy for Parallel Processing. IEEE Transactions on Computers, Vol. C-36, 1987, No. 4, pp. 433–442, doi: 10.1109/tc.1987.1676925.

[32] Tekinerdogan, B.—Arkin, E.: Architectural Framework for Mapping Parallel Algorithms to Parallel Computing Platforms. 2nd International Workshop on Model Driven Engineering for High Performance and Cloud Computing (MDHPCL 2013), Miami, Florida, CEUR Workshop Proceedings, Vol. 1118, 2013, pp. 53–62.

[33] Tjaden, G. S.—Flynn, M. J.: Detection and Parallel Execution of Independent Instructions. IEEE Transactions on Computers, Vol. C-19, 1970, No. 10, pp. 889–895, doi: 10.1109/t-c.1970.222795.

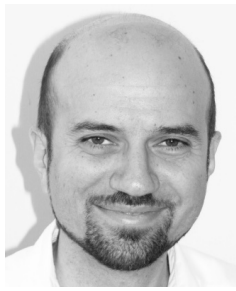
[34] Voevodin, V. V.—Antonov, A. S.—Dongarra, J. J.: AlgoWiki: An Open Encyclopedia of Parallel Algorithmic Features. Supercomputing Frontiers and Innovations, Vol. 2, 2015, No. 1, pp. 4–18, doi: 10.14529/jsfi150101.

[35] Zhong, Z.—Rychkov, V.—Lastovetsky, A.: Data Partitioning on Multicore and Multi-GPU Platforms Using Functional Performance Models. IEEE Transactions on Computers, Vol. 64, 2015, No. 9, pp. 2506–2518, doi: 10.1109/tc.2014.2375202.

Luisa D'Amore received her degree in mathematics in 1988 and her Ph.D. in applied mathematics and computer science in 1995. Since 1996 she works as researcher in numerical analysis and since 2001 she is Associate Professor of numerical analysis. Since 2011 she has been working as an associate staff researcher of the ASC (Advanced Scientific Computing) Division of CMCC (Euro Mediterranean Center on Climate Changes). She is a member of the Academic Board for the Ph.D. in mathematics and informatics at the University of Naples Federico II and a teacher of courses in numerical analysis, scientific computing and parallel computing. Her research activity focuses on scientific computing and it is addressed to the numerical solution of ill-posed inverse problems with applications in image analysis, medical imaging, astronomy, digital restoration of films and data assimilation. The need of computing the numerical solution in a suitable time, induced by the applications, often requires the use of advanced computing architectures.

This involves designing and development of algorithms and software capable of exploiting the high performance of emergingcomputing infrastructures. She is (co)author of about 100 publications in refereed journals and conference proceedings.

Valeria Mele is a Researcher at University of Naples Federico II (Naples, Italy). She obtained a degree in informatics and a Ph.D. in computational science, and for many years has been working as teaching assistant in Parallel and Distributed Computing classes at University of Naples Parthenope and University of Naples Federico II. Meanwhile, her research activity has been mainly focused on the development and the performance evaluation of parallel algorithms and software for heterogeneous, hybrid and multilevel parallel architectures, from multicore to GPU-enhanced machines and modern clusters and supercomputers. After attending the Argonne Training Program on Extreme-Scale Computing (ATPESC) and visiting the Argonne National Laboratory (ANL, Chicago, Illinois, USA), she is now mainly working on the designing, the implementation and the performance prediction/evaluation of software with/for the PETSc (Portable, Extensible Toolkit for Scientific Computation) library, collaborating with applied mathematicians and computer scientists at the Mathematics and Computer Science Division of the ANL.



Diego Romano obtained the Laurea degree (Italian M.Sc. equivalent) in mathematics in 2000, and his Ph.D. degree in computational and computer sciences from the University of Naples Federico II, Italy, in 2012. He obtained a permanent position as researcher at the Italian National Research Council (CNR) in 2008, where he is currently employed at the Institute for High Performance Computing and Networking (ICAR). His research interests include performance and design of GPU computing algorithms. Within this field, he works, for instance, on the global illumination problem in computer graphics, and on a mathematical model for the performance analysis.

Giuliano Laccetti is Professor of computer science at the University of Naples Federico II, Italy. He obtained his Laurea degree (cum laude) in physics from the University of Naples. His main research interests are mathematical software, high performance architecture for scientific computing, distributed computing, grid and cloud computing, algorithms on emerging hybrid architectures (CPU+GPU, ...), Internet of Things. He has been organizer and chair of several workshops joint to larger international conferences. He is author (or co-author) of about 100 papers published in international refereed journals, books, and conference proceedings.

Computing and Informatics, Vol. 38, 2019, 851–882, doi: 10.31577/cai_2019_4_851

PROCESS MATCHING: PERFORMANCE TRADE-OFF BETWEEN SUMMARY AND FULL-LENGTH DESCRIPTIONS

Syed Irtaza Muzaffar, Khurram Shahzad, Faisal Aslam
Madiha Khalid, Kamran Malik

Punjab University College of Information Technology
University of the Punjab
The Mall, Lahore, Pakistan
e-mail: {phdcsf17m503, khurram, faisal.aslam, madiha.khalid, kamran.malik}@pucit.edu.pk

Abstract. Business process models are used by modeling experts to concisely depict the workflow of an organization that plays a pivotal role in the development of ERP systems. A growing number of organizations also maintain the textual process descriptions of these process models as the descriptions are understandable across the board. A recent study has revealed that these textual descriptions can also be used for an accurate process model search. However, the use of textual descriptions is a resource-intensive task due to the sheer size of the descriptions. To that end, in this paper, we have proposed an approach that relies on the use of summary textual descriptions, instead of full-length descriptions, to enhance the performance of process matching. To evaluate the proposed approach, we have used four diverse text summarization techniques, including a state-of-the-art deep learning based technique, for generating summary descriptions, and seven text-matching techniques for finding relevant process specifications. Our empirical study has established that the Vector Space Model is the most effective technique for process matching. Furthermore, the use of Lingo generated summaries, at a compression rate of 50 %, can achieve a higher efficiency as well as effectiveness than the full-length textual process descriptions.

Keywords: Information retrieval systems, process retrieval, text-matching, summary-full description for process matching

1 INTRODUCTION

Business process models are widely established as key artifacts to visually represent, analyze, and enhance the business operations of an enterprise [1, 2]. As these artifacts are developed by modeling experts, they may not be readily understandable by all the stakeholders. In particular, the business users who actually execute processes have difficulties with reading and comprehension of models due to their limited knowledge of process modeling [3, 4]. Therefore, several studies have emphasized maintaining textual descriptions alongside process models. The availability of textual process descriptions has also prompted the use of these descriptions for process model validation [5], inconsistencies detection [6], and process matching [7].

A recent study [8], has proposed to employ the combination of process models with textual descriptions of its activities to enhance the accuracy of querying processes from a process repository. In contrast, another notable study has proposed the use of textual descriptions as an alternative to process models [7]. The two approaches have established that the use of textual descriptions enhances the effectiveness of process matching. However, we contend that the use of textual descriptions could be a time-consuming task due to the sheer size of these descriptions. For instance, an Austrian bank's process collection has 119 textual descriptions of processes with an average length of 13 130 words, and the longest description is composed of 60 558 words [9]. In the presence of such large textual descriptions, the use of full-length textual descriptions may impede the efficiency of the process matching.

To enhance the performance of matching, in this paper, we have proposed a summary description-based approach that relies on the compressed versions of full-length textual descriptions. The significantly reduced size of the summary descriptions should enhance the efficiency of process matching. However, we recognize that such a reduction in the descriptions may impede the accuracy of matching. Therefore, in this paper, we analyze the trade-offs between the summarized descriptions and full-length textual descriptions in terms of efficiency and effectiveness. As far as we are aware, no study has been conducted to evaluate the effectiveness of summary descriptions for process matching. In particular, the key contributions to this paper are as follows:

Proposed Approach: We have proposed a process matching approach that takes an input textual or model-based specifications of a process and returns the specifications of relevant processes. In essence, the approach generates summary textual descriptions by using text summarization techniques. Subsequently, it computes the similarity between query-source pairs using text-matching techniques and returns the specifications of the relevant processes.

Corpora Generation: We have generated corpora of 669 full-length textual descriptions and their summary textual descriptions at five different compression rates, using four diverse summarization techniques, including a state-of-the-art

deep learning based approach. Thus, in total, we have generated $(1 + 4 \times 5 =)$ 21 corpora of textual process descriptions. The techniques used to generate summary descriptions are: a) TF-IDF, which generated a collection of important words, b) LexRank, which employs a graph-based approach to rank, and subsequently choose sentences of higher rank [10], c) Lingo, which employs a clustering-based approach to identify the sentences that include key phrases of the descriptions, and d) K-means clustering with skip-thought embeddings which relies on the use of deep learning based technique to identify key sentences for inclusion in a summary description.

Analysis of Summaries: We have compared the summary textual descriptions generated by all the four summarization techniques, TF-IDF, LexRank, Lingo, and K-means clustering with skip-thought embeddings. To this end, we first generated the pairs of these summary descriptions at each compression rate. Subsequently, we employed two established text-matching techniques, n-gram overlap, and Longest Common Subsequence, to compute the similarity between each pair of textual description. The results have been used to provide valuable insights into the generated summaries.

Efficiency and Effectiveness Experiments: We have performed numerous experiments for full-length textual descriptions and summary descriptions generated by each summarization technique. For that, we have used seven text-matching techniques for each type of experiments. The results have been analyzed to empirically establish the benefit of using summary descriptions as an alternative to the full-length descriptions. Furthermore, the trade-offs between efficiency and effectiveness have been analyzed.

The rest of the paper is organized as follows: Section 2 provides an overview of the proposed approach. Section 3 presents the procedure we have used for generating full and summary textual descriptions corpora and the specifications of the corpora. Section 4 introduces the text-matching techniques that are used for experimentation. Section 5 analyzes the similarity between the summary descriptions generated by the four summarization techniques. Section 6 presents the experimental setup. Analysis of results, as well as the trade-off between efficiency and effectiveness, is presented in Section 7. Related work is presented in Section 8. Finally, conclusions are drawn in Section 9.

2 CONCEPTUAL APPROACH: AN OVERVIEW

In this section, we present an overview of the proposed approach which relies on the use of summary textual descriptions for retrieving the desired specifications of relevant processes, instead of full-length textual process descriptions. The reason to use summary descriptions over full-length descriptions stems from the potential size of process descriptions, which are particularly sizeable for end-to-end processes. For instance, a recent study [9] has highlighted that the collection of an Austrian

bank contains 119 real-world processes having an average length of 13 130 words and a maximum length of 60 558 words. The presence of such sizeable textual descriptions makes process matching a resource-intensive task which impedes the efficiency of matching. A conceptual overview of the proposed approach is presented in Figure 1. As depicted in the figure, the repository is composed of a source collection of process models and their corresponding corpus of textual process descriptions. Furthermore, a mapping can be defined between the two types of process specifications. While recent notable studies [8, 7] advocate that keeping textual process descriptions alongside process models increases the comprehension of business operations of enterprises among users, we have proposed to use summary textual description queries for process matching.

Our approach relies on the use of an automatic approach to generate textual descriptions of a process model using Natural Language Generation System (NLGS) [5]. As far as we are aware, NLGS is the only available tool that can automatically and comprehensively generate textual descriptions of a process model. It uses a well-established technique that takes a process model in the JSON format as input and generates its textual process description. In particular, the input to our proposed approach could be a model-based or textual specification as a query, whereas, the output is the specifications of relevant business processes. The approach involves three main steps: generating textual description, finding similar process descriptions, and returning specifications of the relevant processes.

In the first step, if the input query is a model-based specification, the textual process description of the query is generated using the NLGS. Secondly, the generated textual description of the input query process is compared with the textual descriptions of all the source process models available in the repository, and a similarity score of each query-source pair is computed using a text-matching technique. Subsequently, a ranked list of processes is generated, where the processes are sorted in the descending order of the similarity scores. Finally, the specifications of the top K source processes are returned. In the rest of the paper, we have used summary textual process descriptions of query processes for the process matching experiments and compared its performance with the corresponding full-length textual descriptions.

For a formal specification of the proposed approach, let p_i be a business process whose model-based specification is represented by M_{pi}. A function β can be defined that generates textual process description D_{pi} of the process model M_{pi}. Formally, $D_{pi} = \beta(M_{pi})$. Furthermore, consider P_M be a collection of process models in a repository, and C_D be a corpus of the corresponding textual descriptions. A function γ can be defined that maps textual descriptions of business processes to the model-based specifications of the processes. Formally, $\gamma : M_{pi} \rightarrow D_{pi}$. For an input query process Q_p, the relevant process specifications the query can be extracted using Algorithm 1.

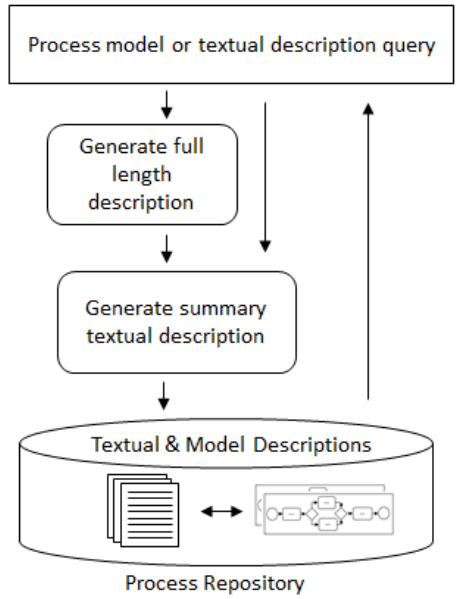


Figure 1. Overview of the proposed approach

Algorithm 1 Summary description for process matching

Input: $Q_p,\ C_D,\ P_M$ `/* query, process corpus, model collection */`
Output: $List[P_{ID},\ int\ y]$

$sim_{Score} = 0$
$L_1 = List[x,\ y]$
if $(Q_p = M_{Q_p})$ **then**
  $\beta(Q_p) \rightarrow D_{Q_p}$
  $\alpha(D_{Q_p}) \rightarrow SD_{Q_p}$
else
  $\alpha(Q_p) \rightarrow SD_{Q_p}$
end
while $D_{p_i} \in C_D$ **do**
  $sim_{Score} = similarity(SD_{Q_p}, D_{p_i})$
  $L_1.append(D_{p_i}, sim_{Score})$
end
$sort_{dec}(L_1, sim_{Score})$
return L_1

3 CORPORA GENERATION

The investigation of performance trade-off between summary and full-length textual descriptions require three artifacts: a) a corpus of full-length textual descriptions of process models, b) corpora of query descriptions used for the matching experiments, and c) corpora of summary textual descriptions of business processes. An overview of the process that we have used for generating these corpora is shown in Figure 2.

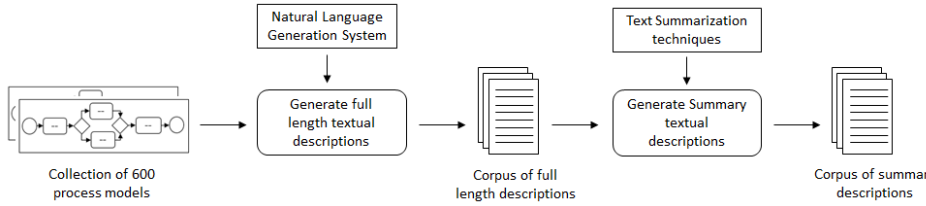


Figure 2. Comparison of textual process descriptions

3.1 Generation of Full-Length Textual Descriptions

We have access to a collection of 669 process models that are designed using Business Process Model and Notation (BPMN), which is the most prominent process modeling language. The process models in the collection are designed in the most recommended process modeling tool [12], Signavio [34]. The two key reasons for choosing this collection of models are the following: a) the collection contains process models with a diverse label and structural features [13, 11], and from several genres, hence, the results generated using such collections are valid for several domains, and b) a recent study [7] has emphasized that a necessary and sufficient pool of queries and human-generated results against these queries are also available, hence, making it a feasible test-bed for the matching experiments.

More precisely, the collection includes: a) 150 Original process models (O), including the two datasets (University Admissions Processes and Birth Registration Processes) used in the Process Model Matching Contest 2015 [14], and b) three other handcrafted variants of these 150 models, Near Copy (NC), Light Revision (LR), and Heavy Revision (HR).

Note that the variants are generated by employing a systematic and rigorous procedure to impart diversity in labels and structure of models in order to challenge the abilities of the matching techniques [13, 11]. The NC variant of a model is generated by slightly changing the formulation of each label of the model, whereas the LR variant is generated by substantially changing the formulation of each label of the model. The HR variant is generated by significantly changing the formulation of each label.

The smallest model in the collection contains 11 activities and the largest model contains 54 activities. Another unique feature of the collection is that the models included in the collection comply most of the process modeling guidelines, presented in [15]. For instance, there is no process model in the collection that contains a split gateway node, without a corresponding join gateway node. The human effort involved in generating the collection can be understood by the number of operations performed while generating the three variants. That is, 24 092 insertion, deletion, and substitution of words were performed to generate three variants of process models.

For generating the full-length textual descriptions of 669 process models, we have used Natural Language Generation System (NLGS). Figure 3 shows an example textual description generated by the NLGS. As far as we are aware, NLGS is the only established tool that can automatically and comprehensively generate a textual description of process models. Note, an empirical evaluation of the textual descriptions generated by NLGS has established that the NLGS generated textual descriptions are superior to the human-generated textual descriptions, in terms of completeness, structure, and linguistic complexity [16]. Furthermore, a users' evaluation of the NLGS generated textual descriptions demonstrate that the descriptions are understandable, and they effectively allow the reader to interpret the semantics of process models [16]. An example textual description of a healthcare process generated using the NLGS is presented in Figure 3. Accordingly, the full-length descriptions' corpus contains 87 772 words, that include 29 493 (33.6 %) stop words. These numbers indicate that the textual descriptions are not merely a collection of activity labels, rather a significant amount of stop words are used in generating the textual descriptions.

3.2 Generating Summary Descriptions

In this section, we first provide an overview of the four diverse techniques that we have used for generating summary descriptions. Subsequently, in Section 3.2.2, the procedure that we have employed for generating the corpora of summary descriptions is presented.

3.2.1 Summarization Techniques

We have used four diverse text summarization techniques, ranging from a collection of most important words to a state-of-the-art deep learning based technique, for generating summary descriptions. In particular, we have used TF-IDF [19] which is a collection of important words based approach, LexRank [10] that employs a graph-based approach to rank sentences, Lingo [32] which is a state-of-the-art approach to identify the sentences that includes key phrases of the input description, and K-means with skip thoughts embeddings [37], which employs a deep learning based approach for generating summaries. A brief overview of each summarization technique is as follows:

The process begins when the hospital inquiry checks data. Then,

- The hospital inquiry finds the information is missing. Afterwards, the hospital inquiry requests the parents to the complete information. Subsequently, the hospital inquiry conducts the information received. Then, the hospital inquiry informs the civil court.
- The hospital inquiry finds the information complete.

Once was the hospital administration confirms the parents accepted baby or not. Afterwards, is.

- The hospital administration sees the parents don't accept baby. Subsequently, the hospital administration sends the information. Then, the hospital administration forwards the case to the higher authority.
- The hospital administration sees the parents accept baby. Afterwards, the hospital administration checks the parents nationality. Subsequently, the hospital administration checks the parents Russian citizenship or not. Then, is.
  - The hospital administration finds the no one has Russian nationality. Afterwards, the hospital administration conducts the trial in court of nationality affairs. Subsequently, the hospital administration receives the citizenship decision. Then, the hospital administration registers the citizenship of the baby.
  - The hospital administration finds the at least 1 is Russian. Afterwards, the hospital administration registers the baby as Russian.

Once was the hospital administration takes the decided name of the baby. Subsequently, the hospital inquiry creates the birth certificate. Then, is.

- The hospital inquiry registers the baby.
- The hospital inquiry sends the birth information to the parents.

Once was the hospital inquiry finds at the least one of the parents has registration. Afterwards, the hospital inquiry getting the citizenship stamp on the birth certificate. Subsequently, the hospital inquiry sends the request about money benefits. Then, the hospital administration formalizes the moth payments.
Afterwards, the process is finished.

Figure 3. Textual description of order process model generated by NLGS

Term Frequency-Inverse Document Frequency (TF-IDF) relies on the importance of words in a document. That is, firstly, a frequency matrix is generated in which columns represent the vocabulary set of all the textual descriptions, whereas, rows represent the identities of textual descriptions in the collection. Secondly, using the formulas presented in Equation (1), the values of the matrix are populated. In the equation given below, t is a vocabulary term, whereas D_i is the i^{th} textual description in the collection. Finally, TF-IDF scores are used to select top N words for each document for inclusion in summary, where N is the number of words that should be included in the summary. Note that the generated summary using this technique is a mere collection of important words that may not be usable for the comprehension of the workflow of the process.

$$TF - IDF = TF(t) \times IDF(t) \tag{1}$$

where

$$TF(t) = \frac{Freq_t^{Di}}{|t \; \forall t \in Di|}$$

and

$$IDF(t) = \log \frac{|Di|}{|Di, \; such \; that, \; t \in Di|}.$$

LexRank is a sentence ranking based approach that relies on the use of Eigenvector Centrality in a graph to compute the importance of each sentence [10]. In the first step of the technique, the source text is tokenized into sentences and each sentence is represented as a vertex in a graph. In the second step, edges between the vertices are marked on the bases of Inverse Document Frequency (IDF). Note that we have adapted the notion of IDF to Inverse Sentence Frequency (ISF). That is, we take the log of the total number of sentences in the process description and divide it the number of sentences in which the word occurs, as shown in Equation (2). Subsequently, if the generated similarity score between two sentences is above a certain threshold value, a value 1 is stored in the respective index of the sentence matrix and increment 1 is performed to the degree values. Otherwise, no increment is performed to the degree value. Lastly, the final score of each sentence is computed using the power method followed by vertices sort.

$$Similarlity(S_i, S_j) = \frac{\sum_{w \in S_i, S_j} tf_{w,S_i} \times tf_{w,S_j} \times (idf_w)^2}{\alpha \times \beta}, \tag{2}$$

$$\alpha = \sqrt{\sum_{x_k \in S_i} (tf_{x_k,S_i} \times idf_{x_k})^2},$$

$$\beta = \sqrt{\sum_{y_k \in S_j} (tf_{y_k,S_j} \times idf_{y_k})^2},$$

$$p(u) = \frac{d}{N} + (1-d) \times \gamma, \tag{3}$$

$$\gamma = \sum_{v \in adj[u]} \frac{similarity(u,v)}{\sum_{z \in adj[v]} similarity(z,v)} p(v).$$

Lingo is a state-of-the-art technique that employs a clustering-based approach to identify the important sentences that include the key phrases of the given description [32]. In the first step, pre-processing is performed on the input text by removing stop-words and applying stemming. In the second step, the phrases are extracted based on the recurring ordered sequences of terms appearing in the document. Subsequently, a term-document (t-d) matrix is generated for each key phrase in the document. In the third step, the matrix is factorized using Singular Value Decomposition (SVD) to find cluster labels, formally called the topic label of the document. In particular, we have used a publicly available implementation [33]. Finally, we generate a summary by extracting those sentences which contain the most important topics.

K-means clustering is an extraction based approach in which sentences are extracted using k-means clustering technique with skip-thought embeddings. As a starting point, each document is decomposed into its constitute sentences. In the second step, encoder, which is the main part of the skip thought mode, encodes the sentences using Recurrent Neural Network with Gated Recurrent Unit. Meaning that a fixed-length vector representation for each sentence is generated [26]. Equations (4)–(8) describe the sequences of steps which are performed to encode the sentences.

$$r^t = \sigma(W_r x^t + U_r h^{t-1}), \tag{4}$$

$$z^t = \sigma(W_z x^t + U_z h^{t-1}), \tag{5}$$

$$\bar{h^t} = \tanh(W x^t + U(r^t \bigodot h^{t-1})), \tag{6}$$

$$h^t = (1 - z^t) \bigodot h^{t-1} + z^t \bigodot \bar{h^t} \tag{7}$$

where r^t is the reset gate, $\bigodot$ represents the element-wise multiplication, z^t is the update gate, and $\bar{h^t}$ is the proposed state update at time t. In the third step, the encoded sentences are clustered using K-means clustering techniques as shown in Equation (8).

$$Kmeans = \sum_{j=1}^{k} \sum_{i=1}^{n} ||x_i^j - c_j|| \tag{8}$$

where $||x_i^j - c_j||$ is selected distance measure between a data point x_i^j and cluster c_j. Finally, the sentences corresponding to sentence embeddings closest to the cluster centers are chosen for inclusion in the summary. The implementation of the technique that we have used in this study can be downloaded from [31].

3.2.2 Generating Summary Descriptions

We have generated the summary descriptions of 669 full-length textual descriptions using the four text summarization techniques discussed above at five different compression rates, 10 %, 20 %, 30 %, 40 % and 50 %. The x % compression rate indicates that the top x % important sentences are preserved in the generated summary. In case, the number of sentences to be preserved is in decimal (50 % of 7 is 3.5), the decimal value was truncated. Accordingly, we yielded a total of twenty summary descriptions corpora, each containing summary descriptions of 669 process models. Table 1 shows an example summary generated by each technique, TF-IDF, LexRank, Lingo, and K-means clustering with skip-thought embeddings, at a compression rate of 50 %. It can be observed from the table that the TF-IDF generated summary is a collection of words rather than complete sentences. Therefore it is not usable for compression of the workflow of the process. On the contrary, the summaries generated by the three other techniques are readable as the techniques rely on ranking sentences and subsequently choosing a subset of sentences for generating summaries. In the example, the bold text represents the sentences that are common between all the three sentence-level summaries, whereas the italic text represents the sentences that are common between two sentence-level summaries.

3.2.3 Query Descriptions and Human Annotations

Typically, the existing studies, such as [17], use merely ten randomly selected queries and a source collection of 100 processes, to evaluate the effectiveness of the matching technique. A key limitation of using such a small and randomly selected queries is that the findings generated in these settings may not be reliable. In contrast to those studies, we have chosen a very large number of 56 queries, a collection of 669 process descriptions, and a recently developed human benchmark [7, 13], for our experimentation.

As discussed in Section 3.1, a key motivation for the choice of the source collection is that it includes a large and carefully handcrafted collection of over 600 process models, which includes three variants of each process model, NC, LR, and HR. A key reason for the choice of 56 queries over a randomly collected pool of merely ten queries is that our queries are selected by employing a systematic and rigorous procedure, without having a pre-defined number in mind. Furthermore, the principal purpose of the procedure was to ensure the inclusion of a necessary and sufficient set of query processes. Essentially, the procedure is composed of four main steps. In the first step, the values of the widely use structural features of each process model were computed. These structural features are size, diameter, sequentiality,

TF-IDF	[the, hospital, administration, inquiry, then, is, parents, afterwards, subsequently, finds, baby, information, citizenship, of, checks, to, was, sends, registers, birth, process, information, conducts , baby , or , not, sees, accept, nationality, russian, one, has, russian, once, certificate, begins, when, a, data, missing, requests, complete]
LexRank	**The process begins when the hospital inquiry checks a data.** *Subsequently, the hospital inquiry conducts the information received.* **Then, the hospital inquiry informs the civil court. Once was the hospital administration confirms the parents accepted baby or not. The hospital administration sees the parents don't accept baby.** Then, the hospital administration forwards the case to the higher authority. The hospital administration finds the no one has Russian nationality. *Afterwards, the hospital administration conducts the trial in court of nationality affairs.* **Subsequently, the hospital administration receives the citizenship decision.** The hospital administration finds the at least 1 is Russian. *Once was the hospital administration takes the decided name of the baby.* Subsequently, the hospital inquiry creates the birth certificate. **Once was the hospital inquiry finds at the least one of the parents has registration. Afterwards, the hospital inquiry getting the citizenship stamp on the birth certificate. Subsequently, the hospital inquiry sends the request about money benefits.** Then, the hospital administration formalizes the moth payments.
Lingo	**The process begins when the hospital inquiry checks a data.** Then is . The hospital inquiry finds the information is missing. *Subsequently the hospital inquiry conducts the information received.* **Then the hospital inquiry informs the civil court. Once was the hospital administration confirms the parents accepted baby or not.** Afterwards is . **The hospital administration sees the parents don't accept baby.** The hospital administration sees the parents accept baby. *Afterwards the hospital administration conducts the trial in court of nationality affairs.* **Subsequently the hospital administration receives the citizenship decision.** *Then the hospital administration registers the citizenship of the baby.* **Once was the hospital inquiry finds at the least one of the parents has registration. Afterwards the hospital inquiry gettings the citizenship stamp on the birth certificate. Subsequently the hospital inquiry sends the request about money benefits.** *Afterwards the process is finished.*
RNN	**The process begins when the hospital inquiry checks a data.** Afterwards, the hospital inquiry requests the parents to the complete information. **Then, the hospital inquiry informs the civil court. Once was the hospital administration confirms the parents accepted baby or not. The hospital administration sees the parents don't accept baby.** Then, is . Afterwards, the hospital administration checks the parents nationality.The hospital inquiry finds the information complete. **Subsequently, the hospital administration receives the citizenship decision.** The hospital administration finds the no one has russian nationality. *Then, the hospital administration registers the citizenship of the baby. Once was the hospital administration takes the decided name of the baby.* **Afterwards, the hospital inquiry gettings the citizenship stamp on the birth certificate. Once was the hospital inquiry finds at the least one of the parents has registration. Subsequently, the hospital inquiry sends the request about money benefits.** *Afterwards, the process is finished.*

Table 1. Summaries generated by TF-IDF, LexRank, Lingo and RNN at compression rate 50 %

network connectivity, token split, etc. [20]. Further details of these metrics can be found in [20]. In the second step, a correlation was computed between every pair of structural metrics. Subsequently, for each pair of metrics having a very high co-relation of over 0.9 one metric was excluded. Hence, ensuring that the values of only adequate metrics are taken into consideration. Thirdly, to ensure the diversity, the process models with the highest, lowest, and average values of each metric were selected as query models. Finally, the steps were repeated for each variant in the collection, NC, LR, and HR, while avoiding redundancy. Accordingly, the generated collection includes 14 process models of each type, Original, NC, LR, and HR, as well as the process model with diverse structural properties.

The full-length textual descriptions of the selected 56 query models are used as an input to the four summarization techniques to generate the summaries of the query processes at different compression rates (10 %, 20 %, 30 %, 40 %, and 50 %). As a result, the summary descriptions of 1 120 queries were generated. These summary descriptions have been used as queries in the rest of the paper for experimentation.

4 MATCHING TECHNIQUES

A notable study [18] has classified text-matching approaches into seven broad categories: overlapping of grams, lexical similarity, string and sequence comparison, fingerprinting, probabilistic methods, NLP methods, and structural methods. The approaches in the former three categories primarily rely on the actual content of the query-source descriptions, whereas, the latter four rely on the use of structural or textual features of the query-source descriptions. In this study, we limit our choice of matching techniques to the former three categories of techniques due to two reasons:

1. summarization may have substantially changed the structure or textual features of the descriptions, which may ultimately affect the matching performance, and
2. the latter four categories of approaches increase the computational overhead of computing structural or textual features of the query-source descriptions.

Below, we provide an overview of the matching techniques used in this study. In particular, we present one technique from the first and second categories (N-gram overlap and Vector Space Model, respectively) and three techniques (Longest Common Subsequence, Local Alignment, and Global Alignment) from the third category.

4.1 N-Gram Overlap

N-gram overlap computes the similarity between a query-source descriptions pair by dividing them into a set of tokens, called grams [21]. It then counts the number of common tokens in the two descriptions and divides it by the number of tokens in one or both descriptions, to get a normalized score between 0 and 1. The value of n determines the number of words in each token. Formally, it is defined as follows:

$$S(Q,S) = \frac{|T(Q) \cap T(S)|}{\min(|T(Q)|, |T(S)|)} \tag{9}$$

where $T(Q)$ and $T(S)$ is the number of token in query and source description, respectively.

4.2 Vector Space Model (VSM)

VSM computes the similarity between a query-source descriptions pair by first representing each description in a vector space, where each word in the description represents a dimension in a vector space [22]. The similarity is then measured by

computing angle between them. Formally, the normalized score is computed as follows:

$$S(Q,S) = \frac{\sum_{i=1}^{n} Q_i \times S_i}{\sqrt{\sum_{i=1}^{n}(Q_i)^2 \times \sum_{i=1}^{n}(S_i)^2}}. \tag{10}$$

4.3 Longest Common Subsequence (LCS)

Longest Common Subsequence computes the similarity between a query-source descriptions pair by identifying the longest consecutive sequence of tokens that are common between the two descriptions and dividing it with the length of the smaller description, to compute a normalized similarity score [18]. Formally, it is defined as follows:

$$S(Q,S) = \frac{|LCS|}{\min(|Q|,|S|)} \tag{11}$$

where LCS is the longest sequence of tokens that are common between the two descriptions.

4.4 Local Alignment (LA)

Local Alignment computes the similarity between a query-source descriptions pair by identifying the identical portion of tokens (small regions) between the two sequences [24]. In particular, for each matching pair of tokens the matching score is incremented by 1, and for each mismatched pair of tokens the matching score is decremented by 1. Subsequently, the normalized score is computed by dividing the similarity score with the minimum length of the query-source description

$$S(Q,S) = \frac{L_{score}}{\min(|Q|,|S|)}. \tag{12}$$

4.5 Global Alignment (GA)

Global Alignment computes the similarity between a query-source descriptions pair by representing both descriptions as a sequence of words and then identifying the identical text between the entire length of the two descriptions [25]. Generally, the technique is recommended for a sequence of equal and near-equal lengths. For each matching pair of tokens, the matching score is incremented by 1, and for each mismatch, the score is decremented by 1. The normalized score is then computed by the following equation:

$$S(Q,S) = \frac{G_{score}}{\min(|Q|,|S|)}. \tag{13}$$

5 COMPARISON OF SUMMARY DESCRIPTIONS

In this section, we have computed the similarity between summary descriptions of queries generated by the four summarization techniques to evaluate how similar or dissimilar are the summary descriptions. The process that we have employed for the comparison of summary descriptions is presented in Figure 4. In particular, we have generated 20 corpora, each containing summary textual descriptions of 56 query processes, i.e., a corpus of summary descriptions generated by each summarization technique at each compression rate 10 %, 20 %, 30 %, 40 %, and 50 %. The comparison of these $56 \times 5 \times 4 = 1\,120$ summary descriptions would require creating numerous pairs of summary descriptions.

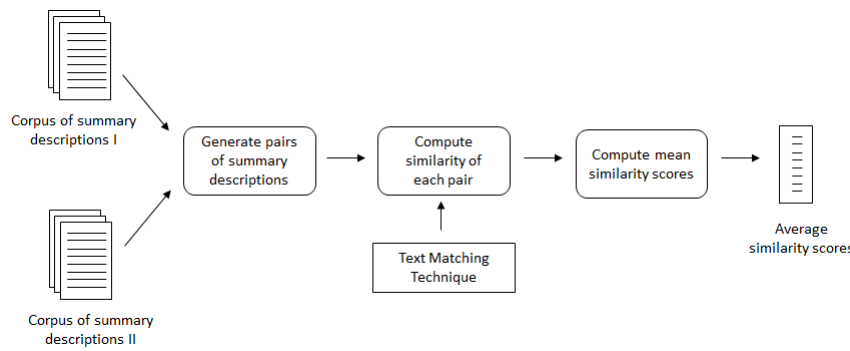


Figure 4. Comparison of textual process descriptions

The manual comparison of these many pairs is a tedious task which requires a substantial human effort. Therefore, we have used two similarity estimation techniques (n-gram overlap and Longest Common Subsequence) to compute the similarities between these pairs. N-gram computes the degree of similarity between a query-source pair by counting the number of unique tokens (common words) and dividing it by the length of the short description to get a normalized score. The similarity score thus represents the content overlap between the query-source pair without taking into consideration the ordering of the words. Due to that limitation, we have also used a variant of LCS – an order-preserving similarity estimation method. LCSnorm, a variant of LCS, computes the similarity by counting the number of edit operations required to transform one text into another and dividing it with the length of the short text.

Table 2 shows the average similarity scores of all possible combinations of pairs of summary descriptions. In the table, the average similarity score of 0.62 at a compression rate 50 % for the 1-gram technique represents that 62 % of the unique words (vocabulary) used by these two algorithms overlap. The key observations from the results are as follows:

Techn.	Comp.	TF-IDF &			LexRank &		Lingo &
		LexRank	Lingo	K-means	Lingo	K-means	K-means
Unigram	50 %	0.62	0.64	0.60	0.86	0.86	0.84
	40 %	0.46	0.50	0.47	0.78	0.77	0.78
	30 %	0.31	0.37	0.38	0.67	0.66	0.72
	20 %	0.23	0.29	0.33	0.58	0.58	0.68
	10 %	0.16	0.20	0.26	0.44	0.43	0.57
LCS	50 %	0.39	0.37	0.34	0.60	0.63	0.60
	40 %	0.35	0.36	0.32	0.54	0.58	0.56
	30 %	0.29	0.34	0.34	0.48	0.52	0.52
	20 %	0.28	0.33	0.39	0.44	0.48	0.49
	10 %	0.26	0.36	0.52	0.36	0.35	0.48

Table 2. Average similarities scores between summaries of 56 query descriptions

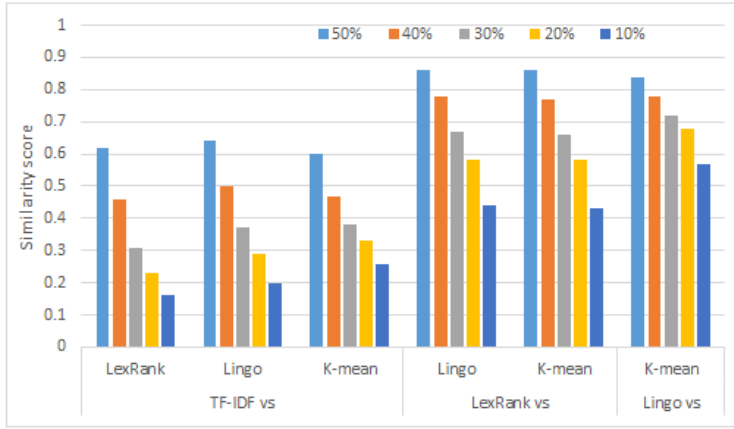


Figure 5. Unigram based comparison of all pairs

Vocabulary overlap. It can be observed from Table 2 that the unigram similarity score for a large majority of the cases, 53 out of 60, are less than or equal to 0.67. These lower values indicate that at least one-third of the vocabulary between these pairs is different. For the remaining seven cases, which are highlighted with gray color, the unigram score is substantially high, i.e., 0.86, 0.86, 0.84, 0.78, 0.77, 0.78, and 0.72. However, it can be observed from the table that the LCS scores of the pairs, where the vocabulary overlap, are also higher, i.e., 0.60, 0.63, and 0.60, 0.54, 0.58, 0.56, and 0.52. These lower values represent that the ordering of the words in these summaries is significantly different from each other, hence, indicating a significant difference between the summaries.

Similarity between types of pairs. Figure 5 plots the n-gram similarity scores between all the pairs of summary descriptions. From the figure, it can be ob-

served that the similarity scores between TF-IDF generated summaries and the ones generated by the remaining techniques are substantially low. On the contrary, the corresponding similarity scores between the other pairs are on the higher side. That is, the similarity score of LexRank & Lingo, LexRank & K-means, and Lingo & K-means are higher than that of TF-IDF & LexRank, TF-IDF & Lingo, and TF-IDF & K-means. A key reason for the differences in the similarity scores stems from the fact that TF-IDF employs an entirely different mechanism from the other three techniques for generating summaries. That is, TF-IDF employs a word-based approach to rank and identify important words for a summary, whereas, the other three techniques employ a sentence-ranking approach to identify a subset of sentences for inclusion in the summary.

Impact of compression rate on the similarity. It can be observed from Figure 5 that as the compression rate decreases from 50 % to 10 %, the vocabulary overlap between the summaries decreases gradually. These decreasing numbers represent that the differences between the summaries in the pair widen with the decrease in the compression rate. Hence, indicating that all the techniques employ a different mechanism to rank words or sentences which becomes more visible when a smaller number of words or sentences are chosen for generating a summary.

From the above discussion, we conclude that the summary textual descriptions generated by the four techniques, TF-IDF, LexRank, Lingo, and K-means, are significantly different from each other. Hence, the choice of the summarization technique is a non-trivial task. This raises several questions, such as, what is the impact of different summarization techniques on process matching? Which summarization technique generates the most appropriate summary for effective, as well as efficient process matching results? What level of compression rate is most appropriate for effective, as well as efficient process matching? To answer these questions, in the remaining part of the paper, we have performed several process matching experiments.

6 PROCESS MATCHING EXPERIMENT SETUP

This section presents the details of the experimental setup that we have used for analyzing the performance trade-off between summary and full-length textual descriptions for process matching. An overview of the experimental setup is presented in Figure 6.

6.1 Dataset and Evaluation Measures

For the experiments, the full-length descriptions of 669 process models have been used as a source collection, and two sets of queries have been used for the process matching:

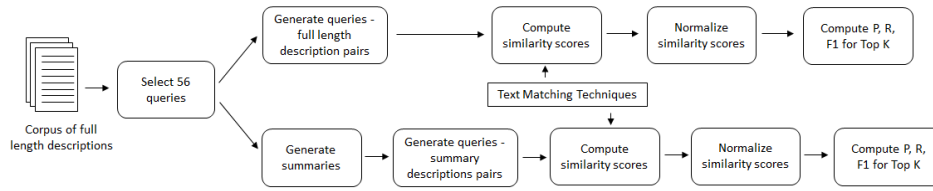


Figure 6. Experimental setup

1. full textual descriptions of 56 query models,
2. four sets of summary textual descriptions of 56 query models at five different compression rates 10 %, 20 %, 30 %, 40 % and 50 %.

The details of the summary descriptions and the human annotations have been discussed in Section 3, whereas the trade-offs between the use of summary and full-length textual descriptions have been analyzed in terms of *effectiveness* and *efficiency*.

For the *effectiveness* of matching, we have used three established measures, Precision (P), Recall (R) and F1 score. Precision represents the percentage of source process models that are retrieved and are relevant. Recall represents the percentage of source process models that are relevant and retrieved. F1 score is a harmonic mean of Precision and Recall.

For the *efficiency*, we have used Retrieval Time (RT) as a measure. For a full-length query description, RT is the time taken by a technique to match the full-length query description with all the source descriptions in the collection. Whereas, for a summary description query, RT is the sum of the time spent to generate a summary description of the query process and the time taken by a technique to match the summary query descriptions with all source descriptions in the collection.

6.2 Evaluation Methodology

We have implemented all the four summarization techniques, TF-IDF, LexRank, Lingo, and K-means, as described in Section 3.2.1. Each implementation takes a full-length textual description as input and generates its summary descriptions at five compression rates 10 %, 20 %, 30 %, 40 %, and 50 %. For measuring the effectiveness, the matching techniques presented in Section 4 are used for experiments. Each technique has been implemented as a program, where each implementation takes a query process description as input and generates its pairs with 669 source process descriptions, formally called query-source pairs. Subsequently, each implementation computes a similarity score between 669 query-source pairs and saves them in a text file in descending order of the similarity score, meaning that the

most relevant processes are on the top. Furthermore, top K processes have been generated by varying the value of K between 4 and 16, with a step size of four. The reason for the varying value of K lies in the nature of the source process collection, i.e., the source collection contains four variants of each model, 150 original process specifications, and three handcrafted variants of each model. Consequently, keeping the step size as 4 has helped us evaluate whether or not all the variants are ranked in the top slots. Finally, Precision, Recall, and F1 scores have been computed after applying pre-processing in order to compare the effect of each pre-processing step. In particular, query and source descriptions have been pre-processed by removing stop words, stemming (using Snowball stemmer), and a combination of both. The process was repeated for full-length query descriptions, as well as for the summary query descriptions generated by TF-IDF, LexRank, Lingo, and K-means, at five different compression rates.

For measuring the efficiency, the implementations of the summarization techniques were modified to include the computation of the summary generation time. Similarly, the implementations of the matching techniques were modified to compute the retrieval time, as defined in Section 6. These implementations take a query-source pair as input and compute the retrieval time of each pair. Subsequently, the retrieval time was saved in a text file. Note that the efficiency experiments have been performed 10 times for 56 queries and at each compression rate. The results presented in this paper are the average of the 10 iterations. Similarly, the summary generation time used in this paper is the average of the 10 iterations.

Technique	**Full Desc.**			**TF-IDF**			**LexRank**		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
1-Gram	0.719	0.455	0.557	0.741	0.471	0.576	0.705	0.444	0.545
2-Gram	0.723	0.454	0.558	0.411	0.253	0.313	0.696	0.441	0.540
3-Gram	0.688	0.438	0.535	0.058	0.031	0.040	0.647	0.420	0.535
GA	0.656	0.426	0.517	0.754	0.475	0.583	0.714	0.449	0.517
LCS	0.634	0.428	0.511	0.737	0.462	0.568	0.674	0.429	0.511
LA	0.616	0.388	0.476	0.250	0.152	0.189	0.580	0.366	0.476
VSM	**0.772**	**0.487**	**0.597**	**0.763**	**0.480**	**0.589**	**0.786**	**0.502**	**0.597**

Table 3. Effectiveness comparison of full and summary descriptions (top 4)

7 RESULTS AND ANALYSIS

7.1 Effectiveness Results

Precision, Recall, and F1 scores provide three different types of measures to gauge the effectiveness of a matching technique. Therefore, we have included the results of all the three measures in Table 3 and Table 4, for all the matching techniques,

Tech.	Full Desc.			Lingo			RNN		
	P	R	F1	P	R	F1	P	R	F1
1-Gram	0.719	0.455	0.557	0.688	0.439	0.536	0.670	0.416	0.513
2-Gram	0.723	0.454	0.558	0.705	0.448	0.548	0.683	0.430	0.528
3-Gram	0.688	0.438	0.535	0.661	0.426	0.518	0.625	0.410	0.495
GA	0.656	0.426	0.517	0.705	0.450	0.549	0.674	0.420	0.518
LCS	0.634	0.428	0.511	0.728	0.465	0.568	0.603	0.385	0.470
LA	0.616	0.388	0.476	0.629	0.405	0.493	0.554	0.341	0.422
VSM	**0.772**	**0.487**	**0.597**	**0.790**	**0.500**	**0.612**	**0.795**	**0.504**	**0.617**

Table 4. Effectiveness comparison of full and summary descriptions (top 4)

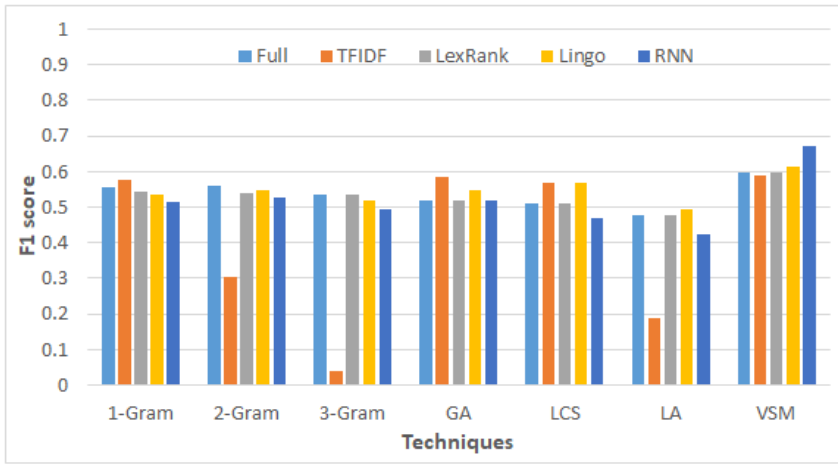


Figure 7. Performing variation across summarization techniques

where full-length textual descriptions and the summary descriptions are used as queries. Note that each value in the table is an average score of the 56 query descriptions. It can be observed from the Table 3 that the Precision scores of full-length descriptions are significantly higher than the Recall scores for all the matching techniques. Furthermore, a similar trend can be observed from Table 3 and Table 4, for each summarization technique. A higher value of Precision represents that among the processes retrieved by a technique the majority of the processes were relevant, whereas, a lower Recall score represents that the majority of the variants were not retrieved. Our synthesis of the Recall results revealed that for each query the identical processes were retrieved, whereas, for a majority of the queries the NC variants were also retrieved. Furthermore, the LR variants of some queries were retrieved, whereas, HR variants of a few queries were retrieved. The key observations based on the analysis of results are as follows:

Most suitable matching technique. It can be observed from the table that the Vector Space Model outperformed all the matching techniques for all types of summary descriptions, as well as for the full-length descriptions. That is, the Precision, Recall, as well as the F1 scores of the VSM are higher than all the other techniques, and for both types of descriptions, summary and full-length descriptions. These highest values are highlighted using gray background color in the table. A possible explanation to the higher values of the evaluation measures stems from the fact that the VSM firstly represents both query and source documents as vectors in a high-dimensional space, and subsequently the similarity is computed by the degree of angle between query-source vectors, rather than merely the overlap in the contents or the alignment of the text.

Variation across summarization techniques. A comparison of the F1 scores of the summary and full-length descriptions is presented in Figure 7. It can be observed from the figure that there is no single summarization technique that outperforms all the other summarization techniques for all the matching techniques. Furthermore, the summary generated by the state-of-the-art deep learning based summarization technique achieved the highest F1 score of 0.617 using the VSM matching technique, whereas Lingo achieved a comparable F1 score of 0.612. These two observations indicate that the most appropriate combination of a summarization and matching technique is the VSM and K-means clustering with skip-thought embeddings, which is a deep learning based approach.

Tech.	**Full Desc.**	**LexRank**	**TF-IDF**	**Lingo**	**RNN**
1-Gram	559	465	359	380	972
2-Gram	558	523	402	433	1 030
3-Gram	561	477	397	414	1 038
GA	2 763	1 180	468	918	1 491
LCS	855	415	241	346	922
LA	6 871	2 955	928	1 116	1 478
VSM	**21**	**20**	**28**	**17**	**603**

Table 5. Retrieval time (in milliseconds) comparison of full and summary descriptions

7.2 Synthesis of Effectiveness

For a thorough analysis of the results, we have synthesized the effectiveness score of 56 queries by dividing them into four types, such that each type has an equal number of queries. The query-types are Original Queries (OQ), Near Copy Queries (NCQ), Light Revision Queries (LRQ), and Heavy Revision Queries (HRQ). For each type of queries, all the experiments have been repeated, and the Precision, Recall, and F1 scores have been recorded in Table 6. The key observations elicited from these results are as follows.

It can be observed from the shaded elements in Table 6 that the Precision, Recall, and F1 score achieved by VSM for *each* query type is higher than the corresponding scores achieved by any other matching technique when RNN generated summary descriptions were used for the process matching. These results are consistent with the results of VSM presented in Table 4, where VSM outperformed all the other techniques based on the average F1 score of 56 queries. Hence, reinforcing that VSM is the most effective matching technique.

It can also be observed from the bold values in Table 6 that the average F1 scores achieved by HRQs are always significantly less than the F1 scores achieved by OQs, NCQs, and LRQs, for all the matching techniques. This observation is valid for the full-length descriptions, as well as for the RNN generated summary descriptions. This lower value is due to the significant differences in the specifications of HR variants from O, NC, and LR variants of processes. Hence, indicating that the identification of HR variant is a challenging task for the matching techniques. It is thus desirable to invent new matching techniques that can effectively retrieve heavily modified variants of processes.

Another notable observation is that for the all types of queries, the F1 score achieved using the RNN generated summary descriptions is either higher than or comparable with the F1 scores achieved by the full-length generated descriptions. This observation is valid across all the matching techniques. In particular, the average differences in F1 scores between the matching techniques are 0.028, 0.062, 0.058, and 0.033 for OQs, NCQs, LRQs, and HRQs, respectively. These results indicate that the RNN generated summary descriptions are equally effective for all query-types and the matching techniques.

7.3 Efficiency Analysis

Table 5 shows a comparison of the Retrieval Time (RT) for the matching techniques where the full-length and summary generated textual descriptions are used as queries. Recall from Section 6 that, for a full-length description query, RT is the time taken by a technique to match the query with all the source descriptions in the collection. On the contrary, for a summary description query, RT is the sum of the time consumed in generating a summary description and the time taken by a technique to match the summarized query description with all the source descriptions in the collection. Each value in Table 5 represents the average RT of 56 queries for 10 iterations. The key observations about efficiency analysis are as follows:

Efficiency of the matching techniques. VSM is the most efficient technique for process matching, as its retrieval time is merely 21 milliseconds in case of full-length query descriptions. Furthermore, it can be observed from the table that the use of summary descriptions substantially reduces the RT of all the matching techniques, with the exception of VSM. That is, the RT of VSM does not decrease substantially. Further analysis of the RT of VSM revealed that the RT for full description queries is minuscule, 21 milliseconds only. Hence, the

		Full-length description			RNN summary		
Queries	**Technique**	**P**	**R**	**F1**	**P**	**R**	**F1**
OQ-14	Unigram	0.804	0.441	0.570	0.804	0.438	0.567
	Bigram	0.786	0.439	0.563	0.786	0.428	0.554
	Trigram	0.786	0.439	0.563	0.732	0.422	0.535
	GA	0.804	0.454	0.580	0.821	0.464	0.593
	LCS	0.786	0.462	0.582	0.768	0.443	0.562
	LA	0.768	0.418	0.541	0.696	0.377	0.489
	VSM	0.821	0.444	0.576	0.893	0.498	0.639
NCQ-14	Unigram	0.714	0.537	0.613	0.661	0.482	0.557
	Bigram	0.696	0.519	0.595	0.679	0.513	0.584
	Trigram	0.714	0.537	0.613	0.714	0.536	0.612
	GA	0.696	0.527	0.600	0.732	0.542	0.623
	LCS	0.732	0.575	0.644	0.661	0.506	0.573
	LA	0.714	0.525	0.605	0.571	0.419	0.483
	VSM	0.804	0.614	0.696	0.786	0.596	0.678
LRQ-14	Unigram	0.696	0.489	0.574	0.625	0.430	0.509
	Bigram	0.714	0.491	0.582	0.625	0.432	0.511
	Trigram	0.607	0.428	0.502	0.500	0.372	0.427
	GA	0.643	0.456	0.534	0.589	0.373	0.457
	LCS	0.589	0.420	0.490	0.518	0.332	0.405
	LA	0.482	0.338	0.397	0.464	0.303	0.367
	VSM	0.696	0.469	0.560	0.714	0.496	0.585
HRQ-14	Unigram	0.661	0.352	**0.459**	0.589	0.315	**0.410**
	Bigram	0.696	0.368	**0.481**	0.643	0.346	**0.450**
	Trigram	0.643	0.349	**0.452**	0.554	0.309	**0.397**
	GA	0.482	0.269	**0.345**	0.554	0.300	**0.389**
	LCS	0.429	0.256	**0.321**	0.464	0.258	**0.332**
	LA	0.500	0.270	**0.351**	0.482	0.263	**0.340**
	VSM	0.768	0.420	**0.543**	0.786	0.426	**0.553**

Table 6. Effectiveness comparison of query types (Top 4)

overhead of summary generation time becomes dominant. That is, the summary generation time of LexRank is 3 milliseconds, whereas its matching time is 17 milliseconds, which represents a slight decrease in the matching time. Similarly, for the other techniques, TF-IDF, Lingo, and K-means, the time required for generating summaries is also minute, 0.015, 0.030, and 584 milliseconds, respectively.

Efficiency of the summarization techniques. It can be observed from the Figure 8 that the RT of a large majority of the summarization techniques is significantly less than that of the full-length query descriptions. A notable observation is that, in contrast to other summarization techniques, the retrieval time of K-means generated summaries is significantly higher than that of the

full-length descriptions. It is due to the reason that K-means employs a deep learning technique that takes into consideration several parameters for generating summaries, hence, making it a resource-intensive task. Furthermore, it can be observed from Figure 8 f) that the RT of K-means generated summaries is manifolds higher than that of the full-length generated descriptions. Our synthesis of the RT revealed that the retrieval time of summary descriptions actually reduces from 21 to 19 milliseconds. However, due to the higher amount of summary generation time, i.e. 584 milliseconds, the overall retrieval time inflates significantly.

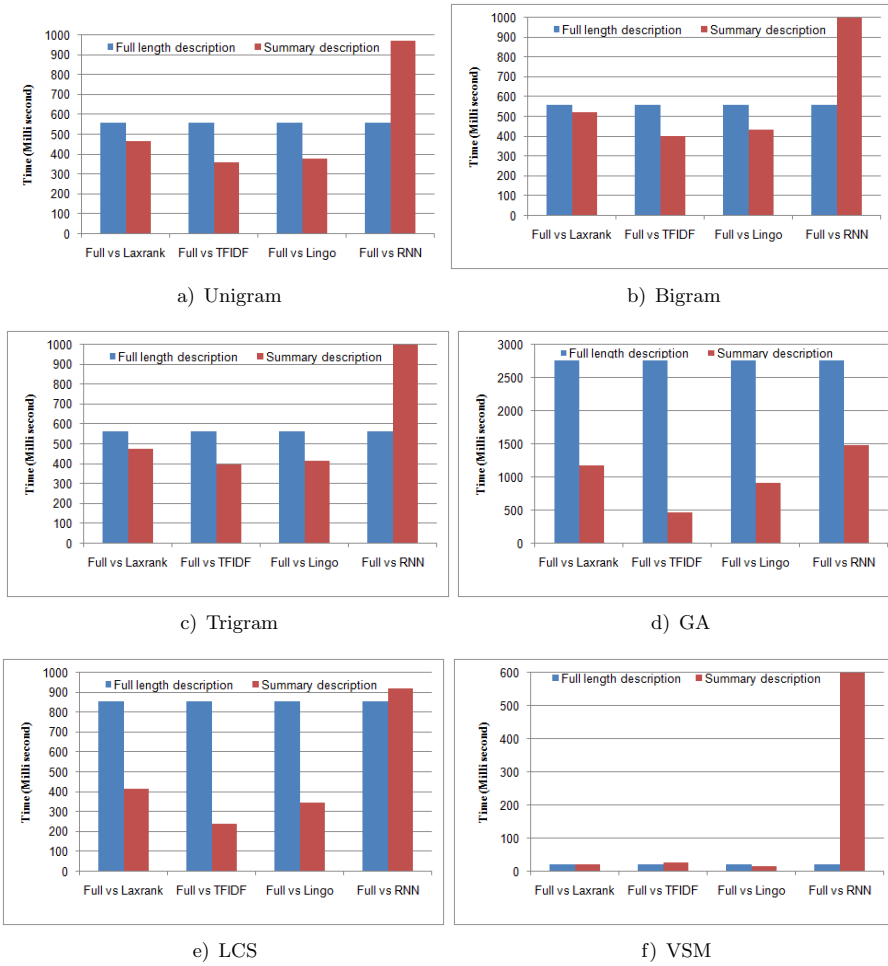
From the discussion we conclude that the choice of summarization technique, as well as the matching technique, significantly contributes to the efficiency of matching. However, one must take into consideration the trade-off between effectiveness and efficiency. Therefore, the subsequent section focuses on analyzing this trade-off in detail.

	Effectiveness				**Efficiency**			
Tech.	TF-IDF	LexRank	Lingo	K-means	TF-IDF	LexRank	Lingo	K-means
1-Gram	+	–	–	–	+	+	+	–
2-Gram	–	–	–	–	+	+	+	–
3-Gram	–	NA	–	–	+	+	+	–
GA	+	NA	+	+	+	+	+	+
LCS	+	NA	+	–	+	+	+	–
LA	–	NA	+	–	+	+	+	+
VSM	–	NA	+	+	–	+	+	–

Table 7. Trade-off between Efficiency (EF) and Effectiveness (EC)

7.4 Efficiency-Effectiveness Trade-Off Analysis

In this section, we discuss the performance trade-off between the summary and full-length textual descriptions. Table 7 provides an overview of the trade-off between these descriptions in terms of efficiency and effectiveness. In the table, a '+' sign for *effectiveness* represents that the use of summary description has a positive impact on the effectiveness of matching, i.e., the average F1 score achieved by the summary queries is higher than the full-length description. Similarly, a '−' sign represents that the use of summaries impedes the effectiveness of matching. On the contrary, a '+' sign for *efficiency* represents that the performance of matching, in terms of efficiency, increases when summary descriptions are used for process matching. That is, the average RT of matching of summary query descriptions is less than the full-length query descriptions, whereas, the '−' sign represents that the matching time of summary descriptions is higher than the full-length descriptions.



a) Unigram b) Bigram

c) Trigram d) GA

e) LCS f) VSM

Figure 8. Performance comparison of summary and full-length description

It can be observed from Table 7 that the efficiency of the matching improves for a large majority of the cases when summary query descriptions are used for process matching. In particular, the efficiency increases when the LexRank generated summary queries are used for matching. However, the use of the LexRank generated summary queries does not increase the effectiveness of matching.

It can also be observed from the table that the efficiency, as well as the effectiveness of matching, does not increase for a large majority of the matching techniques when K-means generated summary descriptions are used as queries. In contrast to that, the use of the summary description queries generated by TF-IDF and Lingo increases the efficiency as well as effectiveness for multiple matching techniques.

Among these two summarization techniques, the effectiveness scores of the Lingo generated summary is higher than that of TF-IDF for a majority of the matching techniques making it more suitable for process matching.

Furthermore, for the most effective matching technique, VSM, the F1 score of 0.612 achieved by Lingo generated summaries is comparable with the F1 score of 0.617 achieved by K-means generated summaries, whereas, the retrieval time of Lingo generated summaries is merely 17 milliseconds, which is less than that of TF-IDF, 28 milliseconds. Hence, we conclude that the summary textual descriptions generated by Lingo at a compression rate of 50 % can achieve a comparable or higher efficiency as well as effectiveness than the summary descriptions generated by the other three techniques, as well as by using full-length textual descriptions.

8 RELATED WORK

The approaches to process matching can be classified into two broad categories:

1. effectiveness enhancement approaches, and
2. efficiency enhancement approaches.

Effectiveness enhancement approaches: This category includes the approaches that aim to enhance the effectiveness of process matching. Several approaches, such as Ref. [17, 27], have proposed to combine structural and behavioral features with label features to decide a query-source pair as equivalent or not equivalent. To compute the similarity between label features, a large majority of the techniques employ syntactic measures, such as distance-based measures [28], to simply count the number of edit operations required to convert one label into another. More advanced techniques, to compute the similarity between label features, use semantic and contextual measures [29]. These measures rely on a lexical database, WordNet [30], to compute semantic similarity between labels. The similarity between label features is combined with graph matching techniques to compute the similarity between a query-source process model pair. Behavioral feature-based approaches [35] compute the similarities between a pair of process models using their execution behaviors, formally called the causal relationship between activities.

Recent studies, such as [36], enhance the accuracy of process matching by integrating the specification of a process model with the textual descriptions of its elements. Another study [7] has proposed the use of textual descriptions as an alternative to the process models.

Efficiency enhancement approaches: This category includes the approaches that aim to enhance the efficiency of the process matching. To the best of our knowledge, this category includes only two approaches, Ref. [38] and [39]. The first approach [38] aims to extract features from process models, and subsequently uses these features to categorize processes as relevant, irrelevant, or

potentially relevant. Whereas, the second approach [39] proposes to use a novel feature of the process models, called Feature-Net (FNet). This approach consists of two phases: indexing and querying. In the first phase, each process graph $\{G1, G2, \dots\}$ in the collection of process models, is indexed. Subsequently, each indexed process graph is split into basic features $\{PF1, PF2, \dots\}$ to construct an FNet, which is used for computing similarity between a query-source pair.

9 CONCLUSION

In this paper, we argue that the use of full-length text descriptions may impede the efficiency of matching techniques, particularly when the textual descriptions are very long. To mitigate this, we promote the use of summary textual descriptions as an alternative to the full-length textual descriptions. To this end, we have thoroughly investigated the trade-off between efficiency and effectiveness between full-length textual descriptions and our proposed alternative of summary textual descriptions.

We have generated a corpus of full-length textual descriptions of 669 process models and use them to generate 20 corpora of summary descriptions. The full-length textual descriptions corpus is generated from the process models in JSON format using an established tool for generating textual descriptions, called NLGS. Whereas, the 20 summary corpora are generated by using diverse text summarization techniques, at five different compression rates, 10 %, 20 %, 30 %, 40 %, and 50 %. The techniques include a word-based summarization technique, TF-IDF, an established graph-based summarization technique, LexRank, a state-of-the-art clustering technique, Lingo, and another state-of-the-art deep learning based technique, K-means clustering with skip-thought embeddings. To establish that the generated summary corpora are substantially different from each other, we have used two text-matching techniques, N-gram overlap, and LCS. For that, we have first generated 1 120 pairs of summary descriptions and subsequently used the two text-matching techniques to compute the similarity between each pair. The results show that the summaries generated by the two summarization techniques are significantly different from one another, hence, the choice of summarization technique is a non-trivial task. Therefore, we conducted process matching experiments to compare the performance of the summary descriptions generated by the four summarization techniques.

The process matching experiments are performed using 56 full-length textual descriptions as queries and 669 full-length textual descriptions as a source. For matching, we have used seven text-matching techniques: Unigram, Bigram, Trigram, Global Alignment, Longest Common Subsequence, Local Alignment, and Vector Space Model. Furthermore, we have performed experiments using 20 sets of 56 summarized query descriptions generated by the four summarization techniques at five compression rates. Our results show that the use of summary description queries, generated by Lingo at a compression rate of 50 %, can achieve a comparable or higher efficiency as well as effectiveness than the full-length descriptions. In the

future, we aim to use other summarization techniques and study their impact on process matching.

REFERENCES

[1] Aysolmaz, B.—Leopold, H.—Reijers, H. A.—Demirörs, O.: A Semi-Automated Approach for Generating Natural Language Requirements Documents Based on Business Process Models. Information and Software Technology, Vol. 93, 2018, pp. 14–29, doi: 10.1016/j.infsof.2017.08.009.

[2] Dumas, M.—La Rosa, M.—Mendling, J.—Reijers, H.: Fundamentals of Business Process Management. Second Edition. Springer, 2018.

[3] Chakraborty, S.—Sarker, S.—Sarker, S.: An Exploration into the Process of Requirements Elicitation: A Grounded Approach. Journal of the Association for Information Systems, Vol. 11, 2010, No. 4, Art. No. 1, doi: 10.17705/1jais.00225.

[4] Sànchez-Ferreres, J.—van der Aa, H.—Carmona, J.—Padró, L.: Aligning Textual and Model-Based Process Descriptions. Data and Knowledge Engineering, Vol. 118, 2018, No. 1, pp. 25–40, doi: 10.1016/j.datak.2018.09.001.

[5] Leopold, H.—Mendling, J.—Polyvyanyy, A.: Supporting Process Model Validation Through Natural Language Generation. IEEE Transactions on Software Engineering, Vol. 40, 2014, No. 8, pp. 818–840, doi: 10.1109/TSE.2014.2327044.

[6] van der Aa, H.—Leopold, H.—Reijers, H. A.: Comparing Textual Descriptions to Process Models – The Automatic Detection of Inconsistencies. Information Systems, Vol. 64, 2017, pp. 447–460, doi: 10.1016/j.is.2016.07.010.

[7] Rana, M.—Shahzad, K.—Adeel Nawab, R. M.—Leopold, H.—Babar, U.: A Textual Description Based Approach to Process Matching. In: Horkoff, J., Jeusfeld, M., Persson, A. (Eds.): The Practice of Enterprise Modeling (PoEM 2016). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 267, 2016, pp. 194–208, doi: 10.1007/978-3-319-48393-1_14.

[8] Leopold, H.—van der Aa, H.—Pittke, F.—Raffel, M.—Mendling, J.—Reijers, H. A.: Searching Textual and Model-Based Process Descriptions Based on a Unified Data Format. Software and Systems Modeling, Vol. 18, 2019, No. 2, pp. 1179–1194, doi: 10.1007/s10270-017-0649-y.

[9] Leopold, H.—van der Aa, H.—Pittke, F.—Raffel, M.—Mendling, J.—Reijers, H. A.: Integrating Textual and Model-Based Process Descriptions for Comprehensive Process Search. In: Schmidt, R., Guédria, W., Bider, I., Guerreiro, S. (Eds.): Enterprise, Business-Process and Information Systems Modeling (BPMDS 2016, EMMSAD 2016). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 248, 2016, pp. 51–65. doi: 10.1007/978-3-319-39429-9_4.

[10] Erkan, G.—Radev, D. R.: LexRank: Graph-Based Lexical Centrality as Salience in Text Summarization. Journal of Artificial Intelligence Research, Vol. 22, 2004, pp. 457–479, doi: 10.1613/jair.1523.

[11] Shahzad, K.—Shareef, K.—Ali, R. F.—Adeel Nawab, R. M.—Abid, A.: Generating Process Model Collection with Divers Label and Structural Features.

2016 Sixth International Conference on Innovative Computing Technology (INTECH 2016), IEEE, 2016, pp. 644–649, doi: 10.1109/intech.2016.7845083.

[12] Snoeck, M.—Moreno-Montes de Oca, I.—Haegemans, T.—Scheldeman, B.—Hoste, T.: Testing a Selection of BPMN Tools for Their Support of Modelling Guidelines. In: Ralyté, J., España, S., Pastor, Ó. (Eds.): The Practice of Enterprise Modeling (PoEM 2015). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 235, 2015, pp. 111–125, doi: 10.1007/978-3-319-25897-3_8.

[13] Shahzad, K.—Adeel Nawab, R. M.—Abid, A.—Sharif, K.—Ali, F.—Aslam, F.—Mazhar, A.: A Process Model Collection and Gold Standard Correspondences for Process Model Matching. IEEE Access, Vol. 7, 2019, pp. 30708–30723, doi: 10.1109/access.2019.2900174.

[14] Antunes, G.—Bakhshandeh, M.—Borbinha, J.—Cardoso, J.—Dadashnia, S.—Di Francescomarino, Ch.—Dragoni, M.—Fettke, P.—Gal, A.—Ghidini, C. et al.: The Process Model Matching Contest 2015. In: Kolb, J., Leopold, H., Mendling, J. (Eds.): Enterprise Modelling and Information Systems Architectures (EMISA 2015). Gesellschaft für Informatik e.V., Bonn, Lecture Notes in Informatics (LNI), Vol. P248, 2015, pp. 127–155.

[15] Mendling, J.—Reijers, H. A.—van der Aalst, W. M. P.: Seven Process Modeling Guidelines (7PMG). Information and Software Technology, Vol. 52, 2010, No. 2, pp. 127–136, doi: 10.1016/j.infsof.2009.08.004.

[16] Leopold, H.—Mendling, J.—Polyvyanyy, A.: Generating Natural Language Texts from Business Process Models. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (Eds.): Advanced Information Systems Engineering (CAiSE 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7328, 2012, pp. 64–79, doi: 10.1007/978-3-642-31095-9_5.

[17] Dijkman, R.—Dumas, M.—Van Dongen, B.—Käärik, R.—Mendling, J.: Similarity of Business Process Models: Metrics and Evaluation. Information Systems, Vol. 36, 2011, No. 2, pp. 498–516, doi: 10.1016/j.is.2010.09.006.

[18] Adeel Nawab, R. M.: Mono-Lingual Paraphrased Text Reuse and Plagiarism Detection. Ph.D. thesis, University of Sheffield, 2012.

[19] Baeza-Yates, R.—Ribeiro-Neto, B.: Modern Information Retrieval. ACM Press New York, 1999.

[20] Mendling, J.: Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness. 1st Edition. Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 6, 2008, doi: 10.1007/978-3-540-89224-3.

[21] Bosanac, S.—Štefanec, V.: N-Gram Overlap in Automatic Detection of Document Derivation. 3rd International Conference "The Future of Information Sciences: INFuture2011 – Information Sciences and e-Society", Zagreb, 2011, pp. 373–382.

[22] Salton, G.—Wong, A.—Yang, C.-S.: A Vector Space Model for Automatic Indexing. Communications of the ACM, Vol. 18, 1975, No. 11, pp. 613–620, doi: 10.1145/361219.361220.

[23] Barrón-Cedeño, A.—Rosso, P.—Benedí, J.-M.: Reducing the Plagiarism Detection Search Space on the Basis of the Kullback-Leibler Distance. In: Gel-

bukh, A. (Ed.): Computational Linguistics and Intelligent Text Processing (CICLing 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5449, pp. 523–534, doi: 10.1007/978-3-642-00382-0_42.

[24] Smith, T. F.—Waterman, M. S.: Identification of Common Molecular Subsequences. Journal of Molecular Biology, Vol. 147, 1981, No. 1, pp. 195–197, doi: 10.1016/0022-2836(81)90087-5.

[25] Needleman, S. B.—Wunsch, Ch. D.: A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. Journal of Molecular Biology, Vol. 48, 1970, No. 3, pp. 443–453, doi: 10.1016/0022-2836(70)90057-4.

[26] Kiros, R.—Zhu, Y.—Salakhutdinov, R. R.—Zemel, R.—Urtasun, R.—Torralba, A.—Fidler, S.: Skip-Thought Vectors. In: Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., Garnett, R. (Eds.): Advances in Neural Information Processing Systems 28 (NIPS 2015). Springer, 2015, pp. 3294–3302.

[27] Weidlich, M.—Dijkman, R.—Mendling, J.: The ICoP Framework: Identification of Correspondences Between Process Models. In: Pernici, B. (Ed.): Advanced Information Systems Engineering (CAiSE 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6051, 2010, pp. 483–498, doi: 10.1007/978-3-642-13094-6_37.

[28] Jabeen, F.—Leopold, H.—Reijers, H. A.: How to Make Process Model Matching Work Better? An Analysis of Current Similarity Measures. In: Abramowicz, W. (Ed.): Business Information Systems (BIS 2017). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 288, 2017, pp. 181–193, doi: 10.1007/978-3-319-59336-4_13.

[29] Dijkman, R.—Dumas, M.—García-Bañuelos, L.: Graph Matching Algorithms for Business Process Model Similarity Search. In: Dayal, U., Eder, J., Koehler, J., Reijers, H. A. (Eds.): Business Process Management (BPM 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5701, 2009, pp. 48–63, doi: 10.1007/978-3-642-03848-8_5.

[30] Wordnet. https://wordnet.princeton.edu/, accessed: 2018-06-23.

[31] Skipthought Source. https://github.com/ryankiros/skip-thoughts/blob/master/skipthoughts.py, accessed: 2019-05-25.

[32] Osiński, S.—Stefanowski, J.—Weiss, D.: Lingo: Search Results Clustering Algorithm Based on Singular Value Decomposition. In: Kłopotek, M. A., Wierzchoń, S. T., Trojanowski, K. (Eds.): Intelligent Information Processing and Web Mining. Springer, Berlin, Heidelberg, Advances in Soft Computing, Vol. 25, 2004, pp. 359–368, doi: 10.1007/978-3-540-39985-8_37.

[33] Carrot2 Project Kernel Description. https://project.carrot2.org/download-workbench-win32-64bit.htm, accessed: 2019-05-25.

[34] Signavio Process Modeling Tool. https://www.signavio.com/, accessed: 2019-05-25.

[35] Baumann, M.—Baumann, M. H.—Jablonski, S.: On Behavioral Process Model Similarity Matching: A Centroid-Based Approach. In: Mayr, H. C., Pinzger, M. (Eds.): Informatik 2016. Gesellschaft für Informatik e.V., Bonn, Lecture Notes in Informatics (LNI), Vol. P259, 2016, pp. 731–732.

[36] Meilicke, Ch.—Leopold, H.—Kuss, E.—Stuckenschmidt, H.—Reijers, H. A.: Overcoming Individual Process Model Matcher Weaknesses Using Ensemble Matching. Decision Support Systems, Vol. 100, 2017, pp. 15–26, doi: 10.1016/j.dss.2017.02.013.

[37] Padmakumar, A.—Saran, A.: Unsupervised Text Summarization Using Sentence Embeddings. Technical Report, University of Texas at Austin, 2016, pp. 1–9.

[38] Yan, Z.—Dijkman, R.—Grefen, P.: Fast Business Process Similarity Search with Feature-Based Similarity Estimation. In: Meersman, R., Dillon, T., Herrero, P. (Eds.): On the Move to Meaningful Internet Systems: OTM 2010. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6426, 2010, pp. 60–77, doi: 10.1007/978-3-642-16934-2_8.

[39] Yan, Z.—Dijkman, R.—Grefen, P.: FNet: An Index for Advanced Business Process Querying. In: Barros, A., Gal, A., Kindler, E. (Eds.): Business Process Management (BPM 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7481, 2012, pp. 246–261, doi: 10.1007/978-3-642-32885-5_20.

Syed Irtaza Muzaffar is a Ph.D. scholar at the Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore. He has M. Phil. in computer science from PUCIT. Currently, he is Lecturer (Visiting) at the University of the Punjab. He has three years of teaching and development experience.

Khurram Shahzad is Assistant Professor at the Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore. He has his Masters and Ph.D. from KTH – Royal Institute of Technology, Stockholm. He is associated with Information Systems Groups at the Technical University Eindhoven, Eindhoven, and the University of Fribourg, Fribourg. He has published more than 35 papers in international conferences and journals.

Faisal Aslam obtained his Ph.D. from the University of Freiburg, Germany and worked as post-doc at the TU Delft, the Netherlands. He was also a research fellow at the Lund University, Sweden. Currently, he is Assistant Professor at the University of the Punjab. He has seven years of his post-Ph.D. experience. He has published papers in reputed journals and conferences.

Madiha Khalid is a faculty member and Ph.D. candidate at the Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan. She holds her M.Sc. and B.Sc. in computer science. She has published several papers in reputed journals.

Kamran Malik is Assistant Professor at the Punjab University College of Information Technology (PUCIT), University of the Punjab, Lahore, Pakistan. He has more than 15 years of teaching and development experience. He holds his Ph.D. degree in computer science and he has authored 1 US patent and published 25 papers in reputed journals and conferences. His research interests include natural language processing, machine learning, and data science. He has provided consultancy to many multinational firms on natural language processing, machine learning, and data science.

Computing and Informatics, Vol. 38, 2019, 883–916, doi: 10.31577/cai_2019_4_883

AN EFFECTIVE METAHEURISTIC FOR MULTIPLE TRAVELING REPAIRMAN PROBLEM WITH DISTANCE CONSTRAINTS

Ha-Bang Ban

School of Information and Communication Technology
Hanoi University of Science and Technology
Hanoi, Vietnam
&
FPT University, Hanoi, Vietnam
e-mail: BangBH@soict.hust.edu.vn

Duc-Nghia Nguyen

School of Information and Communication Technology
Hanoi University of Science and Technology
Hanoi, Vietnam
e-mail: NghiaND@soict.hust.edu.vn

Kien Nguyen

Graduate School of Science and Engineering, Chiba University, Japan
e-mail: nguyen@chiba-u.jp

Abstract. Multiple Traveling Repairman Problem with Distance Constraints (MTRPD) is an extension of the NP-hard Multiple Traveling Repairman Problem. In MTRPD, a fleet of identical vehicles is dispatched to serve a set of customers with the following constraints. First, each vehicle's travel distance is limited by a threshold. Second, each customer must be visited exactly once. Our goal is to find the visiting order that minimizes the sum of waiting times. To solve MTRPD we propose to combine the Insertion Heuristic (IH), Variable Neighborhood Search (VNS), and Tabu Search (TS) algorithms into an effective two-phase metaheuris-

tic that includes a construction phase and an improvement phase. In the former phase, IH is used to create an initial solution. In the latter phase, we use VNS to generate various neighborhoods, while TS is employed to mainly prohibit from getting trapped into cycles. By doing so, our algorithm can support the search to escape local optima. In addition, we introduce a novel neighborhoods' structure and a constant time operation which are efficient for calculating the cost of each neighboring solution. To show the efficiency of our proposed metaheuristic algorithm, we extensively experiment on benchmark instances. The results show that our algorithm can find the optimal solutions for all instances with up to 50 vertices in a fraction of seconds. Moreover, for instances from 60 to 80 vertices, almost all found solutions fall into the range of 0.9 %–1.1 % of the optimal solutions' lower bounds in a reasonable duration. For instances with a larger number of vertices, the algorithm reaches good-quality solutions fast. Moreover, in a comparison to the state-of-the-art metaheuristics, our proposed algorithm can find better solutions.

Keywords: Traveling repairmen problem, distance constraints, insertion heuristic, tabu search, variable neighborhood search

1 INTRODUCTION

The Traveling Repairman Problem (TRP), which is also known as the Minimum Latency Problem (MLP) or the Deliveryman Problem (DMP), has been studied in the number of previous works [1, 2, 3, 4, 5, 6, 10, 11, 18]. The problem arises when repairmen or servers have to accommodate a set of requests to minimize the total or average waiting times [1, 2, 8, 10]. A direct generalization of the TRP is the Multiple Traveling Repairman Problem (MTRP) that considers multiple vehicles or travelers. Similar to TRP, there are several prior studies in the literature for MTRP [22, 9, 23, 28, 29]. Applications of the MTRP can be found in routing Pizza deliverymen or scheduling machines to minimize mean flow time for jobs [17]. In this paper, we study an extension of MTRP, namely the Multiple Traveling Repairmen Problem with Distance Constraints (MTRPD), which involves distance constraints. In MTRPD, the route length or maximum duration of each vehicle cannot exceed a predetermined limit (MD). This type of constraint usually stems from regulations on working hours for workers. Other examples of vehicle routing models that incorporate the distance constraint can be found in [30]. In MTRPD, we consider k vehicles at a main depot s and n customers. The goal is to find a tour such that each vertex is visited exactly once, the distance constraint is respected and the total waiting time of all customers is minimized.

MTRPD is at least as hard as TRP and MTRP. MPTRPD, which is also NP-hard problem, can be formulated as follows.

Given a complete graph K_n with the vertex set $V = \{1, 2, \ldots, n\}$, a symmetric distance matrix $C = \{c(i, j) \mid i, j = 1, 2, \ldots, n\}$, where $c(i, j)$ is the distance between two vertices i and j, and a predetermined limit L. Let $R = (1, 2, \ldots, k)$

be a set of k vehicles which begin at the main depot v_1. Suppose that the tour $T = (R_1, \ldots, R_l, \ldots, R_k)$ is a set of obtained routes from k vehicles. Let $R_l = (v_1, \ldots, v_h, \ldots, v_m)$ $(1 < m \leqslant n)$ be a route of vehicle l $(l \in R)$. $P(v_1, v_h)$ is the path from v_1 to v_h on the route R_l and $l(P(v_1, v_h))$ is its length. The waiting time of a vertex v_h $(1 < h \leqslant m)$ on R_l is the length of the path from starting vertex v_1 to v_h:

$$l(P(v_1, v_h)) = \sum_{i=1}^{h-1} c(v_i, v_{i+1}).$$

The waiting time of R_l is defined as the sum of waiting times of all vertices in this route. It must satisfy the below constraint:

$$W(R_l) = \sum_{h=2}^{m} l(P(v_1, v_h)),$$

$$L(R_l) = \sum_{i=1}^{m-1} c(v_i, v_{i+1}) \leqslant MD.$$

The total waiting time of T is the sum of all the vertices' waiting times:

$$W(T) = \sum_{l=1}^{k} W(R_l).$$

MTRPD asks for a k-route, which starts at a given vertex v_1, visits each vertex in the graph once exactly with the total waiting time of all vertices being minimized. Like other NP-hard problems, there are three main approaches to solve MTRPD:

1. exact algorithms,
2. approximation algorithms, and
3. heuristic algorithms.

The first approach guarantees to find the optimal solution that takes exponential time in the worst case. However, the exact algorithm only solves with up to 50 vertices [25]. In the second approach, we denote an approximation algorithm as p-approximation when the algorithm finds the solution at most p times worse than the optimal one. Here p is the approximation ratio, which has a constant value. Up to date, the best approximation ratio is 16.994 for the MTRP [22], which is still far from the optimal solution. In the third approach, the proposed heuristic algorithms perform well in practice and their performance is validated on an experimental benchmark of interesting instances. The metaheuristic algorithm also falls in the third approach.

Research on the MTRPD has not studied much and only one meta-heuristic approach for this problem has been proposed in [9]. Ban's algorithm in [9] is mainly based on the principles of the Variable Neighborhood Decent (VND). However, Ban's

algorithms might become trapped into cycles. That means they return to the points previously explored in the solution space. Consequently, the algorithms can get stuck in local optima. In this article, we investigate the global structure of the MTRPD solution space. Based on the investigation, a meta-heuristic algorithm that combines the Tabu search (TS) and Variable Neighborhood Search (VNS) is proposed. In the algorithm, TS is used to avoid getting trapped into cycles. Therefore, it supports the search to escape from local optima. In a cooperative way, VNS is employed to generate various neighborhoods for the TS. Moreover, we also introduce a novel neighborhoods' structure for VNS and present a constant time operation for calculating the cost of each neighboring solution. Therefore, the extension of explored part of the solution space obtained by using various neighborhoods, which can increase chances of finding better solution, is not time-consuming in our algorithm. Extensive computational experiments on benchmark instances show that the proposed algorithm is able to find the optimal solutions for all instances with up to 50 vertices in a fraction of seconds. Moreover, almost all found solutions for instances from 60 to 80 vertices fall into the range of 0.9 %–1.1 % of the lower bounds of the optimal solutions at reasonable amount of time. For larger instances, our algorithm obtains good-quality solutions fast and the new best solutions are found in comparison with the state-of-the-art metaheuristics.

The rest of this article is organized as follows. Section 2 introduces the global structure of the solution space of MTRPD. Section 3 presents the proposed algorithm. Section 4 contains the evaluation. Finally, Section 5 concludes the article.

2 INVESTIGATION OF MTRPD SOLUTION SPACE

The structure of the MTRPD solution takes an important part in improving a suitable algorithm to solve the problem. However, to the best of our knowledge, there has not been a previous work that would solve the global structure of the MTRPD problem. That motivates us to investigate the global structure of the MTRPD solution space.

In an intuitive way, the distance between two tours T_1 and T_2 of the problem is defined as the minimum number of transformations from T_1 to T_2, denoted by $d(T_1, T_2)$. Since no polynomial method for computing $d(T_1, T_2)$ has been known, we define $d(T_1, T_2)$ to be n minus the number of vertices which have the same position in both T_1 and T_2. We see that this distance approximates the number of 2-opt operations (2-opt is a local search described in Section 3) required to transform one tour into another, to within a factor of two. Therefore, $d(T_1, T_2)$ is the good measure of proximity between solutions produced by 2-opt.

We have selected two instances (d15112-x and pr1002-x) from the dataset in [25] and implemented with 2-opt. The selection reason is that the optimal solutions of both instances are provided in [25]. Running each instance with 2-opt, we obtain locally optimal solutions. The larger the number of 2-opt runs, the better the visualization of the MTRPD solution space. The pilot experiment shows that the value

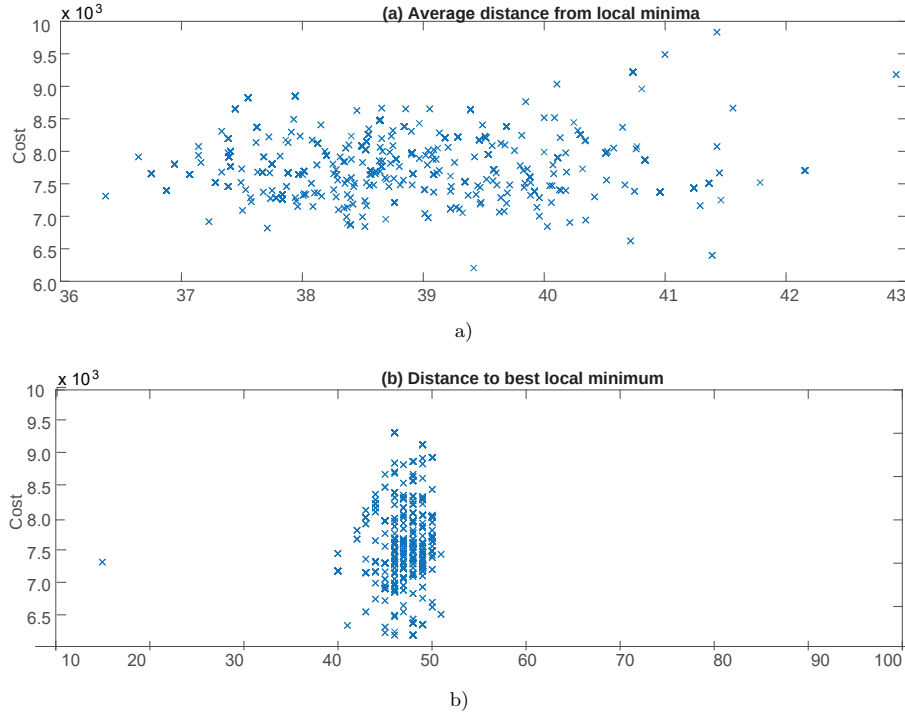


Figure 1. 2 000 random 2-opt local minima for d15112-x. Tour cost (vertical axis) is plotted against a) mean distance to the 1 999 other local minima and b) distance to the global minimum.

of 2 000 is good enough for the investigation. We run 2-opt 2 000 times to produce 2 000 locally optimal solutions. Then, for each of those solutions, we compute the average distance to the other 1999 solutions, measured by the distance metric d. The results for d15112-x and pr1002-x are presented in Figures 1 and 2, respectively.

In Figures 1 a) and 2 a), we can realize a clear correlation as follows. The optimal minimum appears to be central to all other local minima. Moreover, indeed, a prominent valley structure can be said to govern the set of locally minimum solutions. We can gain further insights from Figures 1 b) and 2 b), which plot the costs of the same 2 000 local minima against their distances from the optimal minimum solution found. It indicates that the average distance between two random solutions is just under $(n-2)$. The experiment shows that the MTRPD solution space exhibits a global convex (i.e., the so-called big valley structure in Figure 3). That means the set of local optima appears convex with one central global optimum.

As mentioned earlier, MTRPD has shown to be NP-Hard because it is a generalization of TRP, which is NP-hard [10, 18]. Therefore, a metaheuristic needs to be developed to provide near-optimal solutions within a short computation time

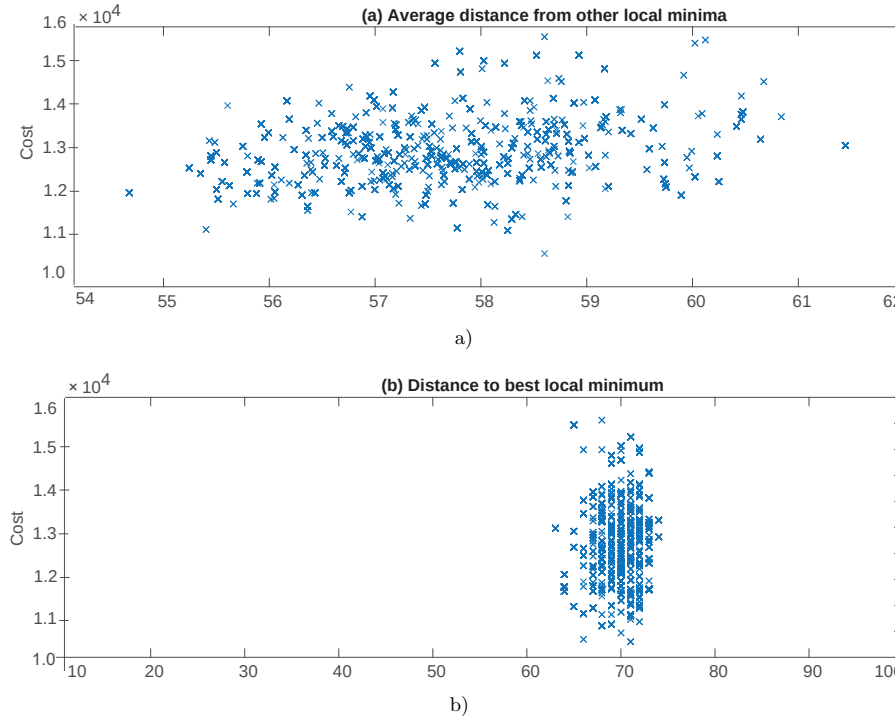


Figure 2. 2 000 random 2-opt local minima for pr1002-x. Tour cost (vertical axis) is plotted against a) mean distance to the 1 999 other local minima and b) distance to the global minimum.

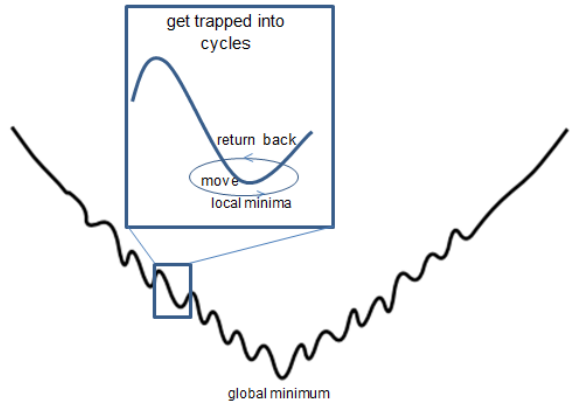


Figure 3. Intuitive scheme of the "big valley" solution space structure

for large instance sizes. Moreover, the big valley structure suggests the idea of the hybrid approach that combines the TS and VNS algorithms. First, in the valley structure, the best elite solutions created by the VNS dispersed over it. Second, TS is perfectly attracted to big valley area. Even though the initial solution was set far from the valley, TS still can prevent from getting trapped into cycles to drive the search to the big valley. The above observations indicate that the combination between TS and VNS is suitable for finding good solutions inside the big valley.

3 OUR METAHEURISTIC APPROACH

We propose the efficient and straightforward metaheuristic algorithm that brings together the components of IH, TS [19], and VNS [27]. The proposed algorithm includes two phases:

1. IH in the construction phase and
2. VNS and TS in the improvement phase.

The two phases could be divided into five detailed steps, as shown in Algorithm 1. In Step 1, the algorithm starts with an initial solution obtained from IH. The following four main steps are repeated until a stop condition is met. In Step 2, we introduce a novel neighborhoods' structure in VNS [27]. Moreover, in order to avoid tabu move, tabu lists are used. The idea of voiding possibility repeatedly in TS [19] is to make tabu lists of the recent types of moves in the space solution, and prohibit reversing these moves. The move here is a transition from one solution to another. In Step 3, a list of promising solutions is built up, and the list serves as input for Step 4. The step aims at exploiting the current solution space. To explore the entire solution space, a diversification phase is added in Step 5. Further in this section we describe the five steps of our algorithm in more details.

Step 1: We use the insertion heuristic which is given in Algorithm 2 for finding an initial solution. Consider a partial tour, and define the set $\overline{V}$ as the set of all non-visited nodes, $\overline{V} \subseteq V$. To improve the partial tour, a node from $\overline{V}$ should be added. This process requires two decisions: which vertex to insert and where to place it in the tour. We use two insertion schemes to keep the balance between pure greediness and overall layout of the tour. The major difference between the two is the order in which the vertices are inserted.

Cheapest insertion: Among all vertices not inserted so far, choose a vertex whose insertion causes the lowest increase in the cost of the tour. The idea behind this strategy is undoubtedly pure greediness.

Farthest insertion: Insert the vertex whose minimal distance to a tour vertex is maximal. The idea behind this strategy is to fix the overall layout of the tour early in the insertion process.

Several main steps in IH-procedure is repeated until a feasible solution is found or a stop condition is met. If any feasible solution is found, it is considered as

Algorithm 1 Our VNS + TS Metaheuristic Algorithm

Input: v_1, K_n, k are a starting vertex, the complete graph, and the number of vehicles, respectively.

Output: the best solution T^*.

```
Step 1 (Generate an initial solution):
T ← IH-Procedure(v_1, V, k); {initiate the best solution}
T* ← T {LT is the list of promising solutions}
LT ← ∅
while stop criteria not met do
  Step 2 (VNS):
  for i : 1 → 10 do
    T' ← arg min N_i(T); {local search}
    if ((W(T') < W(T) and T' is not tabu) or (W(T') < W(T*))) then
      T ← T'
      i ← 1
      update tabu lists;
      if (W(T') < W(T*)) and (T' must be a feasible solution) then
        T* ← T';
      end if
    else
      i++
    end if
  end for
  Step 3 (Built up promising solutions list L_T):
  if W(T) < (1 + ST)W(T*) then
    LT ← LT ∪ T;
  end if
  if (| LT |== sLT) then
    go to Step 2;
  end if
  Step 4 (Implement Intensification):
  for j : 1 → sLT do
    perform VNS as in Step 2 without tabu list with an element of LT as start
    solution;
  end for
  Step 5 (Implement Diversification):
  Clear all tabu lists and update attribute matrix M;
  select a random tour T in LT;
  T ← shaking-procedure(T);
end while
return T*;
```

Algorithm 2 IH-Procedure(v_1, K_n, k)

Input: v_1, K_n, k are a starting vertex, the complete graph, and the number of vehicles, respectively.

Output: An initial solution T. {LIT is the list of infeasible tours}

```
1: LT = φ;
2: T = v1;
3: while stop criteria not met do
4:   for (l = 1; l < k; l++) do
5:     R_l = R_l ∪ v1; {The l^th route of the tour T starts at a main depot v1}
6:   end for {L is the list of visited vertices in K_n}
7:   L = φ;
8:   while |T| < n do
9:     l = random(k); {Choose a route randomly in k routes}
10:    rd = random(2); {Choose an insertion scheme randomly}
11:    if rd == 1 then
12:      Arbitrary select a vertex v that is not yet in the partial route and
         an inserted position j < |R_l| at time t_j so that the cost of R'_l (R'_l =
         Insert(R_l, j, v)) is minimal; {Cheapest Insertion}
13:    else
14:      Arbitrary select a vertex v that is not yet in the partial tour and an in-
         serted position j < |R_l| at time t_j so that c(v_j, v, j) is minimal and the
         cost of R'_l (R'_l = Insert(R_l, j, v)) is maximal; {Farthest Insertion}
15:    end if
16:    R_l ← R'_l
17:  end while
18:  if T is a feasible solution then
19:    return T;
20:  else
21:    LIT = LIT ∪ {T};
22:  end if
23:  if |LIT| > n − 1 then
24:    choose a tour with the minimum cost in LIT;
25:    exit();
26:  end if
27: end while
```

the initial solution. Conversely, it is added into the list LIT that is used to store all infeasible tours. Since the size of LIT is n, we choose a tour with the minimum cost in LIT as the initial solution.

Step 2: In this step, ten neighborhoods investigated are divided into two categories: intro-route, and intra-route. Intro-route is used as a post-optimizer on single vehicle routes. It includes remove-insert, swap-adjacent, swap, move-up(down), move-forward(backward)-k-vertices [21]. Meanwhile, solution improvements can

Algorithm 3 Shaking(T, M, pos)

Input: T, M, pos are the tour, attribute matrix, and the number of swap, respectively.

Output: a new route $T^{'}$.

```
Select randomly a route R_l in T;
T = Shaking-intro-route(R_l, M, l, pos).
Select randomly two routes R_l and R_h in T;
T = Shaking-intra-routes(R_l, R_h, pos).
return T;
```

Algorithm 4 Shaking-intro-route(R_l, M, l, pos)

Input: R_l, M, l, pos are the $l-th$ route, attribute matrix, and the number of times an edge is present in an element of the promising solutions list, the number of swap, respectively.

Output: a new solution T.

```
while (pos > 0) do
  {select i, j from [1, n] at random}
  i ⟵ Random(1, n);
  j ⟵ Random(1, n);
  if (i ≠ j) then
    if (edge(R_l[i], R_l[j]) and edge(R_l[i], R_l[j+1]) are not in M more than l times)
    then
      Insert R_l[i] between R_l[j] and R_l[j + 1];
      pos ⟵ pos − 1;
    end if
  end if
end while
update R_l in T;
return T;
```

Algorithm 5 Shaking-intra-routes(R_l, R_h, pos)

Input: R_l, R_h, pos are the l^{th}, h^{th} route, the number of vehicles and the number of swap, respectively.

Output: a new solution T.

```
while (pos > 0) do
  select i^th and j^th positions from R_l and R_h at random, respectively;
  swap R_l[i] between R_h[j];
  pos = pos − 1;
end while
update T;
return T;
```

be obtained by moving vertices belonging to two or more different routes in intra-route. In this work, we introduce new neighborhoods in intra-route such as swap-2-route, and insert-2-route. For a given current solution T, neighborhood explores the neighboring solution space set $N(T)$ of T iteratively and tries to replace T by the best solution $T' \in N(T)$. The main operation in exploring the neighborhood is the calculation of the cost of a neighboring solution. In a straightforward implementation in the worst case, this operation requires $Tsol = O(n)$. In this paper, by using the known cost of the current solution, we show that this operation can be done in constant time for some considered neighborhoods. Thus, we speed up the running time of exploring these neighborhoods. Now, let $T = (R_1, R_2, \ldots, R_k)$ be a tour with k routes, we then introduce a novel neighborhoods' structure and complexity of its exploration.

For Intro-route: Intro-route is used to optimize on a single route. Assume that R and m ($m < n$) are a route and its length, respectively. We then introduce eight neighborhoods' structure in turn.

Remove-insert neighborhood considers each vertex v_i in the route at the end of the route. This neighborhood of R is defined as a set $N_1(R) = \{R_i = (v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_m, v_i) : i = 2, 3, \ldots, m-1\}$. Obviously, the size of $N_1(R)$ is $O(m)$.

Property 1. The time complexity of exploring $N_1(R)$ is $O(m^2)$.

Proof. Let us consider an initial solution $R = v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_m$. The neighborhood generates a neighboring solution $R_i = v_1, v_2, \ldots, v_{i-1}, v_{i+1}, \ldots, v_m, v_i$. The costs of R and R_i are calculated as follows:

$$\begin{aligned} L(R) = {} & (m-1)c(v_1, v_2) + \ldots + (m-i+1)c(v_{i-1}, v_i) + (m-i)c(v_i, v_{i+1}) \\ & + (m-i-1)c(v_{i+1}, v_{i+2}) + \ldots + c(v_{m-1}, v_m) \end{aligned} \tag{1}$$

and

$$\begin{aligned} L(R_i) = {} & (m-1)c(v_1, v_2) + \ldots + (m-i+1)c(v_{i-1}, v_{i+1}) + (m-i)c(v_{i+1}, v_{i+2}) \\ & + \ldots + 2c(v_{m-1}, v_m) + c(v_m, v_i). \end{aligned}$$

Thus,

$$\begin{aligned} L(R_i) = {} & L(R) - \sum_{k=i-1}^{m-1} (m-k)c(v_k, v_{k+1}) + (m-i+1)c(v_{i-1}, v_{i+1}) \\ & + \sum_{k=i+1}^{m-1} (m-k+1)c(v_k, v_{k+1}) + c(v_m, v_i). \end{aligned} \tag{2}$$

It takes $O(m)$ time to calculate the formulation in (2). Therefore, the time complexity of exploring $N_1(R)$ is $O(m^2)$. □

Swap adjacent neighborhood attempts to swap each pair of adjacent vertices in the route. This neighborhood of R is defined as a set $N_2(R) = \{R_i = (v_1, v_2, \ldots, v_{i-2}, v_i, v_{i-1}, v_{i+1}, \ldots, v_m) : i = 3, 4, \ldots, m-1\}$. The size of the neighborhood is $O(m)$.

Property 2. The time complexity of exploring $N_2(R)$ is $O(m)$.

Proof. The initial tour R and $L(R)$ are the same as in (1). The neighborhood generates a neighboring tour $R_i = v_1, v_2, \ldots, v_{i-2}, v_i, v_{i-1}, v_{i+1}, \ldots, v_m$. The latency of R_i is calculated as follows:

$$\begin{aligned} L(R_i) = {} & (n-1)c(v_1, v_2) + \ldots + (m-i+2)c(v_{i-2}, v_i) + (m-i+1)c(v_i, v_{i-1}) \\ & + (m-i)c(v_{i-1}, v_{i+1}) + (m-i-1)c(v_{i+1}, v_{i+2}) + \ldots + c(v_{m-1}, v_n). \end{aligned}$$

We have

$$\begin{aligned} L(R_i) = {} & L(R) - (m-i+2)c(v_{i-2}, v_{i-1}) - (m-i)c(v_i, v_{i+1}) \\ & + (m-i+2)c(v_{i-2}, v_i) + (m-i)c(v_{i-1}, v_{i+1}). \end{aligned} \tag{3}$$

It is obvious that we can calculate $L(R_i)$ by the formulation (3) in $O(1)$ time. Therefore, the complexity of exploring $N_2(R)$ is $O(m)$. □

Swap neighborhood attempts to swap the positions of each pair of vertices in the route. This neighborhood of R is defined as a set $N_3(R) = \{R_{ij} = (v_1, v_2, \ldots, v_{i-1}, v_j, v_{i+1}, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, v_m) : i = 2, 3, \ldots, m-3; j = i+3, \ldots, m\}$. The size of the neighborhood is $O(m^2)$.

Property 3. The complexity of exploring $N_3(R)$ is $O(m^2)$.

Proof. Initially, we have a solution $R = v_1, v_2, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_{j-1}, v_j, v_{j+1}, \ldots, v_m$ $(i+2 < j)$. Swap generates a neighboring solution $R_{ij} = v_1, v_2, \ldots, v_{i-1}, v_j, v_{i+1}, \ldots, v_{j-1}, v_i, v_{j+1}, \ldots, v_m$. The costs of R and R_i are calculated as follows:

$$\begin{aligned} L(R) = {} & (m-1)c(v_1, v_2) + \ldots + (m-i+1)c(v_{i-1}, v_i) + (m-i)c(v_i, v_{i+1}) \\ & + \ldots + (m-j+1)c(v_{j-1}, v_j) + (m-j)c(v_j, v_{j+1}) \\ & + \ldots + c(v_{m-1}, v_m). \end{aligned} \tag{4}$$

$$\begin{aligned} L(R_{ij}) = {} & (m-1)c(v_1, v_2) + \ldots + (m-i+1)c(v_{i-1}, v_j) + (m-i)c(v_j, v_{i+1}) \\ & + \ldots + (m-j+1)c(v_{j-1}, v_i) + (m-j)c(v_i, v_{j+1}) + \ldots + c(v_{m-1}, v_m). \end{aligned}$$

It follows that

$$\begin{aligned} L(R_{ij}) = L(R) &- (m-i+1)c(v_{i-1},v_i) - (m-i)c(v_i,v_{i+1}) \\ &- (m-j+1)c(v_{j-1},v_j) - (m-j)c(v_j,v_{j+1}) + (m-i+1)c(v_{i-1},v_j) \\ &+ (m-i)c(v_j,v_{i+1}) + (m-j+1)c(v_{j-1},v_i) + (m-j)c(v_i,v_{j+1}). \quad (5) \end{aligned}$$

Hence, we can calculate $L(R_{ij})$ by the formulation (5) in $O(1)$ time. Therefore, the complexity of exploring $L(R_{ij})$ is $O(m^2)$. □

Or neighborhood attempts to reallocate three adjacent vertices to another position of the route. This neighborhood of R is defined as a set $N_4(R) = \{R_i = (v_1, v_2, \ldots, v_{i-1}, v_i, v_{j+1}, \ldots, v_k, v_{i+1}, \ldots, v_j, v_{k+1}, \ldots, v_m) : i = 2, 3, \ldots, m-5, j = 4, \ldots, m-3, k = 6, \ldots, m-1\}$. The size of the neighborhood is $O(m^3)$.

Property 4. The complexity of exploring $N_4(R)$ is $O(m^3)$.

Proof. In fact, this neighborhood implements the swap neighborhoods twice (we exchange between v_{j+1} and v_{i+1} and between v_j and v_k) in the route. It takes $O(1)$ time for swap operation. Therefore, calculating $L(R_{ijk})$ is $2 \times O(1)$ time and the complexity of exploring $L(R_{ijk})$ is $O(m^3)$. □

2-opt neighborhood removes each pair of edges from the solution and reconnects the vertices. This neighborhood of T is defined as a set $N_5(T) = \{T_{ij} = (v_1, v_2, \ldots, v_i, v_j, v_{j-1}, \ldots, v_{i+2}, v_{i+1}, v_{j+1}, \ldots, v_m) : i = 1, \ldots, n-4; j = i+4, \ldots, m\}$. The size of the neighborhood is $O(m^2)$.

Property 5. The complexity of exploring $N_5(T)$ is $O(m^3)$.

Proof. The initial tour and $L(T)$ are the same as in (4). The neighborhood generates a neighboring tour $T_{ij} = (v_1, v_2, \ldots, v_i, v_j, v_{j-1}, \ldots, v_{i+2}, v_{i+1}, v_{j+1}, \ldots, v_m)$. The costs of T and T_i are calculated as follows:

$$\begin{aligned} L(T_{ij}) = (m-1)c(v_1,v_2) + \ldots + (m-i)c(v_i,v_j) + (m-i-1)c(v_j,v_{i+2}) \\ + \ldots + (m-j+1)c(v_{j-1},v_{i+1}) + (m-j)c(v_{i+1},v_{j+1}) + \\ + \ldots + (v_{m-1},v_m). \end{aligned}$$

We have

$$L(T_{ij}) = L(T) - (m-i)c(v_i, v_{i+1}) - \sum_{h=1}^{j-i-1}(m-i-h)c(v_{i+h}, v_{i+h+1}) - (m-j)c(v_j, v_{j+1}) + (m-i)c(v_i, v_j) + \sum_{h=1}^{j-i-1}(m-i-h)c(v_{j-h+1}, v_{j-h}) + (m-j)c(v_{i+1}, v_{j+1}). \quad (6)$$

It is obvious that we can calculate $L(T_{ij})$ by the formulation (6) in $O(m)$ time. Therefore, the complexity of exploring $N_5(T)$ is $O(m^3)$. □

Move-forward-k-vertices neighborhood of T is defined as a set $N_6(T) = \{T_{ijk} = (v_1, v_2, \ldots, v_i, v_{i+k+1}, v_{i+k+2}, \ldots, v_j, v_{i+1}, v_{i+2} \ldots, v_{i+k}, v_{j+1}, \ldots, v_m) : i = 1, 2, \ldots, m-k-1; i+k < j \leqslant m\}$ with $k = 2, \ldots, l$. The size of the neighborhood is $O(m^2)$.

Move-backward-k-vertices neighborhood of T is defined as a set $N_7(T) = \{T_{ijk} = (v_1, v_2, \ldots, v_i, v_{i+k+1}, v_{i+k+2}, \ldots, v_{i+1}, v_{i+2} \ldots, v_{i+k}, v_j, v_{j+1}, \ldots, v_m) : i = 1, 2, \ldots, m-k-1; i+k < j \leqslant m\}$ with $k = 2, \ldots, l$. The size of the neighborhood is $O(m^2)$.

Property 6. The complexity of exploring $N_6(R)$ and $N_7(R)$ is $O(m^3)$.

Proof. We prove Property 6 for move-forward-k-vertices and the same argument holds for move-backward-k-vertices. For a tour $R_{ijl} \in N_6(R)$, it can be shown that

$$L(T_{ijk}) = L(T) - (m-i)c(v_i, v_{i+1}) - \sum_{h=i+1}^{j-1}(m-h)c(v_h, v_{h+1}) - (m-i)c(v_i, v_{i+1}) + (m-i)c(v_i, v_{i+k+1}) + \sum_{h=1}^{j-i-k-2}(m-i-h)c(v_{i+k+h}, v_{i+k+h+1}) + (m-j+k+1)c(v_{j-1}, v_j) + (m-j+k)c(v_j, v_{i+1}) + \sum_{h=1}^{k-1}(m-j+k-h)c(v_{i+h}, v_{i+h+1}) + (m-j+1)c(v_{i+k}, v_{j+1}). \quad (7)$$

It is obvious that we can calculate $L(T_{ijk})$ by the formulation (7) in $O(m)$ time. Therefore, the complexity of exploring $N_6(T)$ is $O(m^3)$. □

It is realized that the calculation of a neighboring solution cost by using the known cost of the current solution in (6) and (7) cannot be done in constant

time. As a result, the algorithm spends $O(m^3)$ operations for a full neighborhood search. However, Silva et al. [31] suggest a move evaluation procedure, which only requires $O(1)$ amortized operations since the number of edge exchanges is bounded. In this work, we use their evaluation procedure for 2-opt and move forward(backward)-k-vertices. Therefore, the time complexity of exploring all neighborhoods in the worst case is performed in $O(m^3)$.

For intra-route: Let R_l, R_h, ml, and mh be two different routes and their sizes in T, respectively. Intra-route is used to exchange vertices between two different routes or remove vertices from a route and then insert them to another as follows.

The swap-2-routes neighborhood tries to exchange the positions of each pair of vertices in R_l and R_h in turn. The swap-2-route neighborhood of R_l and R_h is defined as a set $N_8(T) = \{T_i = (R_1, \ldots, R_2, \ldots, R_l = (v_{1l}, v_{2l}, \ldots, v_{ih}, v_{il+1}, \ldots, v_{ml}), \ldots, R_h = (v_{1h}, v_{2h}, \ldots, v_{il}, v_{ih+1}, \ldots, v_{mh}), \ldots, R_k) : il = 2, 3, \ldots, ml - 1, ih = 2, 3, \ldots, mh - 1\}$. The size of the neighborhood is $O(ml \times mh)$.

Property 7. The complexity of exploring $N_8(T)$ is $O(ml \times mh)$.

Proof. We have an initial tour T and its two routes are $R_l = (v_{1l}, v_{2l}, \ldots, v_{il}, v_{il+1}, v_{il+2} \ldots, v_{ml})$ and $R_h = (v_{1h}, v_{2h}, \ldots, v_{ih}, v_{ih+1}, v_{ih+2}, \ldots, v_{hl})$. The neighborhood generates a neighboring tour $T_i = (R_1, \ldots, R_2, \ldots, R_l^{'} = (v_{1l}, v_{2l}, \ldots, v_{ih}, v_{il+1}, \ldots, v_{ml}), \ldots, R_h^{'} = (v_{1h}, v_{2h}, \ldots, v_{il}, v_{ih+1}, \ldots, v_{mh}), \ldots, R_k)$. The costs of R_l, and R_h are calculated as follows:

$$\begin{aligned}
L(R_l) &= (ml-1)c(v_{1l}, v_{2l}) + \ldots + (ml-i+1)c(v_{il-1}, v_{il}) + (ml-i)c(v_{il}, v_{il+1}) \\
&\quad + (ml-i-1)c(v_{il+1}, v_{il+2}) + \ldots + c(v_{ml-1}, v_{ml}), \\
L^{'}(R_l) &= (ml-1)c(v_{1l}, v_{2l}) + \ldots + (ml-i+1)c(v_{il-1}, v_{ih}) + (ml-i)c(v_{ih}, v_{il+1}) \\
&\quad + (ml-i-1)c(v_{il+1}, v_{il+2}) + \ldots + c(v_{ml-1}, v_{ml}), \qquad (8) \\
L(R_h) &= (mh-1)c(v_{1h}, v_{2h}) + \ldots + (mh-i+1)c(v_{ih-1}, v_{ih}) \\
&\quad + (mh-i)c(v_{ih}, v_{ih+1}) + (mh-i-1)c(v_{ih+1}, v_{ih+2}) \\
&\quad + \ldots + c(v_{mh-1}, v_{mh}), \qquad (9) \\
L^{'}(R_h) &= (mh-1)c(v_{1h}, v_{2h}) + \ldots + (mh-i+1)c(v_{ih-1}, v_{il}) \\
&\quad + (mh-i)c(v_{il}, v_{ih+1}) - (mh-i-1)c(v_{ih+1}, v_{ih+2}) \\
&\quad + \ldots + c(v_{mh-1}, v_{mh}).
\end{aligned}$$

Therefore,

$$\begin{aligned} L(T_i) = L(T) &- (ml - i + 1)c(v_{il-1}, v_{il}) - (ml - i)c(v_{il}, v_{il+1}) \\ &- (mh - i - 1)c(v_{ih-1}, v_{ih}) - (mh - i)c(v_{ih}, v_{ih+1}) \\ &+ (ml - i + 1)c(v_{il-1}, v_{ih}) + (ml - i)c(v_{ih}, v_{il+1}) \\ &+ (mh - i + 1)c(v_{ih-1}, v_{il}) + (mh - i)c(v_{il}, v_{ih+1}). \end{aligned} \tag{10}$$

□

Hence, we can calculate $L(T_i)$ by the formulation (10) in $O(1)$ time. The complexity of exploring $N_8(T)$ is $O(ml \times mh)$.

The insert-2-routes neighborhood considers each vertex v_i in R_l and inserts it into each position in R_h. Insert-2-route neighborhood of R_l and R_h is defined as a set $N_1 1(T) = \{T_i = (R_1, \ldots, R_2, \ldots, R_l = (v_{1l}, v_{2l}, \ldots, v_{ih-1}, v_{ih}, v_{il+1}, \ldots, v_{ml}), \ldots, R_h = (v_{1h}, v_{2h}, \ldots, v_{ih-1}, v_{ih+1}, \ldots, v_{mh}), \ldots, R_k) : il = 2, 3, \ldots, ml - 1, ih = 2, 3, \ldots, mh - 1\}$. The size of the neighborhood is $O(ml \times mh)$.

Property 8. The complexity of exploring $N_9(T)$ is $O(ml \times mh)$.

Proof. The initial tour and $L(R)$ are the same as in (8) and (9). The neighborhood generates a neighboring tour $T_i = (R_1, \ldots, R_2, \ldots, R_l^{'} = (v_{1l}, v_{2l}, \ldots, v_{ih}, v_{il}, v_{il+1}, \ldots, v_{ml}), \ldots, R_h^{'} = (v_{1h}, v_{2h}, \ldots, v_{ih-1}, v_{ih+1}, \ldots, v_{mh}), \ldots, R_k)$. The costs of $R_l^{'}, R_h^{'}$ are calculated as follows:

$$\begin{aligned} L^{'}(R_l) = &(ml - 1)c(v_{1l}, v_{2l}) + \ldots + (ml - i + 1)c(v_{il-1}, v_{il}) + (ml - i)c(v_{il}, v_{ih}) \\ &+ (ml - i - 1)c(v_{ih}, v_{il+1}) + (ml - i - 2)c(v_{il+1}, v_{il+2}) \\ &+ \ldots + c(v_{ml-1}, v_{ml}), \\ L^{'}(R_h) = &(mh - 1)c(v_{1h}, v_{2h}) + \ldots + (mh - i + 1)c(v_{ih-1}, v_{ih+1}) \\ &+ (mh - i)c(v_{ih+1}, v_{ih+2}) + \ldots + c(v_{mh-1}, v_{mh}). \end{aligned}$$

Therefore,

$$\begin{aligned} L(T_i) = L(T) &- (ml - i)c(v_{il}, v_{il+1}) - (mh - i + 1)c(v_{ih-1}, v_{ih}) \\ &- (mh - i)c(v_{ih}, v_{ih+1}) - \sum_{k=1h}^{ih-1} c(v_k, v_{k+1}) \\ &+ (ml - i)c(v_{il}, v_{ih}) + (mh - i - 1)c(v_{ih}, v_{il+1}) \\ &+ (mh - i + 1)c(v_{ih-1}, v_{ih+1}) + \sum_{k=1l}^{il-1} c(v_k, v_{k+1}). \end{aligned} \tag{11}$$

□

Hence, we can calculate $L(T_i)$ by the formulation (11) in $O(max(mh, ml))$ time. Therefore, the complexity of exploring $N_9(T)$ is $O(max(mh, ml) \times ml \times mh)$.

In each iteration, the best neighboring solution is accepted if it is non-tabu and improving, or tabu, but globally improving. Due to the use of different neighborhood structures, three tabu lists are built. A move of the type remove-insert, swap-adjacent, or move-up(down) is stored in the first tabu list, the second is for 2-opt and 2-edge-opt moves, and the last one is for swap-2-routes, insert-2-routes. We do not use tabu list for move-forward-k-vertices, and move-backward-k-vertices.

Step 3: After finding a local optimum in Step 2, Step 3 starts to build up a promising solution LT. When the objective value of any local optimum lies within 5–10 % of the best-found solution, it is added into LT. If the size of LT is equal to sTL, then the algorithm goes to Step 4. The size of LT is chosen to be five because a small value for the size of LT enhances more implementations to the intensification and diversification steps. However, the search can be moved to another area of the solution space without the previous area explored. Otherwise, if the value of sTL is large, less intensification and diversification steps are performed.

Step 4: If the promising solution list LT is full, an intensification step starts. Each solution of LT is returned to Step 2 without any tabu move. When a new local optimum is found, the algorithm goes to Step 5 in which a diverse solution to reinitialize the search is created.

Step 5: We update an attribute matrix M, whose entries represent the number of times edge (i, j) occurred in an element of the promising solutions list. The Shaking procedure in Algorithm 3 will use the matrix; hence allows guiding the search towards an unexplored part of the solution space. In this work, two shaking mechanisms can be used to give a new solution as follows:

1. In intro-route, we use the shaking mechanism in Algorithm 4, called double-bridge, originally developed by [26]. The structure of the double-bridge move derives from a special 4-opt neighborhood where edges added and dropped need not be successively adjacent. This mechanism can also be seen as a permutation of two disjoint segments of a route.
2. In intra-route, we randomly choose two routes and after that, exchange some vertices in them or insert some vertices from a route into another. The steps in the intra-route are described in Algorithm 5. This solution obtained from shaking procedures does not include the edges which appear more than l times in the M matrix. We finally return to Step 2 with this solution.

The last aspect to discuss is the stop criterium of the VNS+TS algorithm. A balance must be made between computation time and efficiency. Here, the algorithm stops if no improvement is found after the number of the loop (NL).

The running time of the VNS + TS algorithm is mostly during the VNS step. In that step, insert-2-routes neighborhood consumes time at least as the others do. Assume that if these neighborhoods are invoked k_1 times, then the complexity of neighborhoods' exploration is $O(k_1 \times max(mh, ml) \times ml \times mh) \sim O(k_1 \times n^3)$ (in the worst case the size of mh or ml is n). It is the theoretical complexity of our algorithm.

4 COMPUTATIONAL EVALUATIONS

4.1 Metrics

In order to evaluate the efficiency of a metaheuristic algorithm, we can compare its solution to

1. the optimal solution (OPT);
2. the lower bound (LB); and
3. the initial solution of the construction phase ($Init.Sol$) or a good upper bound of the state-of-the-art metaheuristic algorithm (UB).

We define the improvement of the algorithm concerning $Best.Sol$, when $Best.Sol$ is the best solution found by our algorithm, in comparison with the optimal solution ($Gap_1[\%]$), a lower bound ($Gap_2[\%]$), and an initial solution ($Improv[\%]$) in percent, respectively, as follows:

$$Gap_1[\%] = \frac{Best.Sol - OPT}{OPT} \times 100\,\%,$$

$$Gap_2[\%] = \frac{Best.Sol - LB}{LB} \times 100\,\%,$$

$$Improv[\%] = \frac{Best.Sol - Init.Sol}{Init.Sol} \times 100\,\%.$$

The exact algorithm can find optimal solutions as in [25]. However, the algorithms only solve the problems with small sizes. The optimal solutions have been unknown with large instance sizes. In such cases, our best solutions can be compared to the tight lower bounds (i.e., defined by Nucamendi-Guillén et al. in [29]) or the initial solutions (i.e., the output of the insertion heuristic). As mentioned ealier, the MTRP is a relaxation of the MTRPD since $MD = 0$. Therefore, we can consider the optimal solutions published by Nucamendi-Guillén et al. [29] for the MTRP as the tight lower bounds in our experiments.

4.2 Datasets

The experimental data includes two datasets. In all instances, every distance between vertices satisfies the triangle inequality. Each instance contains the coordinate

of n vertices and one vertex was arbitrary designated as the depot. We divide these instances into two types (i.e., type 1 and type 2). The former one consists of the instances in which the optimal solutions have been known, otherwise the other depends on type 2.

We inherit several small instances in [25] and name them dataset 1 in our experiments. As a result, we can obtain the optimal solutions for these instances by using the exact algorithm in [25]. The dataset includes six TSP instances from the TSPLIB such as brd14051, d15112, d18512, fnl4461, nrw1379, and pr1002. For each TSP instance, they generate ten MTRPD instances by randomly selecting ten subsets of n vertices, where $n = 30$, 40 and 50. Therefore, in total, fifty MTRPD instances are used in our experiment.

The numerical analysis was performed on a set of benchmark problems for Capacitated VRP in [34]. As testing the proposed algorithm on all instances would be computationally too expensive, we applied our numerical analysis on some selected instances. First, to eliminate the effects of size, problems with approximately 50 up to 561 customers are chosen. Moreover, in order not to bias the results by taking "easy" or "hard" instances we randomly select them. We put them into a group named dataset 2. These are:

1. Christofides et al.: This dataset includes seven instances (CMT6, CMT7, ..., CMT14), which vary the number of vertices from 50 to 200 and vehicles from 5 to 18;
2. Taillard et al.: Nine instances from 75 to 150 vertices are picked randomly, specifically: tai75a, tai75b, tai75c, tai100a, tai100b, tai100c, tai150a, tai150b, tai150c;
3. Augerat et al.: Fifteen instances of dataset P and E are selected, which vary the number of vertices from 30 to 76 and vehicles from 2 to 15. In this dataset, we can obtain the lower bounds of the optimal solutions for the instances in [29];
4. Golden et al.: Six larger instances are picked randomly from G1 to G8, which vary the number of vertices from 240 to 480 and vehicles from 5 to 10;
5. Nucamendi-Guillén et al.: One hundred and fifty instances from 60 to 80 vertices are used in our experiments. The optimal solutions for the instances can be extracted from [29].

Moreover, our algorithm is also tested with some TRP instances. These are:

6. Silva et al. [31]: Three of these sets are generated, where each of them is composed of 20 instances with 50, 100, and 200 customers, respectively;
7. Abeledo et al. [2]: Nine instances from 48 to 100 vertices are chosen. The optimal solutions for these instances are extracted from [2].

In all instances in dataset 1, and several instances in dataset 2 (such as Christofides et al.'s and Golden et al.'s instances), the maximum total distance traveled in a route is available. However, in the others, there does not exist the distance

constraint. For each of these instances, we generated three possible distance constraints as a function of the distance to the farthest vertice from the depot (d_{max}). The distance constraint gets the values $2 \times d_{max}, 2.5 \times d_{max}$, and $3 \times d_{max}$. The similar generation for travel distance limit can be found in [12, 13, 25].

4.3 Results and Discussion

We conducted the experiments on a personal computer, which is equipped with an Intel Pentium Core i7 duo 2.10 GHz CPU and 4 GB RAM.

We experimented with the above datasets. For the instances in dataset 1, their optimal solutions let us evaluate precisely the efficiency of the TS + VNS algorithm. For the instances in dataset 2, because their optimal solutions have been unknown, our solutions only compare to the upper bounds or the known best solutions instead of the optimal ones. Therefore, the TS+VNS algorithm's efficiency is only evaluated relatively.

Through preliminary experiments, we observed that the values $pos = 5$, $sLT = 5$, $l = 5$, and $NL = 50$ resulted in a good trade-off between solution quality and run time. In addition, in a pilot study, the performance of the algorithm relatively depends on the order in which the neighborhoods are used. Generally speaking, the neighborhoods which have a smaller size are explored first. Since the algorithm becomes stuck in local optimum, the larger neighborhoods are used. That is, larger sized neighborhoods may help escape from local optimum. In this paper, the order of the neighborhoods is as follows: swap adjacent, remove-insert, swap, 2-opt, or, swap-2-route, and insert-2-route.

For each instance, our algorithm runs ten times, and the results are shown in Tables 1–18. In all the tables, we denote *Best.Sol* and *Aver.Sol* as the best and average solution of our metaheuristic, respectively. Table 1 includes the comparison between our algorithm and the optimal solutions in [25]. Table 2 shows the average values of Table 1. In Tables 3–11, we compare the results of the algorithm with the lower bound of the optimal solution. In these cases, the optimal solution of MTRP is the lower bound of the optimal solution of MTRPD. The optimal solution of MTRP is obtained in [29]. Moreover, they are also compared with the initial solutions by using insertion heuristic. Table 12 illustrates the evolution of the average deviation to the initial solutions during the iterations in some instances. In Tables 13–18, we show the results of our algorithm against the state-of-the-art metaheuristic algorithms in several MTRPD variants. Let T be the running time in seconds for our metaheuristic. *cTime* represents scaled run times, which is estimated on a Pentium 4, 2.4 GHz by means of the factors of Dongarra in [14] by second (note that: The experiments of Ezzineet et al. (IOE) [15], Ke et al. (CCVRP) [23], Ngueveu (MA1) [28], Nucamendi-Guillén et al. (SNG) [29], Riberio et al. (ALNS) [30], Silva et al. (MS) [31], and Ban (GRASP + VND) [9] were implemented on Pentium 4, 1 GHz, Pentium 4, 2.4 GHz, Pentium 4, 2 GHz, Intel Core 2 Duo 3 GHz, Pentium 4, 2 GHz, Pentium 4, 2.4 GHz, Pentium core i7 2.93 GHz, and Pentium core i7 duo 2.10 GHz, respectively).

4.3.1 Experimental Results for Datasets in Type 1

The experimental results are illustrated in Table 2, which are the average values calculated from Table 1. In Table 2, we denote $\overline{Gap_1}$ and $\overline{T}$ as the average values of Gap_1 and T for each dataset, respectively.

Table 2 shows that the algorithm is capable of finding the optimal solutions for all instances in dataset 1 in a reasonable amount of time, even for the cases of 50 vertices. That means our solutions are better than the ones in our previous work [9], which fails to find the optimal solutions for all instances with 50 vertices.

Instances		n=30, k=6				n=40, k=8				n=50, k=10			
		OPT	*Best.Sol*	*Aver.Sol*	*T*	*OPT*	*Best.Sol*	*Aver.Sol*	*T*	*OPT*	*Best.Sol*	*Aver.Sol*	*T*
pr1002	0	168,188.80	168,188.80	272,908.40	0.19	233,387.20	233,387.20	272,908.40	0.38	272,908.40	272,908.40	273754.69	0.72
	1	195,805.80	195,805.80	249,275.40	0.18	218,781.40	218,781.40	249,275.40	0.38	249,275.40	249,275.40	250561.75	0.73
	2	182,635.00	182,635.00	277,959.10	0.20	211,241.70	211,241.70	277,959.10	0.40	277,959.10	277,959.10	278260.69	0.69
	3	139,784.70	139,784.70	298,846.10	0.18	196,120.40	196,120.40	298,846.10	0.42	298,846.10	298,846.10	299368.26	0.72
	4	164,916.70	164,916.70	262,518.40	0.17	227,450.10	227,450.10	262,518.40	0.50	262,518.40	262,518.40	263424.19	0.68
	5	163,642.30	163,642.30	273,318.80	0.18	194,802.20	194,802.20	273,318.80	0.43	273,318.80	273,318.80	275714.50	0.68
	6	160,585.90	160,585.90	280,317.30	0.19	229,730.10	229,730.10	280,317.30	0.48	280,317.30	280,317.30	281015.54	0.72
	7	166,887.00	166,887.00	246,341.00	0.20	236,896.10	236,896.10	246,341.00	0.39	246,341.00	246,341.00	247693.88	0.73
	8	161,025.80	161,025.80	256,971.40	0.18	230,126.60	230,126.60	256,971.40	0.47	256,971.40	256,971.40	257688.31	0.74
	9	144,167.00	144,167.00	267,596.70	0.18	192,179.90	192,179.90	267,596.70	0.48	267,596.70	267,596.70	268104.67	0.75
brd14051	0	97,380.30	97,380.30	133,642.30	0.19	97,630.70	97,630.70	133,642.30	0.43	133,642.30	133,642.30	134565.72	0.68
	1	96,322.50	96,322.50	123,212.70	0.18	110,671.10	110,671.10	123,212.70	0.42	123,212.70	123,212.70	123459.68	0.69
	2	64,109.40	64,109.40	137,175.10	0.19	127,629.70	127,629.70	137,175.10	0.46	137,175.10	137,175.10	137550.65	0.75
	3	89,582.50	89,582.50	150,209.80	0.18	99,527.60	99,527.60	150,209.80	0.37	150,209.80	150,209.80	151117.47	0.75
	4	87,615.70	87,615.70	116,278.70	0.19	123,881.80	123,881.80	116,278.70	0.49	116,278.70	116,278.70	116771.48	0.75
	5	75,079.50	75,079.50	124,648.20	0.19	98,329.40	98,329.40	124,648.20	0.49	124,648.20	124,648.20	124860.29	0.69
	6	94,540.80	94,540.80	121,190.40	0.20	110,676.60	110,676.60	121,190.40	0.36	121,190.40	121,190.40	122040.23	0.72
	7	81,515.80	81,515.80	124,077.60	0.19	103,775.50	103,775.50	124,077.60	0.42	124,077.60	124,077.60	124459.33	0.68
	8	74,160.80	74,160.80	125,446.00	0.19	101,387.30	101,387.30	125,446.00	0.41	125,446.00	125,446.00	125750.98	0.72
	9	90,628.10	90,628.10	118,925.00	0.18	87,945.10	87,945.10	118,925.00	0.48	118,925.00	118,925.00	119701.47	0.75
fnl4461	0	51,192.20	51,192.20	81,562.20	0.19	63,096.00	63,096.00	81,562.20	0.48	81,562.20	81,562.20	81931.39	0.68
	1	44,154.70	44,154.70	79,804.80	0.18	66,882.60	66,882.60	79,804.80	0.38	79,804.80	79,804.80	80132.27	0.71
	2	46,571.20	46,571.20	73,309.10	0.18	70,151.40	70,151.40	73,309.10	0.39	73,309.10	73,309.10	73657.61	0.69
	3	48,591.40	48,591.40	79,335.10	0.19	58,843.90	58,843.90	79,335.10	0.43	79,335.10	79,335.10	79480.48	0.67
	4	54,485.90	54,485.90	75,052.00	0.18	61,654.90	61,654.90	75,052.00	0.38	75,052.00	75,052.00	75154.18	0.70
	5	47,907.30	47,907.30	76,738.10	0.18	56,144.50	56,144.50	76,738.10	0.42	76,738.10	76,738.10	77076.34	0.75
	6	45,882.10	45,882.10	75,268.90	0.18	61,274.90	61,274.90	75,268.90	0.43	75,268.90	75,268.90	75553.79	0.70
	7	44,545.30	44,545.30	72,956.30	0.20	65,698.30	65,698.30	72,956.30	0.38	72,956.30	72,956.30	73058.48	0.72
	8	50,365.30	50,365.30	70,244.00	0.20	64,260.90	64,260.90	70,244.00	0.48	70,244.00	70,244.00	70593.77	0.72
	9	49,179.60	49,179.60	82,157.00	0.18	58,717.50	58,717.50	82,157.00	0.49	82,157.00	82,157.00	82319.62	0.73
d15112	0	225,070.20	225,070.20	353,657.80	0.18	287,734.80	287,734.80	353,657.80	0.37	353,657.80	353,657.80	354976.61	0.73
	1	213,332.30	213,332.30	355,115.20	0.19	256,987.10	256,987.10	355,115.20	0.39	355,115.20	355,115.20	356258.02	0.72
	2	208,323.60	208,323.60	392,196.10	0.19	307,407.10	307,407.10	392,196.10	0.46	392,196.10	392,196.10	393694.14	0.75
	3	222,870.40	222,870.40	350,821.80	0.19	292,602.70	292,602.70	350,821.80	0.42	350,821.80	350,821.80	351422.23	0.72
	4	216,056.00	216,056.00	341,493.60	0.19	299,259.00	299,259.00	341,493.60	0.35	341,493.60	341,493.60	342212.93	0.74
	5	235,215.80	235,215.80	360,717.40	0.18	269,559.40	269,559.40	360,717.40	0.35	360,717.40	360,717.40	362220.31	0.72
	6	207,139.00	207,139.00	390,251.40	0.19	320,989.40	320,989.40	390,251.40	0.45	390,251.40	390,251.40	392014.44	0.72
	7	280,309.00	280,309.00	327,701.90	0.20	287,270.70	287,270.70	327,701.90	0.41	327,701.90	327,701.90	328900.79	0.73
	8	244,015.40	244,015.40	344,600.50	0.19	303,263.90	303,263.90	344,600.50	0.38	344,600.50	344,600.50	345172.84	0.68
	9	238,976.20	238,976.20	347,783.60	0.20	282,412.30	282,412.30	347,783.60	0.40	347,783.60	347,783.60	348249.82	0.72
nrw1379	0	31,249.40	31,249.40	39,206.10	0.17	35,655.20	35,655.20	39,206.10	0.45	39,206.10	39,206.10	39299.26	0.74
	1	33,138.50	33,138.50	61,449.00	0.17	33,000.70	33,000.70	61,449.00	0.39	61,449.00	61,449.00	44859.42	0.68
	2	31,872.00	31,872.00	45,914.20	0.19	39,928.10	39,928.10	45,914.20	0.49	45,914.20	45,914.20	46111.68	0.72
	3	31,777.10	31,777.10	46,208.00	0.18	36,685.40	36,685.40	46,208.00	0.50	46,208.00	46,208.00	46317.23	0.68
	4	26,671.20	26,671.20	43,557.90	0.18	36,168.60	36,168.60	43,557.90	0.38	43,557.90	43,557.90	43635.93	0.75
	5	29,010.30	29,010.30	46,718.40	0.18	38,005.40	38,005.40	46,718.40	0.42	46,718.40	46,718.40	46946.35	0.71
	6	30,398.10	30,398.10	49,421.10	0.19	31,837.30	31,837.30	49,421.10	0.36	49,421.10	49,421.10	49553.53	0.73
	7	30,765.50	30,765.50	49,960.10	0.19	39,394.80	39,394.80	49,960.10	0.49	49,960.10	49,960.10	50037.67	0.75
	8	28,796.40	28,796.40	41,560.90	0.20	36,674.50	36,674.50	41,560.90	0.37	41,560.90	41,560.90	41703.33	0.74
	9	26,271.20	26,271.20	44,404.00	0.18	36,447.70	36,447.70	44,404.00	0.43	44,404.00	44,404.00	44518.42	0.68

Table 1. Results for dataset type 1

4.3.2 Experimental Results for Datasets in Type 2

Similar to the dataset type 1, we show the average values in Tables 9 and 12. The values in Table 9 are calculated from the ones in Tables 3–8. Meanwhile, Table 12

Instances	n=30		n=40		n=50	
	Gap_1	T	Gap_1	T	Gap_1	T
brd14051-x	0.00	0.19	0.00	0.38	0.00	0.72
d15112-x	0.00	0.18	0.00	0.38	0.00	0.73
fnl4461-x	0.00	0.20	0.00	0.40	0.00	0.69
nrw1379-x	0.00	0.18	0.00	0.42	0.00	0.72
pr1002-x	0.00	0.17	0.00	0.50	0.00	0.68

Table 2. Average results for daset type 1

Instances	LB	$Init.Sol$	$MD = 2\times d_{max}$			$MD = 2.5\times d_{max}$			$MD = 3\times d_{max}$		
			$Best.Sol$	$Aver.Sol$	T	$Best.Sol$	$Aver.Sol$	T	$Best.Sol$	$Aver.Sol$	T
pr1002_60_0	530946.01	660211.35	532444.24	537533.19	1.49	531865.86	533825.5	1.45	531757.16	534609.7	1.45
pr1002_60_1	356469.79	455893.41	359952.84	370184.99	1.42	358368.98	367379.55	1.45	358942.18	367163.14	1.45
pr1002_60_2	344118.14	467498.53	344660.51	354297.21	1.49	344923.03	351734.69	1.42	344310.30	351678.69	1.42
pr1002_60_3	429604.2	579392.35	430930.61	435619.75	1.46	430738.16	433873.27	1.45	430341.65	434434.23	1.45
pr1002_60_4	435655.25	540342.11	436868.42	443304.34	1.41	437187.11	441177.4	1.46	436487.61	441214.38	1.46
pr1002_60_5	668129.73	779776.11	670146.51	674511.75	1.43	669780.58	673040.01	1.47	669092.64	673298.58	1.47
pr1002_60_6	406678.77	495022.53	408185.76	413479.93	1.45	407782.79	411889.62	1.44	407328.01	412153.54	1.44
pr1002_60_7	311254.73	414296.52	315012.06	325939.25	1.49	314351.03	322249.21	1.44	314631.61	323133.36	1.44
pr1002_60_8	469816.84	591638.26	471572.56	478597.47	1.50	470579.14	476756.73	1.50	471383.23	476823.63	1.50
pr1002_60_9	277336.06	377249.41	280140.52	292281.76	1.42	281572.42	288733.31	1.40	278504.55	288503.61	1.40
brd14051_60_0	213420.42	267899.2375	214319.39	216860.88	1.50	213818.46	215295.73	1.49	213614.02	215749.05	1.49
brd14051_60_1	218315.68	312468.7714	218728.14	220264.95	1.49	218880.78	220373.77	1.49	218556.31	220728.3	1.49
brd14051_60_2	151666.85	207799.2353	153198.75	156139.96	1.45	152543.04	154954.36	1.48	153057.91	154911.12	1.48
brd14051_60_3	172199.83	232433.3597	172875.34	175903.46	1.48	172877.53	174923.62	1.41	172882.12	174934.8	1.41
brd14051_60_4	133952.5	167792.608	135660.89	139384.34	1.42	134963.93	138301.02	1.42	135622.01	138319.39	1.42
brd14051_60_5	203145.14	290606.4286	203424.5	205266.4	1.44	203348.6	204742.68	1.43	203576.11	204746.58	1.43
brd14051_60_6	136233.51	171636.975	137309.58	140813.51	1.49	137072.2	139742.31	1.47	137666.96	139931.2	1.47
brd14051_60_7	171879.58	248180.3	173726.21	176795.14	1.48	173395.86	175603.68	1.42	172880.97	175530.15	1.42
brd14051_60_8	191580.79	241067.3882	192145.73	196064.3	1.49	191949.55	194840.17	1.47	192277.69	195207.18	1.47
brd14051_60_9	128178.58	174326.1925	129452.37	132113.8	1.47	129039.81	131038.32	1.41	128554.14	131136.74	1.41
fnl4461_60_0	156260.54	194032.1583	156482.1	157364.48	1.40	156502.53	156867.68	1.47	156508.88	157085.13	1.47
fnl4461_60_1	103190.13	131569.4961	103881.33	105989.96	1.48	103571.39	105059.35	1.45	103533.38	104978.45	1.45
fnl4461_60_2	109739.93	149525.6149	110236.87	111979.43	1.49	110112.9	111088.62	1.48	109795.94	111158.19	1.48
fnl4461_60_3	100792.2	136198.0575	101299.08	103234.98	1.47	100961.18	102382.31	1.47	101131.62	102305.81	1.47
fnl4461_60_4	149638.18	185947.0777	150338.84	151703.66	1.48	150322.42	151338.95	1.49	150154.33	151438.45	1.49
fnl4461_60_5	158679.44	185251.4379	159206.73	160478.43	1.48	158930.62	160072.49	1.49	158926.13	160089.54	1.49
fnl4461_60_6	122266.92	149102.6283	122947.07	124435.02	1.44	122916.9	123926.83	1.43	122817.55	123825.42	1.43
fnl4461_60_7	107469.11	142108.8532	108053.05	109887.69	1.47	107925.52	109376.06	1.47	107716.14	109006.11	1.47
fnl4461_60_8	100531.72	127280.3749	101450.39	104630.87	1.42	101592.06	103707.02	1.42	101118.58	103710.54	1.42
fnl4461_60_9	135829.76	183343.8156	136148.74	137640.52	1.47	136160.29	137330.96	1.40	136163.91	137267.96	1.40
d15112_60_0	684939.42	851498.8482	686712.6	699720.00	1.40	685359.09	694349.73	1.48	686192.80	694446.4	1.48
d15112_60_1	644759.99	819500.5493	647040.61	654258.07	1.43	646425.6	651802.15	1.45	645901.74	652152.95	1.45
d15112_60_2	425069.33	583381.5404	430094.57	444157.23	1.41	428827.48	439403.47	1.45	426883.39	438826.4	1.45
d15112_60_3	528177.95	662371.45	529897.16	541938.08	1.41	529082.69	539617.02	1.49	529129.18	538786.1	1.49
d15112_60_4	586915.82	736112.95	588890.36	596517.59	1.48	587802.34	593603.46	1.46	588157.01	593755.74	1.46
d15112_60_5	422195.61	494729.4263	425174.86	435267.63	1.47	423386.01	432465.17	1.46	422698.69	432637.93	1.46
d15112_60_6	518793.6	633637.8578	522485.21	534777.58	1.43	521841.6	530514.27	1.48	520984.95	529540.23	1.48
d15112_60_7	616918.44	776732.675	621386.14	631195.53	1.49	620561.63	627712.93	1.48	619844.40	627215.03	1.48
d15112_60_8	397619.37	500495.3875	400396.31	417380.61	1.40	400772.54	412773.75	1.46	400191.93	412712.49	1.46
d15112_60_9	673840.81	910298.6184	675975.95	682660.36	1.45	674759.18	681252.34	1.42	675686.08	680839.7	1.42
nrw1379_60_0	64359.77	80086.56654	64587.82	65752.75	1.44	64588.24	65216.27	1.42	64570.11	65388.53	1.42
nrw1379_60_1	83410.67	104646.3375	83717.07	84295.24	1.48	83453.87	84149.95	1.49	83577.60	84304.19	1.49
nrw1379_60_2	52858.87	70986.81333	53240.11	55622.46	1.48	53731.31	55606.98	1.40	53699.79	55545.65	1.40
nrw1379_60_3	62341.36	84434.20476	62799.04	64203.28	1.42	63054.46	64243.88	1.45	63155.92	64314.58	1.45
nrw1379_60_4	56012.13	69680.90881	56337.25	57630.46	1.45	56306.76	57462.91	1.42	56401.26	57331.24	1.42
nrw1379_60_5	58083.8	72973.325	58378.66	59960.51	1.45	58551.84	59986.25	1.50	58611.01	59945.96	1.50
nrw1379_60_6	52224.66	65749.025	52599.22	54626.86	1.46	52433.62	54526.7	1.47	52476.05	54450.03	1.47
nrw1379_60_7	58402.97	73290.6375	58632.51	59997.69	1.47	58495.04	60194.16	1.45	58679.87	59937.75	1.45
nrw1379_60_8	52145.08	66101.85821	52687.3	53873.21	1.48	52682.18	53653.94	1.45	52550.57	53726.75	1.45
nrw1379_60_9	49026.52	66572.84307	49436.13	51019.36	1.43	49471.03	51283.54	1.41	49385.68	50955.34	1.41

Table 3. Results for ANMM-instances ($n = 60$, $k = 12$)

Instances	*LB*	*Init.Sol*	$MD = 2\times d_{max}$			$MD = 2.5\times d_{max}$			$MD = 3\times d_{max}$		
			Best.Sol	*Aver.Sol*	*T*	*Best.Sol*	*Aver.Sol*	*T*	*Best.Sol*	*Aver.Sol*	*T*
pr1002_70_0	429557.7	535485.69	432866.54	439866.58	1.56	432866.54	439866.58	1.86	432866.54	439866.58	1.86
pr1002_70_1	430048.06	530744.28	433424.51	440956.86	1.54	433424.51	440956.86	2.10	433424.51	440956.86	2.10
pr1002_70_2	377233.86	524943.18	379605.76	389076.55	1.34	379605.76	389076.55	1.95	379605.76	389076.55	1.95
pr1002_70_3	429562.01	557187.53	432804.3	441224.85	1.32	432804.3	441224.85	1.85	432804.3	441224.85	1.85
pr1002_70_4	435659.17	574628.71	439726.32	447473.63	1.48	439726.32	447473.63	2.17	439726.32	447473.63	2.17
pr1002_70_5	429584.16	558284.19	431998.56	443795.32	1.67	431998.56	443795.32	2.13	431998.56	443795.32	2.13
pr1002_70_6	344534.44	495280.87	348013.29	355996.4	1.41	348013.29	355996.4	1.95	348013.29	355996.4	1.95
pr1002_70_7	393558.46	543537.08	396618.63	406657.01	1.51	396618.63	406657.01	1.99	396618.63	406657.01	1.99
pr1002_70_8	397072.39	560744.46	400993.88	412836.03	1.37	400993.88	412836.03	1.97	400993.88	412836.03	1.97
pr1002_70_0	429557.7	535485.69	432866.54	439866.58	1.56	432866.54	439866.58	1.86	432866.54	439866.58	1.86
brd14051_70_0	191843.35	248655.93	193433.59	196300.43	1.38	193433.59	196300.43	1.82	192664.90	195906.35	1.82
brd14051_70_1	169340.01	227929.44	169848.29	172027.95	1.49	169848.29	172027.95	1.91	170141.57	171980.54	1.91
brd14051_70_2	216195.95	274964.88	217119.82	218723.61	1.56	217119.82	218723.61	1.78	216904.65	219570.63	1.78
brd14051_70_3	229328.9	287350.77	230828.7	235001.63	1.64	230828.7	235001.63	2.12	230700.27	234893.9	2.12
brd14051_70_4	302498.42	352533.21	303665.65	305149.45	1.67	303665.65	305149.45	1.75	302997.60	305008.65	1.75
brd14051_70_5	179470.31	239730.53	180336.91	182335.7	1.50	180336.91	182335.7	1.78	180402.62	182346.26	1.78
brd14051_70_6	231693.74	299183.67	232654.01	234908.12	1.33	232654.01	234908.12	1.75	232686.01	234963.25	1.75
brd14051_70_7	284960.31	354174.33	285658.8	288385.84	1.33	285658.8	288385.84	1.78	285974.24	288104.83	1.78
brd14051_70_8	167533.17	246354.08	168487.82	171604.89	1.38	168487.82	171604.89	1.89	169011.06	171829.52	1.89
brd14051_70_9	253499.74	304870.99	255311.77	259334.3	1.63	255311.77	259334.3	1.82	255500.92	259488.88	1.82
fnl4461_70_0	154805.67	195026.89	155064.12	156113.94	1.38	155064.12	156113.94	2.16	155064.12	156113.94	2.16
fnl4461_70_1	104585.82	138431.3	105739.55	108511.43	1.62	105739.55	108511.43	1.88	105739.55	108511.43	1.88
fnl4461_70_2	161892.44	202490.16	162356.87	163138.77	1.37	162356.87	163138.77	1.75	162356.87	163138.77	1.75
fnl4461_70_3	99122.23	136888.98	100079.14	101756.85	1.66	100079.14	101756.85	2.15	100079.14	101756.85	2.15
fnl4461_70_4	157106.13	215133.56	157517.01	158373.7	1.42	157517.01	158373.7	2.19	157517.01	158373.7	2.19
fnl4461_70_5	112094.64	154967.72	113643.87	116157.3	1.36	113643.87	116157.3	1.89	113643.87	116157.3	1.89
fnl4461_70_6	121521	163833.57	122307.91	124262.7	1.37	122307.91	124262.7	1.71	122307.91	124262.7	1.71
fnl4461_70_7	175859.51	219145.84	176440.38	177290.01	1.53	176440.38	177290.01	1.79	176440.38	177290.01	1.79
fnl4461_70_8	122141.15	168186.87	122884.09	125065.93	1.47	122884.09	125065.93	1.88	122884.09	125065.93	1.88
fnl4461_70_0	154805.67	195026.89	155064.12	156113.94	1.38	155064.12	156113.94	2.16	155064.12	156113.94	2.16
d15112_70_0	517426.18	692065.87	523553.26	531145.65	1.62	523553.26	531145.65	1.48	523553.26	531145.65	1.79
d15112_70_1	715678.26	886361.97	722362.06	728250.24	1.51	722362.06	728250.24	1.45	722362.06	728250.24	1.98
d15112_70_2	688605.9	892883.4	690001.05	695602.83	1.50	690001.05	695602.83	1.45	690001.05	695602.83	2.04
d15112_70_3	625623.9	852706.46	630340.44	637526.15	1.66	630340.44	637526.15	1.49	630340.44	637526.15	1.77
d15112_70_4	532088.98	747897.34	536030.13	544894.97	1.39	536030.13	544894.97	1.46	536030.13	544894.97	1.72
d15112_70_5	500455.25	639173.85	504012.97	516029.99	1.59	504012.97	516029.99	1.46	504012.97	516029.99	1.82
d15112_70_6	497229.6	708355.1	494630.23	501861.27	1.59	494630.23	501861.27	1.48	494630.23	501861.27	1.82
d15112_70_7	599776.85	766690.15	604254.21	609315.82	1.43	604254.21	609315.82	1.48	604254.21	609315.82	1.88
d15112_70_8	576957.51	734369.78	582333.88	587065.31	1.51	582333.88	587065.31	1.46	582333.88	587065.31	1.93
d15112_70_9	775176.3	1018784.7	776211.48	780571.39	1.30	776211.48	780571.39	1.42	776211.48	780571.39	1.70
nrw1379_70_0	66839.83	91823.05	67133.95	68149.96	1.29	67133.95	68149.96	1.42	67141.89	67842.3	1.79
nrw1379_70_1	65103.43	86177.5	65403.57	67281.39	1.49	65403.57	67281.39	1.49	65836.06	67162.39	2.09
nrw1379_70_2	63480.7	86971.23	64215.06	65872.58	1.60	64215.06	65872.58	1.40	63992.33	65884.47	1.67
nrw1379_70_3	59273.92	78578.67	60111.63	61528.06	1.66	60111.63	61528.06	1.45	59705.82	61585.84	2.16
nrw1379_70_4	70594.56	90450.43	71095.2	71700.76	1.33	71095.2	71700.76	1.42	70953.48	71755.91	2.05
nrw1379_70_5	73884.17	95059.75	74190.23	74969.23	1.51	74190.23	74969.23	1.50	74081.86	75045.23	1.92
nrw1379_70_6	64306.14	87586.7	65019.63	66399.38	1.47	65019.63	66399.38	1.47	64995.77	66682.88	1.97
nrw1379_70_7	90554.87	113522.25	90716.75	91348.07	1.27	90716.75	91348.07	1.45	90743.23	91407.05	1.78
nrw1379_70_8	91738.43	125234.93	91924.59	92556.51	1.41	91924.59	92556.51	1.45	91961.09	92591.73	1.90
nrw1379_70_9	68024.3	92003.32	68219.67	69163.47	1.34	68219.67	69163.47	1.41	68512.64	69234.01	2.18

Table 4. Results for ANMM-instances ($n = 70$, $k = 14$)

contains the average values calculated from Tables 10 to 11. In Tables 9 and 12, we denote $\overline{Gap_i}(i = 1, 2)$ and $\overline{T}$ as the average of $Gap_i(i = 1, 2)$ and T for each dataset, respectively.

From Tables 9 to 11, for all instances, it is indicated that the proposed algorithm can improve the solutions in comparison with the initial solutions. Specifically, the average of *Improv* lines between 19.4 % and 27.1 %. Besides, in Table 9, for most instances, our solutions fall into the range of 0.88 %–3.77 % of the lower bound of the optimal solution. Therefore, we can conclude that for the instances solved, our algorithm finds near-optimal solutions, even for the large instances. In comparison with GRASP + VND [9], our average Gap_2 (about 1.09) is much better than GRASP + VND (about 3.62). Obviously, the VNS + TS algorithm outperforms

Instances	LB	Init.Sol	$MD = 2\times d_{max}$			$MD = 2.5\times d_{max}$			$MD = 3\times d_{max}$		
			Best.Sol	Aver.Sol	T	Best.Sol	Aver.Sol	T	Best.Sol	Aver.Sol	T
pr1002_80_0	491764.64	656239.68	494687.68	504055.32	4.74	494687.82	501827.11	4.52	495216.69	502028.79	4.52
pr1002_80_1	442164.21	613287.46	446829.95	452810.93	4.59	446638.68	451662.89	4.77	446105.29	451219.62	4.77
pr1002_80_2	505524.17	609954.56	509224.22	516775.04	4.66	507827.3	514010.41	4.78	509460.12	513907.95	4.78
pr1002_80_3	436752.96	614611.29	439342.64	448311.86	4.55	439573.59	445577.18	4.65	438557.21	445402.53	4.65
pr1002_80_4	453609.46	587470.83	457842.39	467356.11	4.68	458557.63	464161.5	4.65	454449.43	463546.66	4.65
pr1002_80_5	599492.4	771733.95	601443.35	606392.15	4.58	601220.79	605159.05	4.60	601030.48	605036.78	4.60
pr1002_80_6	619003.36	805206.35	620905.87	626627.49	4.70	620390.26	624888.79	4.77	620980.59	624674.78	4.77
pr1002_80_7	508186.51	658640.85	511992.03	520148.95	4.71	512678.26	518684.16	4.61	511049.54	518270.58	4.61
pr1002_80_8	409733.88	518052.51	414454.61	431925.53	4.72	416386.88	426285.82	4.53	417195.97	426199.67	4.53
pr1002_80_9	557220.48	670387.49	559471.39	566974.28	4.64	560004.93	565634.88	4.73	560663.69	565474.87	4.73
brd14051_80_0	336983.07	403178.34	338615.5	340638.58	4.52	339017.38	341621.98	4.62	338752.62	341253.06	4.62
brd14051_80_1	277861.02	348787.38	278718.88	280954.58	4.57	278712.45	280898.21	4.57	279056.42	281444.44	4.57
brd14051_80_2	265370.92	321922.47	266964.57	271196.43	4.78	266965.71	271284.78	4.62	267098.27	271731.14	4.62
brd14051_80_3	189815.69	240361.74	190981.53	192988.97	4.55	190970.44	193195.6	4.53	190524.94	192859.4	4.53
brd14051_80_4	206068.45	275228.43	207846.42	211233.89	4.75	208142.71	211570.64	4.54	206986.40	211677.53	4.54
brd14051_80_5	303621.75	348578.27	304790.13	307187.47	4.66	304570.04	307066.74	4.78	304731.36	306944.31	4.78
brd14051_80_6	213405.23	266958.37	215496.73	219693.15	4.80	215905.29	220134.25	4.78	214846.74	220392.91	4.78
brd14051_80_7	263737.93	308039.16	265261	267579.79	4.52	265166.47	267383	4.67	265491.60	268425.6	4.67
brd14051_80_8	232967.83	298574.84	234215.29	236778.42	4.63	234262.37	238062.49	4.52	235200.12	238042.96	4.52
brd14051_80_9	317790.55	368183.51	318860.4	320911.18	4.53	319258.92	321512.59	4.57	319305.30	321576.77	4.57
fnl4461_80_0	153124.51	194685.8	154240.65	155854.44	4.79	154017.32	155260.49	4.61	154016.10	155538.61	4.61
fnl4461_80_1	174975.64	224516.74	175564.02	176753.56	4.50	175521.26	176567.39	4.75	175432.81	176488.95	4.75
fnl4461_80_2	162755.5	197782.39	163709.43	165437.58	4.73	163927.72	165265.48	4.51	163681.90	165198.58	4.51
fnl4461_80_3	160819.04	192927.87	161721.6	164211.03	4.75	162093.53	164341.05	4.52	161950.22	164289.55	4.52
fnl4461_80_4	151790.69	187440.09	152909.48	154583.49	4.76	152741.67	154474.03	4.55	152941.34	154525.71	4.55
fnl4461_80_5	131045.47	172293.83	131674.42	134594.78	4.52	132508.99	134525.24	4.69	132384.34	134838.76	4.69
fnl4461_80_6	125405.93	166418.99	126309.93	128634.42	4.62	126716.15	128784.35	4.72	126469.44	128585.69	4.72
fnl4461_80_7	125228.91	164627.71	127382.05	129202.79	4.58	126969.72	129172.82	4.69	126973.64	129372.03	4.69
fnl4461_80_8	185280.87	228208.31	185832.25	187148.81	4.74	185826.34	187068.01	4.64	185922.90	187144.12	4.64
fnl4461_80_9	130304.95	165022.1	131835.98	133896.72	4.63	131690.15	133899.8	4.66	131257.09	134025.52	4.66
d15112_80_0	551900.43	753989.49	556725.13	564166.47	4.78	552906.19	562737.67	4.59	552906.19	562737.67	4.59
d15112_80_1	815029.39	979921.76	816533.21	820893.85	4.55	817987.76	820839.25	4.72	817987.76	820839.25	4.72
d15112_80_2	828114.32	1080571.45	830372.82	834934.02	4.58	831176.05	836837.77	4.55	831176.05	836837.77	4.55
d15112_80_3	689450.94	964458.54	693037.65	702133.55	4.55	694369.37	708522.84	4.71	694369.37	708522.84	4.71
d15112_80_4	560385.47	737417.48	566738.01	578449.8	4.54	566738.01	578449.8	4.55	566738.01	578449.8	4.55
d15112_80_5	821959.4	1030515.79	825064.27	830641.04	4.76	825064.27	830641.04	4.61	825064.27	830641.04	4.61
d15112_80_6	715206.03	882086.8	719021.95	728868.7	4.68	719021.95	728868.7	4.68	719021.95	728868.7	4.68
d15112_80_7	958278.86	1155190.13	960032.55	964445.29	4.66	960032.55	964445.29	4.73	960032.55	964445.29	4.73
d15112_80_8	990277.77	1174384.17	990277.77	992123.77	4.55	990277.77	992123.77	4.52	990277.77	992123.77	4.52
d15112_80_9	672457.47	931587.71	672457.47	677541.47	4.75	672457.47	677541.47	4.78	672457.47	677541.47	4.78
nrw1379_80_0	64831.76	96656.39	65739.01	67598.57	4.68	65725.8	67571.78	4.73	65550.91	67927.83	4.73
nrw1379_80_1	64967.83	88394.72	66163.83	67947.85	4.61	65927.2	67899.4	4.65	65929.48	67713.73	4.65
nrw1379_80_2	73858.13	96499.97	74824.24	76023.55	4.65	74664.85	75881.09	4.63	74495.65	75981.26	4.63
nrw1379_80_3	100592.83	131733.34	101010.22	101650.71	4.62	100706.51	101643.99	4.63	100858.38	101810.63	4.63
nrw1379_80_4	98228.29	126451.61	98545.56	99177.28	4.52	98519.47	99278.86	4.59	98418.60	99239.92	4.59
nrw1379_80_5	75984.21	99492.47	76685.61	78143.45	4.57	76525.87	78206.56	4.65	76524.34	78122.3	4.65
nrw1379_80_6	79165.6	105024.23	79636.62	80402.19	4.54	79757.37	80363.95	4.65	79757.37	80363.95	4.65
nrw1379_80_7	73194.55	105328.52	74106.95	75589.58	4.55	74106.95	75589.58	4.75	74106.95	75589.58	4.75
nrw1379_80_8	83492.62	115793.31	83710.72	85128.66	4.57	83710.72	85128.66	4.74	83710.72	85128.66	4.74
nrw1379_80_9	67034.31	92380.75	67853.71	69638.94	4.62	67853.71	69638.94	4.69	67853.71	69638.94	4.69

Table 5. Results for ANMM-instances ($n = 80$, $k = 16$)

Instances	LB	Init.Sol	$MD = 2\times d_{max}$			$MD = 2.5\times d_{max}$			$MD = 3\times d_{max}$		
			Best.Sol	Aver.Sol	T	Best.Sol	Aver.Sol	T	Best.Sol	Aver.Sol	T
E-n30-k3	1643.3	2604.62	1670.41	1670.41	0.06	1670.41	1670.41	0.06	1670.41	1670.41	0.06
E-n30-k4	1643.3	2490.48	1645.87	1645.87	0.07	1645.87	1645.87	0.06	1645.87	1645.87	0.07
E-n33-k4	2819.43	3545.31	2819.43	2819.43	0.08	2819.43	2819.43	0.08	2819.43	2819.43	0.08
E-n51-k5	2209.64	3357.72	2292.60	2292.60	1.01	2292.60	2292.60	1.05	2292.60	2292.60	0.96
E-n76-k10	2310.09	3775.91	2417.55	2589.31	2.05	2417.55	2417.55	2.12	2417.55	2417.55	2.14
E-n76-k14	2005.4	3237.01	2049.33	2145.23	2.13	2030.43	2030.43	2.09	2030.43	2030.43	2.05
E-n76-k15	1962.47	3116.23	2031.95	2145.90	2.11	2031.95	2047.43	2.17	2031.95	2047.43	2.03

Table 6. Results for E-instances

Instances	*LB*	*Init.Sol*	$MD = 2\times d_{max}$			$MD = 2.5\times d_{max}$			$MD = 3\times d_{max}$		
			Best.Sol	*Aver.Sol*	*T*	*Best.Sol*	*Aver.Sol*	*T*	*Best.Sol*	*Aver.Sol*	*T*
P40k5	1537.79	2146.68	1587.52	1690.1559	0.08	1587.52	1568.86	0.08	1587.52	1568.86	0.08
P45k5	1912.31	2688.95	1968.57	2143.5088	0.09	1968.57	1961.12	0.10	1968.57	1961.12	0.09
P50k7	1547.89	2267.93	1580.65	1726.5957	0.97	1580.65	1575.59	0.97	1580.65	1575.59	1.01
P55k7	1766.56	2716.8	1840.22	2006.9836	1.07	1840.22	1838.15	0.99	1840.22	1838.15	1.07
P60k10	1676.35	2492.09	1723.04	1830.2216	1.18	1723.04	1707.72	1.16	1723.04	1707.72	1.20
P76k4	4686.92	6474.01	5059.4	5666.5279	2.18	5059.4	5023.78	2.16	5059.40	5023.78	2.16
P76k5	3820.02	5962.72	3820.02	3820.02	2.14	3820.02	3820.02	2.04	3820.02	3820.02	2.13
Pn70k10	2097.17	3414.5	2137.3	2332.3804	1.40	2137.3	2332.3804	1.40	2137.3	2332.3804	1.40

Table 7. Results for P-instances

Instances	*Init.Sol*	$MD = 2\times d_{max}$			$MD = 2.5\times d_{max}$			$MD = 3\times d_{max}$		
		Best.Sol	*Aver.Sol*	*T*	*Best.Sol*	*Aver.Sol*	*T*	*Best.Sol*	*Aver.Sol*	*T*
tai75a	6840.64	4840.69	5020.63	2.06	4803.02	5023.08	2.06	4803.02	5023.08	2.06
tai75b	5575.85	3782.95	3931.11	2.14	3793.62	3941.72	2.14	3793.62	3941.72	2.14
tai75c	7110.74	3838.18	4008.01	2.19	3848.39	4008.11	2.25	3848.39	4008.11	2.25
tai75d	6501.43	4762.54	5091.00	2.18	4762.54	5091.00	2.22	4762.54	5091.00	2.22
tai100a	11398.60	7798.82	8167.83	6.35	7772.71	8042.25	6.37	7733.07	8150.46	6.37
tai100b	9775.59	7002.04	7510.64	6.42	6968.49	7452.22	6.34	6859.95	7398.34	6.34
tai100c	7895.75	4773.88	4945.76	6.35	4773.88	4945.76	6.35	4773.88	4945.76	6.35
tai100d	9051.64	5411.22	5695.49	6.38	5384.50	5627.70	6.36	5357.70	5646.42	6.36
tai150a	16188.04	14048.22	14595.99	67.08	14052.37	14594.23	67.77	14075.20	14559.93	67.77
tai150b	14018.07	11225.23	11802.84	67.15	11225.23	11802.84	68.54	11303.73	11804.18	68.54
tai150c	13293.35	9756.99	10177.14	66.69	9763.81	10151.19	68.12	9789.65	10210.75	68.12
tai150d	13001.42	9843.82	10201.89	67.46	9843.82	10201.89	69.01	9846.68	10199.94	69.01

Table 8. Results for Tai-instances

GRASP + VND. On the other hand, the results from the different values of MD in Table 9 indicate that the distance constraint also affects the quality of solutions.

In the CMT and Kelly instances with the maximum total distance traveled in a route (as in Tables 11 and 12), as expected, our proposed algorithm shows a significant improvement comparing to the initial solutions, with average $Improv$ of 24.01 % to 26.31 %.

Table 12 shows the evolution of the average deviation to the initial solutions during the iterations in some instances. The deviations are 26.07 %, 28.05 %, 28.34 %, 28.61 %, 28.86 %, and 28.86 % for the first local optimum, obtained by one, ten, twenty, thirty, fifty and one-hundred calls VNS, respectively. A major part of the descent obtained is about 0.87 % by fifty to three-hundred calls VNS. We can observe that additional iterations give a minor improvement with the big running time. Hence, the first way to reduce the long running time is to use no more than fifty calls to VNS, and the improvement of our algorithm is about 28.61 %. A much faster option is to run the initial construction phase, then improve it by using a single call to VNS. As as a result, we can obtain an average deviation of 26.07 % and average

Instances	$MD = 2 \times d_{max}$			$MD = 2.5 \times d_{max}$			$MD = 3 \times d_{max}$		
	Gap_2	*Impro*	*T*	Gap_2	*Impro*	*T*	Gap_2	*Impro*	*T*
pr1002_60_x	0.53	21.30	1.12	0.48	21.35	1.45	0.34	21.45	1.45
brd14051_60_x	0.66	25.63	1.13	0.46	25.77	1.45	0.53	25.73	1.45
fnl4461_60_x	0.48	20.26	1.12	0.39	20.33	1.45	0.29	20.41	1.45
d15112_60_x	0.56	23.94	1.11	0.39	24.07	1.46	0.31	24.12	1.46
nrw1379_60_x	0.75	25.65	1.12	0.70	25.69	1.45	0.63	25.74	1.45
pr1002_70_x	0.92	24.55	1.48	0.88	24.58	1.97	0.82	24.62	1.97
brd14051_70_x	0.53	21.62	1.49	0.50	21.65	1.84	0.50	21.66	1.84
fnl4461_70_x	0.66	23.53	1.46	0.62	23.56	1.94	0.59	23.58	1.94
d15112_70_x	0.75	23.70	1.51	0.73	23.73	1.85	0.69	23.70	1.85
nrw1379_70_x	0.72	24.17	1.44	0.64	24.23	1.95	0.61	24.25	1.95
pr1002_80_x	0.73	22.23	3.58	0.68	22.26	4.66	0.65	22.28	4.66
brd14051_80_x	0.62	17.85	3.56	0.57	17.89	4.62	0.56	17.89	4.62
fnl4461_80_x	0.79	20.29	3.59	0.73	20.34	4.63	0.72	20.35	4.63
d15112_80_x	0.43	21.59	3.57	0.41	21.61	4.65	0.39	21.62	4.65
nrw1379_80_x	0.95	25.60	3.53	0.85	25.67	4.67	0.81	25.70	4.67
E-instances	5.29	32.69	1.39	5.29	32.69	1.42	5.29	32.69	1.37
P-instances	3.14	30.18	1.48	3.14	30.18	1.45	3.14	30.18	1.48
Tai-instances	-	29.77	32.79	-	32.42	33.20	-	29.83	33.31
Aver	1.09	24.14	3.69	1.03	24.33	4.23	0.99	24.21	4.23

Table 9. Average results for dataset type 2

Instances	$MD = 2 \times d_{max}$			$MD = 2.5 \times d_{max}$			$MD = 3 \times d_{max}$		
	Gap_1	Gap_2	*T*	Gap_1	Gap_2	*T*	Gap_1	Gap_2	*T*
pr1002_60_x	0.53	21.30	1.12	0.48	21.35	1.45	0.34	21.45	1.45
brd14051_60_x	0.66	20.43	1.13	0.46	24.86	1.45	0.53	24.80	1.45
fnl4461_60_x	0.48	21.78	1.12	0.48	21.78	1.45	0.48	21.78	1.45
d15112_60_x	0.56	21.18	1.11	0.56	21.18	1.46	0.56	21.18	1.46
nrw1379_60_x	0.63	20.98	1.12	0.63	20.98	1.45	0.63	20.98	1.45
pr1002_70_x	0.88	24.58	1.14	0.88	24.58	1.52	0.88	24.58	1.52
brd14051_70_x	0.50	21.65	1.15	0.50	21.65	1.41	0.50	21.66	1.41
fnl4461_70_x	0.62	23.56	1.12	0.62	23.56	1.49	0.62	23.56	1.49
d15112_70_x	0.69	23.70	1.16	0.69	23.70	1.13	0.69	23.70	1.42
nrw1379_70_x	0.64	24.23	1.11	0.64	24.23	1.11	0.61	24.25	1.50
pr1002_80_x	0.68	22.26	3.58	0.73	22.23	3.58	0.65	22.28	3.58
brd14051_80_x	0.57	17.89	3.56	0.62	17.85	3.55	0.56	17.89	3.55
fnl4461_80_x	0.73	20.34	3.59	0.79	20.29	3.56	0.72	20.35	3.56
d15112_80_x	0.41	21.61	3.57	0.39	21.62	3.57	0.39	21.62	3.57
nrw1379_80_x	0.95	25.60	3.53	0.85	25.67	3.59	0.81	25.70	3.59
E-instances	2.28	32.78	1.07	2.14	32.68	1.09	2.14	32.68	1.06
P-instances	3.14	30.18	1.14	3.14	30.18	1.11	3.14	30.18	1.14
Tai-instances	-	29.83	25.20	-	29.91	25.63	-	29.98	25.63
Aver	0.88	23.55	3.14	0.86	23.79	3.31	0.84	23.81	3.35

Table 10. Average results for dataset type 2

Instances	*n*	*k*	*MD*	*Init.Sol*	*Best.Sol*	*Aver.Sol*	*Impro*	*T*
kelly01	240	9	650	72143.84	56121.25	57116.60	22.21	177.04
kelly02	320	10	700	152816.69	104163.33	106346.89	31.84	349.84
kelly04	480	10	1600	353374.40	271826.46	278169.07	23.08	566.42
kelly05	200	5	1800	153187.07	116050.81	118758.33	24.24	179.39
kelly06	280	7	1500	157506.03	127733.90	129785.16	18.90	404.47
kelly07	360	8	1300	218214.03	166321.67	170351.03	23.78	615.73
Aver							24.01	382.15

Table 11. Results for Kelly-instances

Dataset	1 iteration		20 iterations		30 iterations		50 iterations		100 iterations		200 iterations	
	$\overline{impro}$	$\overline{T}$	$\overline{impro}$	$\overline{T}$	$\overline{impro}$	$\overline{T}$	$\overline{impro}$	$\overline{T}$	$\overline{impro}$	$\overline{T}$	$\overline{impro}$	$\overline{T}$
E-instances	27.23	0.16	31.62	0.45	32.24	0.64	32.71	1.07	32.71	2.24	32.71	4.15
P-instances	28.01	0.17	29.80	0.48	30.04	0.68	30.18	1.13	30.18	2.26	30.18	4.72
Tai-instances	28.29	1.83	29.56	12.74	29.84	16.38	29.91	25.49	29.91	52.80	29.91	105.59
CMT	24.98	17.45	25.87	23.92	26.31	27.10	26.31	35.06	26.31	61.03	26.31	141.56
Kelly	22.01	188.22	23.55	259.52	23.88	293.74	24.01	382.15	24.01	667.34	24.01	1544.29
Aver	26.10	41.56	28.08	59.42	28.46	67.71	28.62	88.98	28.62	157.13	28.62	360.06

Table 12. Evolution of average *Impro* deviation to *Init.Sol*

time of 41.46 seconds, even for the instances which are up to 560 customers.

Most previous algorithms are proposed for specific variants; hence, they do not apply for the other variants. However, our proposed algorithm is applicable to multiple variants of MTRPD, although it was not designed for solving them. In comparison with the state-of-the-art algorithms in [15, 23, 28, 29, 30, 31], our TS + VNS algorithm's solutions are at least as good as already the existing CCVRP algorithm in [23, 28, 30], MTRP algorithm in [15, 29], TRP algorithm in [31]. Specifically, for CCVRP problem, our algorithm obtains the better solutions for CMT1, CMT2, CMT3, CMT4 or at least similar solutions for the others in Table 14. For the MTRP problem, as shown in Table 13, the quality of our solutions is much better than the algorithm of Ezzine et al. in [15] and comparable with the algorithm of Nucamendi-Guillén et al. Tables 15 to 18 show the experimental results for the TRP problem. Our algorithm outperforms that of Silva et al. [31] for four instances, and has similar performance for the most of instances in TRP-100-Rx. In TRP-200-Rx, although the VNS + TS metaheuristic cannot find any new best solution, our average solution quality is slightly improved. In addition, our algorithm can find the optimal solutions for the problems with 50 to 100 vertices in several seconds, as shown in Tables 15 and 18 (note that the optimal solutions for the instances are extracted from Abeledo et al. [2]).

Our metaheuristic performs well because of two reasons:

1. The algorithm uses more neighborhoods than the others. Therefore, the explored part of the solution space is more substantial. Hence, the chances of finding even better solutions are higher. The extension of explored part is not time-

Instances	IOE		SNG		Our Algorithm		
	Best.Sol	*T*	*Best.Sol*	*T*	*Best.Sol*	*T*	*cTime*
E-n51-k5	3320	2.25	2209.64	0.70	2209.64	1.31	3.25
E-n76-k10	4094	1.48	2310.09	4.20	2310.09	2.67	6.62
E-n76-k14	3762	0.50	2005.40	3.40	2005.40	2.77	6.87
E-n76-k15	3822	0.09	1962.47	2.81	1962.47	2.74	6.80
E-n101-k8	6383	89.4	-	-	4051.47	6.40	15.87
E-n101-k14	5048	5.43	-	-	3288.53	6.74	16.72
P-n50-k7	-	-	1547.89	0.70	1547.89	1.26	3.12
P-n55-k7	-	-	1766.56	1.01	1766.56	1.39	3.45
P-n60-k10	-	-	1676.35	1.42	1676.35	1.54	3.82
P-n76-k4	-	-	4686.92	3.40	4686.92	2.83	7.02
P-n76-k5	-	-	3820.02	3.63	3820.02	2.78	6.89
CMT1	-	-	2209.64	0.70	2209.64	1.40	3.47
CMT2	-	-	2310.09	4.19	2310.09	2.78	6.89
CMT3	-	-	4002.90	14.94	4002.90	6.40	15.87
tai100a	-	-	7809.43	13.63	7733.07	6.37	15.54
tai100b	-	-	7038.60	12.83	6859.95	6.34	15.47
tai100a	-	-	4868.61	14.12	4786.94	6.35	15.49
tai100b	-	-	5422.63	14.23	5357.70	6.36	15.52

Table 13. Comparision with state-of-the-art metaheuristic algorithm for MTRP ($MD = 0$)

Instances	**MA1**		**ALNS**		**L. Ke's algorithm**		**Our Algorithm**		
	Best.Sol	*T*	*Best.Sol*	*T*	*Best.Sol*	*T*	*Best.Sol*	*T*	*cTime*
CMT1	2230.35	10.63	2230.35	30.29	2230.35	17.64	2209.64	1.40	3.47
CMT2	2421.90	27.78	2391.63	60.77	2391.63	22.48	2310.09	2.78	6.89
CMT3	4073.12	97.91	4045.42	172.4	4045.42	60.96	4002.90	6.40	15.8
CMT4	4987.52	449.4	4987.52	235.1	4987.52	92.68	4953.94	63.0	156.
CMT5	5810.12	1035.	5838.32	277.3	5809.59	135.36	5809.59	11.2	92.7
CMT12	3558.92	53.72	3558.92	38.20	3558.92	152.74	3564.24	9.42	23.3

Table 14. Comparision with state-of-the-art metaheuristic algorithm for CCVRP ($MD = 0$)

consuming because of a constant time operation for calculating the latency cost of each neighboring solution.

2. In some cases, while their algorithms get trapped in cycles, our algorithm overcomes the issue and obtains the better solutions.

In Tables 13–14, the average scaled running time of the VNS + TS algorithm is better than those of Ban et al., Ngueveu et al., and Ribeiro et al., and as well as the algorithm of Ke et al. Besides, it grows quite moderately with the algorithm of Nucamendi-Guillén et al.

Instances $k = 1$ MD=0	*OPT*	**Our Algorithm**			
		Best.Sol	*Aver.Sol*	*T*	*cTime*
TRP-50-R1	12198	**12198**	**12198**	0.49	1.20
TRP-50-R2	11621	**11621**	**11674**	0.57	1.41
TRP-50-R3	12139	**12139**	**12139**	0.61	1.50
TRP-50-R4	13071	**13071**	**13071**	0.61	1.49
TRP-50-R5	12126	**12126**	**12284**	0.58	1.44
TRP-50-R6	12684	**12684**	**12684**	0.55	1.36
TRP-50-R7	11176	**11176**	**11176**	0.59	1.45
TRP-50-R8	12910	**12910**	**12945**	0.61	1.50
TRP-50-R9	13149	**13149**	**13149**	0.62	1.53
TRP-50-R10	12892	**12892**	**12892**	0.62	1.53
TRP-50-R11	12103	**12103**	**12181**	0.61	1.44
TRP-50-R12	10633	**10633**	**10633**	0.61	1.47
TRP-50-R13	12115	**12115**	**12115**	0.56	1.47
TRP-50-R14	13117	**13117**	**13117**	0.58	1.47
TRP-50-R15	11986	**11986**	**11986**	0.61	1.47
TRP-50-R16	12138	**12138**	**12138**	0.58	1.47
TRP-50-R17	12176	**12176**	**12176**	0.49	1.48
TRP-50-R18	13357	**13357**	**13357**	0.57	1.48
TRP-50-R19	11430	**11430**	**11430**	0.61	1.48
TRP-50-R20	11935	**11935**	**11935**	0.58	1.48
Aver				0.58	1.47

Table 15. Comparision with state-of-the-art metaheuristic algorithm for TRP (TPR-50-Rx)

Instances $k = 1$ MD=0	**MS**	Our Algorithm			
	Best.Sol	*Best.Sol*	*Aver.Sol*	*T*	*cTime*
TRP-100-R1	32779	32680	32680	5.67	14.00
TRP-100-R2	33435	31598	31598	5.77	14.25
TRP-100-R3	32390	32390	32390	5.67	14.00
TRP-100-R4	34733	35208	35208	5.98	14.76
TRP-100-R5	32598	32598	32598	5.87	14.50
TRP-100-R6	34159	34159	34159	5.46	13.49
TRP-100-R7	33375	33375	33375	5.05	12.47
TRP-100-R8	31780	32479	32479	5.36	13.23
TRP-100-R9	34167	34167	34167	5.98	14.76
TRP-100-R10	31605	31605	31289	5.26	12.98
TRP-100-R11	34188	34188	34188	5.26	13.84
TRP-100-R12	32146	32146	30487	5.46	13.83
TRP-100-R13	32604	31930	31930	5.05	13.78
TRP-100-R14	32433	32433	32433	5.67	13.76
TRP-100-R15	32574	32574	32574	5.87	13.67
TRP-100-R16	33566	33566	33275	5.26	13.58
TRP-100-R17	34198	34198	34198	5.87	13.59
TRP-100-R18	31929	31929	31929	5.46	13.70
TRP-100-R19	33463	33463	33463	6.08	13.75
TRP-100-R20	33632	33363	33363	5.36	13.65
Aver				5.57	13.72

Table 16. Comparision with state-of-the-art metaheuristic algorithm for TRP (TRP-100-Rx)

Instances $k = 1$ MD=0	**MS**		Our Algorithm			
	Best.Sol	*Aver.Sol*	*Best.Sol*	*Aver.Sol*	*T*	*cTime*
TRP-200-R1	88787	88794.60	88789	88794.25	89.51	218.40
TRP-200-R2	91977	92013.10	91977	91989.30	91.77	223.93
TRP-200-R3	92568	92631.20	92570	92570.90	96.31	234.98
TRP-200-R4	93174	93192.30	93174	93178.60	101.97	248.81
TRP-200-R5	88737	88841.20	88737	88740.65	94.04	229.46
TRP-200-R6	91589	91601.90	91591	91590.50	99.70	243.28
TRP-200-R7	92754	92763.20	92754	92759.85	92.91	226.69
TRP-200-R8	89048	89049.00	89048	89051.25	94.04	229.46
TRP-200-R9	86326	86326.00	86326	86327.70	90.64	221.16
TRP-200-R10	91554	91596.50	91554	91555.51	91.77	223.93
TRP-200-R11	92655	92700.60	92658	92658.22	94.04	229.46
TRP-200-R12	91457	91504.10	91457	91458.43	90.64	221.16
TRP-200-R13	86155	86181.40	86159	86178.31	97.44	237.75
TRP-200-R14	91882	91929.10	91882	91890.95	95.17	232.22
TRP-200-R15	88914	88912.40	88914	88928.95	90.64	221.16
TRP-200-R16	89313	89364.70	89313	89316.21	96.31	234.98
TRP-200-R17	89089	89118.30	89089	89092.93	96.31	234.98
TRP-200-R18	93619	93676.60	93619	93632.77	99.70	243.28
TRP-200-R19	93369	93401.60	93369	93371.85	94.04	229.46
TRP-200-R20	86294	86292.00	86294	86294.35	95.17	232.22
Aver					94.60	230.82

Table 17. Comparision with state-of-the-art metaheuristic algorithm for TRP (TRP-200-Rx)

Instances $k = 1$ MD=0	*n*	*OPT*	*UB*	Our algorithm			
				Best.Sol	*Aver.Sol*	*T*	*cTime*
dantzig42	42	12528	12650	**12528**	12528	0.56	1.50
att48	48	209320	25315	**209320**	209320	1.45	3.87
eil51	51	10178	10593	**10178**	10178	1.56	4.17
berlin52	52	143721	15209	**143721**	143721	1.51	4.03
st70	70	20557	25809	**20557**	20557	2.43	6.49
KroA100	100	983128	10912	**983128**	983128	8.25	22.03
KroB100	100	986008	10212	**986008**	986008	8.12	21.68
KroC100	100	961324	11013	**961324**	961324	8.28	22.11
KroD100	100	976965	10253	**976965**	976965	8.19	21.87
Aver						4.48	11.97

Table 18. Comparision with state-of-the-art metaheuristic algorithm for TRP (TSPLIB)

5 CONCLUSIONS

In this paper, we study the global structure of the MTRPD solution space. We have proposed a new effective meta-heuristic algorithm for MTRPD, which combines Insertion Heuristic (IH), Tabu Search (TS), and Variable Neighborhood Search (VNS). Our algorithm has been suitable for the global structure of the solution space. Moreover, we introduce the novel neighborhoods' structure as well as the constant time operation for efficient calculation of the latency cost for each neighboring solution.

The extensive computational experiments on benchmark instances show that the proposed algorithm is able to find the optimal solutions for all instances with up to 50 vertices in a fraction of seconds. Moreover, almost all the found solutions for instances from 60 to 80 vertices fall into the range of 0.9 %–1.1 % of the lower bounds of the optimal solutions at a reasonable amount of time. For the larger number of vertices, our algorithm obtains good-quality solutions fast. Additionally, our algorithm can find better solutions than the state-of-the-art ones.

REFERENCES

[1] Archer, A.—Levin, A.—Williamson, D. P.: A Faster, Better Approximation Algorithm for the Minimum Latency Problem. SIAM Journal on Computing, Vol. 37, 2008, No. 5, pp. 1472–1498, doi: 10.1137/07068151X.

[2] Abeledo, H.—Fukasawa, R.—Pessoa, A.—Uchoa, E.: The Time Dependent Traveling Salesman Problem: Polyhedra and Algorithm. Mathematical Programming Computation, Vol. 5, 2013, No. 1, pp. 27–55, doi: 10.1007/s12532-012-0047-y.

[3] Afrati, F.—Cosmadakis, S.—Papadimitriou, C. H.—Papageorgiou, G.—Papakostantinou, N.: The Complexity of the Travelling Repairman Problem. Informatique Théorique et Applications, Vol. 20, 1986, No. 1, pp. 79–87.

[4] Arora, S.—Karakostas, G.: Approximation Schemes for Minimum Latency Problems. Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC '99), 1999, pp. 688–693, doi: 10.1145/301250.301432.

[5] Ausiello, G.—Leonardi, S.—Marchetti-Spaccamela, A.: On Salesmen, Repairmen, Spiders and Other Traveling Agents. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (Eds.): Algorithms and Complexity (CIAC 2000). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1767, 2000, pp. 1–16, doi: 10.1007/3-540-46521-9_1

[6] Ban, H. B.—Nguyen, D. N.: Improved Genetic Algorithm for Minimum Latency Problem. Proceedings of the 2010 Symposium on Information and Communication Technology (SoICT '10), 2010, pp. 9–15, doi: 10.1145/1852611.1852614.

[7] Ban, H. B.—Nguyen, K.—Ngo, M. C.—Nguyen, D. N.: An Efficient Exact Algorithm for Minimum Latency Problem. Progress in Informatics, No. 10, 2013, pp. 167–174, doi: 10.2201/NiiPi.2013.10.10.

[8] Ban, H. B.—Nguyen, D. N.: A Meta-Heuristic Algorithm Combining Between Tabu and Variable Neighborhood Search for the Minimum Latency Problem. The 2013 RIVF International Conference on Computing and Communication Technologies – Research, Innovation, and Vision for Future (RIVF), 2013, pp. 192–197, doi: 10.1109/RIVF.2013.6719892.

[9] Ban, H. B.: A GRASP + VND Algorithm for the Multiple Traveling Repairman Problem with Distance Constraints. Journal of Computer Science and Cybernetics, Vol. 33, 2017, No. 3, pp. 272–288, doi: 10.15625/1813-9663/33/3/10511.

[10] Blum, A.—Chalasani, P.—Coppersmith, D.—Pulleyblank, B.—Raghavan, P.—Sudan, M.: The Minimum Latency Problem. Proceedings

of the Twenty-Sixth Annual ACM Symposium on Theory of Computing (STOC '94), 1994, pp. 163–171, doi: 10.1145/195058.195125.

[11] Chaudhuri, K.—Godfrey, B.—Rao, S.—Talwar, K.: Paths, Trees and Minimum Latency Tours. Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03), 2003, pp. 36–45.

[12] Li, C.-L.—Simchi-Levi, D.—Desrochers, M.: On the Distance Constrained Vehicle Routing Problem. Operations Research, Vol. 40, 1992, No. 4, pp. 790–799, doi: 10.1287/opre.40.4.790.

[13] Erera, A. L.—Morales, J. C.—Savelsbergh, M.: The Vehicle Routing Problem with Stochastic Demand and Duration Constraints. Transportation Science, Vol. 44, 2010, No. 4, pp. 474–492, doi: 10.1287/trsc.1100.0324.

[14] Dongarra, J. J.: Performance of Various Computers Using Standard Linear Equations Software. Linpack Benchmark Report, University of Tennessee, Computer Science Technical Report, CS-89-85, 2013.

[15] Ezzine, I. O.—Elloumi, S.: Polynomial Formulation and Heuristic Based Approach for the k-Travelling Repairman Problem. International Journal of Mathematics in Operational Research, Vol. 4, 2012, No. 5, pp. 503–514, doi: 10.1504/IJMOR.2012.048928.

[16] Feo, T. A.—Resende, M. G. C.: Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization, Vol. 6, 1995, No. 2, pp. 109–133, doi: 10.1007/BF01096763.

[17] Fischetti, M.—Laporte, G.—Martello, S.: The Delivery Man Problem and Cumulative Matroids. Operations Research, Vol. 41, 1993, No. 6, pp. 1055–1064, doi: 10.1287/opre.41.6.1055.

[18] Goemans, M.—Kleinberg, J.: An Improved Approximation Ratio for the Minimum Latency Problem. Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '96), 1996, pp. 152–158.

[19] Glover, F.: Tabu Search – Part II. INFORMS Journal on Computing, Vol. 2, 1990, No. 1, pp. 4–32, doi: 10.1287/ijoc.2.1.4.

[20] Fakcharoenphol, J.—Harrelson, C.—Rao, S.: The k-Traveling Repairman Problem. Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '03), 2003, pp. 655–664.

[21] Johnson, D. S.—McGeoch, L. A.: The Traveling Salesman Problem: A Case Study in Local Optimization in Local Search. In: Aarts, E. H. L., Lenstra, J. K. (Eds.): Combinatorial Optimization. John Wiley and Sons, New York, 1997, pp. 215–310.

[22] Jothi, R.—Raghavachari, B.: Minimum Latency Tours and the k-Traveling Repairmen Problem. In: Farach-Colton, M. (Ed.): LATIN 2004: Theoretical Informatics. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2976, 2004, pp. 423–433, doi: 10.1007/978-3-540-24698-5_46.

[23] Ke, L.—Feng, Z.: A Two-Phase Metaheuristic for the Cumulative Capacitated Vehicle Routing Problem. Computers and Operations Research, Vol. 40, 2013, No. 2, pp. 633–638, doi: 10.1016/j.cor.2012.08.020.

[24] Laporte, G.—Nobert, Y.—Desrochers, M.: Optimal Routing under Capacity and Distance Restrictions. Operations Research, Vol. 33, 1985, No. 5, pp. 1050–1073,

doi: 10.1287/opre.33.5.1050.

[25] Luo, Z.—Qin, H.—Lim, A.: Branch-and-Price-and-Cut for the Multiple Traveling Repairman Problem with Distance Constraints. European Journal of Operational Research, Vol. 234, 2014, No. 1, pp. 49–60, doi: 10.1016/j.ejor.2013.09.014.

[26] Martin, O.—Otto, S. W.—Felten, E. W.: Large-Step Markov Chains for the Traveling Salesman Problem. Complex Systems, Vol. 5, 1991, No. 3, pp. 299–326.

[27] Mladenović, N.—Hansen, P.: Variable Neighborhood Search. Computers and Operations Research, Vol. 24, 1997, No. 11, pp. 1097–1100, doi: 10.1016/S0305-0548(97)00031-2.

[28] Ngueveu, S. U.—Prins, C.—Wolfler Calvo, R.: An Effective Memetic Algorithm for the Cumulative Capacitated Vehicle Routing Problem. Computers and Operations Research, Vol. 37, 2010, No. 11, pp. 1877–1885, doi: 10.1016/j.cor.2009.06.014.

[29] Nucamendi-Guillén, S.—Martínez-Salazar, I.—Angel-Bello, F.—Moreno-Vega, J. M.: A Mixed Integer Formulation and an Efficient Metaheuristic Procedure for the k-Travelling Repairmen Problem. Journal of the Operational Research Society, Vol. 67, 2016, No. 8, pp. 1121–1134, doi: 10.1057/jors.2015.113.

[30] Ribeiro, G.—Laporte, G.: An Adaptive Large Variable Neighborhood Search Heuristic for the Cumulative Capacitated Vehicle Routing Problem. Computers and Operations Research, Vol. 39, 2012, No. 3, pp. 728–735, doi: 10.1016/j.cor.2011.05.005.

[31] Silva, M. M.—Subramanian, A.—Vidal, T.—Ochi, L. S.: A Simple and Effective Metaheuristic for the Minimum Latency Problem. European Journal of Operational Research, Vol. 221, 2012, No. 3, pp. 513–520, doi: 10.1016/j.ejor.2012.03.044.

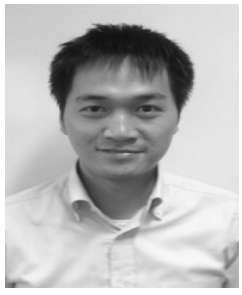
[32] Simchi-Levi, D.—Berman, O.: Minimizing the Total Flow Time of N Jobs on a Network. IIE Transactions, Vol. 23, 1991, No. 3, pp. 236–244, doi: 10.1080/07408179108963858.

[33] Wu, B. Y.—Huang, Z.-N.—Zhan, F.-J.: Exact Algorithms for the Minimum Latency Problem. Information Processing Letters, Vol. 92, 2004, No. 6, pp. 303–309, doi: 10.1016/j.ipl.2004.09.009.

[34] NEO: Capacitated VRP Instances. `http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/`, 2013.

Ha-Bang Ban received his B.E. in information technology in 2006 and the Ph.D. in computer science in 2015, both from the Hanoi University of Science and Technology (HUST), Vietnam. He is currently Lecturer at the School of Information and Communication Technology (SOICT), HUST, Vietnam. His research interests include algorithms, graphs, optimization, logistics, etc. He has published many publications in international peer-reviewed journals and conferences.

Duc-Nghia Nguyen is Associate Professor of computer science at the School of Information and Communication Technology, Hanoi University of Science and Technology (HUST), Vietnam. He received his Ph.D. in computer science in 1988 from Belarusian State University. His current research interests include algorithms and optimization, high performance computing, data science.



Kien Nguyen received his B.E. in electronics and telecommunications from Hanoi University of Science and Technology (HUST), Vietnam, in 2004, and the Ph.D. in informatics from the Graduate University for Advanced Studies, Japan, in 2012. He is currently Assistant Professor at the Graduate School of Engineering, Chiba University, Japan. Before joining Chiba University, he was a researcher at the National Institute of Information and Communications Technology (NICT), Japan, during 2014–2018. His research interests include communication networks, the Internet, and the Internet of Things (IoT). He has published 70+ publications in international peer-reviewed journals and conferences. Besides, he has co-authored submitted patents and Internet Engineering Task Force (IETF) Internet drafts. He is a member of IEICE and a senior member of IEEE.

Computing and Informatics, Vol. 38, 2019, 917–936, doi: 10.31577/cai_2019_4_917

HSIC REGULARIZED LTSA

Xinghua Zheng

School of Data and Computer Science, Sun Yat-sen University
Guangzhou, 510275, China
e-mail: 380567579@qq.com

Zhengming Ma*, Hangjian Che

School of Electronics and Information Technology, Sun Yat-sen University
Guangzhou, 510275, China
e-mail: issmzm@mail.sysu.edu.cn, 1349831461@qq.com

Lei Li

School of Data and Computer Science, Sun Yat-sen University
Guangzhou, 510275, China
e-mail: leili525@126.com

Abstract. Hilbert-Schmidt Independence Criterion (HSIC) measures statistical independence between two random variables. However, instead of measuring the statistical independence between two random variables directly, HSIC first transforms two random variables into two Reproducing Kernel Hilbert Spaces (RKHS) respectively and then measures the kernelled random variables by using Hilbert-Schmidt (HS) operators between the two RKHS. Since HSIC was first proposed around 2005, HSIC has found wide applications in machine learning. In this paper, a HSIC regularized Local Tangent Space Alignment algorithm (HSIC-LTSA) is proposed. LTSA is a well-known dimensionality reduction algorithm for local homeomorphism preservation. In HSIC-LTSA, behind the objective function of LTSA, HSIC between high-dimensional and dimension-reduced data is added as a regularization term. The proposed HSIC-LTSA has two contributions. First,

* Corresponding author

HSIC-LTSA implements local homeomorphism preservation and global statistical correlation during dimensionality reduction. Secondly, HSIC-LTSA proposes a new way to apply HSIC: HSIC is used as a regularization term to be added to other machine learning algorithms. The experimental results presented in this paper show that HSIC-LTSA can achieve better performance than the original LTSA.

Keywords: Dimensionality reduction, RKHS, Hilbert-Schmidt operators, LTSA, HSIC

1 INTRODUCTION

The loss of information is inevitable during dimensionality reduction. Therefore, the main concern in constructing algorithms of dimensionality reduction is what information needs to be preserved during dimensionality reduction. From this viewpoint, the algorithms of dimensionality reduction can be divided into two categories: global-preserving and local-preserving algorithms [1]. The global-preserving algorithms preserve some global features of data during dimensionality reduction [2, 3, 4, 5], while the local-preserving algorithms preserve some local features of data during dimensionality reduction. Local Tangent Space Alignment (LTSA) algorithm is a typical local-preserving algorithm for dimensionality reduction. The local feature LTSA preserves is the local homeomorphism, i.e., the continuous dependence between data within a local region [6]. In recent years, the dimensionality reduction algorithms capable of preserving both local and global features have emerged, such as LPP [7, 8, 9].

Hilbert-Schmidt Independence Criterion (HSIC) measures the statistical independence between two random variables [10]. However, instead of measuring the statistical independence between two random variables directly, HSIC first transforms the two random variables into two reproducing kernel Hilbert spaces (RKHS) respectively and then measures the statistical correlation of the two transformed random variables by using Hilbert-Schmidt operators between two RKHSs. In the application of HSIC to data analysis, the given data can be regarded as the values taken by the random variables. The HSIC formulae for calculating the statistical correlation of data are simple and often used in many applications [11, 12, 13, 14, 15]. However, HSIC involves many mathematical concepts and it is not easy to understand the meaning of HSIC thoroughly. The misunderstanding, or even misuse of HSIC happens from time to time.

In this paper, HSIC is first explored theoretically and then applied to LTSA. LTSA is a local homeomorphism-preserving algorithm for dimensionality reduction. An improved LTSA, called HSIC regularized LTSA, or HSIC-LTSA for short, is proposed in which a HSIC regularization term is added to LTSA's objective function. The HSIC regularization term measures the statistical correlation between the high-

dimensional data and the dimension-reduced data. HSIC-LTSA takes into account both the local and global preserving requirements during dimensionality reduction and achieves a better result than LTSA.

The remaining sections in this paper are arranged as follows: in Section 2, LTSA is elaborated, showing that LTSA is a local-homeomorphism preserving algorithm in nature; in Section 3, RKHS is briefly described; in Section 4, the theory of HSIC is elaborated thoroughly and the HSIC formulae for calculating the statistical correlation between two sets of data is derived. In Section 5, an improved HSIC-LTSA is proposed; in Section 6, the experimental results of LTSA and HSIC-LTSA are presented to show the effectiveness of HSIC-LTSA; in Section 7, some conclusions are presented.

2 LOCAL TANGENT SPACE ALIGNMENT (LTSA)

LTSA [6] is a classical local homeomorphism-preserving algorithm of manifold learning and mainly applied to dimensionality reduction. Generally speaking, the problem of dimensionality reduction can be expressed as follows: given a set of high-dimensional data $X = \{x_1, \ldots, x_N\} \subseteq R^D$, we want to find an another set of data $Y = \{y_1, \ldots, y_N\} \subseteq R^d$ such that y_n is the dimensional-reduced version of x_n, where $d << D$ and $n = 1, \ldots, N$. In manifold learning, Y is also called the global coordinate of X.

Remark 1. In this paper, a dataset can be represented by a set, in which the elements of the set are data, for example, $X = \{x_1, \ldots, x_N\} \subseteq R^D$. The dataset can also be represented by a matrix, in which the column vectors of matrix are data, for example, $X = [x_1, \ldots, x_N] \in R^{D \times N}$. The two representations are equivalent.

The stages of LTSA are as follows:

1. Decompose the high-dimensional data into local groups: LTSA uses K-NN method. For each data x_n, let $x_{n_1}, \ldots, x_{n_K}$ be its K-nearest neighbors, then the n^{th} local group is as follows:

$$X_n = \left[x_{n_1} \ldots x_{n_K}, x_{n_{K+1}}\right] \in R^{D \times (K+1)} \tag{1}$$

   where $x_{n_{K+1}} = x_n$, $n = 1, \ldots, N$. It is clear that $X = \bigcup_{n=1}^{N} X_n$.
2. Reduce the dimension of each local group X_n: LTSA uses PCA method. The local group X_n is first centralized:

$$\hat{X}_n = \left[x_{n_1} - \bar{x}_n, x_{n_{K+1}} - \bar{x}_n\right] = X_n C_{K+1} \tag{2}$$

   where $\bar{x}_n = \frac{1}{K+1} \sum_{k=1}^{K+1} x_{n_k}$, $C_{K+1} = I_{K+1} - \frac{1}{K+1} \Gamma_{K+1} \Gamma_{K+1}^T$, $\Gamma_{K+1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \in R^{K+1}$. C_{K+1} is often called centralizing matrix.

Then, the centralized matrix $\hat{X}_n$ is SVD-decomposed:

$$\hat{X}_n = U_n \Sigma_n V_n^T \tag{3}$$

where both $U_n \in R^{D\times D}$ and $V_n \in R^{(K+1)\times(K+1)}$ are orthonormal matrices, $\Sigma_n \in R^{D\times(K+1)(K+1)}$ is singular value matrix.

At last, let $U_{n,1} \in R^{D\times d}$ be the matrix composed of the first d column vectors of U_n, then the column vectors of $\hat{X}_n$ are projected into the space spanned by the column vectors of $U_{n,1}$, i.e., $spanU_{n,1}$, the coordinates of these projections in $spanU_{n,1}$ are the dimensional-reduced version of X_n:

$$\Theta_n = U_{n,1}^T \hat{X}_n \in R^{d\times(K+1)}. \tag{4}$$

In manifold learning, Θ_n is often called the local coordinate of X_n.

Remark 2. From the viewpoint of manifold, the space $spanU_{n,1}$ can be regarded as the tangent space [16] of the point $\bar{x}_n$, therefore LTSA is called local tangent space method. Furthermore, since $\hat{X}_n$ and Θ_n are homeomorphic [17] to each other within the neighborhood of $\bar{x}_n$, therefore, LTSA belongs to the category of local preserving algorithms.

3. Derive the global coordinate from the local coordinate: Let us denote the global coordinate of X_n as

$$Y_n = \left[y_{n_1} \dots y_{n_{K+1}}\right] \in R^{d\times(K+1)} \tag{5}$$

where y_{n_k} is the global coordinate of x_{n_k}, i.e., the dimensional-reduced version of x_{n_k}, $1 \le n_k \le N$, $k = 1, \dots, K+1$. We want to derive Y_n from Θ_n. LTSA assumes that Y_n is the linear transformation of Θ_n (affine transformation, strictly speaking):

$$\hat{Y}_n = \left[y_{n_1} - \bar{y}_n \dots y_{n_{K+1}} - \bar{y}_n\right] = Y_n C_{K+1} = A_n \Theta_n \tag{6}$$

where $\bar{y}_n = \frac{1}{K+1}\sum_{k=1}^{K+1} y_{n_k}$. The geometric meaning of Equation (6) is that Y_n can be derived from Θ_n by translation, rotation and scale. Furthermore,

$$\hat{Y}_n = A_n \Theta_n \Rightarrow A_n = \hat{Y}_n \Theta_n^+ \tag{7}$$

where Θ_n^+ represents the right pseudo inverse of Θ_n, i.e., Θ_n^+ is the solution to the following problem:

$$\left\|I_d - \Theta_n \Theta_n^+\right\|^2 \xrightarrow[choose\ \Theta_n^+]{} \min. \tag{8}$$

Based on Equation (7), the local objective function can be established:

$$\left\|\hat{Y}_n - A_n\Theta_n\right\|^2 = \left\|\hat{Y}_n\left(I_{K+1} - \Theta_n^+\Theta_n\right)\right\|^2 = \left\|Y_n C_{K+1}\left(I_{K+1} - \Theta_n^+\Theta_n\right)\right\|^2$$

$$= \left\| Y S_n C_{K+1} \left(I_{K+1} - \Theta_n^+ \Theta_n \right) \right\|^2 = \left\| Y L_n \right\|^2 \xrightarrow[choose\ Y]{} \min \tag{9}$$

where $S_n \in R^{N \times (K+1)}$ is the selection matrix such that $Y_n = Y S_n$, i.e., the n_k^{th} element of the k^{th} column vector is 1, other elements are 0, $k = 1, \ldots, K+1$; $L_n = S_n C_{K+1} \left(I_{K+1} - \Theta_n^+ \Theta_n \right)$, called the local pattern of X.

The objective function of LTSA can be derived by summing up all the local objective functions:

$$\begin{aligned} \sum_{n=1}^{N} \left\| Y L_n \right\|^2 &= \sum_{n=1}^{N} tr \left(Y L_n L_n^T Y^T \right) = tr \left(Y \sum_{n=1}^{N} L_n L_n^T Y^T \right) \\ &= tr \left(Y L L^T Y^T \right) \xrightarrow[choose\ Y]{} \min \end{aligned} \tag{10}$$

where $L = [L_1 \ldots L_N]$.

3 REPRODUCING KERNEL HILBERT SPACES (RKHS)

HSIC is based on RKHS. Let $L^2(\Omega) = \left\{ f \,\middle|\, f : \Omega \to R, \int_\Omega |f(x)|^2 < +\infty \right\}$ be the space of square integrable functions. An inner product $\langle \bullet, \bullet \rangle$ can be defined over $L^2(\Omega)$ [18]:

$$\langle f, g \rangle = \int_\Omega f(x)\, g(x)\, dx. \tag{11}$$

It can be proven that $H = \left(L^2(\Omega), \langle \bullet, \bullet \rangle \right)$ is a complete inner product space, i.e., Hilbert space.

Definition 1 ([18])**.** Let $H = \left(L^2(\Omega), \langle \bullet, \bullet \rangle \right)$, if there is a function $k : \Omega \times \Omega \to R$ such that

- for all $x \in \Omega$, $k_x = k(\bullet, x) \in H$,
- for all $f \in H$, $f(x) = \langle f, k(\bullet, x) \rangle$,

then H is called a reproducing kernel Hilbert space (RKHS) and k is called the reproducing kernel of H.

The reproducing kernel k can be used to define a map: $\varphi : \Omega \to H$ such that for all $x \in \Omega$,

$$\varphi(x) = k(\bullet, x) \in H. \tag{12}$$

It can be easily proven that

$$\langle \varphi(x), \varphi(y) \rangle = \langle k_x, k(\bullet, y) \rangle = k_x(y) = k(y, x) = k(x, y). \tag{13}$$

The above equation is often used in many kernel methods of machine learning such as kPCA [3], kLDA [19], kSVM [20], etc.

Furthermore, if X is a random variable on Ω, then $\varphi(X)$ is a random process and its mean function is defined:

$$\mu_X(u) = E_X[\varphi(X)(u)] = E_X[k(u,X)] = \int_\Omega k(u,x)\, p_X(x)\, \mathrm{d}x. \tag{14}$$

Then, for all $f \in H$,

$$\begin{aligned}\langle \mu_X, f\rangle &= \int_\Omega \mu_X(u)\, f(u)\, \mathrm{d}u = \int_\Omega \left(\int_\Omega k(u,x)\, p_X(x)\, \mathrm{d}x\right) f(u)\, \mathrm{d}u \\ &= \int_\Omega \left(\int_\Omega k(u,x)\, f(u)\, \mathrm{d}u\right) p_X(x)\, \mathrm{d}x = \int_\Omega \langle f, k(\bullet, x)\rangle\, p_X(x)\, \mathrm{d}x \\ &= \int_\Omega f(x)\, p_x(x)\, \mathrm{d}x = E_x[f(X)]. \end{aligned} \tag{15}$$

In mathematics, it can be proven that RKHS can be generated from kernel functions. The definition of kernel functions is as follows.

Definition 2 ([21])**.** Let $k : \Omega \times \Omega \to R$, if k satisfies the following conditions:

- Symmetric: for all $x, y \in \Omega$, $k(x,y) = k(y,x)$,
- Square integrable: for all $x \in \Omega$, $k_x = k(\bullet, x)$ is square integrable,
- Positive definite: for all $x_1, \ldots, x_N \in \Omega$, the matrix $\begin{bmatrix} k(x_1,x_1) & \ldots & k(x_1,x_N) \\ \vdots & \ddots & \vdots \\ k(x_N,x_1) & \ldots & k(x_N,x_N) \end{bmatrix}$ is positive definite,

then k is called a kernel function.

Remark 3. Kernel functions and reproducing kernels are not the same concept. Kernel functions are defined on their own, while reproducing kernels are defined based on RKHS.

Theorem 1 ([18])**.** A kernel function k can be used to generate a unique RHHS H_k such that k becomes the reproducing kernel of H_k.

Based on this theorem, as long as we define a kernel function, we define an RKHS.

4 HILBERT-SCHMIDT INDEPENDENCE CRITERION (HSIC)

4.1 HS Operators

HSIC is defined by using Hilbert-Schmidt operators.

Definition 3 ([22]). Let H_X and H_Y be two separable Hilbert spaces, $\{e_i \,|i \in I\}$ the orthonormal basis of H_X, $T : H_X \to H_Y$ a compact operator, if $\sum_{i\in I} \|Te_i\|_Y^2 < +\infty$, then T is called a Hilbert-Schmidt (HS) operator.

Remark 4. In this paper, $\langle \bullet, \bullet \rangle_X$ represents the inner product of H_X, $\|\bullet\|_X = \sqrt{\langle \bullet, \bullet \rangle_X}$ the norm of H_X. Similarly, $\langle \bullet, \bullet \rangle_Y$ represents the inner product of H_Y, $\|\bullet\|_Y = \sqrt{\langle \bullet, \bullet \rangle_Y}$ the norm of H_Y.

Let $HS\,(H_X \to H_Y)$ be the space of all HS operators from H_X to H_Y. An inner product $\langle \bullet, \bullet \rangle_{HS}$ can be defined on $HS\,(H_X \to H_Y)$ to make $(HS(H_X \to H_Y), \langle \bullet, \bullet \rangle_{HS})$ become a Hilbert space.

Theorem 2 ([10]). If for all $T, S \in HS\,(H_X \to H_Y)$, $\sum_{i\in I} |\langle Te_i, Se_i \rangle_Y| < +\infty$, then $(HS\,(H_X \to H_Y), \langle \bullet, \bullet \rangle_{HS})$ is a Hilbert space, where the inner product $\langle \bullet, \bullet \rangle_{HS}$ is defined as follows:

$$\langle T, S \rangle_{HS} = \sum_{i\in I} \langle Te_i, Se_i \rangle_Y. \tag{16}$$

Tensor product operators are a kind of HS operators.

Theorem 3 ([10]). Let H_X and H_Y be two separable Hilbert spaces, $f_0 \in H_X$, $g_0 \in H_X$, define $f_0 \otimes g_0 : H_X \to H_Y$ such that for all $f \in H_X$, $f_0 \otimes g_0\,(f) = \langle f_0, f \rangle_X g_0 \in H_Y$, then $f_0 \otimes g_0$ is a HS operator, i.e., $f_0 \otimes g_0 \in HS\,(H_X \to H_Y)$.

Remark 5. $f_0 \otimes g_0$ is called the tensor product of f_0 and g_0.

4.2 Cross Covariance Operators

Generally speaking, HSIC involves two RKHSs.

Let $H_1 = (L^2\,(\Omega_1), \langle \bullet, \bullet \rangle_1)$ be an RKHS, $k_1 : \Omega_1 \times \Omega_1 \to R$ the reproducing kernel of H_1. Define $\varphi_1 : \Omega_1 \to H_1$ such that for all $x \in \Omega_1$, $\varphi_1\,(x) = k_1\,(\bullet, x) \in H_1$. Note that $\langle \varphi_1\,(x'), \varphi_1\,(x'') \rangle_1 = k_1\,(x', x'')$.

Similarly, let $H_2 = (L^2\,(\Omega_2), \langle \bullet, \bullet \rangle_2)$ be an RKHS, $k_2 : \Omega_2 \times \Omega_2 \to R$ the reproducing kernel of H_2. Define $\varphi_2 : \Omega_2 \to H_2$ such that for all $y \in \Omega_2$, $\varphi_2\,(y) = k_2\,(\bullet, y) \in H_2$. Note that $\langle \varphi_2\,(y'), \varphi_2\,(y'') \rangle_2 = k_2\,(y', y'')$.

Furthermore, let X be a random variable on Ω_1, Y a random variable on Ω_2.

Theorem 4 ([10]). Let $\Phi : HS\,(H_1 \to H_2) \to R$ such that for all $T \in HS(H_1 \to H_2)$

$$\Phi\,(T) = E_{XY}\,[\langle \varphi_1\,(X) \otimes \varphi_2\,(Y), T \rangle_{HS}]. \tag{17}$$

If $E_{XY}\,[\|\varphi_1\,(X) \otimes \varphi_2\,(Y)\|_{HS}] < +\infty$, then Φ is continuous linear functional on $HS\,(H_1 \to H_2)$.

According to the representation theorem of continuous linear functionals (Riesz theorem [18]), there must be a unique HS operator $T_\Phi \in HS\,(H_X \to H_Y)$ such that for all HS operators $T \in HS\,(H_X \to H_Y)$,

$$\Phi\,(T) = E_{XY}\,[\langle \varphi_1\,(X) \otimes \varphi_2\,(Y), T \rangle_{HS}] = \langle T, T_\Phi \rangle_{HS}. \tag{18}$$

This HS operator T_Φ is called as cross covariance operator and often denoted as C_{XY}.

4.3 Hilbert-Schmidt Independence Criterion (HSIC)

Definition 4 ([10])**.** The HSIC of two random variables X and Y is defined as

$$HSIC(X,Y) = E_{XY}\left[\|(\varphi_1(X)-\mu_X)\otimes(\varphi_2(Y)-\mu_Y)\|_{HS}^2\right]. \tag{19}$$

It can be easily proven [10] that:

$$\begin{aligned} HSIC(X,Y) &= E_{XY}\left[\|(\varphi_1(X)-\mu_X)\otimes(\varphi_2(Y)-\mu_Y)\|_{HS}^2\right] \\ &= \|C_{XY}-\mu_X\otimes\mu_Y\|_{HS}^2 \\ &= \langle C_{XY},C_{XY}\rangle_{HS} - 2\langle C_{XY},\mu_X\otimes\mu_Y\rangle_{HS} \\ &\quad + \langle\mu_X\otimes\mu_Y,\mu_X\otimes\mu_Y\rangle_{HS}. \end{aligned} \tag{20}$$

In practice, two sets of data $\{x_1,\ldots,x_N\}\subseteq\Omega_1$ and $\{y_1,\ldots,y_N\}\subseteq\Omega_2$ are given and can be regarded as some sample taken by the random variables X and Y. Therefore, the calculation of HSIC can be approximated by replacing statistical average with sample average [10].

At first, for all HS operators $T\in HS(H_1\to H_2)$, since

$$\begin{aligned} \langle C_{XY},T\rangle_{HS} &= E_{XY}\left[\langle\varphi_1(X)\otimes\varphi_2(Y),T\rangle_{HS}\right] \\ &\approx \frac{1}{N}\sum_{n=1}^{N}\langle\varphi_1(x_n)\otimes\varphi_2(y_n),T\rangle_{HS} \\ &= \left\langle\frac{1}{N}\sum_{n=1}^{N}\varphi_1(x_n)\otimes\varphi_2(y_n),T\right\rangle_{HS} \end{aligned} \tag{21}$$

then

$$C_{XY}\approx\frac{1}{N}\sum_{n=1}^{N}\varphi_1(x_n)\otimes\varphi_2(y_n). \tag{22}$$

Similarly, for all functions $f\in H_1$, since

$$\langle f,\mu_X\rangle_1 = E_X\left[\langle\varphi_1(X),f\rangle_1\right]\approx\frac{1}{N}\sum_{n=1}^{N}\langle\varphi_1(x_n),f\rangle_1 = \left\langle\frac{1}{N}\sum_{n=1}^{N}\varphi_1(x_n),f\right\rangle_1$$

then

$$\mu_X\approx\frac{1}{N}\sum_{n=1}^{N}\varphi_1(x_n). \tag{23}$$

By the same deduction, we have

$$\mu_Y \approx \frac{1}{N}\sum_{n=1}^{N}\varphi_2(y_n). \tag{24}$$

Substituting Equations (34), (35), (36) into Equations (31), (32), (33) gives:

$$\begin{aligned}\langle C_{XY}, C_{XY}\rangle_{HS} &\approx \left\langle \frac{1}{N}\sum_{i=1}^{N}\varphi_1(x_i)\otimes\varphi_2(y_i), \frac{1}{N}\sum_{j=1}^{N}\varphi_1(x_j)\otimes\varphi_2(y_j)\right\rangle_{HS} \\ &= \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}k_1(x_i,x_j)\,k_2(y_i,y_j) = \frac{1}{N^2}tr(K_1K_2), \end{aligned}\tag{25}$$

$$\begin{aligned}\langle C_{XY}, \mu_X\otimes\mu_Y\rangle_{HS} &\approx \left\langle \frac{1}{N}\sum_{i=1}^{N}\varphi_1(x_i)\otimes\varphi_2(y_i),\right. \\ &\quad \left.\left(\frac{1}{N}\sum_{p=1}^{N}\varphi_1(x_p)\right)\otimes\left(\frac{1}{N}\sum_{q=1}^{N}\varphi_2(y_q)\right)\right\rangle_{HS} \\ &= \frac{1}{N^3}\sum_{i=1}^{N}\sum_{p=1}^{N}\sum_{q=1}^{N}k_1(x_i,x_p)\,k_2(y_i,y_q) = \frac{1}{N^3}\Gamma_N^T K_1K_2\Gamma_N \end{aligned}\tag{26}$$

$$\begin{aligned}&\langle \mu_X\otimes\mu_Y, \mu_X\otimes\mu_Y\rangle_{HS} = \langle\mu_X,\mu_X\rangle_1\langle\mu_Y,\mu_Y\rangle_2 \\ &\approx \left\langle \frac{1}{N}\sum_{i=1}^{N}\varphi_1(x_i), \frac{1}{N}\sum_{j=1}^{N}\varphi_1(x_j)\right\rangle_1 \left\langle \frac{1}{N}\sum_{i=1}^{N}\varphi_2(y_i), \frac{1}{N}\sum_{j=1}^{N}\varphi_2(y_j)\right\rangle_2 \\ &= \frac{1}{N^4}\Gamma_N^T K_1\Gamma_N\Gamma_N^T K_2\Gamma_N \end{aligned}\tag{27}$$

where

$$K_1 = \begin{bmatrix} k_1(x_1,x_1) & \dots & k_1(x_1,x_N) \\ \vdots & \ddots & \vdots \\ k_1(x_N,x_1) & \dots & k_1(x_N,x_N)\end{bmatrix}, \quad K_2 = \begin{bmatrix} k_2(y_1,y_1) & \dots & k_2(y_1,y_N) \\ \vdots & \ddots & \vdots \\ k_2(y_N,y_1) & \dots & k_2(y_N,y_N)\end{bmatrix}. \tag{28}$$

Substituting (37), (38), (39) into Equation (30) gives:

$$\begin{aligned}HSIC(X,Y) &= \langle C_{XY},C_{XY}\rangle_{HS} - 2\langle C_{XY}, \mu_X\otimes\mu_Y\rangle_{HS} + \langle\mu_X\otimes\mu_Y,\mu_X\otimes\mu_Y\rangle_{HS} \\ &\approx \frac{1}{N^2}tr(K_1K_2) - \frac{2}{N^3}\Gamma_N^T K_1K_2\Gamma_N + \frac{1}{N^4}\Gamma_N^T K_1\Gamma_N\Gamma_N^T K_2\Gamma_N \qquad (29) \\ &= \frac{1}{N^2}tr(K_2C_NK_1C_N) \qquad (30)\end{aligned}$$

where $C_N = I_N - \frac{1}{N}\Gamma_N\Gamma_N^T$ is the centralized matrix.

5 HSIC REGULARIZED LTSA (HSIC-LTSA)

5.1 The Objective Function of HSIC-LTSA

In manifold learning, LTSA is among the few algorithms which are created based on the mathematical properties of manifolds. Therefore, LTSA achieves better performance in the process of manifold data. However, the so-called manifolds are topological spaces which are locally homeomorphic to Euclidean spaces. Therefore, it is natural for LTSA to be a local homeomorphism-preserving algorithm. Many improvements to LTSA try to turn LTSA into one capable of preserving both local and global properties of data during dimensionality reduction. For example, in [23], the dimension-reduced data Y are set to the linear transformation of the high dimensional data X, i.e., $Y = WX$, where $W \in R^{d\times D}$. Y is then replaced with $Y = WX$ in the objective function of LTSA:

$$tr\left(YLL^TY^T\right) \xrightarrow[\text{choose } Y]{} \min \Rightarrow tr\left(WXLL^TX^TW^T\right) \xrightarrow[\text{choose } W]{} \min \tag{31}$$

However, the setting $Y = WX$ will destroy the nonlinear nature of LTSA.

In this paper, an improved LTSA, called HSIC regularized LTSA (HSIC-LTSA for short), is proposed in which a HSIC regularization term is added to the objective function of LTSA:

$$tr\left(YLL^TY^T\right) - \lambda HSIC\left(X,Y\right) = tr\left(YLL^TY^T\right) - \lambda tr\left(K_2C_NK_1C_N\right) \xrightarrow[\text{choose } Y]{} \min \tag{32}$$

where $\lambda > 0$ is the regularization coefficient.

$HSIC\left(X,Y\right)$ measures the statistical dependence of two random processes $\varphi_1\left(X\right)$ and $\varphi_2\left(Y\right)$. Therefore, the objective function HSIC-LTSA shown in Equation (32) means that X and Y will be kept statistically dependent as much as possible during dimensionality reduction of LTSA.

Furthermore, the dimension-reduced data Y is hidden in the kernel matrix K_2 in $HSIC\left(X,Y\right)$. In order to facilitate the optimization of Y, the proposed HSIC-LTSA sets the kernel function k_2 based on the linear kernel: $k_2 : R^d \times R^d \to R$, for all $y', y'' \in R^d$,

$$k_2\left(y', y''\right) = {y'}^T y'' + \kappa\delta\left(y', y''\right) \tag{33}$$

where $\kappa > 0$ and $\delta\left(y', y''\right) = \begin{cases} 1, & y' = y'' \\ 0, & \text{others} \end{cases}$. The addition of δ ensures the positive

definiteness of k_2. The kernel matrix K_2 is then to be:

$$K_2 = \begin{bmatrix} k_2(y_1, y_1) & \cdots & k_2(y_1, y_N) \\ \vdots & \ddots & \vdots \\ k_2(y_N, y_1) & \cdots & k_2(y_N, y_N) \end{bmatrix} = \begin{bmatrix} y_1^T y_1 & \cdots & y_1^T y_N \\ \vdots & \ddots & \vdots \\ y_N^T y_1 & \cdots & y_N^T y_N \end{bmatrix} + \kappa I_N = Y^T Y + \kappa I_N. \tag{34}$$

In this setting of K_2, $HSIC(X, Y)$ will become:

$$\begin{aligned} HSIC(X, Y) &= tr(K_2 C_N K_1 C_N) = tr\left(Y^T Y C_N K_1 C_N\right) + \kappa tr(C_N K_1 C_N) \\ &= tr\left(Y C_N K_1 C_N Y^T\right) + \kappa tr(C_N K_1 C_N). \end{aligned} \tag{35}$$

$tr(C_N K_1 C_N)$ has nothing to do with Y, therefore the objective function of HSIC-LTSA becomes:

$$tr\left(Y L L^T Y^T\right) - \lambda tr\left(Y C_N K_1 C_N Y^T\right) \xrightarrow[\text{choose } Y]{} \min \tag{36}$$

where

$$K_1 = \begin{bmatrix} k_1(x_1, x_1) & \cdots & k_1(x_1, x_N) \\ \vdots & \ddots & \vdots \\ k_1(x_N, x_1) & \cdots & k_1(x_N, x_N) \end{bmatrix}. \tag{37}$$

The kernel function k_1 can be chosen according to the applications at hand. Therefore, HSIC-LTSA provides much flexibility for different applications.

5.2 The Solution to HSIC Regularized LTSA

The objective function of HSIC-LTSA shown in Equation (37) can be rewritten in an equivalent form:

$$\frac{tr\left(Y L L^T Y^T\right)}{tr\left(Y C_N K_1 C_N Y^T\right)} \xrightarrow[\text{choose } Y]{} \min. \tag{38}$$

In Equation (38), since for all constant vectors $z \in R^N$, $C_N z = 0$, $C_N K_1 C_N$ is then positive semi-definite, not positive definite. However, from another viewpoint, C_N is the centralizing matrix, YC_N means the centralization of Y. In geometry, YC_N means translation of Y to the origin of the Euclidean space R^n. Obviously, the translation of Y has no impact on the result of dimensionality reduction. Therefore, it is reasonable to assume that $YC_N = Y$. Under this assumption, the objective function shown in Equation (38) can be refined as follows:

$$\frac{tr\left(Y L L^T Y^T\right)}{tr\left(Y K_1 Y^T\right)} \xrightarrow[\text{choose } Y]{} \min.$$

Equation (38) can be solved according to the following stages:

1. Cholesky Decomposition of K_1: the kernel function of K_1 is symmetric and positive definite, and can be Cholesky-decomposed:

$$K_1 = VV^T \tag{39}$$

where $V \in R^{N\times N}$ is a low-triangular matrix and the diagonal elements are all positive.

2. Let $Z = YV \in R^{d\times N}$, then $Y = ZV^{-1}$ and

$$\frac{tr\left(YLL^TY^T\right)}{tr\left(YK_1Y^T\right)} = \frac{tr\left(YLL^TY^T\right)}{tr\left(YVV^TY^T\right)} = \frac{tr\left(ZV^{-1}LL^T(V^{-1})^TZ^T\right)}{tr\left(ZZ^T\right)}. \tag{40}$$

Furthermore, let us denote $Z = \begin{bmatrix} Z_{1Row} \\ \vdots \\ Z_{dRow} \end{bmatrix}$, where $Z_{iRow} \in R^{1\times N}$ represents the row vector of Z, $i = 1, \ldots, d$, then

$$\frac{tr\left(ZV^{-1}LL^T(V^{-1})^TZ^T\right)}{tr\left(ZZ^T\right)} = \frac{\sum_{i=1}^{d} Z_{iRow}V^{-1}LL^T(V^{-1})^TZ_{iRow}^T}{\sum_{i=1}^{d} Z_{iRow}Z_{iRow}^T}. \tag{41}$$

3. Eigen Decomposition of $V^{-1}LL^T(V^{-1})^T$. If Z_{iRow}^T is an eigenvector of $V^{-1}LL^T(V^{-1})^T$, i.e.,

$$V^{-1}LL^T\left(V^{-1}\right)^TZ_{iRow}^T = \lambda_i Z_{iRow}^T \tag{42}$$

then

$$\lambda_{\min} \le \frac{\sum_{i=1}^{d} Z_{iRow}V^{-1}LL^T(V^{-1})^TZ_{iRow}^T}{\sum_{i=1}^{d} Z_{iRow}Z_{iRow}^T} = \frac{\sum_{i=1}^{d} \lambda_i Z_{iRow}Z_{iRow}^T}{\sum_{i=1}^{d} Z_{iRow}Z_{iRow}^T} \le \lambda_{\max} \tag{43}$$

where $\lambda_{\max}$ and $\lambda_{\min}$ represent the maximum and minimum eigenvalues of $V^{-1}LL^T(V^{-1})^T$, respectively.

It is clear that the d row vectors of Z should be chosen to be the eigenvectors corresponding to the d minimum eigenvalues of $V^{-1}LL^T(V^{-1})^T$.

4. $Y = ZV^{-1}$.

6 EXPERIMENTS

In this section, some experimental results of LTSA and HSIC-LTSA are presented for comparison.

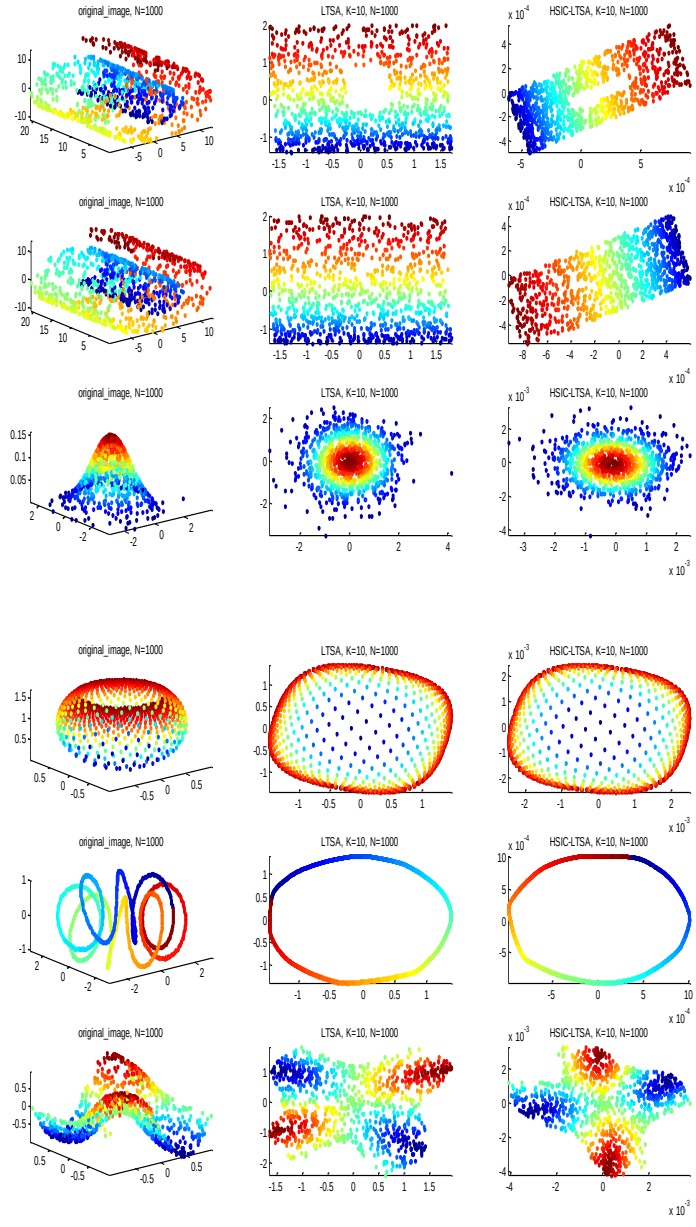


Figure 1. The experimental results of LTSA and HSIC-LTSA on toy data

6.1 Toy Data

Figure 1 shows the experimental results on toy data. The toy data as well as the experimental results of LTSA on the toy data are all produced by using MANI. MANI is a platform commonly used in manifold learning and can be downloaded free from internet. It can be seen from Figure 1 that the experimental results of HSIC-LTSA are reasonable and comparable with those of LTSA. In some toy data, HSIC-LTSA seems even better than LTSA. For example, in Swill Roll with rectangle hole in the middle, HSIC-LTSA reproduces the rectangle more faithfully.

6.2 Face Image Data

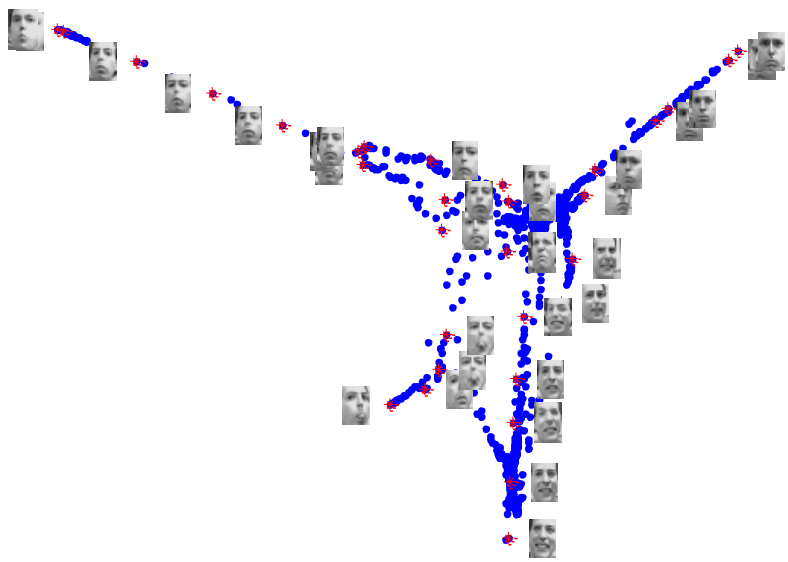


Figure 2. The experimental results of LTSA on face images

Figures 2 and 3 show the experimental results of LTSA and HSIC-LTSA on the dataset of faces. This dataset is often used in many literatures of manifold learning. The face in the dataset only changes in gesture and expression. Therefore, although the faces are represented with high-dimensional vectors, it may be enough to represent these faces with 2-dimensional vectors. In Figures 2 and 3, the faces are dimensionally reduced to 2-dimension plane with LTSA and HSIC-LTSA, respectively. Some face images are also shown at the corresponding positions. It can be seen from Figures 2 and 3 that from up to bottom the face expression

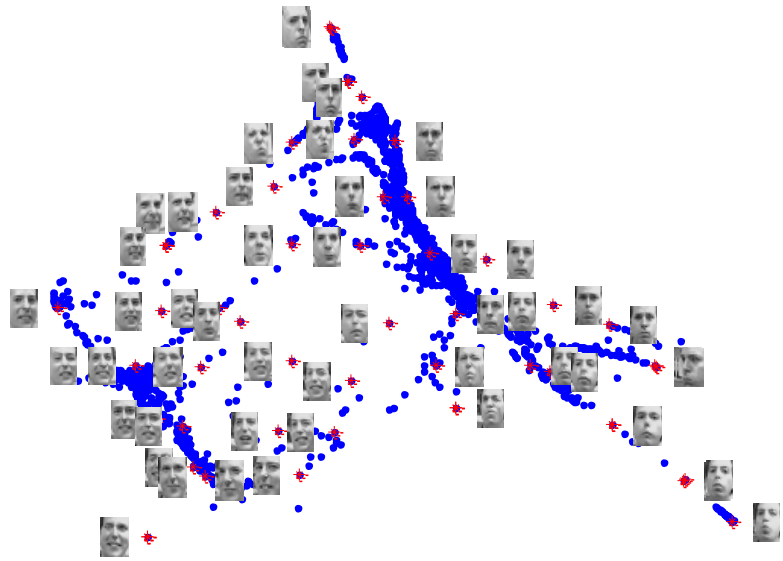


Figure 3. The experimental results of HSIC-LTSA on face images

changes from serious to happy, while from left to right, the face gesture changes from eastward to westward. The impression of HSIC-LTSA seems better than that of LTSA.

6.3 Classification Experiments

The experimental results shown in Figures 1, 2 and 3 are qualitative, not quantitative, and are judged entirely by subjective feelings. In order to compare LTSA and HSIC-LTSA objectively, a number of classification experiments are presented, where data are first dimensionally reduced with LTSA and HSIC-LTSA, respectively, and then classified with K-NN method. The accuracy rates of classification are listed in Table 1.

The datasets used in the classification experiment are MNIST, USPS, YaleB, Binaryalphadigs, AR, UMIST, ORL and Vehicle. All these datasets can be downloaded from Internet and commonly used in many literatures of machine learning. Both MNIST and USPS are the datasets of handwritten digits. Binaryalphadigs is the dataset of handwritten digits and English letters. YaleB, AR, UMIST and ORL are all the datasets of face images. Vehicle is the dataset of vehicle images. The classification method used in the experiments is 3-NN method. The kernel used in HSIC is linear kernel.

In Table 1, the numbers shown in the leftmost column are the reduced dimensions; the numerical values shown next to the names of datasets are the accuracy rates of classification without dimensionality reduction. Since the dimension of feature vectors of vehicle image is only 18, the reduced dimensions are then not larger than 18.

Generally speaking, the performance of HSIC-LTSA is better than LTSA.

RD: the reduced dimension; unit: %
The numbers next to the names of datasets are the accuracy rates of classification without dimensionality reduction

RD	MNIST/88.0		USPS/84.1		YaleB/61.5		AR/32.13	
	LTSA	HSIC-LTSA	LTSA	HSIC-LTSA	LTSA	HSIC-LTSA	LTSA	HSIC-LTSA
10	74.6	86.2	69.4	85.9	7.5	19.5	15.8	17.8
20	80.4	88.6	79.4	87.4	28.0	67.7	20.4	23.0
30	82.4	89.3	80.5	86.4	46.3	78.3	23.4	28.6
40	82.2	88.5	80.6	87.1	61.4	80.5	26.0	31.7
50	85.5	88.7	82.8	86.6	73.6	83.1	28.9	37.3
60	86.0	88.4	82.4	85.4	77.5	83.2	33.7	44.3
80	86.0	88.4	84.7	85.3	82.6	84.7	47.6	51.5
100	87.1	88.1	84.5	84.1	85.3	86.0	58.9	58.3

RD	ORL/82.5		Binaryalphadigs/69.5		RD	Vehicle/63.7	
	LTSA	HSIC-LTSA	LTSA	HSIC-LTSA		LTSA	HSIC-LTSA
10	64.0	77.0	53.1	7.40	2	48.6	48.5
20	75.2	81.8	66.3	31.4	3	48.7	51.3
30	81.9	81.7	65.6	31.4	4	48.0	51.7
40	82.0	77.0	67.4	31.4	5	51.8	50.5
50	81.7	72.5	63.5	43.0	10	66.5	62.3
60	77.6	65.2	59.0	40.0	15	74.0	66.7
80	70.6	53.9	52.4	27.6	16	74.7	70.4
100	66.1	47.4	41.1	18.9	17	75.1	66.9

Remark: The datasets as well as the source codes will be available on request.

Table 1. The accuracy rates of classification

7 CONCLUSIONS

The theory of HSIC sounds a little complicated and seems too difficult to understand for AI engineers. In this paper, a brief and self-sufficient introduction to HSIC is presented for better understanding of HSIC. Since it was first proposed around 2005, HSIC has found many applications in machine learning and some of them are similar to dimensionality reduction [24, 25, 26]. However, HSIC has never been applied to machine learning in regularization form so far. The proposed HSIC-LTSA may be the first try of HSIC regularization.

The so-called regularization means to add regularization terms behind objective functions of other algorithms. The proposed HSIC-LTSA adds HSIC regularization to LTSA, we can also add HSIC regularization to Laplacian Eigenmap algorithm [1] to form HSIC-LE algorithm, to Local Linear Embedded algorithm [2] to form HSIC-LLE algorithm, and so on. HSIC regularization would likely greatly expand the application scope of HSIC, just like what manifold regularization [3] has done. Manifold regularization makes the application scope of manifold learning expand from dimensionality reduction initially to various aspects of machine learning now.

Acknowledgments

This research has been supported by the National Natural Science Foundation of China (NSFC) under Grants 61773022.

REFERENCES

[1] Belkin, M.—Niyogi, P.: Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. Proceedings of the 14th International Conference on Neural Information Processing Systems (NIPS 2001). Advances in Neural Information Processing Systems, Vol. 14, 2001, pp. 585–591.

[2] Roweis, S. T.—Saul, L. K.: Nonlinear Dimensionality Reduction by Locally Linear Embedding. Science, Vol. 290, 2000, No. 5500, pp. 2323–2326, doi: 10.1126/science.290.5500.2323.

[3] Belkin, M.—Niyogi, P.—Sindhwani, V.: Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. Journal of Machine Learning Research, Vol. 7, 2006, No. 11, pp. 2399–2434.

[4] Weinberger, K. Q.—Sha, F.—Saul, L. K.: Learning a Kernel Matrix for Nonlinear Dimensionality Reduction. Proceedings of the Twenty-First International Conference on Machine Learning (ICML 2004), ACM, International Conference Proceeding Series, Vol. 69, 2004, doi: 10.1145/1015330.1015345.

[5] Lafon, S.—Lee, A. B.: Diffusion Maps and Coarse-Graining: A Unified Framework for Dimensionality Reduction, Graph Partitioning, and Data Set Parameterization. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 28, 2006, No. 9, pp. 1393–1403, doi: 10.1109/tpami.2006.184.

[6] Zhang, Z.—Zha, H.: Principal Manifolds and Nonlinear Dimensionality Reduction via Tangent Space Alignment. SIAM Journal on Scientific Computing, Vol. 26, 2004, No. 1, pp. 313–338, doi: 10.1137/s1064827502419154.

[7] He, X.—Niyogi, P.: Locality Preserving Projections. Proceedings of the 16th International Conference on Neural Information Processing Systems (NIPS 2003). Advances in Neural Information Processing Systems, Vol. 16, 2003, pp. 186–197.

[8] Chen, J.—Ma, Z.—Liu, Z.: Local Coordinates Alignment with Global Preservation for Dimensionality Reduction. IEEE Transactions on Neural Networks and Learning Systems, Vol. 24, 2013, No. 1, pp. 106–117, doi: 10.1109/tnnls.2012.2225844.

[9] Liu, X.—Wang, L.—Zhang, J.—Yin, J.—Liu, H.: Global and Local Structure Preservation for Feature Selection. IEEE Transactions on Neural Networks and Learning Systems, Vol. 25, 2014, No. 6, pp. 1083–1095, doi: 10.1109/tnnls.2013.2287275.

[10] Gretton, A.—Bousquet, O.—Smola, A.—Schölkopf, B.: Measuring Statistical Dependence with Hilbert-Schmidt Norms. In: Jain, S., Simon, H. U., Tomita, E. (Eds.): Algorithmic Learning Theory (ALT 2005). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 3734, 2005, pp. 63–77, doi: 10.1007/11564089_7.

[11] Yan, K.—Kou, L.—Zhang, D.: Learning Domain-Invariant Subspace Using Domain Features and Independence Maximization. IEEE Transactions on Cybernetics, Vol. 48, 2018, No. 1, pp. 288–299, doi: 10.1109/tcyb.2016.2633306.

[12] Damodaran, B. B.—Courty, N.—Lefèvre, S.: Sparse Hilbert Schmidt Independence Criterion and Surrogate-Kernel-Based Feature Selection for Hyperspectral Image Classification. IEEE Transactions on Geoscience and Remote Sensing, Vol. 55, 2017, No. 4, pp. 2385–2398, doi: 10.1109/tgrs.2016.2642479.

[13] Gangeh, M. J.—Zarkoob, H.—Ghodsi, A.: Fast and Scalable Feature Selection for Gene Expression Data Using Hilbert-Schmidt Independence Criterion. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), Vol. 14, 2017, No. 1, pp. 167–181, doi: 10.1109/tcbb.2016.2631164.

[14] Xiao, M.—Guo, Y.: Feature Space Independent Semi-Supervised Domain Adaptation via Kernel Matching. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 37, 2015, No. 1, pp. 54–66, doi: 10.1109/tpami.2014.2343216.

[15] Zhong, W.—Pan, W.—Kwok, J. T.—Tsang, I. W.: Incorporating the Loss Function into Discriminative Clustering of Structured Outputs. IEEE Transactions on Neural Networks, Vol. 21, 2010, No. 10, pp. 1564–1575, doi: 10.1109/tnn.2010.2064177.

[16] Jost, J.: Riemannian Geometry and Geometric Analysis. Springer Science and Business Media, 2008.

[17] Spivak, M.: A Comprehensive Introduction to Differential Geometry, Vol. 4. 3rd edition. Publish or Perish Press, 1999.

[18] Kreyszig, E.: Introductory Functional Analysis with Applications. 1st edition. Wiley, New York, 1989, pp. 547–553.

[19] Mika, S.—Ratsch, G.—Weston, J.—Scholkopf, B.—Mullers, K. R.: Fisher Discriminant Analysis with Kernels. Neural Networks for Signal Processing IX. Proceedings of the 1999 IEEE Signal Processing Society Workshop, 1999, doi: 10.1109/nnsp.1999.788121.

[20] Cortes, C.—Vapnik, V.: Support-Vector Networks. Machine Learning, Vol. 20, 1995, No. 3, pp. 273–297, doi: 10.1007/bf00994018.

[21] Shawe-Taylor, J.—Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, 2004, doi: 10.1017/cbo9780511809682.

[22] Gohberg, I.—Goldberg, S.—Kaashoek, M. A.: Hilbert-Schmidt Operators. Classes of Linear Operators, Vol. I. Birkhäuser, Basel, Operator Theory: Advances and Applications, Vol. 49, 1990, pp. 138–147, doi: 10.1007/978-3-0348-7509-7_9.

[23] Xiang, S.—Nie, F.—Pan, C.—Zhang, C.: Regression Reformulations of LLE and LTSA with Locally Linear Transformation. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), Vol. 41, 2011, No. 5, pp. 1250–1262, doi: 10.1109/tsmcb.2011.2123886.

[24] Gangeh, M. J.—Ghodsi, A.—Kamel, M. S.: Kernelized Supervised Dictionary Learning. IEEE Transactions on Signal Processing, Vol. 61, 2013, No. 19, pp. 4753–4767, doi: 10.1109/tsp.2013.2274276.

[25] Gangeh, M. J.—Fewzee, P.—Ghodsi, A.—Kamel, M. S.—Karray, F.: Multiview Supervised Dictionary Learning in Speech Emotion Recognition. IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP), Vol. 22, 2014, No. 6, pp. 1056–1068, doi: 10.1109/taslp.2014.2319157.

[26] Barshan, E.—Ghodsi, A.—Azimifar, Z.—Jahromi, M. Z.: Supervised Principal Component Analysis: Visualization, Classification and Regression on Subspaces and Submanifolds. Pattern Recognition, Vol. 44, 2011, No. 7, pp. 1357–1371, doi: 10.1016/j.patcog.2010.12.015.

Xinghua Zheng received his B.Sc., M.Sc. and Ph.D. degrees from the Sun Yat-sen University, Guangzhou, China, in 2006, 2011 and 2018, respectively. His research interests include machine learning.

Zhengming Ma received his B.Sc. and M.Sc. degrees from the South China University of Technology, Guangzhou, China, in 1982 and 1985, respectively, and his Ph.D. degree in pattern recognition and intelligent control from Tsinghua University, Beijing, China, in 1989. He is currently Professor with the School of Electronics and Information Technology, Sun Yat-sen University, Guangzhou, China. His current research interest is machine learning.

Hangjian Che received his B.Sc. degree in electronic information science and technology from Sun Yat-sen University, Guangzhou, China, in 2017. He is currently pursuing the master's degree with the School of Electronics and Information Technology, Sun Yat-sen University, Guangzhou. His current research interests include machine learning.

Lei Li received his Ph.D. degree in computer science from Claude Bernard Lyon 1 University, France, in 1988. He is currently Professor with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. His current research interest is machine learning.

Computing and Informatics, Vol. 38, 2019, 937–962, doi: 10.31577/cai_2019_4_937

REVISITING MULTIPLE PATTERN MATCHING

Robert Susik, Szymon Grabowski

Institute of Applied Computer Science, Lodz University of Technology
Al. Politechniki 11, 90 924 Łódź, Poland
e-mail: rsusik@kis.p.lodz.pl, sgrabow@kis.p.lodz.pl

Kimmo Fredriksson

School of Computing, University of Eastern Finland
P.O.B. 1627, FI-70211 Kuopio, Finland
e-mail: kimmo.fredriksson@uef.fi

Abstract. We consider the classical exact multiple string matching problem. The proposed solution is based on a combination of a few ideas: using q-grams instead of single characters, pattern superimposition, bit-parallelism and alphabet size reduction. We discuss the pros and cons of various alternatives to achieve the possibly best combination of techniques. The main contribution of this paper are different alphabet mapping methods that allow to reduce memory requirements and use larger q-grams. The experimental results show that the presented algorithm is competitive in most practical cases. One of the tests shows also that tailoring our scheme to search over a byte-encoded text results in speedups in comparison to searching over a plain text.

Keywords: Text algorithms, pattern matching, multiple pattern matching, q-grams, bit-parallelism, compressed pattern matching

Mathematics Subject Classification 2010: 68W32

1 INTRODUCTION

The problem of multiple pattern matching can be stated as follows: Given text T of length n and pattern set $\mathcal{P} = \{P_0, \ldots, P_{r-1}\}$, in which each pattern is of length m, and all considered sequences are over common alphabet Σ of size σ, find all pattern occurrences in T. The pattern equal length requirement may be removed (although not all algorithms easily handle uneven patterns). Multiple pattern matching is a classic problem, with over 40 years of history and applications in intrusion detection, anti-virus software, spam filtering and bioinformatics, to name a few. As this problem is a straightforward generalization of a single pattern matching, it is perhaps no surprise that many techniques worked out for a single pattern are borrowed for efficient algorithms for multiple patterns.

Our paper presents a novel algorithm for multiple pattern matching, being a careful combination of several known techniques: using q-grams combined with pattern superimposition, bit-parallelism and alphabet size reduction. The experimental results will show that the presented solution, MAG (Multi AOSO on q-Grams), usually dominates over its competitors on texts with diverse characteristics.

1.1 Related Work

A naive approach to searching for r patterns in a text is to use any single pattern search algorithm r times. As the performance grows linearly with r, such a technique cannot be reasonably used when, e.g., r exceeds 10. Non-trivial algorithms for the presented problem can roughly be divided into three different categories, based on the location of pattern substrings they try to find (and then to extend). More specifically, these algorithms are based on

1. prefix searching,
2. suffix searching and
3. factor searching, respectively.

Another taxonomy of the existing solutions classifies them according to whether they are based on character comparisons, hashing, or bit-parallelism. Yet another view is to say that they are based on filtering, aiming for good average case complexity, or on some kind of "direct search" with good worst case complexity guarantees. These different categorizations are of course not mutually exclusive, and many solutions are hybrids that borrow ideas from several techniques. For a good overview of the classical solutions, the reader is referred to, e.g., [27, 19, 10]. We briefly review some of them in the following paragraphs.

Perhaps the most famous solution to the multiple pattern matching problem is the Aho-Corasick (AC) [1] algorithm, which generalizes the Knuth-Morris-Pratt algorithm [22] for a single pattern. The AC algorithm follows the prefix-based approach and it builds a pattern trie with extra (failure) links. One can say that AC

works in linear time. More precisely, however, AC total time is $O(M+n+z)$ for constant alphabet, where M, the sum of pattern lengths, is the preprocessing cost, and z is the total number of pattern occurrences in T. For an integer alphabet, i.e., when $\sigma = n^{O(1)}$, it obtains $O(M + n \log \sigma + z)$ time [12]. Fredriksson and Grabowski [18] showed an average-optimal filtering variant of the classic AC algorithm. They built the AC automaton over superimposed subpatterns, which allows to sample the text characters in regular distances, not to miss any match (i.e., any verification). This algorithm is based on similar ideas as the current work.

Another classic algorithm is Commentz-Walter [8], which generalizes the ideas of Boyer-Moore (BM) algorithm [4] for a single pattern, to solve the multiple pattern matching problem (suffix-based approach). Set Horspool (SH) [15, 27] may be considered its more practical simplification, exactly in the way that Boyer-Moore-Horspool (BMH) [20] is a simplification of the original BM. Set Horspool makes use of a generalized bad character function. The Horspool technique was used in a different way in an earlier algorithm by Wu and Manber (WM) [32]. These methods are based on backward matching over a sliding text window, which is shifted based on some rule, with the hope that many text characters can be skipped altogether.

The first factor-based algorithms were DAWG-match [9] and MultiBDM [11]. Like Commentz-Walter and Set Horspool, they are based on backward matching. However, instead of recognizing the pattern suffixes, they recognize the factors, which effectively means that they work more per window, but in return they are able to make longer shifts of the sliding window, and in fact they obtain optimal average case complexity. At the same time they are linear in the worst case. The drawback is that these algorithms are reasonably complex and not very efficient in practice. A more practical approach is the Set Backward Oracle Matching (SBOM) algorithm [2], which is based on the same idea as MultiBDM, but uses simpler data structures and is very efficient in practice. Yet another variant is the Succinct Backward DAWG Matching algorithm (SBDM) [17], which is practical for huge pattern sets due to replacing the suffix automaton with succinct index. The factor-based algorithms usually lead to average-optimal [25] time complexity of $O(n \log_\sigma(rm)/m)$.

Bit-parallelism can be used to replace the various automata in the methods mentioned earlier, to obtain very simple and yet competitive incarnations of many classical algorithms. The most standard bit-parallel solution for a single pattern is Shift-Or [3]. The idea is to encode (non-deterministic) automaton as a bitvector, i.e., a small integer value, and simulate all the states in parallel using Boolean logic and arithmetic. The result is often the most practical method for the problem, but the drawback is that the scalability is limited by the number of bits in a computer word, although there exist ways to alleviate this problem somewhat, see [28, 7]. Another way that is applicable to huge pattern sets is to combine bit-parallelism with q-grams; our method is also based on this, and we review the idea and related previous work in detail in the next section. Another practical solution based on q-grams (used for a cache-efficient index structure), where a multiple pattern matching algorithm is

applied in intrusion detection software and an antivirus system, is SigMatch [21]. Using q-grams in searches over DNA (for one or multiple patterns) is recommended in the survey by Rivals et al. [29].

In a somewhat different way, parallelism for multiple pattern matching has been obtained with a GPU. In particular, Kouzinopoulos et al. [24] adopted the well-known (and mentioned earlier) AC, SH, SBOM, WM algorithms, as well as a more recent one, SOG [30], to the CUDA programming model, to obtain from about 10- to 30-fold speedup compared to one CPU core. Their test GPU was an Nvidia GTX280, and we can guess that the speedup would be significantly higher on a more modern GPU. The same authors [23] proposed another solution based on hybrid OpenMP/MPI technique, focusing on searching in biological databases.

Some recent work also recognizes the neglected power of the SIMD instructions, which have been available on commodity computers well over a decade. For example, Faro and Külekci [13] make use of the Intel Streaming SIMD Extensions (SSE) technology, which gives wide registers and many special purpose instructions to work with. They develop (among other things) a wsfp (*word-size fingerprint instruction*) operation, based on hardware opcode for computing CRC32 checksums, which computes an α-bit fingerprint from a w-bit register handled as a block of α characters. Similar values are obtained for all α-sized factors of all the patterns in the preprocessing, and wsfp can therefore be used as a simple yet efficient hash-function to identify text blocks that may contain a matching pattern. The same authors [14] presented later a SIMD-based solution for multiple string matching, focusing on searching short patterns ($16 \leq m < 32$) in genome sequences and English texts.

Our paper is organized as follows. Section 2 describes and discusses the two key concepts underlying our work, q-grams and pattern superimposition. Section 3 presents the description of our algorithm, together with its complexity analysis. Section 4 contains experimental results. The last section concludes and points some avenues for pursuing further research.

A preliminary version of our work appeared in Proc. PSC 2014 [31].

2 ON SUPERIMPOSING Q-GRAMS

A q-gram is usually defined as a contiguous substring (factor) of a string comprising q characters, although, noteworthily, non-contiguous q-grams have also been considered [6]. In what follows, q can be considered a small constant ($2, \ldots, 6$ in practice), although we may analyze the optimal value for a given problem instance. We note that q-grams have been widely used in approximate (single and multiple) string matching, where they can be used to obtain fast filtering algorithms based on exact matching of a set of q-grams. Obviously these algorithms work for the exact case as well, as a special case, but they are not within the scope of this paper. Another use (which is not relevant in our case) is to speed up exact matching of a single pattern by treating the q-grams as a superalphabet [16].

In multiple pattern matching q-grams may be combined using an interesting technique called superimposition. Consider a set of patterns $\mathcal{P} = \{P_0, \ldots, P_{r-1}\}$. We form a single pattern P where each position $P[i]$ is no longer a single character, but a *set* of characters, i.e., $P[i] \subseteq \Sigma$. More precisely, $P[i] = \bigcup_j P_j[i]$. Now P can be used as a *filter*: we search candidate text substrings that might contain an occurrence of any of the patterns in $\mathcal{P}$. In other words, if $T[i+j] \in P[j]$, for all $j \in 0, \ldots, m-1$, then $T[i \ldots i+m-1]$ may match with some pattern in $\mathcal{P}$.

Let us present a simple example. Let $r = 2$ and $\mathcal{P} = \{abba, bbac\}$. The superimposed pattern is then $P = \{a,b\}\{b\}\{a,b\}\{a,c\}$, and there are a total of 8 different strings of length 4 that can match with P (and trigger verification). Therefore we immediately notice one of the problems with this approach, i.e., the probability that some text character t matches a pattern character p is no longer $1/\sigma$ (assuming uniform random distribution), it can be up to r/σ. This gets quickly out of hands when the number of patterns r grows.

To make the technique more useful, we first generate a new set of patterns, and then superimpose. The new patterns have the q-grams as the alphabet, which means the new alphabet has size σ^q, and the probability of a false positive candidate will be considerably lower. There are two main approaches: overlapping and non-overlapping q-grams.

Consider first the overlapping q-grams. For each P_i we generate a new pattern such that $P'_i[j] = P_i[j \ldots j+q-1]$, for $j \in 0 \ldots m-q$, that is, each q-gram $P_i[j \ldots j+q-1]$ is treated as a single "super character" in P'_i. Note also that the pattern lengths are decreased from m to $m-q+1$. Taking the previous example, if $\mathcal{P} = \{abba, bbac\}$ and now $q = 2$, the new pattern set is $\mathcal{P}' = \{[ab][bb][ba], [bb][ba][ac]\}$, where we use the brackets to denote the q-grams. The corresponding superimposed pattern is then $P' = \{[ab],[bb]\}\{[bb],[ba]\}\{[ba],[ac]\}$. To be able to search for P', the text must be factored in exactly the same way.

The other possibility is to use non-overlapping q-grams. In this case we have $P'_i[j] = P_i[(j-1)q+1 \ldots jq]$, for $j \in 0 \ldots \lfloor m/q \rfloor - 1$, and for our running example we get $P' = \{[ab],[bb]\}\{[ba],[ac]\}$. Again, the text must be factored similarly. But the problem now is that only every q^{th} text position is considered, and to solve this problem we must consider all q possible shifts of the original patterns. That is, given a pattern P_i, we generate a set $\hat{P}_i = \{P_i[0 \ldots m-1], P_i[1 \ldots m-1], \ldots, P_i[q-1 \ldots m-1]\}$, and then generate $\hat{P}'_i$, and finally superimpose them.

The two alternatives given above both have some benefits and drawbacks. For overlapping q-grams we have:

- pattern length is large $(m-q+1)$, which implies fewer verifications,
- text length is practically unaffected $(n-q+1)$.

On the other hand, for the non-overlapping ones:

- pattern length is short (m/q), which means potentially more verifications, but bit-parallelism works for bigger m,
- text is shorter too (n/q),

- more patterns to superimpose (factor of q).

In the end, the benefits and drawbacks between the two approaches mostly cancel out each other, except bit-parallelism remains more applicable to non-overlapping q-grams.

To illustrate the power of this technique, let us have, for example, a random text over an alphabet of size $\sigma = 16$ and patterns generated according to the same probability distribution; q-grams are not used yet (i.e., we assume $q = 1$). If $r = 16$, then the expected size of a character class in the superimposed pattern is about 10.3, which means that a match probability for a single character position is about 64 %. Even if high, this value may yet be feasible for long enough patterns, but if we increase r to 64, the character class expected size grows to over 15.7 and the corresponding probability to over 98 %. This implies that match verifications are likely to be invoked for most positions of the text. Using q-grams has the effect of artificially growing the alphabet. In our example, if we use $q = 2$ and thus $\sigma' = 16^2 = 256$, the corresponding probabilities for $r = 16$ and $r = 64$ become about 6 % and 22 %, respectively, so they are significantly lower.

The main problem that remains is to decide between the two choices, properly choose a suitable q, and finally find a good algorithm to search the superimposed pattern. To this end, Salmela et al. [30] presented three algorithms combining the known mechanisms: Shift-Or, BNDM [26] and BMH, with overlapping q-grams; the former two of these algorithms are bit-parallel ones. The resulting algorithms were called SOG, BG and HG, respectively. In general, larger q means better filtering, but on the other hand the size of the data structures (tables) that the algorithms use is $O(\sigma^q)$, which can be prohibitive. BGqus (BG with q-grams, Unrolling and s-bit shift hash method) [33] tries to solve the problem by combining BG with hashing.

Actually, not many classic algorithms can be generalized to handle superimposed patterns (character classes) efficiently, but bit-parallel methods generalize trivially. In the next section we describe our choice, FAOSO [18].

3 OUR ALGORITHM

In [18] a general technique of how to skip text characters, with any (linear time) string matching algorithm that can search for multiple patterns simultaneously was presented, alongside with several applications to known algorithms. In the following we review the idea, and for the moment assume that we already have done all factoring to q-grams, and that we have only a single pattern.

3.1 Average-Optimal Character Skipping

The method takes a parameter k, and from the original pattern generates a set $\mathcal{K}$ of k new patterns $\mathcal{K} = \{P^0, \ldots, P^{k-1}\}$, each of length $m' = \lfloor m/k \rfloor$, as follows:

$$P^j[i] = P[j + ik], \quad j = 0 \ldots k-1, i = 0 \ldots \lfloor m/k \rfloor - 1.$$

In other words, k different alignments of the original pattern P is generated, each alignment containing only every k^{th} character. The total length of the patterns P^j is $k\lfloor m/k \rfloor \leq m$.

Assume now that P occurs at $T[i \ldots i + m - 1]$. From the definition of P^j it directly follows that

$$P^j[h] = T[i + j + hk], \quad j = i \mod k, h = 0 \ldots m' - 1.$$

This means that the set $\mathcal{K}$ can be used as a filter for the pattern P, and that the filter needs only to scan every k^{th} character of T. Figure 1 serves as an illustration.

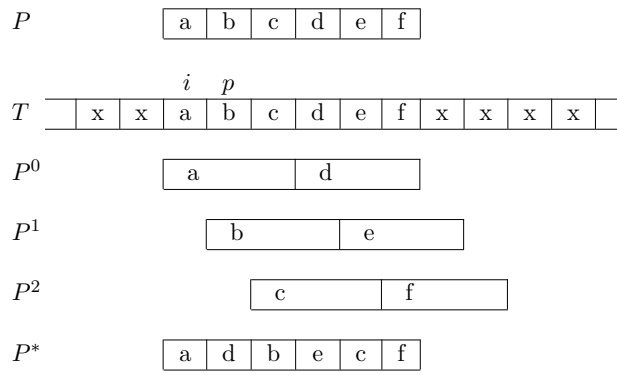


Figure 1. An example. Assume that $P =$ `abcdef` occurs at text position $T[i \ldots i+m-1]$, and that $k = 3$. The current text position is $p = 10$, and $T[p] =$ `b`. The next character the algorithm reads is $T[p+k] = T[13] =$ `e`. This triggers a match of $P^{p \mod k} = P^1$, and the text area $T[p-1 \ldots p-1+m-1] = T[i \ldots i+m-1]$ is verified.

The occurrences of the patterns in $\mathcal{K}$ can be searched for simultaneously using any multiple string matching algorithm. Assuming that the selected string matching algorithm runs generally in $O(n)$ time, then the filtering time becomes $O(n/k)$, as only every k^{th} symbol of T is read. The filter searches for the exact matches of k patterns, each of length $\lfloor m/k \rfloor$. Assuming that each character occurs with probability $1/\sigma$, the probability that P^j occurs (triggering a verification) in a given text position is $(1/\sigma)^{\lfloor m/k \rfloor}$. A brute force verification cost is in the worst case $O(m)$. To keep the total time $O(n/k)$ on average, we select k so that $nm/\sigma^{m/k} = O(n/k)$. This is satisfied for $k = m/(2\log_\sigma(m))$, where the verification cost becomes $O(n/m)$

and filtering cost $O(n \log_\sigma(m)/m)$. The total average time is then dominated by the filtering time, i.e., $O(n \log_\sigma(m)/m)$, which is optimal [34].

3.2 Multiple Matching with q-Grams

To apply the previous idea to multiple matching, we just assume that the (single) input pattern (for the filter) is the non-overlapping q-gram factored and superimposed pattern set. The verification phase just needs to be aware that there are possibly more than one pattern to verify. The analysis remains essentially the same: now the text length is n/q, pattern lengths are m/q, there are r patterns to verify, and the probability of a match is p instead of $1/\sigma$, where $p = O(1 - (1 - (1/\sigma^q))^{qr}) = O((qr)/\sigma^q)$. That is, the filtering time is $O(qn/(kq)) = O(n/k)$, verification cost is $O(rqm)$, and its probability is $O(p^{\lfloor m/(kq) \rfloor})$ for each of the n/q text positions. However, now we have two parameters to optimize, k and q, and the optimal value of one depends on the other.

In practice we want to choose q first, such that the verification probability is as low as possible. This means maximizing q, but the preprocessing cost (and space) grows as $O(\sigma^q)$, and we do not want this to exceed $O(rm)$ (or the filtering cost for that matter). So we select $q = \log_\sigma(rm)$, and then choose k as large as possible. Repeating the above analysis gives then

$$k = O\left(\frac{m}{\log_\sigma(rm)} \cdot \frac{\log_\sigma 1/\rho}{\log_\sigma(rm) + \log_\sigma 1/\rho}\right) \tag{1}$$

where $\rho = \log_\sigma(rm)/m$. We note that this is not average-optimal anymore, although we are still able to skip text characters.

To search the superimposed pattern, we use FAOSO [18], which is based on Shift-Or. The fact that the pattern consists of character classes is not a problem for bit-parallel algorithms, since it only affects the initial preprocessing of a single table. For details see [18]. The filter implemented with FAOSO runs in $O(n/k \cdot \lceil (m/q)/w \rceil)$ time in our case, where w is the number of bits in computer word (typically 64).

We note that Salmela et al. [30] have tried a similar approach, but abandoned it early because it did not look promising for short patterns in their tests.

3.2.1 Implementation

The q-gram, i.e., the super character, must have some suitable representation, and the convenient way is to compute a numerical value in the range $0 \ldots \sigma^q - 1$, which is done as $\sum_{i=0}^{q-1} S[i] \cdot \sigma^i$ for a q-gram $S[0 \ldots q-1]$. This is computed using Horner's method to avoid the exponentiation. We have experimented with two different variants. The first encodes the whole text prior to starting the actual search algorithm, which is then more streamlined. This also means that the total complexity is $\Omega(n)$, the time to encode the text. We call the resulting algorithm SMAG (short of Simple Multi AOSO on q-grams). The other alternative is to keep the text intact, and

compute the numerical representation of the q-gram requested on the fly. This adds just constant overhead to the total complexity. We call this variant MAG (short of Multi AOSO on q-grams). We have verified experimentally that MAG is generally better than SMAG.

3.3 Alphabet Mapping

If the alphabet is large, then selecting a suitable q may become a problem. The reason is that some value q' may be too small to facilitate good filtering capability, yet, using $q = q' + 1$ can be problematic, as the preprocessing time and space grow with σ^q (note that q must be an integer). The other view of using strings of length q as super characters is that we may then say that our characters have $q \log_2 \sigma$ bits, and we want to have more control of how many bits we use. One way to achieve this is to reduce the original alphabet size σ.

We note that in theory this method cannot achieve much, as reducing the alphabet size generally only worsens the filtering capability and therefore forces larger q, but in practice this allows better fine tuning of the parameters.

3.3.1 Histogram Based Alphabet Mapping (HAM)

What we do is that we select some $\sigma' < \sigma$, compute a mapping $\mu : \Sigma \mapsto 0 \ldots \sigma' - 1$, and use $\mu(c)$ whenever the (filtering) algorithm needs to access some character c from the text or the pattern set. Verifications still obviously use the original alphabet. A simple method to achieve this is to compute the histogram of character distribution of the pattern set, and assign code 0 to the most frequent character, 1 to second most frequent, and so on, and put the $\sigma' - 1 \ldots \sigma - 1$ most frequent characters to the last bin, i.e., giving them code $\sigma' - 1$. The text characters not appearing in the patterns also will have code $\sigma' - 1$.

A better strategy is to try to distribute the original characters into σ' bins so that each bin will have (approximately) equal weight, i.e., each $\mu(c)$, where $c \in 0 \ldots \sigma' - 1$ will have (approximately) equal probability of appearance. This is NP-hard optimization problem, so we use a simple greedy heuristic which can be described in few steps:

1. compute the symbol frequencies on the pattern set (using, e.g., hashing to avoid possibly large tables);
2. choose some suitable σ', the size of the mapped alphabet;
3. use method of choice (e.g., bin-packing) to reduce the number of symbols, i.e., map them to range $0 \ldots \sigma' - 1$;
4. optionally use hashing to store the mapping.

3.3.2 Histogram Based Alphabet Mapping on q-Grams (HAMq)

The HAM method can be applied also to q-grams. This variant allows control of table size, and with combined hashing it can accommodate very large q as well. In this case the q-grams are used instead of the alphabet symbols, but the whole process is very much the same. The only issue here is the need of additional hashing to store mapping, along with the corresponding bitvectors needed by FAOSO. We did not implement this method in this paper.

3.3.3 Combined Alphabet Mapping and q-Gram Generation (CAMq)

Yet another method to reduce the alphabet is to combine the q-gram computations with some bit magic. The benefit is that the mapping tables need not to be preprocessed, and this allows further optimizations as we will see shortly. The drawback is that the quality of the mapping is worse than what is achieved with approaches like bin-packing.

Consider a (text sub-)string $S[0 \dots q-1]$ over alphabet Σ of size σ. A simple way to reduce the alphabet is to consider only the ℓ low-order bits of each $S[i]$, where $\ell < \log_2 \sigma$. We can then compute $(q\ell)$-bit q-gram s simply as

$$\begin{aligned} s = (S[0] \,\&\, b) + (S[1] \,\&\, b) << \ell + (S[2] \,\&\, b) \\ << 2\ell + \cdots + (S[q-1] \,\&\, b) << (q-1)\ell \end{aligned} \tag{2}$$

where $b = (1 << \ell) - 1$ and $<<$ denotes the left shift and & the bitwise and.

There is a possibility to have four unique values in ASCII DNA alphabet by right shift (CAMq(dna)).

$$\begin{aligned} s = ((S[0] >> 1) \,\&\, b) + ((S[1] >> 1) \,\&\, b) \\ << \ell + \cdots + ((S[q-1] >> 1) \,\&\, b) << (q-1)\ell. \end{aligned} \tag{3}$$

The main benefit of this approach is that a sequence of shifts and adds can be often replaced by a multiplication (which can be seen as an algorithm performing just that). As an illustrative example, consider the case $\ell = 2$ and hence $b = 3$ (which coincides to DNA nicely). As an implementation detail, assume that the text is 8-bit ASCII text, and it is possible to address the text, a sequence of characters, as a sequence of 32-bit integers (which is easy, e.g., in C). Then to compute a 8-bit 4-gram s we can simply apply the transform:

$$s = (((x >> 1) \,\&\, 0\text{x}03030303) * 0\text{x}40100401) >> 24 \tag{4}$$

where x is the 32-bit integer containing $S[0 \dots 3]$. Assuming 4-letter DNA alphabet, with a right shift (by 1) and the (parallel) masking we obtain unique (and case insensitive) 2-bit codes for all four characters. If the alphabet is larger (many DNA sequences have rare extra symbols), those will be mapped into the same range,

0 . . . 3. The multiplication then shifts and adds all those codes into an 8-bit value, and the final shift moves the 4-gram to the lower bits (CAMq(opt)). Larger q-grams can be obtained by repeating the code.

3.4 Preprocessing and Verification

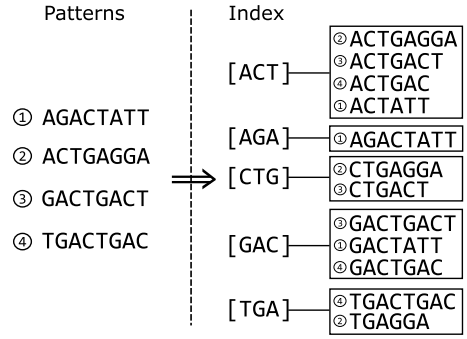


Figure 2. An example of the data structure used for verification; $r = 4$, $q = 3$

The algorithm only returns the positions of possible matches so each of the position needs to be verified. There is no information if the string on returned position belongs to the list of the patterns or not so the verification is necessary. To verify which (if any) of the patterns is found on a given position, the algorithm maintains a dictionary with q-grams as keys and lists of the patterns which contain the key as a substring starting at position $0, \ldots, q-1$ as the associated values. The total length of all the lists stored in the dictionary is rq.

Figure 2 presents a simple example of the described dictionary. The patterns are shown on the left and the dictionary (I) of mappings from 3-grams to patterns containing them (only their respective suffixes are presented) are given on the right. For example, the second element on the list I[“CTG”] is 3 because “CTG” is the third q-gram in the 3rd pattern. The pattern occurs in two other lists: I[“GAC”] and I[“ACT”]. The pseudocode of building such structure is presented as Algorithm 1.

The presented structure is later used in verification process whose pseudocode is presented as Algorithm 2. After a tentative match at position *pos* is reported, the list of associated patterns, $I[get\_q\_gram(T[pos \ldots pos + q - 1])]$, is searched with the binary search (line 3). The *bsearch* function searches over $I[idx]$ with the *pat* field as the key and returns the maximum range of indexes (i, j) such that for all $i \leq a \leq j$, $T[pos - I[idx][a].off \ldots pos - I[idx][a].off + m - 1] = I[idx][a].pat$.

3.5 Algorithm

The combination of all the described techniques yields a fast algorithm for multiple pattern matching, presented as Algorithm 3. The search phase is based on FAOSO

Algorithm 1 MAG_Build_Index

```
1: function MAG_Build_Index(P)
2:     for j ← 0 ... r − 1 do
3:         for i ← 0 ... q − 1 do
4:             idx ← get_q_gram(P_j[i ... q + i − 1])
5:             el.off ← i
6:             el.pat ← P_j
7:             I[idx] ← I[idx] ∪ {el}
8:         for i ← 0 ... | I | do
9:             sort(I[i], key = pat)                    ▷ Sort by pat
10:    return I
```

Algorithm 2 MAG_Verification

```
1: function MAG_Verification(T, pos, I)
2:     idx ← get_q_gram(T[pos ... pos + q − 1])
3:     [i, j] ← bsearch(I[idx])
4:     for el ← I[idx][i] ... I[idx][j] do
5:         report match at pos − el.off
```

(an algorithm for searching a single pattern) but the preprocessing (lines 3–16) and verification (Algorithm 2) are novel. The preprocessing involves multiple patterns (there are r of them) that consist of character classes built on top of q-grams. There is a possibility that the found position is aligned with one of the first q q-grams and this is why such shifts have been taken into account (line 11). Note that the pseudocode presents the variant with combined alphabet mappings and q-gram generation called *get_q_gram*.

3.6 Searching in Compressed Text

Text compression is a technique widely used to decrease the amount of memory needed to store (or transmit) textual data. Some text compression methods allow to search directly in the compressed text, without prior decompression, which is not only elegant but may also improve the search speed, even compared to searching over non-compressed text. In this section, we adapt our solution to searching in End-Tagged Dense Code (ETDC) [5] compressed text, where the ETDC codes are assigned to words. ETDC is a variable-length byte code in which 7 bits per byte store data and 1 bit (by convention, the highest one) is set in the last byte of each codeword and unset otherwise. The data bits are used fully, which means that (in accordance to the golden rule of data compression, which assigns shorter codewords to more frequent symbols) 1-byte codewords are assigned to 128 most frequent words in the text, 2-byte codewords are assigned to $2^{14} = 16\,384$ next most frequent words, and so on (in practice, for unilingual texts it is enough to maintain at most 3-byte

Algorithm 3 Multi AOSO on q-Grams

```
1:  function MAG(P, T[0...n-1])
2:      I ← MAG_Build_Index(P)
3:      for i ← 0...σ^q − 1 do b[i] = ∼0
4:      m' ← (⌊m/q⌋) − (1 − ⌊(m mod q)/(q − 1)⌋)
5:      j ← 0
6:      h ← 0
7:      mm ← 0
8:      for j ← 0...k − 1 do
9:          for i ← 0...m'/k − 1 do
10:             for z ← 0...r − 1 do
11:                 for o ← 0...q − 1 do
12:                     b[get_q_gram(P_z[ikq̇ + jq̇ + o])] &= ∼(1 << h)
13:         for l ← 0...U − 1 do
14:             mm ← mm | (1 << (h − 1))
15:             h ← h + 1
16:         h ← h − 1
17:     /* Search superimposed pattern in T using FAOSO and verify each
            tentative match at position pos with MAG_Verification(T, pos, I) */
```

codewords). The flag bits not only make the code a prefix one, but also allow instant synchronization in an ETDC stream, which in turn conveniently enables to plug in any character-skipping pattern searching algorithms. Note also that such encoding of a text perceived as a sequence of words is intended to make the text shorter (to about 35 % of its original length in practice), yet the encoded pattern (a phrase consisting of whole words) also tends to be shorter, which mitigates the expected search speedup.

3.6.1 Implementation

We encode the input text with ETDC and build a helper array A which allows us to find the position of the pattern in the original (non-compressed) text. More precisely, before the encoding, the original text T is transformed to a simpler form T' by removing all commas, tabulation symbols, excessive spaces and EOL characters (a period is treated as a separate word). After the ETDC encoding we obtain three streams:

1. the encoded text $E(T')$,
2. the dictionary, which maps the distinct words to their corresponding variable-length codewords, and
3. the array A.

Assume that the length of T' is n' (characters, and also bytes). The array A is of length n'/h (integers), where $h > 1$ is a parameter; larger h produces a smaller array yet reporting a match position is more time consuming. Each $A[i]$, $0 \leq i < n'$, stores a value j such that $T'[j]$ is the first symbol of the word whose codeword in $E(T')$ contains the byte $T'[ih]$. We assume that h is at least the maximum number of bytes per codeword (in practice, it is much larger, otherwise the overhead of A would be significant).

Searching in $E(T')$ is very similar to searching in the plain text T. We first run the MAG code for the encoded patterns over $E(T')$, to obtain their positions in the encoded text. Then, in a postprocessing, we map the found positions onto the original positions in T'.

The mapping, making use of the array A, will be shown on an example. Let us have a short text (extracted from `english.200MB`) $T' =$ "the fire on the hearth filled the chamber ." and a pattern $P =$ "the hearth" (for clarity we use only one pattern, but it easily generalizes to multiple patterns). Figure 3 presents the (simplified) text T', the encoded text $E(T')$ and the helper table A. The goal is to obtain all positions of pattern P in text T' from the found occurrences of $E(P)$ in $E(T')$.

Let us assume that $E(P)$ is the 3-byte sequence $[(128)(36, 189)]$ and let $h = 4$ (note that $|A| = \lceil |E(T')|/h \rceil = \lceil 13/4 \rceil = 4$). The first (and only in the example) occurrence of $E(P)$ in $E(T')$ is at position $pos = 4$. As $pos \bmod h = 0$, we have that $A[\lfloor pos/h \rfloor]$ is aligned with the match position and the returned value $A[\lfloor pos/h \rfloor] = A[1] = 12$ is the match position of P in T'.

In the second case, i.e., when $pos \bmod h \neq 0$, things are slighly more complicated. Again, let us use an example; this time we look for $P =$ "chamber". It is encoded as $E(P) = [(85, 138)(129)]$. Its (first and only) occurrence in $E(T')$ is at position $pos = 10$. As $pos \bmod h \neq 0$, to the largest value of A sampling $E(T')$ *before* pos, that is, $A[\lfloor pos/h \rfloor] = A[2] = 23$, we add the decoded lengths of all the words before the found pattern, starting with the word aligned with $A[\lfloor pos/h \rfloor]$. In our example, there are two such words, spanning $E(T[7 \ldots 9])$ and decoded to "filled the " (note the blank space following the last word), of length 11. Therefore, the returned position of P in T' is $23 + 11 = 34$. Note that the ETDC dictionary is not shown in Figure 3.

```
                  012345678901234567890123456789012345678901234567890123456789012
              T' = the fire on the hearth filled the chamber .
                   0    1  2    3    4    5  6    7  8    9    10 11   12
          E(T') = (128)(112,130)(150)(128) (36,189)   (66,133) (128)  (85,138) (129)
              A =  0              12               23                   42
```

Figure 3. The excerpt from an ETDC-encoded text with the helper array ($h = 4$)

4 EXPERIMENTAL RESULTS

In order to evaluate the performance of our approach, we ran several experiments, using 200 MB versions of selected datasets (`dna`, `english` and `proteins`) from the widely used Pizza & Chili corpus (`http://pizzachili.dcc.uchile.cl/`). MAG variants were implemented in C++ and compiled with `g++` 4.8.1 with `-O3` optimization. The experiments were run on a desktop PC with an Intel i3-2100 CPU clocked at 3.1 GHz with 128 KB L1, 512 KB L2 and 3 MB L3 cache, sporting 4 GB of 1 333 MHz DDR3 RAM and running Ubuntu 13.04 64-bit OS with kernel 3.11.0-17. The MAG source codes can be found at the URLs given below:

1. MAG – `https://github.com/rsusik/mag`,
2. MAG on ETDC – `https://github.com/rsusik/magetdc`.

4.1 Multi AOSO on q-Grams

In this section we compare several variants (cf. Section 3.3.1) of our solution. We use the following naming convention:

- `_dna`, adapted for the `dna` alphabet (each symbol is right shifted),
- `_opt`, an optimized variant (shifts and adds replaced by a multiplication),
- `_l`x, where x is the value of the ℓ parameter ($\sigma = 2^\ell$).

We have nine variants in total:

- `mag` – HAM (Section 3.3.1),
- `mag_l2` – CAMq (Equation (2)), with $\ell = 2$,
- `mag_l3` – as above, $\ell = 3$,
- `mag_l4` – as above, $\ell = 4$,
- `mag_dna_l2` – CAMq(dna) (Equation (3)), $\ell = 2$,
- `mag_dna_l3` – as above, $\ell = 3$,
- `mag_dna_l4` – as above, $\ell = 4$,
- `mag_dna_opt_l2` – CAMq(opt) (Equation (4)), with $\ell = 2$,
- `mag_dna_opt_l3` – as above, $\ell = 3$.

We call CAMq and CAMq(dna) as the generic variants, and CAMq(opt) as the optimized variant (shifts and adds replaced by a multiplication). There are a few parameters to set for the algorithms such as q-gram size, FAOSO striding parameter (k) and the quantized alphabet size σ' (only for HAM). We tested HAM with multiple σ' values ($\{4, 5, \ldots, 26\}$) and experimentally chose only a few later used in the tests, namely $\sigma' = 5$ for `dna`, $\sigma' = 14$ for `english` and `proteins` ($r \in \{10, 100\}$) and $\sigma' = 21$ for `proteins` ($r \in \{1\,000, 10\,000\}$). For all variants we set $q = \min(10, \max(2, \lfloor \log_{\sigma'}(rm) \rceil))$ and the FAOSO parameter k as mentioned

in Equation (1), but rounded to the nearest value from $\{1, 2, 4\}$, and the pattern length as follows:

$$m' = \begin{cases} \lfloor m/q \rfloor - 1, & \text{if } m \bmod q < q - 1, \\ \lfloor m/q \rfloor, & \text{otherwise.} \end{cases}$$

It is possible to tune the parameters for a certain dataset and achieve better results (because the formulas on q and k are designed to be optimal for random text). The RAM usage of our implementation can be roughly expressed as $16\sigma'^q + 24r + rm$ bytes, where 16 and 24 are the internal data structure sizes (in bytes). The variants other than HAM are less flexible in terms of σ', but, on the other hand, do not have the preprocessing phase and an additional array to store the alphabet mapping. We tested both methods generic and optimized (shifts and adds replaced by a multiplication). The tests were performed on a 64-bit machine, so there was a limitation for the maximum alphabet size and the value of q. For example, if $\ell = 2$, then we can (directly) use $q \leq 5$, therefore to deal with a larger q, we need to combine two smaller q-grams, e.g., take $S[0 \ldots 2]S[3 \ldots 5]$ to obtain a 6-gram.

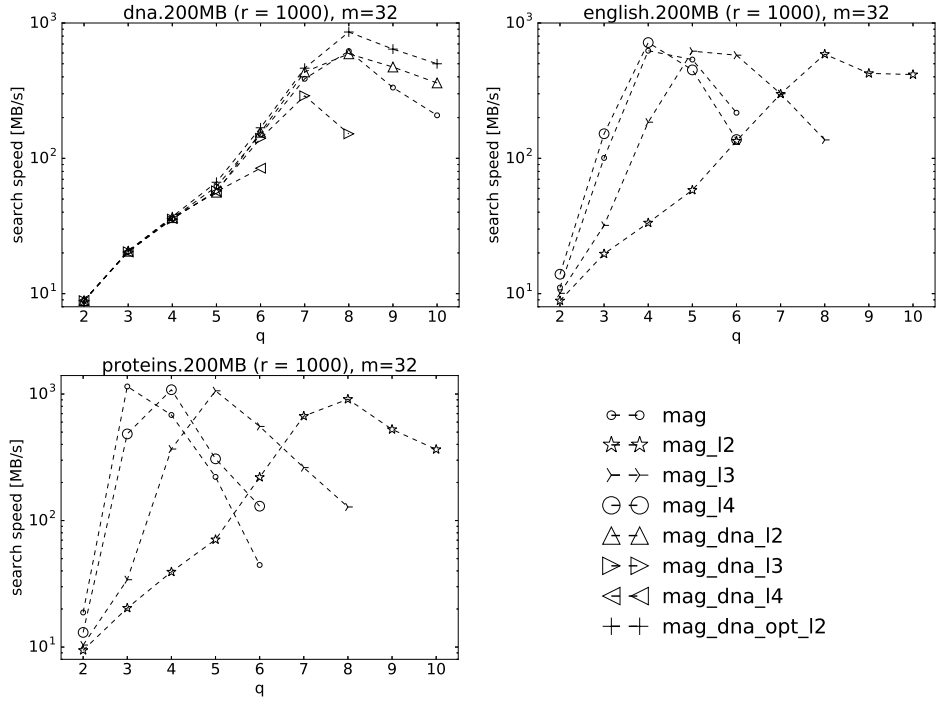


Figure 4. Search speeds for 1 000 patterns, $m = 32$, in function of q

Figure 4 presents search speeds for a collection of $r = 1\,000$ patterns of length $m = 32$ of the MAG algorithm variants using different alphabet mappings, a histogram based mapping (`mag`), and a combined mapping. The main advantage (and disadvantage) of HAM variant is the σ' flexibility that allows use of quite a large range of alphabet mapping. This parameter lets us adapt the HAM (`mag`) in terms of performance to certain dataset or amount of RAM. On the other hand, it forces us to find the most suitable configuration, what is quite tricky and highly depends on the text characteristic. We removed some variants from charts for clarity. The `CAMq(dna)` and `CAMq(opt)` variants were removed for `proteins` and `english` as the results were worse or similar to `CAMq`. The `mag_dna_opt_l3` was removed because the search speed was almost the same as `mag_dna_l3`. The next thing that may be noticed is the difference between larger and smaller alphabets. On `dna` the benefits in performance are visible for a higher q than on the `english` and `proteins` datasets. Another parameter whose optimal value may vary for different alphabet sizes is ℓ. As expected, a larger value of ℓ value is beneficial for larger alphabets (`english` and `proteins`). Note also that there is no point in setting ℓ above 2 for `dna` as its alphabet size is small (four symbols). Finally, the `dna` adapted variant (right shift) proved successful.

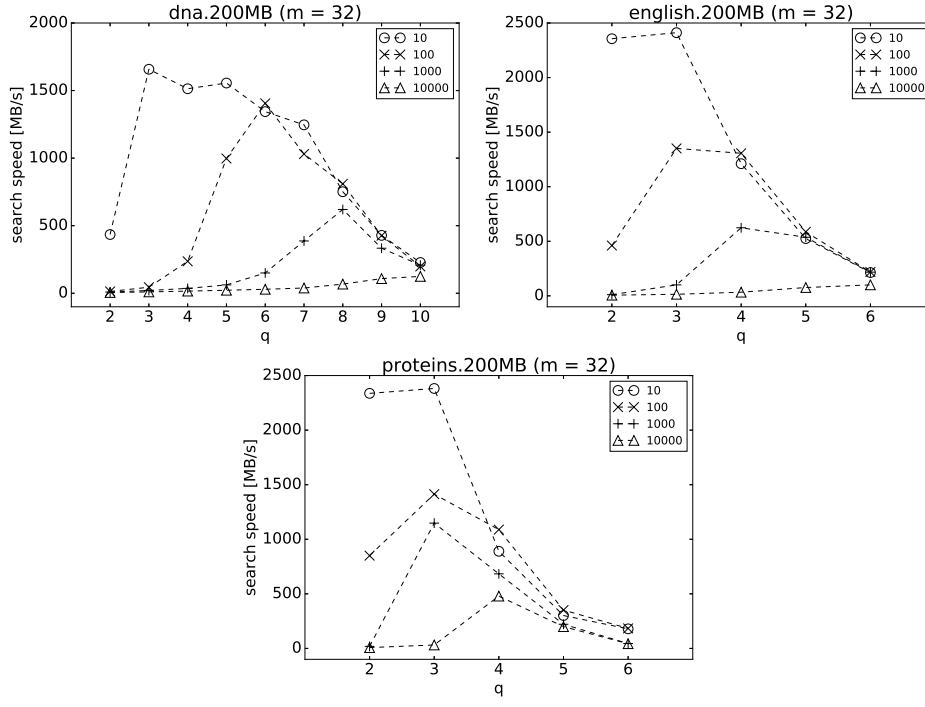


Figure 5. Search speeds for the pattern length $m = 32$ and varying q

We also show how MAG performance changes with growing q for different number of patterns (Figure 5). The figure presents HAM variant for different $r = \{10, 100, 1\,000, 10\,000\}$ as a separate function for multiple values of q values ($q \in \{2, 3, \ldots, 10\}$ for `dna` and $q \in \{2, 3, \ldots, 6\}$ for `english` and `proteins`). As expected, larger q makes sense for increasing number of patterns (r), but a too large value of it slows down the search, presumably due to many cache misses. If the alphabet size is small (such as `dna`) the optimal q-gram size is larger than for bigger alphabet (such as `english` and `proteins`). Note that due to the quantization the original alphabet size does not (significantly) affect the choice of q, but may affect the choice of optimal σ' as it is expected to be smaller than σ.

4.2 Other Solutions

After testing our variants we decided to compare them to other relevant algorithms from the literature, namely:

- BNDM on q-grams (BG) [30],
- Shift-Or on q-grams (SOG) [30],
- BMH on q-grams (HG) [30],
- Rabin-Karp combined with binary search and two-level hashing (RK) [30],
- Multibom and Multibsom, variants of Set Backward Oracle Matching [2],
- Succinct Backward DAWG Matching (SBDM) [17],
- Multi AOSO on q-grams (MAG) (this work).

We tested the competitors' algorithms for a variety of parameters and chose the fastest ones. Namely, there were executed: four variants of BG (using 2-grams, 3-grams, 3-grams with 4 subsets, 3-grams with 2 subsets), two variants of RK (with default parameters and second level hashing), two variants of HG (with default settings and 3-grams), three variants of SOG (2-grams, 2-grams AOSO and 3-grams), and one variant of SBDM ("-A -B -F 1", rank_g and CUSTSIGMA for `english`; some other configurations were excluded after preliminary experiments), Multibom and Multibsom (both with default parameters). All presented results include preprocessing and search times.

In Figure 6 we show the results of all the listed algorithms on `english`, with a fixed pattern length m and growing number of patterns r. The used pattern lengths (one for each plot) are $\{8, 16, 32, 64\}$. Note that some algorithms (or rather their available implementations) cannot handle longer patterns ($m = 64$). To make the chart clear we chose only two variants of our solution that seem to be the most interesting in this case, which are `mag` and `mag_l4`. The `mag_l4` dominates for longer patterns (32, 64) and its performance is mixed for $m = 8$ and $m = 16$. For short ($m = 8, 16$) and many ($r = 10\,000$) patterns SBDM achieves the best results. As expected, for all algorithms the search speed deteriorates with a growing number of patterns, and for $r = 10\,000$ and relatively long patterns ($m = 32$) only MAG

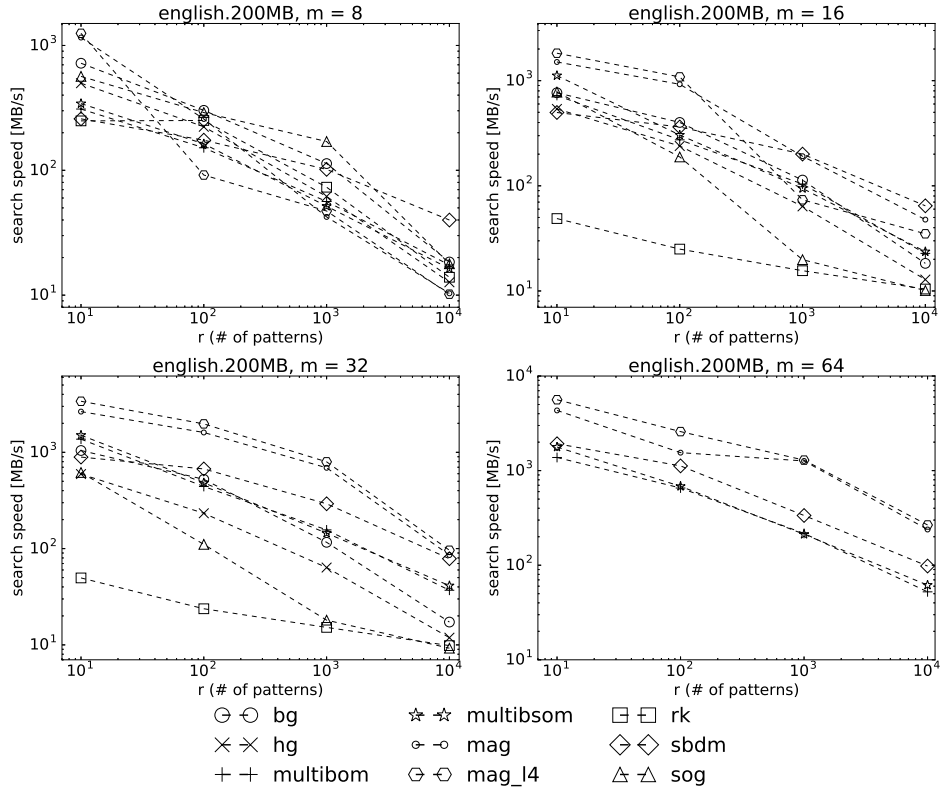


Figure 6. Search speeds for varying number of patterns $r = \{10, 100, 1\,000, 10\,000\}$ and pattern lengths $m = \{8, 16, 32, 64\}$

almost achieves 100 MB/s (the worst ones here, SOG and RK, are 10 times slower). However, the parameters of MAG are calculated using a formula designed for random text, so better results are possible with tuning the parameters for a particular dataset (actually, we achieved almost 200 MB/s for above example). Yet, this approach is rather inelegant and time-consuming in the construction phase.

In Figure 7 the number of patterns r is fixed (1 000), but m grows. We chose three variants, `mag` (the HAM variant) for all datasets, `mag_l4` for `english` and `proteins`, and `mag_dna_opt_l2` for `dna` set. MAG usually wins on `english` and `proteins` (except for the shortest patterns), yet is dominated by a few algorithms on `dna`. Overall, in the experiments the toughest competitor to MAG was SBDM, but in some cases (the shortest patterns ($m = 8$) on `english` and `proteins`) the winner was SOG.

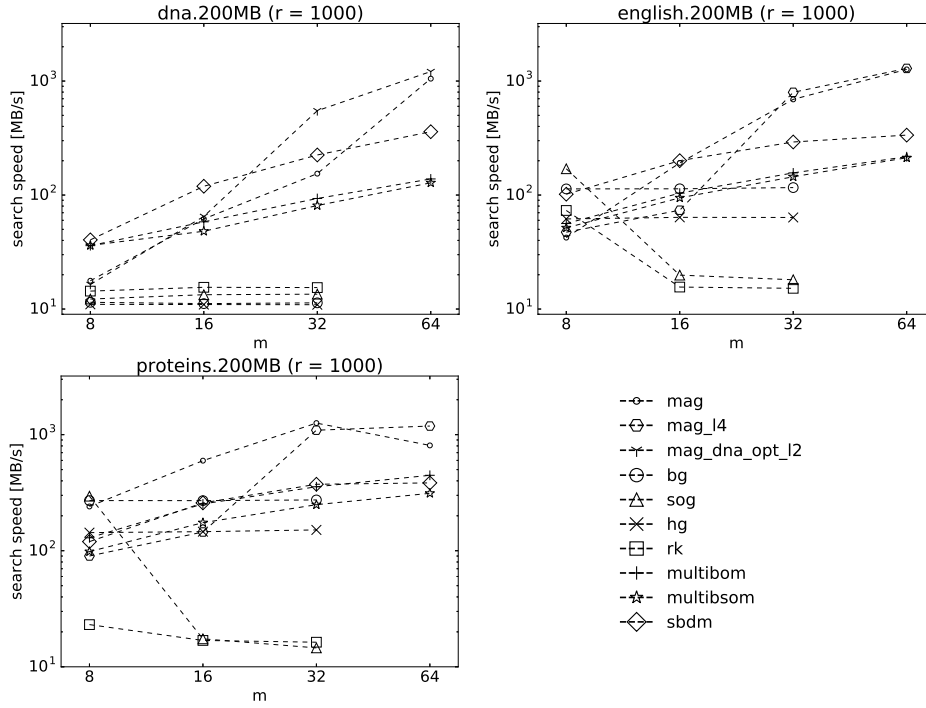


Figure 7. Search speeds for the number of patterns $r = 10\,00$ and varying pattern length m

4.3 MAG on ETDC

It is interesting to test multiple pattern matching also on compressed text. To this end, we chose ETDC (see Section 3.6), a popular byte code scheme, applied to the words of the text. The benefits of using ETDC are less text to search in, possibly less RAM used, and (as we expect) reduced search time. Similarly to the experiments on plain text, we use now different alphabet mapping methods. In this experiment we use $\sigma' = 26$ for `etdc_mag`, and for other variants the same σ' parameters as in the experiments with plain text. The variant naming convention is preserved. The ETDC-encoded `english` text (including the helper array A) takes between 33 (for the smallest A) and 35 (for the largest A) percent of the original text.

In Figure 8 the performance difference between all variants of MAG algorithm adapted for ETDC is presented. The ETDC variant has quite better performance than the corresponding solution on plain text. ETDC compression allows to obtain the compression ratio of factor 2.9 (as the ETDC-encoded file, the dictionary and the array A take in total 35 % of the original file). The most successful variant is `etdc_mag_l4`, which achieves not only the highest speed among all the variants but

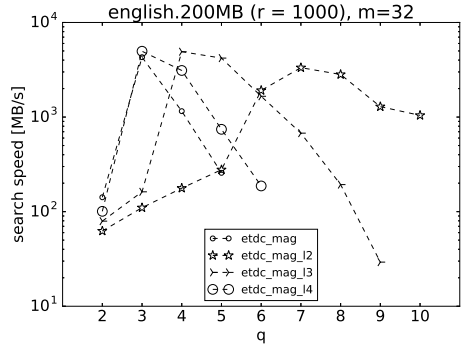


Figure 8. Search speeds for the pattern length $m = 32$ and varying q

then also makes use of a relatively small q (4), beneficial for RAM usage. This figure also shows the growth of the optimal q for decreasing ℓ parameter (i.e., for $\ell = 4$ the optimal q is 3, for $\ell = 3$ the optimal q is 4 and for $\ell = 2$ it is 7).

It is not trivial to compare the results of MAG in our two scenarios: on plain text and ETDC-compressed text. The reason is that in the compressed scenario the patterns must consist of words (which are obviously of varying length) and the length (in bytes) of the ETDC codewords for the input words tends to differ. Taking these into consideration, we decided to compare these algorithms by calculating the average length of decoded patterns (for the ETDC case) and then execute MAG tests with corresponding pattern lengths on plain text.

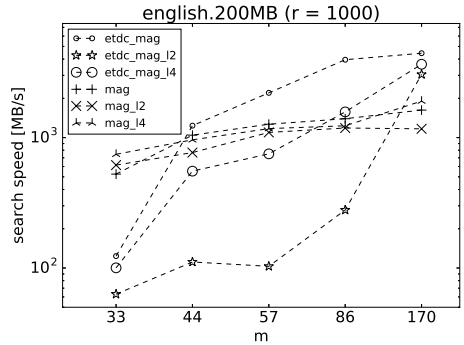


Figure 9. Search speeds (comparison of plain text variants against ETDC variants) for varying pattern lengths $m = \{33, 44, 57, 86, 170\}$

Figure 9 presents search speeds of MAG on plain text and MAG on ETDC-compressed text with varying pattern length in $\{33, 44, 57, 86, 170\}$ (averages for ETDC). The corresponding word counts (in case of ETDC) for each pattern length are $\{6, 8, 10, 16, 32\}$. As expected, the performance of MAG on ETDC data is much

better than on plain text. The speed of the best result of MAG on ETDC is by a factor of 2.8 higher (for $m = 86$ (16)) compared to the corresponding best result of MAG on plain text.

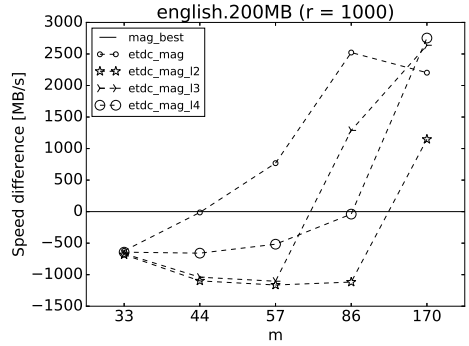


Figure 10. Difference of search speeds between the best plain text variant and ETDC variants for varying pattern lengths $m = \{33, 44, 57, 86, 170\}$

In Figure 10 we compare variants of MAG on ETDC (`etdc_mag`, `etdc_mag_l2`, `etdc_mag_l3`, `etdc_mag_l4`) against the best result of MAG on plain text (`mag`, `mag_l2`, `mag_l3`, `mag_l4`) which is labeled as `mag_best`. Overall, we can see a clear upward trend in the speed growth when pattern length is increasing. Note that the machine word size (64 bits) is limiting the MAG (`mag_best`) performance in terms of large pattern sizes, this is why we expected drastic speed increase for patterns longer than 64 characters. On the other hand, a longer machine word can also improve the performance of ETDC variants for even longer patterns.

5 CONCLUSIONS AND FUTURE WORK

Multiple string matching is one of the most explored problems in stringology. The presented algorithm, MAG, usually wins with its competitors on the three test datasets (`english`, `proteins` and `dna`). We discuss the pros and cons of various alternatives to achieve a possibly best combination of techniques. The main contribution and one of the key successful ideas was alphabet quantization such as bin-packing which is performed in a greedy manner, after sorting the original alphabet by frequency (HAM). We also proposed a different implementation of alphabet quantization, called combined alphabet mapping (CAMq, in some variants), that has a few advantages like faster preprocessing (there is no need to create a histogram), no array access (because there is no histogram) and less operations. The disadvantage, which may be called reduced flexibility, is strong dependence on machine word size, what significantly constraints the values of q and ℓ.

There is a number of interesting questions that we can present here. We analytically showed that the presented approach is sublinear on average, yet not average optimal. Therefore, is it possible to choose the algorithm's parameters in order to reach average optimality (for $m = O(w)$)?

Our experiments confirmed that dense codes (ETDC) for words not only serve for compressing data (texts), but also enable faster searching, for long enough patterns. Thanks to the fact that the encoded pattern is much shorter than the original, the actual input pattern length may be increased, which effectively raises the limit on the machine word size and provides better performance for longer patterns.

Real computers nowadays have a hierarchy of caches in their CPU-related architecture and it could be interesting to apply the I/O model (or cache-obvious model) for the multiple pattern matching problem. The cache efficiency issue may be crucial for very large pattern sets.

The underexplored power of the SIMD instructions also seems to offer great opportunities, especially for bit-parallel algorithms.

REFERENCES

[1] Aho, A. V.—Corasick, M. J.: Efficient String Matching: An Aid to Bibliographic Search. Communications of the ACM, Vol. 18, 1975, No. 6, pp. 333–340, doi: 10.1145/360825.360855.

[2] Allauzen, C.—Raffinot, M.: Factor Oracle of a Set of Words. Technical Report 99-11, Institut Gaspard-Monge, Université de Marne-la-Vallée, 1999.

[3] Baeza-Yates, R. A.—Gonnet, G. H.: A New Approach to Text Searching. Communications of the ACM, Vol. 35, 1992, No. 10, pp. 74–82, doi: 10.1145/135239.135243.

[4] Boyer, R. S.—Moore, J. S.: A Fast String Searching Algorithm. Communications of the ACM, Vol. 20, 1977, No. 10, pp. 762–772, doi: 10.1145/359842.359859.

[5] Brisaboa, N. R.—Fariña, A.—Navarro, G.—Paramá, J. R.: New Adaptive Compressors for Natural Language Text. Software: Practice and Experience, Vol. 38, 2008, No. 13, pp. 1429–1450, doi: 10.1002/spe.882.

[6] Burkhardt, S.—Kärkkäinen, J.: Better Filtering with Gapped q-Grams. Fundamenta Informaticae, Vol. 56, 2003, No. 1–2, pp. 51–70.

[7] Cantone, D.—Faro, S.—Giaquinta, E.: A Compact Representation of Nondeterministic (Suffix) Automata for the Bit-Parallel Approach. Information and Computation, Vol. 213, 2012, pp. 3–12, doi: 10.1016/j.ic.2011.03.006.

[8] Commentz-Walter, B.: A String Matching Algorithm Fast on the Average. In: Maurer, H. A. (Ed.): Automata, Languages and Programming (ICALP 1979). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 71, 1979, pp. 118–132, doi: 10.1007/3-540-09510-1_10.

[9] Crochemore, M.—Czumaj, A.—Gąsieniec, L.—Lecroq, T.—Plandowski, W.—Rytter, W.: Fast Practical Multi-Pattern Matching. Information Processing Letters, Vol. 71, 1999, No. 3–4, pp. 107–113, doi: 10.1016/s0020-0190(99)00092-7.

[10] Crochemore, M.—Hancart, Ch.—Lecroq, T.: Algorithms on Strings. Cambridge University Press, New York, USA, 2007, doi: 10.1017/cbo9780511546853.

[11] Crochemore, M.—Rytter, W.: Text Algorithms. Oxford University Press, 1994.

[12] Dori, S.—Landau, G. M.: Construction of Aho Corasick Automaton in Linear Time for Integer Alphabets. Information Processing Letters, Vol. 98, 2006, No. 2, pp. 66–72, doi: 10.1016/j.ipl.2005.11.019.

[13] Faro, S.—Külekci, M. O.: Fast Multiple String Matching Using Streaming SIMD Extensions Technology. In: Calderón-Benavides, L., González-Caro, C., Chávez, E., Ziviani, N. (Eds.): String Processing and Information Retrieval (SPIRE 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7608, 2012, pp. 217–228, doi: 10.1007/978-3-642-34109-0_23.

[14] Faro, S.—Külekci, M. O.: Towards a Very Fast Multiple String Matching Algorithm for Short Patterns. Proceedings of the Prague Stringology Conference, Prague, 2013, pp. 78–91.

[15] Fisk, M.—Varghese, G.: Fast Content-Based Packet Handling for Intrusion Detection. Technical report, DTIC Document, 2001, doi: 10.21236/ada406413.

[16] Fredriksson, K.: Shift-Or String Matching with Super-Alphabets. Information Processing Letters, Vol. 87, 2003, No. 4, pp. 201–204, doi: 10.1016/s0020-0190(03)00296-5.

[17] Fredriksson, K.: Succinct Backward-DAWG-Matching. ACM Journal of Experimental Algorithmics, Vol. 13, 2009, Art. No. 8, doi: 10.1145/1412228.1455263.

[18] Fredriksson, K.—Grabowski, Sz.: Average-Optimal String Matching. Journal of Discrete Algorithms, Vol. 7, 2009, No. 4, pp. 579–594, doi: 10.1016/j.jda.2008.09.001.

[19] Gusfield, D.: Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1997.

[20] Horspool, R. N.: Practical Fast Searching in Strings. Software: Practice and Experience, Vol. 10, 1980, No. 6, pp. 501–506, doi: 10.1002/spe.4380100608.

[21] Kandhan, R.—Teletia, N.—Patel, J. M.: SigMatch: Fast and Scalable Multi-Pattern Matching. Proceedings of the VLDB Endowment, Vol. 3, 2010, No. 1–2, pp. 1173–1184, doi: 10.14778/1920841.1920987.

[22] Knuth, D. E.—Morris, J. H.—Pratt, V. R.: Fast Pattern Matching in Strings. SIAM Journal on Computing, Vol. 6, 1977, No. 1, pp. 323–350, doi: 10.1137/0206024.

[23] Kouzinopoulos, C. S.—Michailidis, P. D.—Margaritis, K. G.: Parallel Processing of Multiple Pattern Matching Algorithms for Biological Sequences: Methods and Performance Results. Chapter 8. Systems and Computational Biology – Bioinformatics and Computational Modeling, InTech, 2011, pp. 161–182, doi: 10.5772/18488.

[24] Kouzinopoulos, C. S.—Michailidis, P. D.—Margaritis, K. G.: Multiple String Matching on a GPU Using CUDAs. Scalable Computing: Practice and Experience, Vol. 16, 2015, No. 2, pp. 121–137, doi: 10.12694/scpe.v16i2.1085.

[25] Navarro, G.—Fredriksson, K.: Average Complexity of Exact and Approximate Multiple String Matching. Theoretical Computer Science, Vol. 321, 2004, No. 2–3, pp. 283–290, doi: 10.1016/j.tcs.2004.03.058.

[26] Navarro, G.—Raffinot, M.: Fast and Flexible String Matching by Combining Bit-Parallelism and Suffix Automata. ACM Journal of Experimental Algorithmics, Vol. 5, 2000, Art. No. 4, doi: 10.1145/351827.384246.

[27] Navarro, G.—Raffinot, M.: Flexible Pattern Matching in Strings – Practical On-Line Search Algorithms for Texts and Biological Sequences. Cambridge University Press, 2002. ISBN 0-521-81307-7, 280 pp., doi: 10.1017/cbo9781316135228.

[28] Peltola, H.—Tarhio, J.: Alternative Algorithms for Bit-Parallel String Matching. In: Nascimento, M. A., de Moura, E. S., Oliveira, A. L. (Eds.): String Processing and Information Retrieval (SPIRE 2003). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 2857, 2003, pp. 80–93, doi: 10.1007/978-3-540-39984-1_7.

[29] Rivals, E.—Salmela, L.—Tarhio, J.: Exact Search Algorithms for Biological Sequences. Chapter 5. Algorithms in Computational Molecular Biology: Techniques, Approaches and Applications, John Wiley and Sons, Inc., 2010, pp. 91–111, doi: 10.1002/9780470892107.ch5.

[30] Salmela, L.—Tarhio, J.—Kytöjoki, J.: Multipattern String Matching with q-Grams. ACM Journal of Experimental Algorithmics, Vol. 11, 2006, Art. No. 1.1, doi: 10.1145/1187436.1187438.

[31] Susik, R.—Grabowski, Sz.—Fredriksson, K.: Multiple Pattern Matching Revisited. Proceedings of the Prague Stringology Conference, Prague, 2014, pp. 59–70.

[32] Wu, S.—Manber, U.: A Fast Algorithm for Multi-Pattern Searching. Report TR-94-17, Department of Computer Science, University of Arizona, Tucson, AZ, 1994.

[33] Yang, P.—Liu, L.—Fan, H.—Huang, Q.: Fast Multi-Pattern String Matching Algorithms Based on q-Grams Bit-Parallelism Filter and Hash. In: Lu, W., Cai, G., Liu, W., Xing, W. (Eds.): Proceedings of the 2012 International Conference on Information Technology and Software Engineering. Springer, Berlin, Heidelberg, Lecture Notes in Electrical Engineering, Vol. 211, 2013, pp. 487–495, doi: 10.1007/978-3-642-34522-7_52.

[34] Yao, A. C.-C.: The Complexity of Pattern Matching for a Random String. SIAM Journal on Computing, Vol. 8, 1979, No. 3, pp. 368–387, doi: 10.1137/0208029.

Robert Susik received his M.Eng. and Ph.D. degrees from the Lodz University of Technology in 2012 and 2018, respectively. Currently, apart from string matching algorithms, his interests mostly focus on data archiving, machine learning and data science.

Szymon Grabowski received his M.Sc. degree from the University of Lodz in 1996, his Ph.D. degree from AGH University of Science and Technology in Cracow in 2003, and the habilitation degree from the Systems Research Institute of the Polish Academy of Sciences in Warsaw in 2011. His former research, including Ph.D. dissertation, involved nearest neighbor classification methods in pattern recognition, also with applications in image processing. Currently, his main interests are focused on string matching and text indexing algorithms, and data compression. Some of his particular research topics include various approximate string matching problems, compressed text indexes, and XML compression. He has published over 120 papers in journals and conferences. He is currently Professor at the Institute of Applied Computer Science of the Lodz University of Technology.

Kimmo Fredriksson received his computer science M.Sc. and Ph.D. degrees from the University of Helsinki in 1997 and 2001, respectively. His research interests include wide variety of string matching problems as well as indexing techniques for searching in metric spaces. He has published about 60 papers on these topics in international conferences and journals. He has the position of Adjunct Professor at the University of Eastern Finland.

Computing and Informatics, Vol. 38, 2019, 963–985, doi: 10.31577/cai_2019_4_963

OPTIMIZATION OF THE MORPHER MORPHOLOGY ENGINE USING KNOWLEDGE BASE REDUCTION TECHNIQUES

Gábor Szabó, László Kovács

Institute of Information Technology
University of Miskolc
Miskolc-Egyetemváros, H 3515, Hungary
e-mail: szgabsz91@gmail.com, kovacs@iit.uni-miskolc.hu

Abstract. Morpher is a novel morphological rule induction engine designed and developed for agglutinative languages. The Morpher engine models inflection using general string-based transformation rules and it can learn multiple arbitrary affix types, too. In order to scale the engine to training sets containing millions of examples, we need an efficient management of the generated rule base. In this paper we investigate and present several optimization techniques using rule elimination based on context length, support and cardinality parameters. The performed evaluation tests show that using the proposed optimization techniques, we can reduce the average inflection time to 0.52 %, the average lemmatization time to 2.59 % and the number of rules to 2.25 % of the original values, while retaining a high correctness ratio of 98 %. The optimized model can execute inflection and lemmatization in acceptable time after training millions of items, unlike other existing methods like Morfessor, MORSEL or MorphoChain.

Keywords: Machine learning, natural language processing, inflection, lemmatization, agglutination, morphology, optimization, rule base reduction

Mathematics Subject Classification 2010: 68T50

1 INTRODUCTION

Natural language processing is one of the actively researched scientific areas nowadays. One of the goals of these research projects is to create automated methods

for processing free texts. Developing efficient NLP applications requires processing modules on several abstraction layers, among which the lowest layer is morphology.

Morphology deals with the inner components of words called morphemes, that are the smallest units of the language with meaning [1]. The grammatically correct root form of a word is called a lemma, whose base meaning can be modified by adding affixes to it. These affixes can appear before the lemma (prefix), after the lemma (suffix) or even inside the word (infix). The complexity of automated morphological processing of words comes from the fact that in some cases adding affixes can modify the root form as well, e.g. *(try, tried)*. Inflection is the operation that produces the inflected form from the lemma, while lemmatization determines the lemma and the list of affix types in an arbitrary word.

Historically the first successfully applied, popular model for inducing inflection rules of agglutinative languages was the two-level morphology model [6]. The name of the model comes from the fact that the inflected word forms are represented on two levels: the surface level stores the written form, and the lexical level stores the morphological structure. Dictionaries were used to collect the valid lemmas and affix types of the target language, while FSTs (finite-state transducers) were trained to apply the required transformations on the input words. The FST model appears in several other publications as well, because this model fits the need of transforming one word to another based on some pre-learnt rules. One of the most widely used FST category is the subsequential transducer model [9, 10] that can be trained using the OSTIA algorithm [4, 12].

A very simple transformation engine called the tree of aligned suffix rules (TASR) is proposed in [15]. It is a supervised model that generates elementary rules from the training word pairs and stores them in a tree. Unfortunately the TASR model can only handle suffix rules and not prefix or infix rules. According to the experiments of [7], for languages that contain only suffix rules, the TASR model can be used very efficiently.

A more recent unsupervised segmentation model is Morfessor [3, 20]. This model uses a statistical training method to determine the morpheme boundaries inside the words. One downside of the original Morfessor model is that it only uses global frequencies and not local probabilities, so it does not take into account which morpheme can come after which other morphemes. However, there are several other methods that either use the Morfessor model or extend it.

The MorphoChain engine [11] is one such extension of Morfessor. The main addition of this improved model is that it also uses orthographic and semantic views of the input words, thus improving the segmentation correctness. Another improvement was presented in [2] that adapts the MorphoChain segmentation system in a way that it can identify identical morphemes with spelling differences. Based on the results, this model outperforms both the original MorphoChain system and MORSEL [8].

The target language of our research is Hungarian, that is highly agglutinative and morphologically complex language. It has many affix types and each word can contain multiple affixes. Most of the affix types are suffixes, but there are a couple

of prefix and prefix-suffix affix types as well, like comparative and superlative[1]. In many cases, adding an affix to a word also changes some characters in the base form.

As an example, *meg|szent|ség|telen|ít|hetetlen|ség|es|ked|és|eitek|ért* is one of the longest Hungarian artificial words that means *"for your [plural] continued behaviour as if you could not be desecrated"*. It contains 11 affixes. On the other hand, one lemma might have several inflected forms: in our data set, the word *konfigurál* (*configure*) has 86 different forms in our data set, including *átkonfigurálom* (*I reconfigure it*), *konfigurálásukkal* (*with their configuration*), etc. Such examples show the complexity of the target language.

For Hungarian, Hunmorph-Ocamorph [18], Hunmorph-Foma [5], Humor [14] and Hunspell [13] are four popular morphology tools. These analyzers can determine all the possible morphological structures of input words, including their lemma and the list of affix types found in the input word. According to the analysis of these tools [16], Hunmorph-Ocamorph is the most advanced among them. Its engine (Ocamorph) is language independent, and Morphdb.hu [19] is the language dependent knowledge database that stores the set of affix types and their related transformation rules. Hunmorph-Ocamorph can recognize more than 4 million words using the lexical database of Morphdb.hu, but unfortunately this database has been constructed by human experts, and not learnt by an automated learning algorithm.

The Morpher[2] system [17] is a morphology model that can statistically learn prefix, suffix and infix transformation rules from a training set containing (word, lemma, part of speech, morphosyntactic tags) tuples. The model is suitable for complex agglutinative languages like Hungarian, and it can handle multiple affix types. According to the evaluation, Morpher can learn 100 000 training items in about 4 seconds, then execute inflection and lemmatization in about 2.4 milliseconds and 2.4 seconds, respectively, reaching about 97 % of average correctness ratio for previously unseen words. The known limitation of the model is that the lemmatization time increases exponentially as we increase the training data set.

The main goal of this paper is to perform space and time complexity analysis of the baseline Morpher model and introduce several optimization techniques for rule base reduction so that the engine can scale more easily to large training data sets containing millions of items.

2 THE MORPHER MODEL

The training data of the Morpher model is a D_{train} set containing training items in the form of $\left(w, \bar{w}, \bar{T}_0, \langle T_i \rangle_{i=1}^{k}\right)$, where $w \in W$ is the inflected form, $\bar{w} \in \bar{W}$ is the lemma of w, $\bar{T}_0 \in \bar{\mathbb{T}}$ is the part of speech, and $\langle T_i \rangle_{i=1}^{k}$ is the ordered list of k affix types in w, $T_i \in \mathbb{T}$.

[1] *Jó, jobb, legjobb* means *good, better, best.*

[2] `https://github.com/szgabsz91/morpher`

From this training data, Morpher can learn string-based inflection and lemmatization rules, the conditional probabilities of the affix types, as well as the valid lemmas and their parts of speech.

Figure 1 displays the overall structure of the model.

Manager: manages the inflection and lemmatization tasks by coordinating the transformation engines generated for each affix type. Has a knowledge about the valid lemmas of the target language and the conditional probabilities of the affix type chains.

Transformation engines: one transformation engine can learn the transformation rules of a single affix type. For every affix type, a separate transformation engine is generated and trained.

Probabilities: during the training phase, all the possible affix type chains are analyzed and the conditional probabilities are stored and updated so that the manager knows which affix type can come after which other affix types and with how much probability.

Lemmas: all the valid lemmas are stored, as well as their associated parts of speech so that during lemmatization the engine knows when it can stop processing an affix type chain candidate.

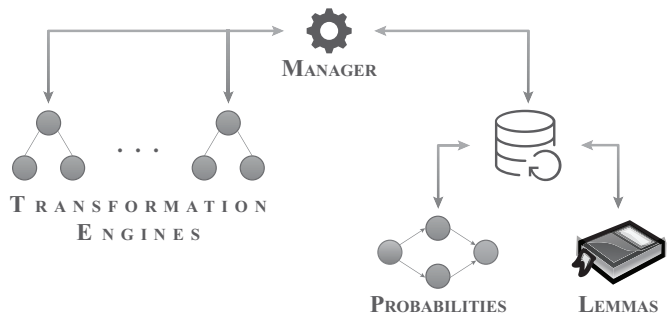


Figure 1. The main components of the Morpher model

2.1 Transformation Engine

The responsibility of the transformation engine model is to learn simple string-based transformation rules from word pairs of an affix type induced by the manager. The transformation engines can inflect and lemmatize input words based on the induced transformation rules. We build a transformation engine for every affix type of the target language.

During the training phase, after the word pairs for an affix type are induced by the manager, we identify the changing parts of the base forms, and generate

a number of atomic rewrite rules for them in the form of $R = (\alpha, \sigma, \tau, \omega)$, where α is the prefix, σ is the changing part of the base form, τ is the replacement string and ω is the suffix. The first atomic rule is called the core atomic rule $\hat{R}$ for which $|\alpha| = |\omega| = 0$. The other rules are extended from this core atomic rule by prepending and appending one character at a time, from the words to the context, essentially filling in α and ω.

For each atomic rule, we calculate different statistics, including the support value and the word frequency. The support of a rule equals the number of training word pairs that the rule matches, while the word frequency is the sum of frequencies of the related training words. A word's frequency is equal to its number of occurrences in the input free text sources, from which the training data was generated.

During inflection and lemmatization, the task is to find the best matching atomic rules for the input word. For this, we define a fitness function that returns the goodness value of an atomic rule R for the input word w. This fitness function is

$$f(R \mid w) = \frac{|\gamma(R)|}{|w|} \cdot \delta(\gamma(R), w)$$

where

- R is the atomic rewrite rule in question,
- w is the input word that needs to be inflected or lemmatized,
- $\gamma(R)$ is the context of the atomic rewrite rule,
- δ is a function that either returns 0 if the context is not found in the input word, or 1 otherwise. (In this sense, it is similar to the Kronecker delta, but could be implemented differently.)

The rule context is $\alpha+\sigma+\omega$ during inflection and $\alpha+\tau+\omega$ during lemmatization. Using the fitness function, we can select the matching atomic rewrite rules for the input word.

2.2 Conditional Probabilities

During the training phase, the Morpher model learns all the possible affix type chains, and their conditional probabilities. M is the function that can return the probability of an affix type chain:

$$M\left(\bar{T}_0, T_1, \ldots, T_i\right) = \begin{cases} P\left(\bar{T}_0\right), & \text{if } i = 0, \\ P\left(\bar{T}_0\right) \cdot \prod_{j=1}^{i} P\left(T_j \mid \bar{T}_0, T_1, \ldots, T_{j-1}\right), & \text{if } i = 1, 2, \ldots \end{cases}$$

We use the relative frequencies in the training data set D_{train} for probability calculation.

2.3 Manager

The manager submodule coordinates all the other submodules to learn the required morphological features of the training word pair set, as well as perform inflection and lemmatization, producing complex responses with multiple steps.

The input of the inflection operation is a lemma $\bar{w}_0$ and a set of affix types $\{T_1, T_2, \ldots, T_k\}$. The output is a set of n items in the form of

$$\left(\bar{T}_0, \langle (T_i, w_i) \rangle_{i=1}^{m}, \vartheta\right)$$

containing

- the $\bar{T}_0$ part of speech,
- the (T_i, w_i) steps, where $\langle T_i \rangle_{i=1}^{m}$ is a valid permutation of the input affix types according to M, and $\langle w_i \rangle_{i=1}^{m}$ are the produced inflected forms, and
- the ϑ aggregated weight of the response.

The responses are sorted by ϑ, in a descending order.

Similarly, the input of the lemmatization operation is an arbitrary inflected word form w, and the output is a set of n items in the form of

$$\left(\langle (T_i, w_i) \rangle_{i=m}^{1}, \bar{T}_0, \vartheta\right)$$

containing

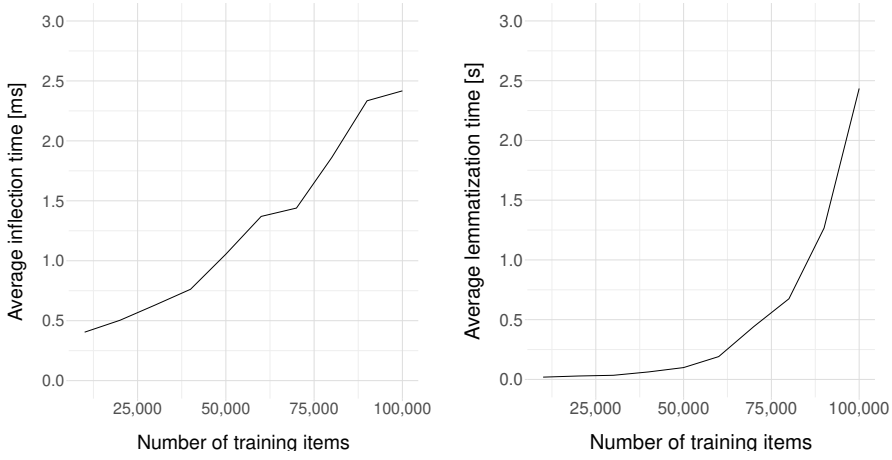
- the (T_i, w_i) steps ($w_m = w$), where $\langle T_i \rangle_{i=m}^{1}$ is a valid affix type chain, and $\langle w_i \rangle_{i=m}^{1}$ are the produced base forms,
- the $\bar{T}_0$ part of speech, and
- the ϑ aggregated weight of the response.

The responses are sorted by ϑ, in a descending order. The ϑ aggregated weight is calculated using the normalized affix type conditional probability, and the aggregated fitness of the output words in the steps.

3 OPTIMIZATION TECHNIQUES

Morpher's generalization ability seemed to be promising, reaching about 97 % of average correctness ratio in case of a training data set containing 100 000 random items and evaluating 10 000 previously unseen random words. On the other hand, we can see a linear increase of average inflection time and an exponential increase of lemmatization time, as shown in Figures 2 a) and 2 b).

This increase is due to the nature of the problems. For example the exponential increase of the lemmatization process is caused by the fact that the manager needs to try all the possible preceding affix type candidates at every affix type. This cannot be changed, because we do not know the exact set of affix types found in the



a) Average inflection time b) Average lemmatization time

Figure 2. Average inflection and lemmatization times of the baseline Morpher model

input word. However, we can extend the number of training items with which the Morpher system can perform inflection and lemmatization in acceptable time.

In order to reduce the size of the knowledge base, we need to modify the training phase of the individual transformation engines associated with the affix types of the target language, since the other components of the Morpher model (the lemma database and the conditional probability store) cannot lose information without a significant loss of correctness ratio.

3.1 Eliminating the Redundant Atomic Rewrite Rules

The main idea behind this optimization technique is to drop the redundant atomic rules that are covered by other rules in the rule base.

Definition 1 (Redundant atomic rule)**.** The atomic rule $R_i = (\alpha_i, \sigma_i, \tau_i, \omega_i)$ is a redundant rule if and only if there exists another $R_j = (\alpha_j, \sigma_j, \tau_j, \omega_j)$ atomic rule in the rule base such that $\gamma(R_j) \subseteq \gamma(R_i)$, $\sigma_i = \sigma_j$ and $\tau_i = \tau_j$. In this case we say R_i is covered by R_j.

As an example, the contexts of $R_1 = (\text{alm}, \text{a}, \text{át}, \#)$[3] and $R_2 = (\text{-}, \text{a}, \text{át}, \#)$ are $\gamma(R_1) = \text{alma}\#$ and $\gamma(R_2) = \text{a}\#$, respectively. If both rules are in the rule base, we can say that R_1 is redundant, since $\gamma(R_2) \subseteq \gamma(R_1)$ and the transformation (σ and τ components) are also the same, i.e. R_1 is covered by R_2.

On the other hand, if $R_3 = (\text{toll}, \text{-}, \text{at}, \#)$ and $R_4 = (\text{l}, \text{-}, \text{t}, \#)$ are part of the same rule base, they do not cover each other. Although $\text{l}\# = \gamma(R_4) \subseteq \gamma(R_3) = \text{tollat}\#$,

[3] # is the special word-end symbol.

R_3 is not redundant, because the transformations are different: R_3 appends *'at'* at the end of the word, while R_4 only appends *'t'*.

The learning algorithm can detect the redundant rules during the rule generation process, and it immediately blocks their generation. This means that the redundant rules are not stored in the rule base, and we do not have to execute a second wave of rule filtering after the original training phase. We introduce a new p_{max} optimization parameter. If for an arbitrary word the atomic rules $\hat{R}_1$, R_2, ..., R_k were generated ($\hat{R}_1$ being the core atomic rule), then using p_{max} only $\hat{R}_1$, R_2, ..., R_l will be retained, where $l = min\,(p_{max}, k)$. Those atomic rewrite rules that have a longer context than R_l are simply omitted.

Proposition 1. Using $p_{max} = 1$, storing only $\hat{R}_1$ for each word pair and dropping the other $R_2, \ldots, R_k$ extended atomic rules is equivalent with generating every possible atomic rule and then dropping all the redundant atomic rules.

Proof. According to Definition 1, an atomic rule $R_i = (\alpha_i, \sigma_i, \tau_i, \omega_i)$ is redundant if and only if there exists another atomic rule $R_j = (\alpha_j, \sigma_j, \tau_j, \omega_j)$ such that $\gamma\,(R_j) \subseteq \gamma\,(R_i)$, $\sigma_i = \sigma_j$ and $\tau_i = \tau_j$.

We can assume indirectly that by executing the first part of the proposition, there remains at least one redundant atomic rule $\tilde{R} = (\tilde{\alpha}, \tilde{\sigma}, \tilde{\tau}, \tilde{\omega})$. This means that there is at least one other atomic rule $R = (\alpha, \sigma, \tau, \omega)$ such that $\gamma\left(\tilde{R}\right) \subseteq \gamma\,(R)\,, \tilde{\sigma} = \sigma$ and $\tilde{\tau} = \tau$.

From these formulae, we can see that $\gamma\left(\tilde{R}\right) = \tilde{\alpha} + \tilde{\sigma} + \tilde{\omega} \subseteq \alpha + \sigma + \omega = \gamma\,(R)$ and since $\tilde{\sigma} = \sigma$, we can see that $\tilde{\alpha} + \sigma + \tilde{\omega} \subseteq \alpha + \sigma + \omega$.

This means that $|\tilde{\alpha} + \sigma + \tilde{\omega}| \leq |\alpha + \sigma + \omega|$. There are two cases to check:

- If $|\tilde{\alpha} + \sigma + \tilde{\omega}| = |\alpha + \sigma + \omega|$, then $\tilde{R} = R$ (as all the components are equal due to both rules being core atomic rules due to $p_{max} = 1$), so $\tilde{R}$ is a non-redundant item in the rule database.
- If $|\tilde{\alpha} + \sigma + \tilde{\omega}| < |\alpha + \sigma + \omega|$, then $|\tilde{\alpha}| = |\tilde{\omega}| = |\alpha| = |\omega| = 0$ since both $\tilde{R}$ and R are core atomic rules. This means that $|\sigma| < |\sigma|$, which is a contradiction.

From both cases we get a contradiction, which means that the proposition is true. □

3.2 Limiting the Generalization Factor

The potential problem with p_{max} optimization, especially using $p_{max} = 1$ is that we only retain atomic rewrite rules with very short contexts. This means that the matching rules for the input word might have very different $\sigma \Rightarrow \tau$ transformations, increasing the number of outputs at each affix type step, making the inflection and lemmatization processes extremely slow. This effect is called overgeneralization and can be prevented by also retaining some redundant rules to increase the information in the system.

In order to avoid this overgeneralization effect, we introduce another parameter called p_{gen}, that identifies the minimum context length of the generated atomic rewrite rules. This means that for all the retained atomic rules, $\gamma(R) \geq p_{gen}$. Every other atomic rule is dropped during the rule generation process.

This way we can limit the number of atomic rules from below, while p_{max} is an upper limit on the atomic rule context length. We can also combine these two parameters, retaining a slice of all the possible atomic rewrite rules. For example, $p_{max} = 2$ and $p_{gen} = 3$ will retain rules whose context contains at least two characters, but only 3 of these rules per each word pair. In this way, we drop the most general rules ($\gamma(R) = 1$ and $\gamma(R) = 2$), and also drop the most specific rules where $\gamma(R) \geq 2 + 3 = 5$ in case of suffix rules.

3.3 Indirect Data Cleaning

While the p_{max} and p_{gen} parameters drop the atomic rewrite rules based on their contexts, we can also drop some rules based on the statistical attributes of the training data set.

For each atomic rule, we calculate a support value and a word frequency value, as described in Section 2. The support value is the number of training word pairs that the rule matches, while the word frequency is the sum of frequencies of the related training words. A word's frequency is equal to the number of occurrences in the input free text sources, from which the training data was generated.

For the support and word frequency based elimination method, we introduce the p_{supp} and p_{freq} parameters that drop every atomic rule whose support is less than p_{supp}, or whose word frequency is less than p_{freq}.

This optimization method is based on the widely used frequency based reduction concept that can be found in other research areas as well, such as association rule mining, where only frequent item sets are considered during rule generation. This also means that we perform an indirect data cleaning, since rare rules apply to fewer words. As the training data is generated automatically, it can contain words with typos or otherwise meaningless words that can be omitted using these two parameters.

4 SPACE AND TIME COMPLEXITY ANALYSIS

4.1 Space Complexity

The number of transformation engines is equal to the number of affix types, so the space complexity is $\Omega(|\mathbb{T}|)$.

The number of conditional probability values for inflection is equal to the number of valid affix type orders: $\Omega\left(\left|\left\{\left(\bar{T}_{i_0}, T_{i_1}, \ldots, T_{i_k}\right) \mid M\left(\bar{T}_{i_0}, T_{i_1}, \ldots, T_{i_k}\right) > 0\right\}\right|\right)$.

The upper size limit of the lemma database can be approximated with the number of training items. In the worst case, every word in the training data set has different lemmas, and as such, the size complexity is $O(|D_{\text{train}}|)$.

The number of generated rules depends on the number of word pairs for the related affix type from D_{train}. In the worst case, it equals the number of training items where the last affix type is the associated affix type of the transformation engine. Therefore the number of word pairs generated to train the transformation engine of the affix type T can be approximated with $O\Big(\Big|\Big\{\Big(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\Big) \in D_{\text{train}} \mid T_k = T\Big\}\Big|\Big)$.

For every deduced word pair we generate a number of atomic rules. The approximation of the number of generated atomic rewrite rules for the word pair (w_1, w_2) is $\max(|w_1|, |w_2|) - |\sigma|$ without any optimizations. For the whole transformation engine related to the affix type T, the approximation of the generated atomic rewrite rules is

$$O\left(\left|\left\{\left(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\right) \in D_{\text{train}} \mid T_k = T\right\}\right| \cdot \left(\max\left(|w_{j_1}|, |w_{j_2}|\right) - |\sigma_j|\right)\right) \tag{1}$$

where the j index refers to the word pair for which the right component is maximal.

The optimization techniques introduced in Section 3 optimize the right component of the above formula. The p_{max} optimization parameter (Subsection 3.1) makes sure that the right component is at most p_{max}:

$$O\left(\left|\left\{\left(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\right) \in D_{\text{train}} \mid T_k = T\right\}\right| \cdot \min\left(p_{max}, \max\left(|w_{j_1}|, |w_{j_2}|\right) - |\sigma_j|\right)\right). \tag{2}$$

Using $p_{max} = 1$, this formula will be as simple as

$$O\left(\left|\left\{\left(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\right) \in D_{\text{train}} \mid T_k = T\right\}\right|\right).$$

The p_{gen} optimization parameter (Subsection 3.2) will result in a space complexity of

$$O\left(\left|\left\{\left(w, \bar{w}, \bar{T}_0, \langle T_i\rangle_{i=1}^{k}\right) \in D_{\text{train}} \mid T_k = T\right\}\right| \cdot \left(\max\left(|w_{j_1}|, |w_{j_2}|\right) - |\sigma_j| - \Upsilon_{gen}\right)\right) \tag{3}$$

where Υ_{gen} denotes the minimum number of generated atomic rules that have a context shorter than p_{gen} among the training word pairs.

Asymptotically this means that in the worst case, no reduction occurs, if all the generated atomic rules have at least as long context as p_{gen}. Additionally the redundant rules are eliminated after processing each word pair or after each training iteration. However, this cannot be estimated, since the final number of retained atomic rules depends on the quality of D_{train}.

The space complexity of the p_{supp} and p_{freq} optimization parameters, i.e. how many atomic rewrite rules are retained by them, depends greatly on the training data set D_{train}. While the support value only refers the number of words for which the given atomic rule is generated, the word frequency also contains information about the free text sources from which the training items in D_{train} were constructed.

4.2 Time Complexity

The training phase consists of three main parts:

- Steps with $O(1)$ time complexity like storing the lemmas or updating the conditional probability values for each training item.
- Generating word pairs from the training items for each affix type.
- Generating atomic rewrite rules from the training word pairs for each affix type.

The generation of word pairs can be done in $O(|D_{\text{train}}|^2)$ time, since we need to find all the possible item pairs that are adjacent in the affix type chains. However, we can improve the pairing algorithm by only going through the items with the same lemma as the item under processing. This way the majority of the search space remains untouched. Also, for the set of items that have only one affix type, we can generate a word pair using the lemma and the word of the same item without any searching, in $O(1)$ time.

For every training word pair, we first have to find the core. This can be done in $O(\max(|w_1|, |w_2|))$ time for the word pair (w_1, w_2). Then we need to generate the required atomic rewrite rules. Every rule can be generated in $O(1)$ time, so the approximation depends on the number of generated atomic rewrite rules, see Formulae (1), (2) and (3). Without any optimization, the whole generation process can be done in $O(\max(|w_{j_1}|, |w_{j_2}|) - |\sigma_j|)$ time per word pair. This can be reduced to $O(p_{max})$ using p_{max} optimization that will result in $O(1)$ time complexity using $p_{max} = 1$. Using p_{gen} optimization, the generation time of the atomic rules will also be $O(\max(|w_{j_1}|, |w_{j_2}|) - |\sigma_j| - \Upsilon_{gen})$. The time complexity of the p_{supp} and p_{freq} optimization parameters cannot be approximated accurately, since the number of retained atomic rules depends on the quality of the training data.

The first task during inflection is to generate all the valid orders of the given k affix types. This can be done in at most $O(k!)$ steps. In the worst case, all the possible permutations are valid, and for each permutation we have to check if the chain's conditional probability is positive, which can be done in $O(1)$. For every possible valid order, we need to go through the affix type chain and perform local inflection based on the atomic rewrite rules of the appropriate affix types one by one. We assume that applying an atomic rule on a word and checking if an atomic rule matches a word can be done in constant time, so at every affix type the generation of the inflected forms can be approximated with $O(|\{R\}|)$, which will be either Formula (1), (2) or (3) as we saw earlier, based on the applied optimization techniques.

As for lemmatization, the provided input does not contain any information about how many and which affix types will appear in the word to lemmatize. In the worst case, there may be $O(|\mathbb{T}|)$. At every step, the number of atomic rules to process and potentially apply can be approximated with Equation (1), (2) or (3) based on the optimization technique we used during training. The number of previous affix types

that need to be checked is $O(|\mathbb{T}| - 1)$. This means that all in all lemmatization can be done in exponential time, roughly in $O(|D_{\text{train}}|^{|D_{\text{train}}|})$ time at most.

Although the size of the knowledge base might seem to be the biggest reason of the increase of average inflection and lemmatization costs, there are also other factors to be considered. For example if we eliminate too many redundant atomic rules with longer contexts, only atomic rules with shorter contexts will remain in the rule base. This can cause many rules to match the input words, resulting in many inflection and lemmatization responses for each affix type. Having long affix type chains, the execution time can increase combinatorically.

5 TEST SYSTEM

For the evaluation of the baseline Morpher model and the optimization techniques, we implemented a test system using the Java 11 programming language. With the modern language features we could introduce clean interfaces among the different submodules, using the modularization techniques introduced by Java 9. Moreover parallel streams were used for processing large amounts of data in parallel, in a functional manner. These features make the implemented system more maintainable and efficient.

Figure 3 presents the data pipeline of the evaluation test system.

- Initially, a large number of Hungarian words were collected from the web using the site of the National Széchenyi Library[4].
- Hunmorph-Ocamorph [18] was used to analyze these words, creating a pre-annotated corpus.
- The corpus was fed to the data generator that emitted both training data and evaluation data.
- The training data was given to the trainer submodule that returned a trained instance of the model.
- The evaluator submodule received the trained model and the evaluation data, and performed several tests to evaluate the model against different metrics.

The number of sample word pairs generated by this data pipeline was 3 612 494.

6 EVALUATION

We examine several metrics during evaluation so that we can compare the baseline Morpher method with other existing models, and evaluate the optimization parameters. These metrics include

- the average training, inflection and lemmatization time,

[4] `http://mek.oszk.hu`

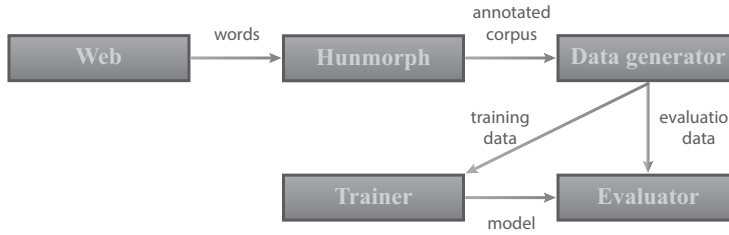


Figure 3. Test system pipeline

- the average number of responses and the average index of the expected response,
- the average correctness ratio,
- the average number of atomic rewrite rules and the average size of the knowledge base.

All of these metrics should be minimized except for the average correctness ratio. The performed tests are performed using the following steps:

1. Generate n_1 training items and train the model.
2. Generate n_2 evaluation items and evaluate the model using the previously mentioned metrics.
3. Repeat the test n_3 times and calculate the average of the examined metrics.

The n_1 parameter is increased from 10 000 to 100 000 with 10 000 increments in Subsections 6.1 and 6.2, while it is increased from 500 000 to 3 million with 500 000 increments in Subsection 6.3. The n_2 parameter is 10 000 in all cases, and n_3 is chosen to be 10. Every training and evaluation item is chosen randomly for each test scenario, but the training and evaluation item sets are always disjoint.

The test machine is a Macbook Pro with 3.1 GHz Intel Core i7 processor and 16 GB memory.

6.1 Comparing the Baseline Morpher Model with Other Morphology Engines

To compare the baseline Morpher model with existing methods, we executed the same evaluation tests on the baseline Morpher model, Morfessor, MORSEL, MorphoChain and Hunmorph-Ocamorph. Since the interface and functionality of these tools differ in some points, we could not perform all the tests on every existing methods:

- MorphoChain failed with an *OutOfMemoryError* using 50 000 training items, so we dropped this tool, since we could not compare its final metric values.
- Morfessor and MORSEL are segmentation tools, so they could not perform inflection.

- Since we used Hunmorph-Ocamorph for training data preprocessing, it did not make sense to include it in the comparisons other than the database size.

Figure 4 displays the average correctness ratio of the investigated methods. The baseline Morpher method's correctness ratio increases from about 73 % to about 97 %, while Morfessor only reaches 62 %, and MORSEL produces the worst results with 20 % at the 100 000 training item mark.

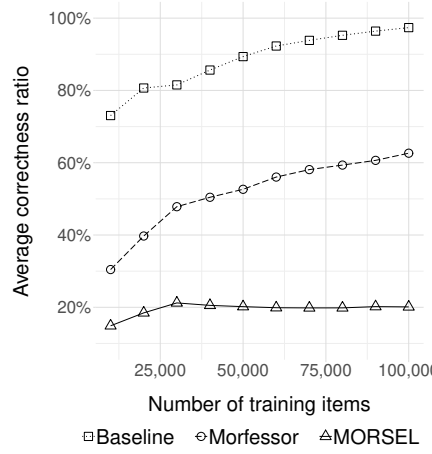
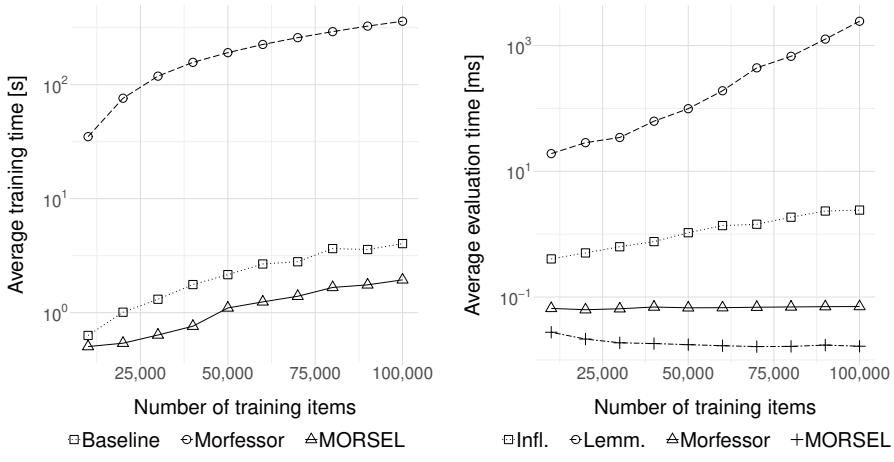


Figure 4. Average correctness ratio of the baseline Morpher model and other morphology engines

Figure 5 a) displays the average training time of the three methods in seconds, using exponential scale on the y axis. Morfessor has the worst training time, since it can learn the segmentation of 10 000 words in 35 seconds, while it takes 6 minutes to learn the segmentation of 100 000 words. MORSEL has a slightly faster training phase and similar characteristics to the baseline Morpher model. The baseline Morpher model trains in 4.03 seconds, while MORSEL finishes learning in 1.94 seconds at the 100 000 training item mark. However, since the baseline Morpher model has a much higher correctness ratio, this is not a big problem.

In Figure 5 b) we can see the average inflection and lemmatization times of the baseline Morpher model, as well as the average segmentation time of Morfessor and MORSEL in milliseconds, using exponential scale on the y axis. Morfessor and MORSEL (the bottom two lines) can perform the segmentation of a word in an average of 70 and 16 microseconds, respectively. This is almost constant time, which lets us draw the conclusion that they work with a map-like structure, identifying the pre-learnt segments in the input words without much searching. Since both inflection and lemmatization are more complex problems, they have a slightly steeper curve (the top two lines). While inflection increases from 0.4 milliseconds to 2.4 milliseconds, lemmatization takes an average of 19.1 milliseconds to 2.4 seconds. This confirms that lemmatization has an exponential time complexity.



a) Average training time

b) Average evaluation time

Figure 5. Average training and evaluation times of the baseline Morpher model and other morphology engines using exponential scale on the y axis

Table 1 contains the average knowledge base size of the baseline Morpher model, Morfessor and Hunmorph-Ocamorph using 100 000 training items. MORSEL does not have an option to export its knowledge base. Morfessor has the smallest database with 3.5 megabytes, but it needs to store much less information. The baseline Morpher model's knowledge base is 6.4 megabytes, but it contains the possible affix type chains, their conditional probabilities, the valid lemmas and more complex transformation rules as well. Hunmorph-Ocamorph has the biggest database with 22.7 megabytes.

Model	**File Size [MB]**
Baseline	6.4
Morfessor	3.5
Hunmorph-Ocamorph	22.7

Table 1. Average knowledge base size of the baseline Morpher model and other morphology engines using 100 000 training items

6.2 Evaluation of the Optimization Techniques

For this evaluation we analyzed the four optimization parameters using a smaller training data set of 100 000 random items to decide which one is worth using with larger data sets.

Figure 6 shows the average correctness ratio on the y axis, and the number of retained atomic rules on the x axis using exponential scale. From this graph we

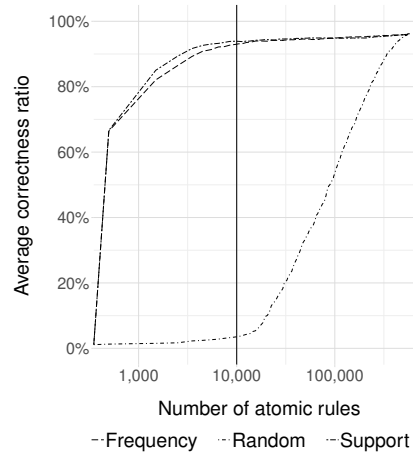
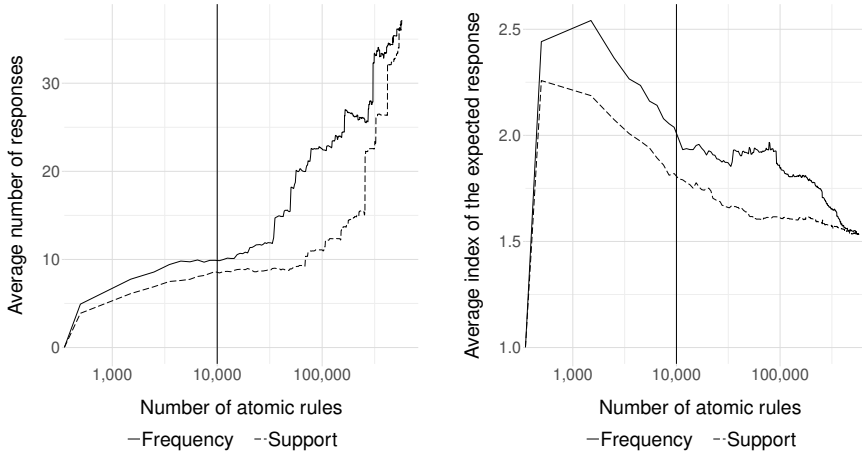


Figure 6. Average correctness ratio over the number of retained atomic rules after p_{supp} and p_{freq} optimization, using exponencial scale on the x axis

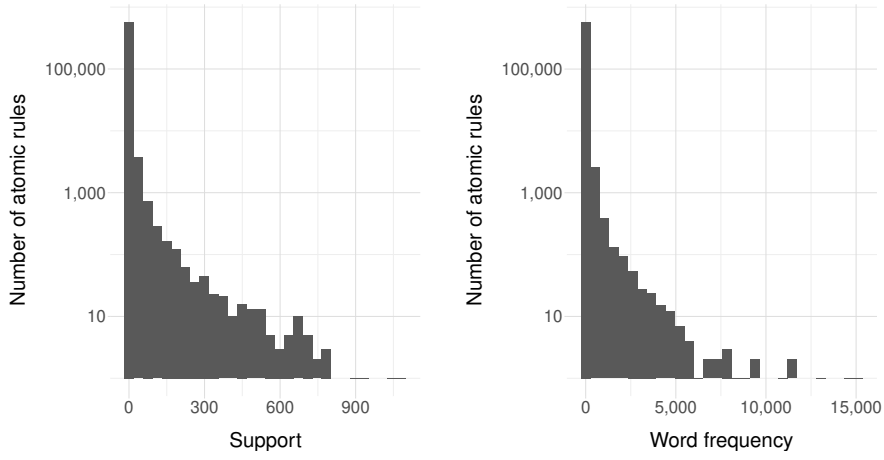
can see that if we drop atomic rewrite rules randomly, the correctness ratio drops dramatically.



a) Average number of responses b) Average index of the expected response

Figure 7. Average number of responses and average index of the expected response over the number of retained atomic rules after p_{supp} and p_{freq} optimization

On the other hand, if we use one of the two optimization parameters, we can reduce the rule base size to about 1.73 % of the original and still inflect and lemmatize about 93 % of the previously unseen words correctly. We added a vertical line to the



a) Support b) Word frequency

Figure 8. Histogram of atomic rules based on their support and word frequency

10 000 atomic rule mark, while the total number of atomic rules was 578 497. We can also see that using p_{supp} we can keep a slightly higher correctness ratio with the same amount of atomic rules, but the two parameters result in very similar results otherwise.

In Figure 7 a) we can see the average number of responses. Similarly to the correctness ratio, p_{supp} performs better, producing slightly fewer responses using the same amount of retained atomic rules.

Figure 7 b) displays the average index of the expected response, i.e., which response is the expected (correct) one. Although it is not guaranteed that the expected response is the correct one due to the large volumes of evaluation data, this metric is a good approximation. The p_{supp} parameter performs better, and the worst value with a small number of retained atomic rules does not increase above 3, meaning that the first 3 responses always contain the expected one.

From the above figures, we can choose $p_{supp} = 10$ as the optimization parameter. The histogram of the atomic rules in Figure 8 shows that choosing a relatively low threshold will drop a lot of rules from the rule base.

Table 2 displays the average number of retained atomic rules, correctness ratio, number of responses and expected response index using different (p_{gen}, p_{max}) combinations. The most responses are produced when we only keep the most general rules, and the correctness ratio is one of the lowest values as well. With $p_{gen} = 3$, the correctness ratio dropped to about 80 %.

p_{gen}	p_{max}	Rules	Correctness [%]	Responses	Response Index
–	–	578 497	96.19	37.11	1.53
1	1	5 019	85.78	126.01	31.94
1	2	16 923	92.62	114.34	7.34
1	3	46 506	94.89	89.45	2.44
1	4	103 533	96.17	56.23	1.60
1	5	175 334	96.18	41.78	1.54
2	1	10 501	91.91	10.92	5.42
2	2	36 186	92.62	9.71	2.05
2	3	90 710	94.02	6.47	1.47
2	4	161 132	94.28	4.60	1.39
2	5	238 879	94.29	4.04	1.39

Table 2. Average number of retained atomic rules, correctness ratio, number of responses and expected response index using different (p_{gen}, p_{max}) combinations

6.3 Evaluating the Optimal Optimization Parameter Using Large Training Data Sets

We wanted to evaluate $(p_{gen} = 1, p_{max} = 1)$, as well as $p_{supp} = 10$ using big training data sets containing up to 3 million training items, but we had to omit the first case, as it could not even handle 500 000 training items due to the exponential growth of responses. Figure 9 shows the average training time of the optimized Morpher model using $p_{supp} = 10$, increasing about linearly up to about 74.61 seconds.

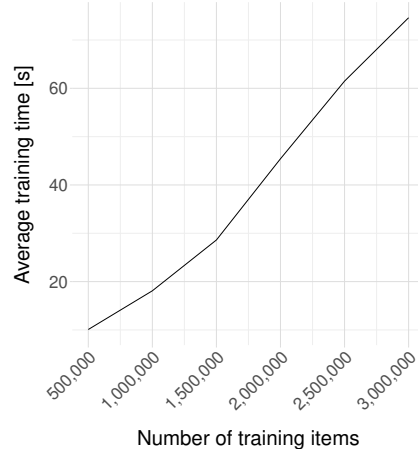
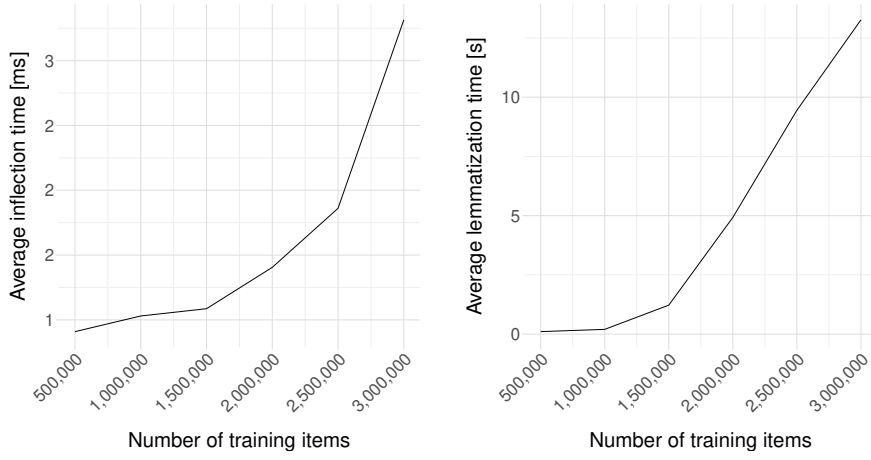


Figure 9. Average training time using big training data sets and $p_{supp} = 10$

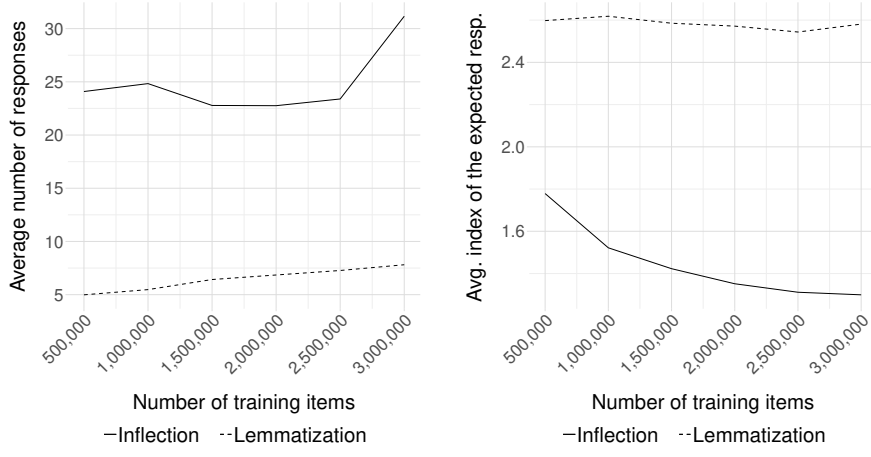
In Figures 10 a) and 10 b) we can see the average inflection and lemmatization times: 3.31 seconds for inflection and 13.26 seconds for lemmatization using 3 million training items.



a) Average inflection time b) Average lemmatization time

Figure 10. Average inflection and lemmatization times using big training data sets and $p_{supp} = 10$

Figure 11 a) displays the average number of responses. It is surprising that inflection produces more responses in average (31.16 vs 7.81).



a) Average number of responses b) Average index of the expected response

Figure 11. Average number of responses and average index of the expected response using big training data sets and $p_{supp} = 10$

The reason is that many lemmatization responses are filtered out due to not ending in a valid lemma. However, Figure 11 b) proves that the index of the expected response does not go above 1.3 and 2.58, respectively.

In Figure 12 we can see the average correctness ratio, that increases from about 96.22 % to about 98.11 % in average.

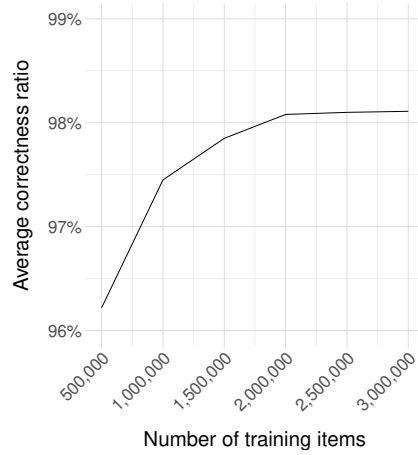


Figure 12. Average correctness ratio using big training data sets and $p_{supp} = 10$

Table 3 contains the summary of the examined metrics, including their baseline values and the measured values for the optimized case ($p_{supp} = 10$), as well as their ratio, using 3 million training items. Just for comparison we executed this test using the baseline Morpher model, but with less evaluation data to save time. The table shows that there are huge improvements, except for the training time: lemmatization of one word would take about 8.5 minutes in average without any optimizations, compared to 13.26 seconds in case of $p_{supp} = 10$, which means that using this optimization parameter value, the lemmatization becomes 2.59 % of the original value.

	Baseline	$p_{supp} = 10$	**Ratio**
Training time [s]	53.41	74.61	139.69 %
Inflection time [s]	640	3.31	0.52 %
Lemmatization time [s]	511.83	13.26	2.59 %
Number of atomic rules	11 354 255	255 867	2.25 %
Knowledge base size [MB]	130.6	5.5	4.21 %

Table 3. Summary of the average values and improvements of the examined metrics using a big training data set containing 3 million items and $p_{supp} = 10$

7 CONCLUSION

In this paper we performed the space and time complexity analysis of the Morpher morphological rule induction model, and introduced several optimization techniques.

The four new optimization parameters aim to reduce the number of retained transformation rules during the training phase. The first optimization parameter (p_{max}) limits the number of generated rules per word pair, while another one (p_{gen}) sets a lower boundary on the context length of the retained rules. We can also reduce the rule base size using statistics calculated from the training data: there is a p_{supp} and a p_{freq} optimization parameter with which we can drop rules that have a support value or a word frequency value less than these threshold parameters.

The complexity analysis showed that these optimization parameters improve the memory requirements and average runtime of the original Morpher model dramatically. The winning configuration was $p_{supp} = 10$ that managed to reduce the number of rules to the 1.73 % of the original set, still keeping an average correctness ratio of about 93 % and finished in acceptable time using up to 3 million training items.

REFERENCES

[1] Bauer, L.: Introducing Linguistic Morphology. Edinburgh University Press, Edinburgh, 2003.

[2] Bergmanis, T.—Goldwater, S.: From Segmentation to Analyses: A Probabilistic Model for Unsupervised Morphology Induction. Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2017), Vol. 1, 2017, pp. 337–346, doi: 10.18653/v1/e17-1032.

[3] Creutz, M.—Lagus, K.: Inducing the Morphological Lexicon of a Natural Language from Unannotated Text. Proceedings of the International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR '05), 2005, pp. 106–113.

[4] de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, 2010, doi: 10.1017/cbo9781139194655.

[5] Hulden, M.: Foma: A Finite-State Compiler and Library. Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics: Demonstrations Session (EACL '09), 2009, pp. 29–32, doi: 10.3115/1609049.1609057.

[6] Koskenniemi, K.: Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production. Department of General Linguistics, University of Helsinki, Finland, 1983.

[7] Kovács, L.—Szabó, G.: String Transformation Approach for Morpheme Rule Induction. Procedia Technology, Vol. 22, 2016, pp. 854–861, doi: 10.1016/j.protcy.2016.01.060.

[8] Lignos, C.: Learning from Unseen Data. Proceedings of the Morpho Challenge 2010 Workshop, 2010, pp. 35–38.

[9] Jurafsky, D.—Martin, J. H.: Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. 2nd Edition. Prentice Hall, 2008.

[10] Mohri, M.: Finite-State Transducers in Language and Speech Processing. Computational Linguistics, Vol. 23, 1997, No. 2, pp. 269–311.

[11] Narasimhan, K.—Barzilay, R.—Jaakkola, T.: An Unsupervised Method for Uncovering Morphological Chains. Transactions of the Association for Computational Linguistics, Vol. 3, 2015, pp. 157–167, doi: 10.1162/tacl_a_00130.

[12] Oncina, J.—García, P.—Vidal, E.: Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 15, 1993, No. 5, pp. 448–458, doi: 10.1109/34.211465.

[13] Pirinen, T.—Lindén, K.: Creating and Weighting Hunspell Dictionaries as Finite-State Automata. Investigationes Linguisticae, Vol. 21, 2010, pp. 1–16, doi: 10.14746/il.2010.21.1.

[14] Prószéky, G.—Kis, B.: A Unification-Based Approach to Morpho-Syntactic Parsing of Agglutinative and Other (Highly) Inflectional Languages. Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics on Computational Linguistics (ACL '99), 1999, pp. 261–268, doi: 10.3115/1034678.1034723.

[15] Shalonova, K.—Flach, P. A.: Morphology Learning Using Tree of Aligned Suffix Rules. ICML Workshop: Challenges and Applications of Grammar Induction, 2007.

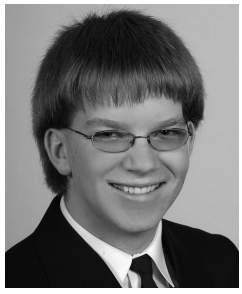
[16] Szabó, G.—Kovács, L.: Benchmarking Morphological Analyzers for the Hungarian Language. Annales Mathematicae et Informaticae, Vol. 49, 2018, pp. 141–166, doi: 10.33039/ami.2018.05.001.

[17] Szabó, G.—Kovács, L.: An Advanced Semi-Supervised Morphological Rule Induction Model for the Hungarian Language. Submitted to a Journal. Available at: `https://users.iit.uni-miskolc.hu/~szabo84/articles/optimization`, 2018.

[18] Trón, V.—Kornai, A.—Gyepesi, Gy.—Németh, L.—Halácsy, P.—Varga, D.: Hunmorph: Open Source Word Analysis. Proceedings of the Workshop on Software (Software '05), 2005, pp. 77–85, doi: 10.3115/1626315.1626321.

[19] Trón, V.—Halácsy, P.—Rebrus, P.—Rung, A.—Vajda, P.—Simon, E.: Morphdb.hu: Hungarian Lexical Database and Morphological Grammar. Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC '06), 2006.

[20] Virpioja, S.—Smit, P.—Grönroos, S.—Kurimo, M.: Morfessor 2.0: Python Implementation and Extensions for Morfessor Baseline. Aalto University, 2013.



Gábor Szabó is currently Ph.D. student at the Institute of Information Technology at the University of Miskolc, Hungary. He has his B.Sc. in software information technology since 2012 and his M.Sc. in engineering information technology since 2014. His main research area is solving the morphological rule induction problem for morphologically complex agglutinative languages.

László Kovács is the Head of the Institute of Information Technology at the University of Miskolc, Hungary, where he is Associate Professor. He received his Ph.D. in computer science in 1998. His broad research areas include morphology, ontologies and database systems, among others.

Computing and Informatics, Vol. 38, 2019, 986–1008, doi: 10.31577/cai_2019_4_986

A NEW OPEN INFORMATION EXTRACTION SYSTEM USING SENTENCE DIFFICULTY ESTIMATION

Vahideh Reshadat

Miyaneh Technical and Engineering Faculty
University of Tabriz, Tabriz, Iran
e-mail: v.reshadat@gmail.com

Heshaam Faili

School of Electrical and Computer Engineering
College of Engineering
University of Tehran, Tehran, Iran
e-mail: hfaili@ut.ac.ir

Abstract. The World Wide Web has a considerable amount of information expressed using natural language. While unstructured text is often difficult for machines to understand, Open Information Extraction (OIE) is a relation-independent extraction paradigm designed to extract assertions directly from massive and heterogeneous corpora. Allocation of low-cost computational resources is a main demand for Open Relation Extraction (ORE) systems. A large number of ORE methods have been proposed recently, covering a wide range of NLP tools, from "shallow" (e.g., part-of-speech tagging) to "deep" (e.g., semantic role labeling). There is a trade-off between NLP tools depth versus efficiency (computational cost) of ORE systems. This paper describes a novel approach called Sentence Difficulty Estimator for Open Information Extraction (SDE-OIE) for automatic estimation of relation extraction difficulty by developing some difficulty classifiers. These classifiers dedicate the input sentence to an appropriate OIE extractor in order to decrease the overall computational cost. Our evaluations show that an intelligent selection of a proper depth of ORE systems has a significant improvement on the effectiveness and scalability of SDE-OIE. It avoids wasting resources and achieves almost the same performance as its constituent deep extractor in a more reasonable time.

Keywords: Information extraction, open information extraction, relation extraction, knowledge discovery, fact extraction

1 INTRODUCTION

Information Extraction is the task of automatically extracting structured data from unstructured text. One of the core information extraction tasks is the relation extraction, which aims at extracting semantic relations among entities from natural language text. Relation extraction can potentially benefit a wide range of NLP tasks such as: Web search, question answering, ontology learning, summarization, building knowledge bases, etc. [1].

The huge and fast-growing scale, a mixed genre of documents and infinite types of relations are challenges of the Web-scale relation extraction [2]. The traditional approaches to information extraction assume a fixed set of predefined target relations and usually do not scale to corpora where the number of target relations is very large [3]. An alternative paradigm OIE aims to scale information extraction methods to the size and diversity of the Web corpus. OIE systems extract relational tuples from texts, without requiring a pre-specified vocabulary [4].

The key goals of OIE are: 1. domain independence, 2. unsupervised extraction, and 3. scalability to large amounts of text [5]. Scalability of OIE systems relies on the different sophistication levels of the NLP tools they use. Shallow extractors try to improve performance by limiting extraction procedure to shallow linguistic analysis. Although the ORE approaches in this category (such as TextRunner [6], WOEpos [7], ReVerb [8], R2A2 [9] and SONEX [10]) are fast and more scalable, they are unable to deal with complicated structures such as long distance relations. In addition, due to usage only shallow syntactic features, high performance is not guaranteed, thus resulting in a significant drop of effectiveness.

In contrast to shallow extractors, some approaches (such as Wanderlust [11], WOEparse [7], KrakeN [12], OLLIE [4], ZORE [13], DepOE [14], SRL-IE-Lund [15], SRL-IE-UIUC [15], the methods proposed in [16] and [17]) use deep syntactic or semantic analysis tools such as dependency parsing. These extractors are generally more expensive than the previous extractors; they trade efficiency for improved precision and recall [5]. The former extractors are rapid, guarantee scalability and require less effort due to usage shallow syntactic analysis, while the latter extractors are efficient for precision and recall but time consuming and require considerable effort due to usage deep syntactic analysis in the extraction process [18].

Given the pros and cons of shallow and deep extractors, we proposed an approach for automatic estimation of ORE difficulty. We developed different classifiers that recognize sentences that are hard for ORE task and pass them to a deep extractor. Thus, it attempts to categorize them with the aim of reducing computational cost. The proposed approach is a combination of two types of OIE systems and

we employed ReVerb [8] and EXEMPLAR [19] as shallow and deep OIE extractors, respectively.

According to the results in [19], shallow methods handle ten times more sentences than deep ones. We examined the trade-off between effectiveness (F-measure) and efficiency (computational cost) and found that using a deep extractor on the intelligent subset of input sentences can yield a substantial improvement in F-measure. We present a novel approach for predicting ORE difficulty using different classifiers with light-weight features. The classifiers recognize sentences that are hard for ORE task and pass input sentences to a deep extractor only if needed. Therefore, our difficulty classifiers prioritize the sentences likelihood of improving performance and lead to better allocation of computational resources. Sentence difficulty is used in many applications of natural language processing such as measuring translation difficulty [21], evaluating the reliability of parses [22], measuring text difficulty [23] and text readability [24], etc. The idea of this work can be ported into other tasks in natural language processing. Application systems such as Speech Processing, Question Answering and Search Engines can benefit from automatic detection of difficult subtasks.

The rest of this paper is organized as follows. Section 2 introduces previous works in the areas of OIE systems. Our proposed approach is described in Section 3. We present results of our experiments in Section 4 and end with the conclusion in Section 5.

2 RELATED WORKS

In this section we review some related works on OIE, in particular works on ORE. OIE has received much attention recently. It covers a wide range of NLP tools, from shallow (e.g., part-of-speech tagging (POS)) to deep (e.g., semantic role labeling (SRL)). These systems can be divided into two main categories based on the linguistic analysis which is applied for relation extraction task [18, 5, 14]. In the following two subsections, we examine these two categories.

2.1 Deep Open Information Extraction Systems

ORE approaches which use parsing-based or SRL-based tools are grouped in deep OIE systems. Most deep OIE systems apply dependency tree paths to learn extraction patterns. Wanderlust [11] uses hand-labeled training data to learn extraction patterns on the dependency tree. The authors of this system annotated 10 000 sentences parsed with LinkGrammar. This system learns 46 general link paths as patterns for relation extraction. WOEparse [7] is a pattern classifier learned from dependency path patterns which uses typed dependencies as features [18]. PATTY [25] extracts textual patterns from sentences based on paths in the dependency tree between the two named entities. It finds the shortest path in the dependency tree that connects the two named entities. The TreeKernel approach [26] first inspects

whether there is a relation between a pair of entities in a sentence and then whether there are explicit relation words for this pair. A set of syntactic patterns is used for generating candidate relations. One of the main drawbacks of dependency-based deep OIE systems is restricting extraction to the paths of dependency tree.

Some approaches use bootstrapping to learn patterns. OLLIE [4] is a hybrid approach based on bootstrapping which learns pattern templates automatically from a training set that is bootstrapped from relations extracted by ReVerb. OLLIE produces n-ary extractions by merging binary relations and has 1.9 to 2.7 times more area under precision-yield curves [1] compared to ReVerb and WOE. BONIE [27] is an open numerical relation extractor, for extracting OIE tuples where one of the arguments is a number or a quantity-unit phrase. BONIE also uses bootstrapping to learn the specific dependency patterns that express numerical relations in a sentence. Bootstrapping methods have some limitations because extraction samples can vary considerably depending on initial seed selection.

Some OIE methods are designed for languages other than English. Similarly, most of them are based on rules or patterns. ZORE [13] is a syntax-based Chinese ORE system that extracts relations and semantic patterns from Chinese texts. The approach proposed in [17] also focuses on Chinese ORE. This system can be considered as a pipeline of word segmentation, POS tagging and parsing [18]. An OIE system for German language was proposed in [28]. It is a straightforward approach for adapting PropS, a rule-based predicate-argument analysis for English, to a new language, German. DepOE [14] is a multilingual OIE system based on fast dependency parsing. It uses DepPattern [29], a multilingual dependency-based parser, to analyze sentences and obtain fine-grained information. Then, a small set of extraction rules is applied and the target verb-based triples are generated. There is a more recent version of DepOE system, called ArgOE [30]. ArgOE is a multilingual rule-based OIE method that obtains as input dependency parses in the CoNLL-X format, recognizes argument structures within the dependency parses, and extracts a set of basic propositions from each argument structure. Since most of the OIE systems designed in languages other than English are based on rules or patterns, they have the same problems as rule-based and pattern-based methods.

Most OIE approaches usually extract binary facts and are not designed to capture n-ary relations. KrakeN [12] addresses this limitation by capturing unary, binary and higher order n-ary facts. It has been built specifically for capturing complete facts from sentences and can extract more facts per sentence with high precision. EXEMPLAR [19] addresses the problem of extracting n-ary relations by using handcrafted rules over dependency trees. These rules are applied to each candidate argument individually by inspecting the path between an entity and a relational word. OIE approaches which deal with n-ary relations can increase the number of correct and informative extractions and achieve high precision and recall.

[1] Receiver Operating Characteristic (ROC)

Some deep OIE methods separate the detection of "useful" pieces of information expressed in a sentence from their representation in terms of extractions. ClauseIE [5] is a clause-based approach which uses linguistic knowledge about the grammar of the English language to first detect clauses in an input sentences and to subsequently identify the type of each clause according to the grammatical function of its constituents [18]. CSD-IE [31] is a method that uses contextual sentence decomposition for OIE. A sentence is decomposed into the parts that are semantically dependent and then the (implicit or explicit) verb in each part is identified and the facts are obtained [8]. The performance of decomposition-based OIE systems is highly dependent on the detection of effective pieces which produce the facts.

Although the majority of deep OIE systems are parser-based, there is a limited quantity of approaches that exploit semantic role labelers. A deep OIE system based on SRL has been proposed in [15]. This system has been developed based on two SRL systems: UIUC [32] and LUND [33]. It produces the extractions by applying some rules on the outputs of these SRL systems. The authors proposed two hybrid methods that employ SRL only on a specific subset of TextRunner outputs. This work is similar to our approach in terms of combining two OIE systems, but there are some differences. Applying TextRunner to all input sentences and using SRL via some restrictions rules on TextRunner outputs are the main differences. Another version of SRL-IE was implemented in [20] by relying on the output of two SRL systems: LUND [34] and SwiRL [35]. Efficiens [20] has a module for each NLP tool. The Efficiens[POS] module relies on POS tagging, while the Efficiens[DEP] and Efficiens[SRL] rely on dependency parsing and SRL, respectively. Since SRL-based deep methods need more computational time than parse-based deep methods, they are computationally expensive, even though they are robust to noisy text. The related surveys are summarized in Table 2.

Although deep OIE systems have a high performance, the cost of leveraging deep NLP tools and scarcity of them in other languages are the main challenges of deep OIE methods. In this paper, we present an approach to alleviate these critical challenges. We developed a strategy which mitigates these challenges by intelligent use of different methods.

2.2 Shallow Open Information Extraction Systems

ORE methods, which are based on shallow NLP tools (such as POS taggers), are grouped in shallow OIE systems. Some shallow OIE systems use classifiers with some lightweight features to recognize the relation between name entities in a sentence. TextRunner [36] is the first OIE system. It applies a Naive Bayes classifier which determines whether the context between a pair of noun phrases in a sentence describes a relation instance or not. WOEpos [7] is also inspired by TextRunner and limited to shallow features like POS tags. WOEpos exploits the relations in Wikipedia Infoboxes to match corresponding sentences in an unlabelled corpus that mention these relations. It uses these examples as relation-independent training data to learn an unlexicalized extractor. R2A2 [8] uses an argument learning com-

	N	D	S	E	P	R
Wanderlust [11]	×	✓	×	✓	✓	×
$WOE_{parse,\ 7}$	×	✓	×	✓	✓	×
PATTY [25]	×	✓	×	✓	✓	✓
TreeKernel [26]	×	✓	×	✓	✓	×
OLLIE [4]	✓	✓	×	✓	✓	×
BONIE [27]	×	✓	×	✓	✓	×
ZORE [13]	×	✓	×	×	✓	×
Chinese OIE [17]	×	✓	×	×	×	✓
German OIE [28]	✓	✓	×	×	×	✓
DepOE [14]	×	✓	×	✓	×	✓
ArgOE [30]	×	✓	×	✓	×	✓
KrakeN [12]	✓	✓	×	✓	×	✓
EXEMPLAR [19]	✓	✓	×	✓	×	✓
ClauseIE [5]	✓	✓	×	✓	×	✓
CSD-IE [31]	×	✓	×	✓	×	✓
SRL-IE [15]	✓	×	✓	✓	×	✓
Efficiens [20]	✓	✓	✓	✓	×	×

Table 1. Comparison of different deep OIE methods. N: extracts N-ary relations? D: extracts relations based on dependency parse tree? S: extracts relations based on SRL? E: extracts relations in English language? P: extracts relations based on patterns? R: extracts relations based on rules?

ponent. It makes use of a number of classifiers to identify the arguments of a verb phrase (based on hand-labeled training data). Two classifiers identify the left and right bounds for the first argument and one classifier identifies the right bound of the second argument.

Some shallow OIE systems are based on patterns. ReVerb [8] is a strong and successful pattern-based shallow OIE system. It makes use of a simple POS tag sequence as a syntactic constraint in order to extract relation phrases and eliminate incoherent extractions and also reduce uninformative extractions. ReVerb exploits a lexical constraint that aims to alleviate the amount of over-specified extractions. Experiments show ReVerb outperforms TextRunner and its performance is more than twice as much as that of TextRunner [7, 18, 37]. SONEX [10] extends ReVerb by detecting patterns with appositions and possessives [19]. It identifies every entity pair (e.g., "Google", "Apple Inc.") and all sentences where this pair is mentioned together. From these sentences, SONEX extracts a context (e.g., a list of surrounding words) for the pair and applies clustering techniques to group together pairs with similar contexts. SONEX sees each cluster of entity pairs as a relation. LSOE [38] is also a pattern-based system which exploits two kinds of patterns: 1. generic patterns, 2. rules from Cimiano and Wenderoth proposal [39]. The performance of LSOE was compared with two other OIE systems: ReVerb and DepOE. The results show that LSOE extracts relations that are not learned by other extractors and also achieves compatible precision.

There are some shallow OIE methods such as R-OpenIE [40] which are based on rules. R-OpenIE defines some text-based rules to generate relation extraction templates. It applies the cascaded finite-state transducer model to match the satisfied relational tuples.

The main drawback of all the above shallow OIE approaches is that they are inefficient for high performance. With the fast growth of the Internet and the emerging problem of information overload, the computational cost of processing a large volume of information is becoming an increasingly important issue of artificial intelligence researches. Based on the methods discussed above, despite the high performance of pure deep OIE systems, applying them is time consuming. Unlike deep OIE systems, shallow ones are fast and do not achieve high performance measures. Therefore, each one of these categories has their own pros and cons and raises the question of what is the trade-off between NLP depth (and associated computational cost) versus effectiveness. In this paper we develop some probabilistic classifiers that apply different combination parameters as features, for different classes of extractors. This approach is not limited to specific types of system. It divides the input sentences to proper extractors.

3 SENTENCE DIFFICULTY ESTIMATION FOR OPEN INFORMATION EXTRACTION SYSTEMS

Various levels of linguistic analysis tools from shallow (e.g. POS tagging) to deep (e.g. SRL) were used to develop OIE systems. Applying expensive NLP tools for extracting facts from huge and heterogeneous corpora in reasonable time is time-consuming and costly. This problem worsens when such methods are applied on World Wide Web documents. In addition, tools for automatic deep analysis are available only for a limited number of natural languages, and produce imperfect results. Manual deep analysis, on the other hand, is time consuming and expensive [41]. Automatic tools for approaches that rely only on a shallow linguistic analysis are available for many languages and sufficiently reliable [41]. These extractors are usually fast, but the restriction to shallow syntactic analysis reduces maximum recall and/or may lead to a significant drop of precision at higher points of recall [5]. Indeed, there is a need to have a system that enables effective use of available time and offers a reasonable balance of precision and recall. The advantages of these two kinds of extractors motivate us to focus on developing a method that gets the best of both worlds. A hybrid OIE paradigm by incorporating strengths of ReVerb and EXEMPLAR is suggested. Figure 1 presents the general framework of our proposed approach.

Preprocessor. The preprocessor converts raw web pages into a sequence of sentences. It takes web pages as input and transforms them to plain sentences using pre-processing tools. The pages were then segmented into sentences, tokenized, tagged with POS and chunked using the OpenNLP[2] package.

[2] Downloadable at `http://opennlp.sourceforge.net`.

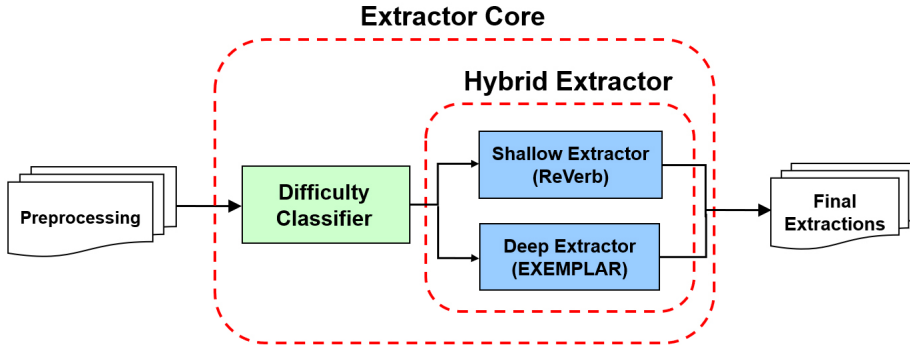


Figure 1. SDE-OIE's framework: Difficulty classifier exploits the best of both the shallow and the deep OIE extractors

Extractor Core. This component takes each sentence and assigns it to a proper extractor. Extractor Core consists of two main subcomponents: Sentence Difficulty Estimator and Hybrid Extractor.

Difficulty Classifier. The difficulty classifier and the hybrid extractor are the main parts of SDE-OIE. SDE-OIE reads each sentence sequentially. Given a sentence, the difficulty classifier extracts a set of features and predicts whether it is difficult or not. In other words, for each sentence, the difficulty classifier finds the most appropriate system for processing it. In regards to binary classification of sentences, we use different classifiers; a Logistic Regression, a Naive Bayes and a Decision Tree. Due to strong classification results of these classifiers, they have been used for many classification problems in computational linguistics. In addition to classification, we need to find difficult sentences, where we care about the severity of the extraction difficulty. For this purpose, we benefit from the classification score as a difficulty measure.

SDE-OIE focuses on the difficulty estimation of a relation extraction task for input sentences of OIE systems. We formulate this problem as a classification problem, where the goal is to assign a class label of *easy* or *difficult* to a candidate sentence s based on a classifier c and then pass it to an appropriate extractor. $c{:}s \rightarrow \{easy, difficult\}$

For this purpose, different probabilistic classifiers were used. We used Naive Bayes, Logistic Regression and Decision Tree to automatically assign a difficult/easy class to each input instance. Naive Bayes is a simple and common generative classifier that chooses the most probable extractor class out of a set of possible classes given a feature vector [42]. The features of data samples are independent. Naive Bayes employs the normal distribution to model numeric attributes.

Logistic regression belongs to the family of classifiers known as the exponential or log-linear classifiers. Like Naive Bayes, it works by extracting some set of weighted features from the input, taking logs, and combining them linearly [43]. In order to train a model to classify with the least amount of error possible, the cost function should be minimized. Gradient descent is our learning algorithm that finds values for the parameters that result in the best parameter values and a smaller minimum error.

Binary Decision Tree consists of terminal vertices and nonterminal vertices. Compared to Naive Bayes, decision tree is a somewhat more transparent approach that lends itself to inspection [42]. Our decision tree was built by C4.5. For implementation of these classifiers we used the Weka package. A variety of features have been used to train the classifiers. These features are discussed with more detail in Section 3.1 and the appendix.

Hybrid Extractor. The hybrid extractor also includes two main subcomponents, a shallow (ReVerb) and a deep (EXEMPLAR) OIE system. A complete and fair experimental comparison of 10 approaches have been presented in [19] and [20]. According to that research, ReVerb is the fastest method based on matching patterns over POS tags. SONEX is a shallow ORE system which produces results comparable to ReVerb. It focuses on overcoming the challenges in deploying ORE systems in the blogosphere and uses a clustering algorithm to group pairs with similar context together in a large scale. Beside the challenges with large-scale clustering (time and space), it recognizes instances at corpus-level. Since our system is sentence-based (meaning the extraction process can take individual sentences as input), we employ ReVerb as the shallow constituent extractor. Through these adaptations, we gain a range of extraction quality improvements at the sentence level. EXEMPLAR is based on using rules over dependency trees. It outperforms ReVerb and differs greatly in efficiency. It achieves the best effectiveness and is faster than the deeper methods such as SRL-based OIE extractors. Thus, EXEMPLAR's processing time is much less. More details about these OIE systems were presented in Section 2. As illustrated in Figure 1, final extractions are obtained by taking the union of these two extractors' outputs.

The proposed approach has some advantages in the following aspects:

- While in the structure of previous similar approaches, a pure shallow or deep linguistic analysis tool is applied to all input sentences at least once; to our knowledge, we are the first to propose an approach to partition the input to an appropriate extractor in order to achieve higher performance.
- The constituent systems of the extractor core are based on shallow and deep linguistic analysis tools and neither ReVerb nor EXEMPLAR needs training data. Therefore, SDE-OIE's performance will be independent from training parameters.

- The proposed approach is independent of its constituent systems and can be designed by other shallow or deep systems. In other words, it is a general framework and it is not designed for certain OIE systems. Hence, it can be designed by incorporating different systems with different depths.
- As will be discussed in Section 4, our experiments indicate that extraction difficulty can be modeled and automatically predicated with decent accuracy. Detecting difficult sentences has significant influence on the extraction time and quality. It prevents wasting resources and helps to achieve approximately the same performance as the deep constituent extractor. SDE-OIE is particularly effective when there is a large dataset and the processing time is limited. In this case, our hybrid extractor makes effective use of available time and runs the best algorithm given the available computation time.

In case all input sentences are difficult, using a difficulty classifier would be an overhead operation rather than applying a deep extractor individually. Additionally, sometimes only a few sentences in the whole dataset produce better instances with deep NLP tools. In this case, a classifier that applies deep extractor for most sentences will be wasting computational resources for the rest of the sentences in that dataset. SDE-OIE will prefer shallow extractor when both extractors produce correct extraction and therefore efficiency improves.

3.1 Feature Set

Deep features could improve precision and recall over shallow syntactic features, but at the cost of extraction speed. For instance, parser-based features can help to handle complicated and long distance relations in sentences. Such cases usually cannot be detected by shallow features. Regarding the computational cost associated with rich syntactic features, we used 61 light-weight features. All features are independent of applied classifiers, scalable, domain independent, and can be evaluated at extraction time without the use of expensive tools.

These features allow the difficulty classifier to estimate the challenge that the system faces in extracting instances from a sentence. Although these features can be extracted from the underlying systems, they are collected from the syntactic and semantic structure of the sentence. Since our difficulty modeling is system-independent, we particularly do not incorporate knowledge (features) from the underlying OIE systems into the difficulty classifier. Additionally, we use source-language features which bring deeper linguistic knowledge into our modeling and classification.We list below some important features which the difficulty classifier uses to recognize the class of an input sentence s:

- F1: s contains at least two name entities where the context between them has a verb phrase.
- F2: Number of capital words in s is greater than 6.

- F3: s contains communication verbs [1].
- F4: Number of stop words in s is equal or greater than 10.
- F5: s contains 'if'.
- F6: s contains at least one coordinating conjunction (and, but, for, nor, or, so, yet).
- F7: s contains 'say'.
- F8: s contains at least one pronoun (PRP, PRP\$, WP, WP\$).
- F9: s contains 'that' or 'whether'.
- F10: s contains at least one relative pronoun.
- F11: s contains 'there'.
- F12: s contains feature1 (F1) and the first name entity is a pronoun.
- F13: s contains F1 and the second name entity is a pronoun.
- F14: s contains F1 and there is a preposition ('to' or 'in') in s.
- F15: s contains F1 and there is a verb before the first name entity.
- F16: s contains F1 and there is a verb after the second name entity.
- F17: s contains F1 and the first name entity is a proper noun.
- F18: s contains at least one entity pair where there is a verb after the second name entity.
- F19: Length(s) is greater than 10.
- F20: s contains cognition verbs [2].

Examples of other features include presence of punctuation, capitalization, WH-words, comma, quotation, parentheses, specific POS tag sequences, a verb with a specific tag (such as vbz, vbg, vbd, vbn, vbp, vb) in the sentence and a specific preposition at the end of the sentence (such as to, in, for, of, on). Following feature extraction, this set of automatically labeled feature vectors is used for training the classifier; then each sentence is passed to an extractor based on the classifier output.

4 EXPERIMENTAL RESULTS

In this section, we first describe the benchmark dataset and performance metrics, and then give the evaluation results obtained by our approach, baseline methods and state-of-the-art approaches.

4.1 Dataset

A gold standard data and a set of features are required to train the difficulty classifier. The lack of standard dataset is one of the main challenges of the OIE systems [20]. The current evaluation approaches rely on manual evaluation (e.g., [7, 8,

4, 12, 14, 5, 31, 44, 26]), whose main limitation is that it is not scalable. Combining available datasets to make a large one has some difficulties. Differences in annotation and evaluation methodology are some of these challenges. Manual creation of a large dataset, on the other hand, is time consuming and expensive.

Based on available resources, we used two state-of-the-art datasets [19] to validate and compare our approach with other methods developed for extracting open relations from the Web. These datasets contain relatively more data than the others (e.g., [7, 8, 4, 12, 14, 5, 30, 26, 44]). They are standard datasets which have been used in several recent studies such as [45, 46, 47].

This datasets try to alleviate the problems related to the lack of ground truth and differences in evaluation methodologies by providing reusable annotations that are flexible and can be used to evaluate a wide range of methods [19, 20]. They cover sentences from the New York Times (NYT-500), the Penn Treebank (PENN-100), a popular Web corpus (WEB-500) and a much larger dataset from the New York Times which has been annotated automatically. WEB-500 is a commonly used dataset, developed for the TextRunner experiments [3]. This dataset contains 500 sentences extracted from search engine snippets. These sentences are often incomplete and grammatically unsound, representing the challenges of dealing with web text. NYT-500 represents the other end of the spectrum with individual sentences from formal, well written new stories from the New York Times corpus [48]. PENN-100 contains sentences from the Penn Treebank recently used in an evaluation of the TreeKernel method [26]. The NYT-500 and the WEB-500 are used as training data and the PENN-100 is used as test data. We also randomly selected 300 sentences from the data source which was built automatically from Freebase and WordNet [19] as our test set.

The gold standard data is a set of sentences which have easy or difficult labels. Given a corpus, SDE-OIE should select sentences for the shallow/deep extractor. We manually annotated these datasets. We label a sentence as easy if the shallow extractor generates a correct result. In cases where the shallow extractor generates an incorrect result, it is labeled as difficult, except for cases where the deep extractor also generates an incorrect result. A sentence is also labeled as easy if the shallow extractor has no output for that sentence, but the deep extractor generates an incorrect result. In this case, if the deep extractor generates a correct result, the sentence will be labeled as difficult.

4.2 Performance Measures

Our evaluation focuses on the extraction of relation instances at sentence level. The metrics used in the evaluations are: Precision (P), Recall (R) and F-measure (F). Precision is the ratio of the number of correctly extracted instances to the total number of extracted instances. Recall is the ratio of the number of correctly extracted instances to the total number of correct instances in the dataset. The F-measure is the harmonic mean of precision and recall [18].

$$P = \frac{\text{number of correctly identified relation instances}}{\text{total number of identified relation instances}},$$
$$R = \frac{\text{number of correctly identified relation instances}}{\text{total number of correct relation instances}},$$
$$F = \frac{2 \times P \times R}{P + R}.$$

4.3 Numerical Results and Discussion

The effect of applying the difficulty classifier to the input sentences was evaluated and the behavior of the shallow and the deep extractors was explored. In our experiments, we used the datasets previously described in Section 4.1. We trained three different classifiers which read a sentence and decide if the sentence is easy or difficult for the extraction of relation.

To collect syntactic features, we need to perform POS tagging and chunking. Therefore, we use the OpenNLP package. We modeled extraction difficulty for sentences. Our modeling of sentence difficulty was binary: sentences are easy or difficult to extract for a system. Given a corpus, SDE-OIE should select sentences for the shallow/deep extractor so as to maximize the number of correctly extracted instances. In other words, it selects the extractor which produces a correct instance when the other extractor generates an incorrect result.

After testing, relation instances with score values equal to or higher than a specific threshold are considered to belong to the class 1. The instances with score values lower than this threshold are considered to belong to class 0. Different values of the classifiers scores were examined. It was observed that the threshold of 0.6 for both Logistic Regression and Decision Tree, and 0.7 for Naive Bayes yields the highest performance.

We ran different OIE systems on these datasets[3]. Table 3 shows the precision and recall of each system on two different datasets. NB, DT and LR subscripts are used for Naive Bayes, Decision Tree and Logistic Regression, respectively. There is an insignificant difference between the precision of SDE-OIE and its constituent systems. ReVerb and EXEMPLAR have relatively high precision due to designing good patterns for relation extraction; thus this can lead to a higher rate of precision in SDE-OIE. The best result for the precision of SDE-OIE was obtained by the Logistic Regression classifier. High precision of SDE-OIE is caused by its primary elements. This is also the case for recall. As a result, selecting the main components of the proposed approach has a direct effect on the overall precision and recall. In terms of precision, SONEX outperforms all other approaches

[3] We used the source codes of these OIE systems for implementation. The source code of SDE-OIE is available at `https://github.com/VahidehRt/SDE-OIE`.

since its pattern-based design is able to detect predicates triggered by a noun properly.

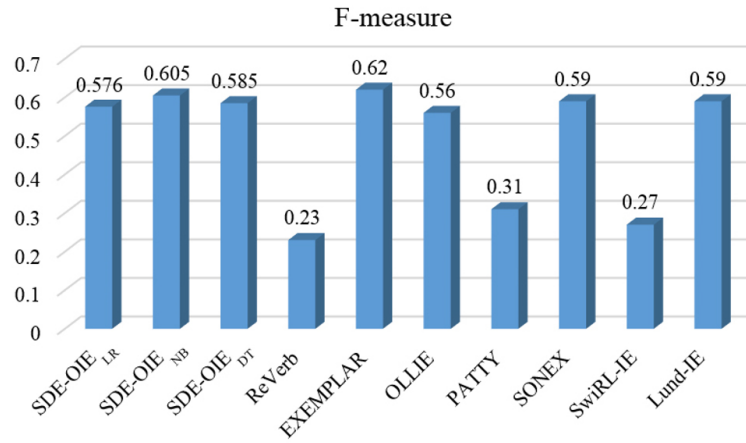
SDE-OIE's recall is higher than that of ReVerb. Moreover, it is lower than that for EXEMPLAR. EXEMPLAR has the highest precision among all the systems. It is superior mainly because it can recognize more correct instances, particularly those with verb + noun predicates [20]. The higher precision and the lower recall in comparison to EXEMPLAR reflect that our approach finds less relation instances than EXEMPLAR but most of the retrieved instances are accurate. Since both ReVerb and EXEMPLAR have no output for some sentences, the number of missing instances may be increased in comparison with EXEMPLAR individually. The best result for SDE-OIE's recall was achieved by the Naive Bayes classifier.

	Penn Treebank		New York Times	
Method	Precision	Recall	Precision	Recall
SDE-OIE$_{NB}$	0.78	0.49	0.8	0.3
SDE-OIE$_{DT}$	0.77	0.49	0.8	0.3
SDE-OIE$_{LR}$	0.79	0.43	0.81	0.26
ReVerb	0.78	0.14	0.8	0.13
EXEMPLAR	0.79	0.51	0.82	0.31
SONEX	0.92	0.43	0.84	0.22
OLLIE	0.81	0.43	0.81	0.25
PATTY	0.46	0.24	0.82	0.21
SwiRL-IE	0.89	0.16	0.84	0.2
Lund-IE	0.86	0.35	0.83	0.24

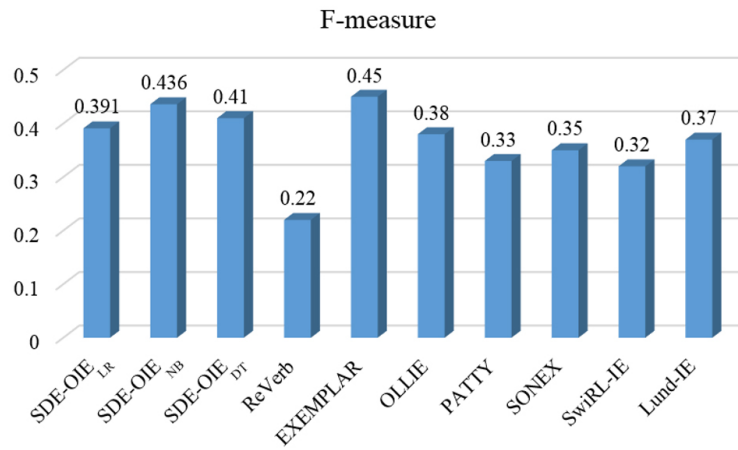
Table 2. Results for the task of extracting relations

Figure 2 shows the F-measure of each system. EXEMPLAR outperforms all methods. This is mainly because of its relatively higher recall in comparison with other methods. SDE-OIE$_{NB}$ has the best F-measure among the other SDE-OIE methods. SDE-OIE$_{NB}$ and EXEMPLAR are both at a very close level of F-measure. Based on the description given above, this can be interpreted in terms of precision and recall. ReVerb and EXEMPLAR have relatively high precision, therefore, SDE-OIE$_{NB}$'s precision is also high. SDE-OIE$_{NB}$ and EXEMPLAR have the same precision. SDE-OIE$_{NB}$'s recall is significantly higher than ReVerb, but slightly lower than EXEMPLAR. Thus, SDE-OIE$_{NB}$'s F-measure is slightly lower than that of EXEMPLAR, as it is defined as a harmonic mean of precision and recall.

SDE-OIE achieves an F-measure that is almost triple that of ReVerb. ReVerb has a lower recall than other approaches because of the intrinsic weakness of shallow tools in detecting relation instances. This leads to a significant drop in its F-measure. The experiment results demonstrate that proper incorporation of shallow and deep extractors decreases the number of incorrect extractions and increases the correct ones, resulting in higher performance. On the other hand, our hybrid method is able



a)



b)

Figure 2. The F-measure of our method in comparison with other methods drawn from a) Penn Treebank b) New York Times. SDE-OIE$_{NB}$ and EXEMPLAR have almost the same F-measure. Their F-measure is better than that of the others.

to cover the limitations of the shallow OIE system and provides significant boost in its performance. The satisfactory level of F-measure indicates that our approach is at least as good as its deep constituent system.

The total computing time of each method was measured. We excluded the time for initializing or loading any libraries or models into memory. To ensure a fair comparison, we make sure each method runs in a single-threaded mode, thus utilizing a single computing core at all times. The results are reported in Table 4. The best results of our approach are highlighted in the table.

Method	Penn Treebank	New York Times
SDE-OIE$_{NB}$	0.41	0.73
SDE-OIE$_{DT}$	0.4	0.71
SDE-OIE$_{LR}$	**0.38**	**0.68**
ReVerb	0.02	0.01
EXEMPLAR	0.62	1.19
SONEX	0.04	0.03
OLLIE	0.14	0.23
PATTY	0.66	1.27
SwiRL-IE	2.17	3.49
Lund-IE	5.21	11.20

Table 3. Computing time (per second) for each method

The processing times vary with different types of extractors. There is an explicit differentiation of almost one order of magnitude among approaches based on semantic parsing (SwiRL-IE and Lund-IE), dependency parsing (EXEMPLAR, OLLIE and PATTY) and shallow parsing (ReVerb and SONEX). ReVerb is the fastest method since it uses shallow patterns and does not rely on any deep tool.

As the results show, deep extractors usually have a high computational cost. There is always a trade-off between performance and speed when selecting a deep extractor. Deep extractors usually have high computational cost. In general, the deeper the extractor, the higher is the incurred computational cost. In the same period of time, shallow extractors process several times more sentences than dependency parsing extractors, which in turn process several times more sentences than semantic parsing extractors. Since our main purpose is to achieve a high-performance system both in time and performance, we have leveraged the best of shallow and deep OIE systems. Selecting shallow and deep components was made according to a fair and objective experimental comparison of 10 state-of-the-art approaches which is presented in [19] and [20]. Thus, SDE-OIE makes effective use of available time and achieves a reasonable balance of precision and recall. As experiment results show, the efficiency of the whole system has been affected by optimizing the processing time which has been reduced more than 33 % for all classifiers. The processing time for SDE-OIE$_{NB}$, SDE-OIE$_{DT}$ and SDE-OIE$_{LR}$ was reduced by 33 %, 35 % and 38 %, respectively, in the Penn Treebank dataset and 38 %, 40 % and 42 %, respectively, in the New York Times corpus.

SDE-OIE$_{LR}$ is the fastest model among the other SDE-OIE models. Generally, SDE-OIE has approximately the same F-measure as EXEMPLAR, but at a much lower processing time. This becomes important in large inputs such as Web-scale data. When the number of sentences processed by ReVerb is high, the total time reduces to the processing time of ReVerb. An interesting result is that despite achieving high accuracy, the methods based on semantic parsing (SwiRL-IE and Lund-IE) have lower F-measure than SDE-OIE and also need too much computational time.

4.4 Evaluation of Sentence Difficulty Estimator

The distribution of easy and difficult sentences is 39 % and 61 %, respectively (difficult being the majority class). Table 5 shows the distribution of the values in the confusion matrix for all classifiers (TN, FP, FN, TP). In general, there are two types of errors in our hybrid extractor. The first type is related to intrinsic weakness of the constituent systems and amending these errors depends on the improvement of the main extractors of the hybrid method. This kind of error is not caused by difficulty classifier; it can be generated even if the proper extractor is selected. The second type of error occurs with the incorrect selection of the extractor by the difficulty classifier. This type of error almost always occurs as a direct result of the first type of error. For example, in this case, the input sentence should be processed by the shallow extractor but it is processed by the deep extractor. Thus, the optimal result is not gained. Although this error may not affect the performance of the hybrid method, it is not beneficial for the extraction speed.

Table 5 shows the accuracy and error rate for each classifier on the two datasets. The accuracy for SDE-OIE$_{\text{NB}}$, SDE-OIE$_{\text{DT}}$ and SDE-OIE$_{\text{LR}}$ is 72 %, 68 % and 73 %, respectively, in the Penn Treebank dataset and 72 %, 67 % and 74 %, respectively, in the New York Times corpus. The results show that SDE-OIE$_{\text{LR}}$ is the most accurate classifier among the three classifiers. We observe that the dominant error in SDE-OIE$_{\text{LR}}$ is classifying a difficult sentence as easy. In general, a sentence difficulty classifier with a high accuracy results in a reasonable trade-off between time and performance, because selecting the proper OIE system leads to a significant reduction on the computational time of the whole system.

		SDE-OIE$_{\text{NB}}$		SDE-OIE$_{\text{DT}}$		SDE-OIE$_{\text{LR}}$	
		Difficult	Easy	Difficult	Easy	Difficult	Easy
Penn Treebank	Difficult	81.8 %	18.1 %	72.7 %	27.2 %	68.1 %	31.8 %
	Easy	37.7 %	62.2 %	35.7 %	64.2 %	21.4 %	78.5 %
New York Times	Difficult	82.7 %	17.3 %	71.6 %	28.3 %	69.3 %	30.6 %
	Easy	37.3 %	62.6 %	36.5 %	63.4 %	20.3%	79.6 %

Table 4. The confusion matrix for the performance of the sentence difficulty classifiers

	Penn Treebank		New York Times	
	Accuracy	Error rate	Accuracy	Error rate
SDE-OIE$_{\text{DT}}$	68.5 %	27.3 %	67.5 %	32.4 %
SDE-OIE$_{\text{NB}}$	72 %	28 %	72.6 %	27.3 %
SDE-OIE$_{\text{LR}}$	73.3 %	26.6 %	74.5 %	25.4 %

Table 5. The accuracy and error rate of the difficulty classifiers

5 CONCLUSIONS

We presented a new method for the automatic estimation of sentence difficulty in ORE systems. In other words, this work explores the notion of relation extraction difficulty and the ways how the difficulty information can be used to enhance extraction quality in shallow extractors. We applied different classifiers with a set of efficient sentence-based features to incorporate strengths of a shallow OIE system with a deep one. Our sentence difficulty classifier detects difficult sentences for processing by the deep extractor. We detected the best trade-off between efficiency (computational cost) and effectiveness (F-measure). Experiment results demonstrate that the proposed approach achieves significantly better F-measure than its shallow extractor. SDE-OIE also has approximately the same level of F-measure as its deep constituent extractor, but at a much lower processing time. This shows that isolation of difficult sentences from the rest of the sentences creates flexibility for applying different types of system customizations.

The aim of OIE is to scale information extraction methods to the size and diversity of the Web domain. SDE-OIE passes an input sentence to a deep extractor only if it is needed. SDE-OIE is able to better allocate computational resources and avoid wasting them, and thus it is suitable in cases where the computing time is limited and high performance is desired.

We believe that an extended work on difficulty modeling should incorporate different sophistication levels of NLP tools; thus this method can be extended to a multi-class problem as well. A SRL-based approach can be applied as the deepest underlying extractor. The proposed features are very fast to compute, which is an important property from a practical implementation perspective. In addition to our proposed features, some features from the underlying systems can be incorporated into the difficulty classifier. Using semantic features can also bring deeper linguistic knowledge into our model, but at the cost of time.

REFERENCES

[1] Piskorski, J.—Yangarber, R.: Information Extraction: Past, Present and Future. In: Poibeau, T., Saggion, H., Piskorski, J., Yangarber, R. (Eds.): Multi-Source, Multilingual Information Extraction and Summarization, Springer, 2013, pp. 23–49, doi: 10.1007/978-3-642-28569-1_2.

[2] Min, B.—Shi, S.—Grishman, R.—Lin, C.-Y.: Towards Large-Scale Unsupervised Relation Extraction from the Web. International Journal on Semantic Web and Information Systems (IJSWIS), Vol. 8, 2012, No. 3, pp. 1–23, doi: 10.4018/jswis.2012070101.

[3] Banko, M.—Etzioni, O.: The Tradeoffs Between Open and Traditional Relation Extraction. Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics, ACL, 2008, pp. 28–36.

[4] Mausam—Schmitz, M.—Bart, R.—Soderland, S.—Etzioni, O.: Open Language Learning for Information Extraction. Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '12), 2012, pp. 523–534.

[5] Del Corro, L.—Gemulla, R.: ClausIE: Clause-Based Open Information Extraction. Proceedings of the 22nd International Conference on World Wide Web (WWW '13), 2013, pp. 355–366, doi: 10.1145/2488388.2488420.

[6] Etzioni, O.—Banko, M.—Soderland, S.—Weld, D. S.: Open Information Extraction from the Web. Communications of the ACM, Vol. 51, 2008, No. 12, pp. 68–74, doi: 10.1145/1409360.1409378.

[7] Wu, F.—Weld, D. S.: Open Information Extraction Using Wikipedia. Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL '10), 2010, pp. 118–127.

[8] Fader, A.—Soderland, S.—Etzioni, O.: Identifying Relations for Open Information Extraction. Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP '11), 2011, pp. 1535–1545.

[9] Etzioni, O.—Fader, A.—Christensen, J.—Soderland, S.—Mausam, M.: Open Information Extraction: The Second Generation. Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence – Volume One (IJCAI '11), 2011, pp. 3–10.

[10] Merhav, Y.—Mesquita, F.—Barbosa, D.—Yee, W. G.—Frieder, O.: Extracting Information Networks from the Blogosphere. ACM Transactions on the Web (TWEB), Vol. 6, 2012, Art. No. 11, doi: 10.1145/2344416.2344418.

[11] Akbik, A.—Bross, J.: Wanderlust: Extracting Semantic Relations from Natural Language Text Using Dependency Grammar Patterns. Proceedings of the 2009 Semantic Search Workshop at the 18th International World Wide Web Conference, 2009, pp. 6–15.

[12] Akbik, A.—Löser, A.: Kraken: N-ary Facts in Open Information Extraction. Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-Scale Knowledge Extraction (AKBC-WEKEX), 2012, pp. 52–56.

[13] Qiu, L.—Zhang, Y.: ZORE: A Syntax-Based System for Chinese Open Relation Extraction. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1870–1880, doi: 10.3115/v1/d14-1201.

[14] Gamallo, P.—Garcia, M.—Fernández-Lanza, S.: Dependency-Based Open Information Extraction. Proceedings of the Joint Workshop on Unsupervised and Semi-Supervised Learning in NLP, 2012, pp. 10–18.

[15] Christensen, J.—Mausam—Soderland, S.—Etzioni, O.: An Analysis of Open Information Extraction Based on Semantic Role Labeling. Proceedings of the Sixth International Conference on Knowledge Capture (K-CAP '11), 2011, pp. 113–120, doi: 10.1145/1999676.1999697.

[16] Kim, M. H.—Compton, P.: Improving Open Information Extraction for Informal Web Documents with Ripple-Down Rules. In: Richards, D., Kang, B. H. (Eds.): Knowledge Management and Acquisition for Intelligent Systems (PKAW 2012). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7457, 2012, pp. 160–174, doi: 10.1007/978-3-642-32541-0_14.

[17] Tseng, Y.-H.—Lee, L.-H.—Lin, S.-Y.—Liao, B.-S.—Liu, M.-J.—Chen, H.-H.—Etzioni, O.—Fader, A.: Chinese Open Relation Extraction for Knowledge Acquisition. Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2014), 2014, pp. 12–16.

[18] Reshadat, V.—Hoorali, M.—Faili, H.: A Hybrid Method for Open Information Extraction Based on Shallow and Deep Linguistic Analysis. Interdisciplinary Information Sciences, Vol. 22, 2016, pp. 87–100, doi: 10.4036/iis.2016.r.03.

[19] Mesquita, F.—Schmidek, J.—Barbosa, D.: Effectiveness and Efficiency of Open Relation Extraction. Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, 2013, pp. 447–457.

[20] de Sá Mesquita, F.: Extracting Information Networks from Text. Ph.D. Thesis, University of Alberta, 2015.

[21] Mohit, B.—Liberato, F.—Hwa, R.: Language Model Adaptation for Difficult to Translate Phrases. Proceedings of the 13th Annual Conference of the EAMT, 2009, pp. 160–167.

[22] Kawahara, D.—Uchimoto, K.: Learning Reliability of Parses for Domain Adaptation of Dependency Parsing. Proceedings of the Third International Joint Conference on Natural Language Processing: Volume-II (IJCNLP), 2008, pp. 709–714.

[23] Kauchak, D.—Leroy, G.—Hogue, A.: Measuring Text Difficulty Using Parse-Tree Frequency. Journal of the Association for Information Science and Technology, Vol. 68, 2017, No. 9, pp. 2088–2100, doi: 10.1002/asi.23855.

[24] Hale, J. T.: Heuristic Search in a Cognitive Model of Human Parsing. Proceedings of the 11th International Conference on Parsing Technologies (IWPT '09), 2009, pp. 230–233, doi: 10.3115/1697236.1697283.

[25] Nakashole, N.—Weikum, G.—Suchanek, F.: PATTY: A Taxonomy of Relational Patterns with Semantic Types. Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '12), 2012, pp. 1135–1145.

[26] Xu, Y.—Kim, M.-Y.—Quinn, K.—Goebel, R.—Barbosa, D.: Open Information Extraction with Tree Kernels. Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (HLT-NAACL), 2013, pp. 868–877.

[27] Saha, S.—Pal, H.—Mausam: Bootstrapping for Numerical Open IE. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), 2017, pp. 317–323, doi: 10.18653/v1/p17-2050.

[28] Falke, T.—Stanovsky, G.—Gurevych, I.—Dagan, I.: Porting an Open Information Extraction System from English to German. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2016, pp. 892–898, doi: 10.18653/v1/d16-1086.

[29] Otero, P. G.—López, I. G.: A Grammatical Formalism Based on Patterns of Part of Speech Tags. International Journal of Corpus Linguistics, Vol. 16, 2011, No. 1, pp. 45–71, doi: 10.1075/ijcl.16.1.03gam.

[30] Gamallo, P.—Garcia, M.: Multilingual Open Information Extraction. In: Pereira, F., Machado, P., Costa, E., Cardoso, A. (Eds.): Progress in Artificial Intelligence (EPIA 2015). Springer, Cham, Lecture Notes in Computer Science, Vol. 9273, 2015, pp. 711–722, doi: 10.1007/978-3-319-23485-4_72.

[31] Bast, H.—Haussmann, E.: Open Information Extraction via Contextual Sentence Decomposition. 2013 IEEE Seventh International Conference on Semantic Computing (ICSC), 2013, pp. 154–159, doi: 10.1109/icsc.2013.36.

[32] Punyakanok, V.—Roth, D.—Yih, W.-T.: The Importance of Syntactic Parsing and Inference in Semantic Role Labeling. Computational Linguistics, Vol. 34, 2008, No. 2, pp. 257–287, doi: 10.1162/coli.2008.34.2.257.

[33] Johansson, R.—Nugues, P.: The Effect of Syntactic Representation on Semantic Role Labeling. Proceedings of the 22nd International Conference on Computational Linguistics (COLING '08), Vol. 1, 2008, pp. 393–400, doi: 10.3115/1599081.1599131.

[34] Johansson, R.—Nugues, P.: Dependency-Based Semantic Role Labeling of PropBank. Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '08), 2008, pp. 69–78, doi: 10.3115/1613715.1613726.

[35] Surdeanu, M.—Harabagiu, S.—Williams, J.—Aarseth, P.: Using Predicate-Argument Structures for Information Extraction. Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (ACL '03), Vol. 1, 2003, pp. 8–15, doi: 10.3115/1075096.1075098.

[36] Banko, M.—Cafarella, M. J.—Soderland, S.—Broadhead, M.—Etzioni, O.: Open Information Extraction for the Web. Proceedings of the 20th International Joint Conference on Artifical Intelligence (IJCAI '07), 2007, pp. 2670–2676.

[37] Mesquita, F.: Clustering Techniques for Open Relation Extraction. Proceedings of the SIGMOD/PODS 2012 Ph.D. Symposium, 2012, pp. 27–32, doi: 10.1145/2213598.2213607.

[38] Castellã Xavier, C.—Strube de Lima, V. L.—Souza, M.: Open Information Extraction Based on Lexical-Syntactic Patterns. 2013 Brazilian Conference on Intelligent Systems (BRACIS), 2013, pp. 189–194, doi: 10.1109/bracis.2013.39.

[39] Cimiano, P.—Wenderoth, J.: Automatically Learning Qualia Structures from the Web. Proceedings of the ACL-SIGLEX Workshop on Deep Lexical Acquisition (DeepLA '05), 2005, pp. 28–37, doi: 10.3115/1631850.1631854.

[40] Lin, H.—Wang, Y.—Zhang, P.—Wang, W.—Yue, Y.—Lin, Z.: A Rule Based Open Information Extraction Method Using Cascaded Finite-State Transducer. In: Bailey, J., Khan, L., Washio, T., Dobbie, G., Huang, J., Wang, R. (Eds.): Advances

in Knowledge Discovery and Data Mining (PAKDD 2016). Springer, Cham, Lecture Notes in Computer Science, Vol. 9652, 2016, pp. 325–337, doi: 10.1007/978-3-319-31750-2_26.

[41] Ebadat, A.-R.—Claveau, V.—Sébillot, P.: Using Shallow Linguistic Features for Relation Extraction in Bio-Medical Texts. Traitement Automatique des Langues Naturelles (TALN 2011), 2011, pp. 125–130.

[42] Jurafsky, D.—Martin, J. H.: Speech and Language Processing. Pearson London, 2014.

[43] Ng, A. Y.—Jordan, M. I.: On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes. Advances in Neural Information Processing Systems 14 (NIPS 2001), 2002, pp. 841–848.

[44] Castellã Xavier, C.—Strube de Lima, V. L.—Souza, M.: Open Information Extraction Based on Lexical Semantics. Journal of the Brazilian Computer Society, Vol. 21, 2015, Art. No. 4, 14 pp., doi: 10.1186/s13173-015-0023-2.

[45] Léchelle, W.—Langlais, P.: An Informativeness Approach to Open IE Evaluation. In: Gelbukh, A. (Ed.): Computational Linguistics and Intelligent Text Processing (CICLing 2016). Springer, Cham, Lecture Notes in Computer Science, Vol. 9624, 2018, pp. 523–534, doi: 10.1007/978-3-319-75487-1_40.

[46] Xu, Y.: Relation Extraction and Its Application to Question Answering. Ph.D. Thesis, University of Alberta, 2017.

[47] Schmidek, J.—Barbosa, D.: Improving Open Relation Extraction via Sentence Re-Structuring. Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), 2014, pp. 3720–3723.

[48] Sandhaus, E.: The New York Times Annotated Corpus. Philadelphia, PA: Linguistic Data Consortium. `https://catalog.ldc.upenn.edu/LDC2008T19`, 2008.

Vahideh Reshadat received her B.Sc. and M.Sc. both in computer engineering in 2008 and 2012, respectively. She has been teaching as Lecturer in different universities of Iran since 2012. Her main interests are natural language processing, machine learning, deep neural networks and text mining.

Heshaam Faili received his B.Sc. and M.Sc. degrees in computer engineering and his Ph.D. in artificial intelligence from Sharif University of Technology in 1997, 1999 and 2006, respectively. He joined the Artificial Intelligence and Robotic Group of the School of Electrical and Computer Engineering, University of Tehran in 2008. He is now Associate Professor and his main research interests include AI, statistical approaches, text processing, and mining methods.