Mathematical Modelling and Analysis Publisher: Taylor&Francis and VGTU
Volume 20 Number 5, September 2015, 619–640 http://www.tandfonline.com/TMMA
http://dx.doi.org/10.3846/13926292.2015.1088903 ISSN: 1392-6292
© Vilnius Gediminas Technical University, 2015 eISSN: 1648-3510

Parallel Optimization Algorithm for Competitive Facility Location

Algirdas Lančinskas[a], **Pilar Martínez Ortigosa**[b] **and Julius Žilinskas**[a]

[a] *Vilnius University, Institute of Mathematics and Informatics*
Akademijos str. 4, LT-08663 Vilnius, Lithuania
[b] *Universidad de Almería*
ceiA3, ctra. Sacramento s/n, La Canada de San Urbano 04120, Almería, Spain
E-mail: `algirdas.lancinskas@mii.vu.lt`
E-mail: `ortigosa@ual.es`
E-mail(*corresp.*): `julius.zilinskas@mii.vu.lt`

Received June 10, 2015; revised August 17, 2015; published online September 15, 2015

Abstract. A stochastic search optimization algorithm is developed and applied to solve a bi-objective competitive facility location problem for firm expansion. Parallel versions of the developed algorithm for shared- and distributed-memory parallel computing systems are proposed and experimentally investigated by approximating the Pareto front of the competitive facility location problem of different scope. It is shown that the developed algorithm has advantages against its precursor in the sense of the precision of approximation. It is also shown that the proposed parallel versions of the algorithm have almost linear speed-up when solving competitive facility location problems of different scope reasonable for practical applications.

Keywords: optimization, facility location, parallel computing.

AMS Subject Classification: 90C29; 68W10.

1 Introduction

It is believed that Facility Location (FL) as a science has originated from Pierre de Fermat, Battista Cavallieri, and Evangelistica Torricelli since they independently proposed the basic Euclidean spatial median problem early in the seventeenth century [6]. However formally, the most important starting point in the history of location science is assumed to be the Alfred Weber's book [23]. The location of facilities is important for the firms providing services to customers in a certain geographical region. There are a lot of models of FL proposed in literature, e.g. [8, 10, 18, 20], which differ on their properties such as location space, describing possible locations for the facilities, attractiveness of facilities, or behavior of customers when choosing a facility to get a service.

1.1 Competitive FL for firm expansion

Consider a firm F_A is planning to locate a set of facilities (or a single facility) in a region where other firms already have facilities and provide a service. The firm F_A has to compete for the market share in the region and take the competition into account when determining locations for the new facilities. Such a kind of FL problems are known as Competitive Facility Location Problems (CFLP).

Suppose that the firm F_A is already in the market (has preexisting facilities and already provides service to the customers in the region) and planning to extend its market share by establishment a set of new facilities. Thus the firm F_A faces the CFLP for Firm Expansion (CFLP/FE). Despite the maximization of the market share captured by the new facilities, the impact of the new facilities on the preexisting ones belonging to the expanding firm should be simultaneously taken into account in CFLP/FE.

Consider the firms F_A and F_B, providing a service to a set I of demand points. The firm F_A has a set of n_A preexisting facilities and the firm F_B has a set of n_B preexisting facilities, providing service for customers in a given geographical region and competing for the market share. The firm F_A wants to increase its market share by establishing a set F_X of n_X new facilities. The new facilities can attract new customers from the facilities of the firm F_B thus increasing the total market share of F_A. On the other hand the new facilities can attract customers who are already served by own facilities of the firm F_A, thus giving raise of the effect of cannibalism [9]. Therefore the firm F_A faces a bi-objective optimization problem with the following objectives: (1) to maximize the market share of the facilities being located and (2) to minimize the loss of market share of the preexisting facilities of the firm F_A (the effect of cannibalism).

A single evaluation of the utility of the new facilities and the effect of cannibalism requires to determine the most attractive facility for every demand point. In order to determine the most attractive facility for a certain demand point $i \in I$, the attractiveness of the demand point to each of the facilities in F_A, F_B, and F_X must be evaluated, thus requiring to perform $n_A + n_B + n_X$ such evaluations. Thus, considering the evaluation of the attractiveness which customers from a certain demand point feel to a certain facility as the basic operation, the complexity of the evaluation of the utility and the effect of the cannibalism can be expressed as $n_I \times (n_A + n_B + n_X)$, where n_I stands for the number of demand points. After such a number of the evaluations of the attractiveness, we are able to determine the most attractive facility for each demand point and evaluate the utility of the facilities in F_X by summing the buying power of all demand points which feel the maximum attractiveness to the facilities from F_X; the effect of cannibalism can be evaluated by the difference of the market share of the facilities in F_A before and after the expansion. Although the complexity of the procedure above can be reduced by optimizing the computational work according to the certain model of customers' behavior, we are interested in development of a general parallel algorithm which would be independent on the model of customers' behavior.

1.2 Multi-objective optimization

In general a CFLP/FE can be expressed as a mathematical optimization problem to find a *decision vector*

$$\mathbf{x} = (x_1, x_2, \ldots, x_d),$$

describing the locations of the the new facilities, which would be optimal with respect to the values of the *objective functions*: $f_1(\mathbf{x})$ describing market share of the new facilities and $f_2(\mathbf{x})$ describing the effect of cannibalism. The decision vector $\mathbf{x}$ can be selected from a set $\mathbb{D} \subset \mathbb{R}^d$ of all possible decision vectors, called *search space*, where d is the number of problem variables.

Due to conflicting objectives usually it is impossible to find a single solution which would be the best by both objectives. Moreover, comparison of two decision vectors by the value of a single objective is meaningless as a decision vector better by one objective can be worse or even the worst by another one. On the other hand, the fitness of two different decision vectors $\mathbf{x}$ and $\mathbf{y}$ can be compared by the *dominance relation.* In terms of multi-objective optimization two different decision vectors $\mathbf{x}$ and $\mathbf{y}$ can be related with each other in three different ways: $\mathbf{x}$ *dominates* $\mathbf{y}$ and vice versa, as well as none of them are dominated by the other. It is said that the decision vector $\mathbf{x}$ dominates the decision vector $\mathbf{y}$ (denoted by $\mathbf{x} \succ \mathbf{y}$) if (1) $\mathbf{x}$ is not worse than $\mathbf{y}$ by any of the objectives and (2) $\mathbf{x}$ is strictly better than $\mathbf{y}$ by at least one objective. If $\mathbf{x} \succ \mathbf{y}$, then $\mathbf{x}$ is called a *dominator* of $\mathbf{y}$; if none of two decision vectors can be distinguished as a dominator of the other, they are considered as *indifferent* in the sense of the dominance relation.

A decision vector $\mathbf{x}$ which has no dominators in the whole search space $\mathbb{D}$ is called *non-dominated*, or *Pareto-optimal*, and a set of non-dominated decision vectors is called the *Pareto set.* The corresponding set of the values of the objective functions for non-dominated decision vectors is called the *Pareto front.*

Determination of the exact Pareto front of a multi-objective optimization problem usually is a hard and time consuming task, which can be even intractable within an acceptable time. On the other hand solution of practical problems usually does not require to find the exact Pareto front, but rather its approximation by a limited set of non-dominated objective vectors. Therefore multi-objective optimization methods approximating the Pareto front are usually used to tackle practical problems. A well known class of such algorithms are Evolutionary Algorithms (EAs), which require little knowledge about the problem being solved, are easy to implement, and can be parallelized [1].

1.3 Related works

There is a great amount of work devoted to an approximation of the Pareto front of CFLP. For example, Redondo et al. [19] proposed a general multi-objective optimization heuristic algorithm and applied it for CFLP; Zitzler et al. [24] proposed the Strength Pareto Evolutionary Algorithm (SPEA2) and Huapu and Jifeng [11] utilized it to solve a bi-level optimization problem related to distribution centers; Deb et al. [5] proposed the Non-dominated Sorting Genetic Algorithm (NSGA-II) and Villegas et al. [22] utilized it to solve a bi-objective FL problem by minimizing operational cost of Colombian Coffee supply network and maximizing the demand.

Although EAs are popular due to applicability to various practical problems, their performance can be notably improved by incorporating a local search thus deriving so called memetic algorithms. For example, Medaglia et al. [17] utilized hybrid NSGA-II and mixed-integer programming approach to solve bi-objective obnoxious FL problem related to the hospital waste management network; Multi-objective Single Agent Stochastic Search (MOSASS) has been proposed in [13] and has been incorporated into the NSGA-II, thus developing a hybrid multi-objective optimization algorithm called NSGA/LSP.

There are also some works devoted to parallel algorithms to tackle FL problems. For example, Belloch and Tangwongsan [2] analyzed metric FL, k-median, k-means, and k-center problems, de Silva and Abramson [4] developed a parallel interior point method for FL.

A parallel algorithm suitable for multi-objective FL, in particular, NSGA-II have been proposed in [14], some version of which have been applied to solve the CFLP using a large scale computing system in [15]. Consequently the parallel version of NSGA/LSP, suitable for hybrid distributed-shared memory architecture of parallel computing system, has been proposed and applied to solve the CFLP in [16]. The main disadvantage of the parallel NSGA/LSP appears to be a sequential local optimization of a single solution, which leads to an idle time of some processing units during the local optimization procedure.

In this paper we will focus on development and investigation of the parallel multi-objective local search algorithms, by modifying previously proposed Multi-Objective Single Agent Stochastic Search (MOSASS), as well as on application of the developed parallel algorithms to solve bi-objective CFLP/FE using shared- and distributed-memory parallel computing systems. The proposed shared-memory parallel algorithm is based on a general model for shared-memory parallel computing [12], where the main concern is to guarantee the consistent access to the shared memory. The proposed distributed-memory parallel algorithm is based on the master-slave model [12], where one of the processors is responsible for the organization of the workload of the slaves as well as for the communication between them. For the application of the master-slave model to parallelize genetic and other algorithms we refer to [3,7,21]. The main concern in development of the distributed-memory parallel algorithm is the organization of the effective workload of the slave (and master) processing units, what is not a trivial task due to the nature of the algorithm being parallelized.

The remainder of the paper is organized as follows: Section 2 consists of description of multi-objective optimization stochastic search algorithm derived from MOSASS; Section 3 describes the parallel multi-objective stochastic search algorithms for shared- and distributed-memory parallel computing systems; Section 4 presents description and results of the experimental investigation of the proposed parallel algorithms.

2 Multi-objective stochastic search algorithm

Multi-Objective Stochastic Search (MOSS) algorithm has been developed from its precursor MOSASS [13]. The concept of MOSASS as well as of its precursor SASS for single-objective optimization is based on generation of a new solution $\mathbf{x}'$ in a neighborhood of a single decision vector $\mathbf{x}$ and consideration of an opposite decision vector $\mathbf{x}''$ if $\mathbf{x}'$ is not acceptable; the decision vector $\mathbf{x}$ is updated only if a dominating decision vector is found. Since the decision to consider $\mathbf{x}''$ can be made after the evaluation of $\mathbf{x}'$, parallelization of process is complicated. Therefore we propose to use multi-agent search strategy, where parent decision vector is randomly selected from the archive of non-dominated decision vectors, thus giving availability to simultaneously use an arbitrary number of decision vectors $\mathbf{x}$ and parallelize the process.

Scheme of MOSS algorithm is presented in Figure 2. The algorithm begins with an initial non-empty archive $\mathbb{A}$ of the decision vectors which are non-dominated among themselves; the initial archive can also consist of a single decision vector, which is naturally non-dominated in the archive. A new decision vector $\mathbf{x}'$ is generated by applying *mutation* to the decision vector $\mathbf{x}$ sampled from $\mathbb{A}$. The mutation is performed by adding *mutation vector*

$$\boldsymbol{\xi} = (\xi_1, \xi_2, \dots \xi_d)$$

to the sampled decision vector $\mathbf{x}$:

$$\mathbf{x}' = \mathbf{x} + \boldsymbol{\xi}. \tag{2.1}$$

Each element ξ_i of the mutation vector is generated following the expression

$$\xi_i = \begin{cases} \mathcal{N}(0,\sigma), & \text{if} \quad r_i \le 1/d, \\ 0, & \text{if} \quad r_i > 1/d, \end{cases} \tag{2.2}$$

where $\mathcal{N}(0,\sigma)$ stands for a random number, generated following the Gaussian distribution with the zero bias and the standard deviation σ, r_i is a random number uniformly generated over $[0,1]$, d is the number of problem variables, and $i = 1, 2, \dots, d$. Such a probabilistic method for generation of a neighbor decision vector leads to the change of a single coordinate in average; see [13] for details and advantages of the method.

The newly generated decision vector can obtain one of the following three states: (a) the decision vector $\mathbf{x}'$ is non-dominated in $\mathbb{A}$ and does not dominate any of decision vectors in $\mathbb{A}$; (b) the decision vector $\mathbf{x}'$ is non-dominated in $\mathbb{A}$ and dominates at least one decision vector in $\mathbb{A}$; (c) the decision vector $\mathbf{x}'$ is dominated by at least one decision vector in archive $\mathbb{A}$.

In the first case, the archive $\mathbb{A}$ is supplemented by the decision vector $\mathbf{x}'$, and the algorithm proceeds to the next iteration assuming the current iteration to be successful. If $\mathbf{x}'$ is non-dominated in $\mathbb{A}$ and there exists at least one decision vector dominated by $\mathbf{x}'$ (the second case), then the archive is updated by including $\mathbf{x}'$ and removing the dominated decision vectors:

$$A \leftarrow (A \cup \{\mathbf{x}'\}) \setminus \{\mathbf{y} \in A : \mathbf{x}' \succ \mathbf{y}\}.$$

The algorithm proceeds to the next iteration assuming the current iteration as successful. In the last case the decision vector $\mathbf{x}'$ is rejected assuming the iteration to be failed and the opposite decision vector

$$\mathbf{x}'' = \mathbf{x} - \boldsymbol{\xi} \tag{2.3}$$

is considered following the same scheme as it was done for $\mathbf{x}'$.

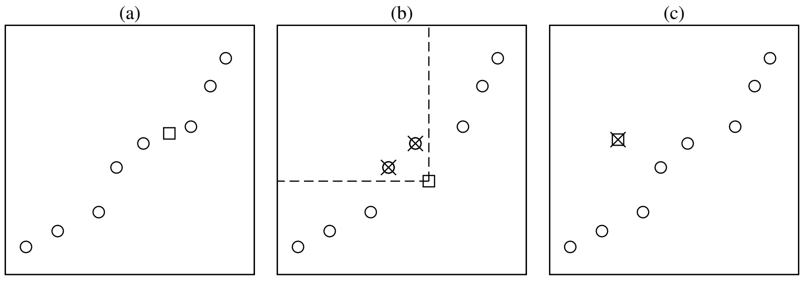

Figure 1. Illustration of three cases of the newly generated decision vector.

The principle of updating the archive is illustrated in Figure 1, where the left image illustrates inclusion of the newly generated decision vector into the archive; the middle image – inclusion of the decision vector with removal of the dominated ones; the right image – rejection of the newly generated decision vector. The square in the images stands for the newly generated decision vector, and crossed points – for the rejected decision vectors.

The standard deviation of the Gaussian perturbation in (2.2) is dynamically adjusted with respect to the number of repetitive successful and failed iterations: if the number of repetitive successful iterations exceeds 3, then the standard deviation is doubled; in the case of 3 repetitive failures the standard deviation is reduced by half. If the standard deviation reaches its lower bound, then it is set to the upper bound in order to avoid stuck in a local optimum; the check for violation of the upper bound is not reasonable as the standard deviation will be automatically reduced due to repetitive failed iterations when the search starts to be chaotic enough.

3 Parallel multi-objective stochastic search

The MOSS algorithm can be briefly separated into three main parts: the initialization of the algorithm, where loading of initial data and assignment of initial values of the parameters take place, the main loop, where iterative process of approximation of the Pareto set takes place, and the finalization, where processing and output of the obtained results take place.

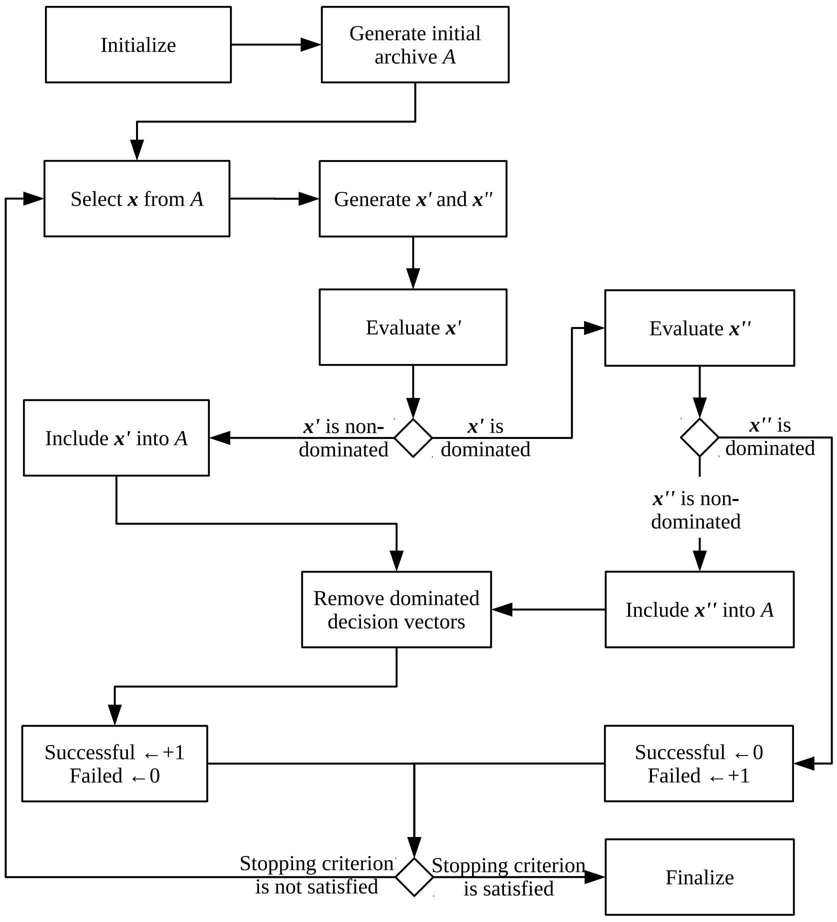
Initialize
Generate initial archive A
Select $\boldsymbol{x}$ from A
Generate $\boldsymbol{x}'$ and $\boldsymbol{x}''$
Evaluate $\boldsymbol{x}'$
Evaluate $\boldsymbol{x}''$
Include $\boldsymbol{x}'$ into A
$\boldsymbol{x}'$ is non-dominated
$\boldsymbol{x}'$ is dominated
$\boldsymbol{x}''$ is dominated
$\boldsymbol{x}''$ is non-dominated
Remove dominated decision vectors
Include $\boldsymbol{x}''$ into A
Successful $\leftarrow$+1 Failed $\leftarrow$0
Successful $\leftarrow$0 Failed $\leftarrow$+1
Stopping criterion is not satisfied
Stopping criterion is satisfied
Finalize

Figure 2. Scheme of MOSS algorithm.

The second part of the algorithm usually is the most time consuming as the evaluation of objective function values usually takes the main computational effort. The first and the third parts are much less time-consuming and their consideration as sequential parts should not make a significant impact on the performance of a parallel algorithm. Assuming that the evaluations of the objective values of different decision vectors can be considered as independent tasks, they can be assigned to different processing units. In such a distribution of tasks the information about the archive $\mathbb{A}$ of all non-dominated decision vectors found so far as well as values of other parameters of the algorithm (the standard deviation σ, the counters of repetitive successful and failed iterations as well as the total number of performed iterations) must be accessed by all processing units.

3.1 Shared memory ParMOSS

In shared-memory parallel computing systems all processing units have equal privileges to access the information stored in the shared-memory. On the other hand a consistent access to the archive $\mathbb{A}$ and parameters of the algorithm must be guaranteed, i.e. if one of the processing units is updating a parameter (or the archive), access of any other processing unit to that parameter (or the archive) is blocked in order to keep the memory or data consistency. Taking the latter considerations into account, the parallel version of MOSS, called ParMOSS/OMP, has been developed using Application Programming Interface (API) specification OpenMP for shared-memory parallel programing.

ParMOSS/OMP begins with initialization of the parameters of the algorithm as well as the data and parameters of the optimization problem to be solved. This part of the algorithm is insignificant in the sense of computational effort, and therefore, is performed by a single processing unit – the *master*. The master processing unit initializes the archive $\mathbb{A}$ of non-dominated decision vectors; the archive can consist of a single decision vector generated at random over the search space, or can be loaded from an external resource, e.g. non-dominated decision vectors found by other multi-objective optimization algorithm.

Further the computational effort is distributed among all processing units being used for the computations. Each of the units randomly selects a decision vector $\mathbf{x}$ from $\mathbb{A}$, generates its neighbors $\mathbf{x}'$ and $\mathbf{x}''$ following (2.1) and (2.3), and evaluates objective values of $\mathbf{x}'$. After the values of the objectives are evaluated, the dominance relation of $\mathbf{x}'$ with those stored in $\mathbb{A}$ is checked. During this procedure the access for updating the archive $\mathbb{A}$ is blocked for other processing units due to requirements for consistent memory access.

3.2 Distributed memory ParMOSS

In contrast with shared-memory parallel programming, processing units of a distributed-memory parallel computing system do not have a common memory. Therefore information about decision vectors in the archive $\mathbb{A}$ and values of parameters of the algorithm must be transfered by passing messages through

Message Passing Interface (MPI). In order to guarantee consistent communication between processing units one of them is devoted for the management of the communication and overall process of the algorithm. Thus the parallel version of MOSS algorithm ParMOSS/MPI for distributed-memory parallel computing systems is developed following the master-slave strategy, where the master processing unit is responsible for management and communication whereas the slaves evaluate objective values of decision vectors requested by the master.

The ParMOSS/MPI algorithm begins with an initial non-empty archive $\mathbb{A}$ of non-dominated decision vectors which is initialized by the master processing unit. The master selects a random decision vector $\mathbf{x}_i$ from $\mathbb{A}$, generates a pair of candidate solutions $(\mathbf{x}_i', \mathbf{x}_i'')$ following (2.1) and (2.3), and sends the generated pair to the i-th processing unit (a slave) with the request to evaluate the first decision vector $\mathbf{x}_i'$. Here i varies from 1 to the number of processing units p thus ensuring that a pair of decision vectors will be generated for each processing unit (the master processing unit normally is indexed by 0). After each slave i is equipped by a pair of decision vectors $(\mathbf{x}_i', \mathbf{x}_i'')$, the master processing unit proceeds to the main loop and waits for the response from any of the slaves with an evaluated decision vector.

Although all slaves are requested to evaluate $\mathbf{x}_i'$, some of them can also be requested to evaluate $\mathbf{x}_i''$ in the later stage of the algorithm. In general the master processing unit proceeds depending on whether evaluation of $\mathbf{x}_i'$ or $\mathbf{x}_i''$ is received and the fitness of the received decision vector with respect to decision vectors in the archive $\mathbb{A}$. The possible cases are the following:

(a) The evaluation of $\mathbf{x}_i'$ is received and $\mathbf{x}_i'$ is non-dominated in $\mathbb{A}$. Then $\mathbb{A}$ is updated by including $\mathbf{x}_i'$ and removing all decision vectors dominated by $\mathbf{x}_i'$. A new pair $(\mathbf{x}_i', \mathbf{x}_i'')$ is generated and sent to i-th processing unit with request to evaluate $\mathbf{x}_i'$.

(b) The evaluation of $\mathbf{x}_i'$ is received, but $\mathbf{x}_i'$ is dominated in $\mathbb{A}$. Then $\mathbf{x}_i'$ is rejected and i-th processing unit is requested to evaluate $\mathbf{x}_i''$.

(c) The evaluation of $\mathbf{x}_i''$ is received and $\mathbf{x}_i''$ is non-dominated in $\mathbb{A}$. Then $\mathbb{A}$ is updated by including $\mathbf{x}_i''$ and removing all decision vectors dominated by $\mathbf{x}_i''$. A new pair $(\mathbf{x}_i', \mathbf{x}_i'')$ is generated and sent to i-th processing unit with request to evaluate $\mathbf{x}_i'$.

(d) The evaluation of $\mathbf{x}_i''$ is received, but $\mathbf{x}_i''$ is dominated in $\mathbb{A}$. Then $\mathbf{x}_i''$ is rejected. A new pair $(\mathbf{x}_i', \mathbf{x}_i'')$ is generated and sent to i-th processing unit with request to evaluate $\mathbf{x}_i'$.

If the archive is supplemented either by $\mathbf{x}'$ or $\mathbf{x}''$ then the counter of repetitive successful iterations is increased by 1 and the counter of repetitive failed iterations is set to zero; otherwise the counter of repetitive failed iterations is increased by 1 and the counter of repetitive successful iterations is set to zero.

Such an iterative process is continued till the counter of function evaluations reaches $E-(p-1)$, where E is the maximum number of function evaluations, and p is the number of processing units. The check for stopping criterion is performed before sending request for evaluation of a decision vector to a slave.

It means one slave as well as the master do not have evaluated decision vector, but $p-2$ evaluations will be received from the remaining slaves. Therefore, if $E-(p-1)$ function evaluations is reached, the master starts sending requests for finalization (instead of request for evaluation of a decision vector) to the slaves.

The scheme of the ParMOSS/MPI algorithm for the master processing unit is presented in Figure 3. The request for evaluation of a decision vector or finalization of the algorithm is transfered through tags of the message: the tag value 1 means request for evaluation of decision vector $\mathbf{x}'_i$, the tag value 2 – request for evaluation of decision vector $\mathbf{x}''_i$, and the tag value 3 – request for the finalization.

All slave processing units work independently from each other and communicate with the master only. Each slave evaluates the requested decision vector and sends it together with values of the objective functions to the master. At the beginning of the algorithm all slaves are requested to evaluate $\mathbf{x}'_i$, though later some of the slaves can be requested to evaluate the second decision vector – $\mathbf{x}''_i$; i.e. if $\mathbf{x}'_i$ is dominated by other non-dominated solution found so far (see description of MOSS in Section 2). The slave processing unit is active till request for finalization is received from the master.

The scheme of the ParMOSS/MPI algorithm for the slave processing unit is presented in Figure 4. Whether evaluation of $\mathbf{x}'_i$ or $\mathbf{x}''_i$ is being sent is indicated by tag of the message.

Considering α and β ($\alpha+\beta=1$) as the ratio of time required for the function evaluations (α) and for other computational effort (β) in the sequential MOSS algorithm, the speed-up of the parallel ParMOSS algorithm on p processing units can be approximately evaluated by

$$\frac{p-1}{\alpha+\beta(p-1)} - \epsilon(p), \tag{3.1}$$

where $\epsilon(p)$ stands for the ratio of computational effort required for communication between p processing units. Expression (3.1) shows that increment of α and thus reduction of β, increases the speed-up of the algorithm and vice versa – reduction of α and thus increase of β, reduces the speed-up of the algorithm.

The master processing unit might be idle for some time, especially using a small number of slaves. In order to avoid that, a modified version of ParMOSS/MPI – the ParMOSS/MPIwM has been developed, where the master performs some function evaluations.

A single function evaluation requires to determine the most attractive facility for every demand point. That means that for CFLP with 5000 demand points we need to determine the most attractive facility for 5000 demand points. Therefore a single function evaluation can be separated into 5000 subtasks which is performed in a sequential fashion by a single processor. If the master processor detects that there are no messages from the slaves, it continues with performing subtasks of a function evaluation and checks for incoming messages after completion of each subtask. If the master detect an incoming message, it stops with function evaluation procedure and continues with process of the received decision vector in the same fashion as ParMOSS/MPI; after processing

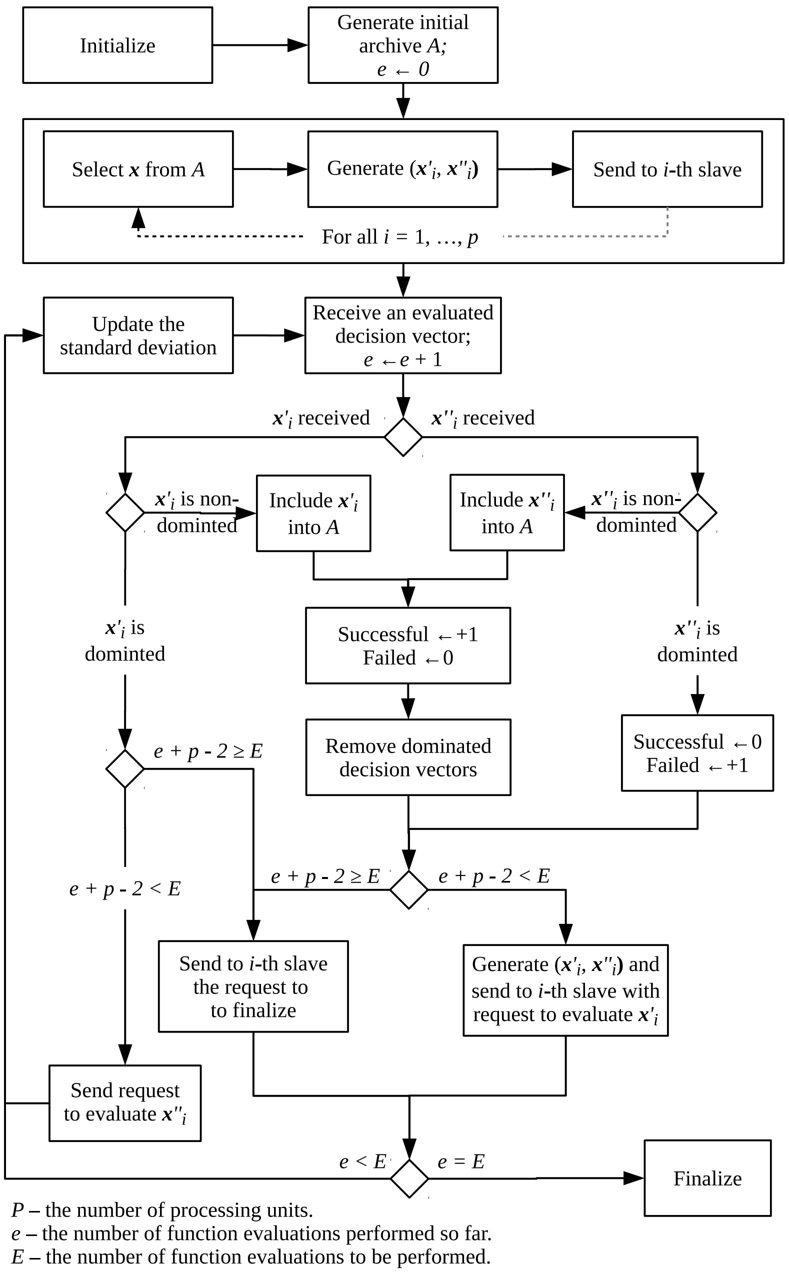
Initialize
Generate initial archive A; $e \leftarrow 0$
Select $\mathbf{x}$ from A
Generate $(\mathbf{x}'_i, \mathbf{x}''_i)$
Send to i-th slave
For all $i = 1, \ldots, p$
Update the standard deviation
Receive an evaluated decision vector; $e \leftarrow e + 1$
$\mathbf{x}'_i$ received
$\mathbf{x}''_i$ received
$\mathbf{x}'_i$ is non-dominted
Include $\mathbf{x}'_i$ into A
Include $\mathbf{x}''_i$ into A
$\mathbf{x}''_i$ is non-dominted
$\mathbf{x}'_i$ is dominted
Successful $\leftarrow$+1 Failed $\leftarrow$0
$\mathbf{x}''_i$ is dominted
$e + p - 2 \geq E$
Remove dominated decision vectors
Successful $\leftarrow$0 Failed $\leftarrow$+1
$e + p - 2 < E$
$e + p - 2 \geq E$
$e + p - 2 < E$
Send to i-th slave the request to to finalize
Generate $(\mathbf{x}'_i, \mathbf{x}''_i)$ and send to i-th slave with request to evaluate $\mathbf{x}'_i$
Send request to evaluate $\mathbf{x}''_i$
$e < E$
$e = E$
Finalize
P – the number of processing units.
e – the number of function evaluations performed so far.
E – the number of function evaluations to be performed.

Figure 3. Scheme of ParMOSS algorithm for the master processing unit.

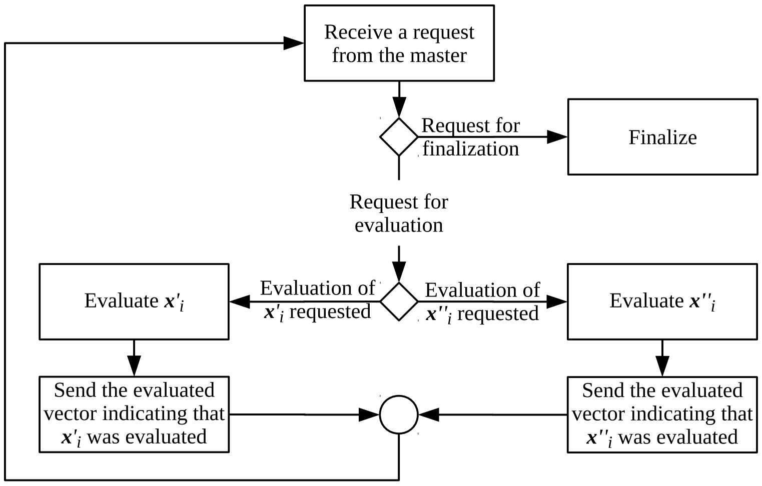
Receive a request from the master
Request for finalization
Finalize
Request for evaluation
Evaluation of x'_i requested
Evaluation of x''_i requested
Evaluate x'_i
Evaluate x''_i
Send the evaluated vector indicating that x'_i was evaluated
Send the evaluated vector indicating that x''_i was evaluated

Figure 4. Scheme of ParMOSS algorithm for the i-th slave processing unit, where $i = 1, 2, \ldots, p$.

the received decision vector, the master continues with the function evaluation if there are no messages from the slaves.

4 Numerical experiments

The proposed algorithm MOSS (Section 2) as well as its parallel versions ParMOSS/OMP (Section 3.1) and ParMOSS/MPI (Section 3.2) have been experimentally investigated by solving bi-objective CFLP/FE.

The real data, consisting of coordinates and population of 5000 cities and towns in Lithuania has been used in the investigation. It was assumed that the firms F_A and F_B have $n_A = n_B = 10$ preexisting facilities. The facilities of the firm F_B are located in the largest towns with populations from 543071 to 31142, whereas facilities of the firm F_A are located in the next 10 largest cities with populations from 29850 to 17244. The simplest model of behavior of the customers when choosing the most attractive facility has been considered, assuming that all customers from a single demand point choose the nearest facility. In case of equal distances to more than one facility, the demand is equally divided among all equidistant facilities.

Its was considered that the firm F_A wants to extend its market share by establishing 3 new facilities thus solving the bi-objective optimization problem with 6 variables (2 variables per facility); see Section 1.1 for details of the optimization problem. The Pareto front of the problem has been approximated by 25000 function evaluations. Due to stochastic nature of the algorithms, each experiment has been performed 100 times, using different randomly generated initial decision vector, and the average results have been computed.

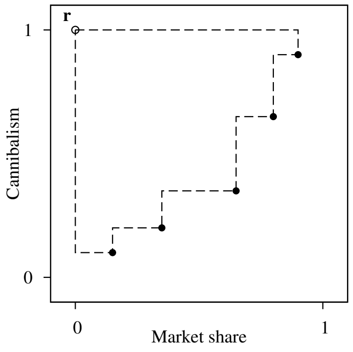

Figure 5. Illustration of the hyper-volume metric.

4.1 Metrics of precision and performance

The precision of the approximation of the Pareto front has been evaluated by the Hyper-Volume (HV) metric, proposed by Zitzler and Thiele in [25]. The HV measures the area captured by the points in the obtained Pareto front approximation and the given reference point **r**. The concept of the HV metric is illustrated in Figure 5, where the filled points stand for non-dominated decision vectors in the objectives space, hollow point – the reference point, and the dashed line marks the boundaries of the dominated area. The larger dominated area means better approximation of the Pareto front. The obtained approximation of the Pareto front has been scaled to the interval $[0, 1]^2$ with respect to the extreme values of the objectives – the maximal possible utility (the market share of the new facilities), which is equal to the total market share of both firms F_A and F_B, and the maximal possible cannibalism, which is equal to the market share of the firm $\mathbb{A}$. The reference point is then chosen to be $(0, 1)$.

The performance of the parallel algorithm has been evaluated by the speed-up $S_p = T_0/T_p$ of the algorithm, where T_0 stands for the time needed to solve the problem using the sequential algorithm and T_p stands for the time needed to solve the problem using a parallel algorithm on p processing units. Here $p = 1, 2, \ldots$ and T_1 might differ from T_0 as the behavior of a parallel algorithm on a single processing unit might differ from the behavior of a sequential algorithm.

4.2 Impact on the precision

Since MOSS has been derived from MOSASS by changing the strategy for selection of the decision vector from the set of non-dominated ones, the impact of the modification on the quality of the approximation must be investigated. It has been investigated by solving the CFLP with 5000 demand points using sequential versions of MOSASS and MOSS.

The obtained results are presented in Figure 6. One can see from the figure that the proposed MOSS algorithm notably outperforms its precursor MOSASS independent on the number of function evaluations devoted for the approximation.

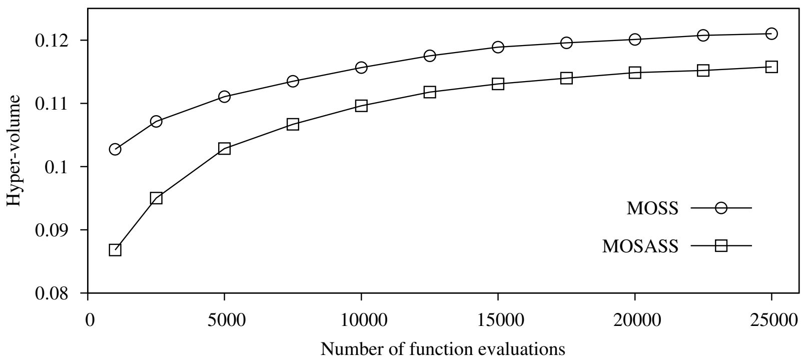

Figure 6. Values of hyper-volume, obtained using different algorithms and different numbers of function evaluations

The parallel algorithms ParMOSS/OMP and ParMOSS/MPI do not have exactly the same behavior as the sequential MOSS, therefore it must be verified if the quality of the approximation has not been reduced when parallelizing. The verification has been done by solving the CFLP of different scope: 1000 and 5000 demand points. The problems have been solved by the parallel algorithm ParMOSS/OMP using different numbers of processing units: 2, 4, 8, and 16. The obtained results have been additionally compared with the results obtained by the sequential version of the algorithms. The corresponding investigation has been performed for the distributed-memory parallel algorithm ParMOSS/MPI, using the same numbers of demand points, but different numbers of processing units: 32, 64, 128, 160, and 192. The obtained results are illustrated in Figures 7 and 8, where two first pairs of columns show average HV, obtained by sequential versions of MOSASS and MOSS, respectively, whereas the following ones show average HV, obtained by the ParMOSS/OMP (Figure 7) and ParMOSS/MPI (Figure 8) algorithms using different numbers of processing units; different columns in a pair represent different number of demand points.

One can see from the figures that the variation of values of HV, obtained using ParMOSS/OMP and ParMOSS/MPI on different numbers of processing units is very slight and can be considered as insignificant. This leads to the conclusion that the parallelization of MOSS does not negatively effect the precision of the approximation.

4.3 Speed-up of the parallel algorithms

The performance of the parallel algorithms ParMOSS/OMP and ParMOSS/MPI have been evaluated by solving the CFLP/FE using four different

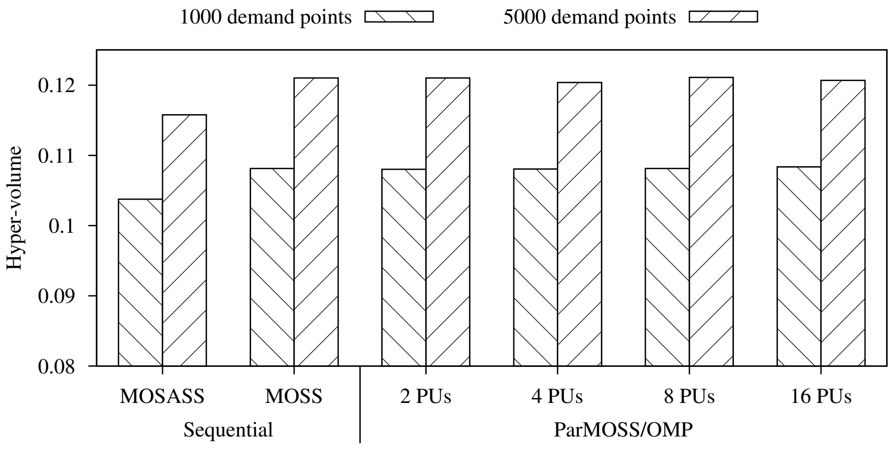

Figure 7. Average values of hyper-volume, obtained by sequential MOSASS and MOSS and parallel shared-memory MOSS (ParMOSS/OMP) using different numbers of processing units.

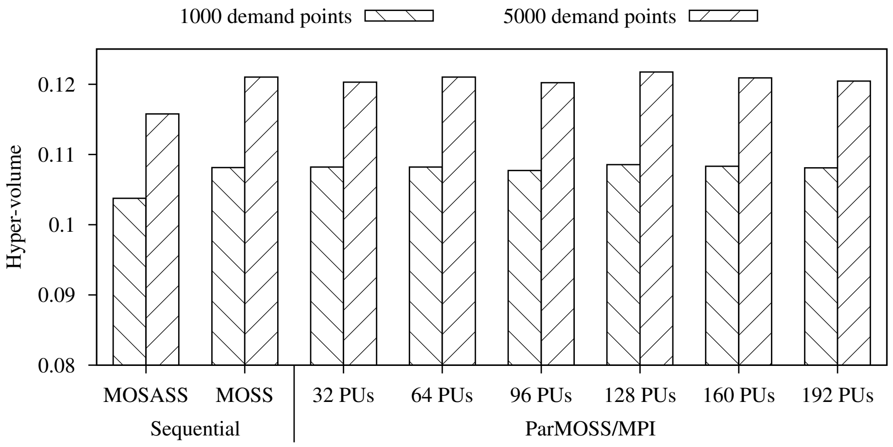

Figure 8. Average values of hyper-volume, obtained by sequential MOSASS and MOSS and parallel distributed-memory MOSS (ParMOSS/MPI) using different numbers of processing units.

numbers of demand points: 5000, 1000, 500, and 100 for ParMOSS/OMP; 5000 and 1000 – for ParMOSS/MPI. The Pareto front of each instance has been approximated by 25000 function evaluations. Each experiment has been run for 100 times and average duration has been evaluated. The average duration of a single approximation by sequential MOSS was around 728 seconds using 5000 demand points, around 145 seconds – using 1000 demand points, around 73 – using 500 demand points, and around 14 seconds – using 100 demand points.

The main effort of the computational resources (more than 99%) has been devoted to the function evaluation in all the cases, except the smallest one – 100 demand points, when function evaluations require a little bit less than 99% of all computational resources.

The time needed to approximate the Pareto front of the problem with 5000 demand points using MOSS was 0.3–0.4% larger comparing with its precursor MOSASS whereas ParMOSS/OMP on one processing unit requires 0.3–0.4% more time than sequential MOSS. Due to the nature of the ParMOSS/MPI algorithm it cannot be executed on a single processing unit – the smallest number of processing units is 2. The behavior of the ParMOSS/MPI algorithm using 2 processing units coincides with the behavior of MOSS, but requires around 0.5% more time due to additional cost for message passing.

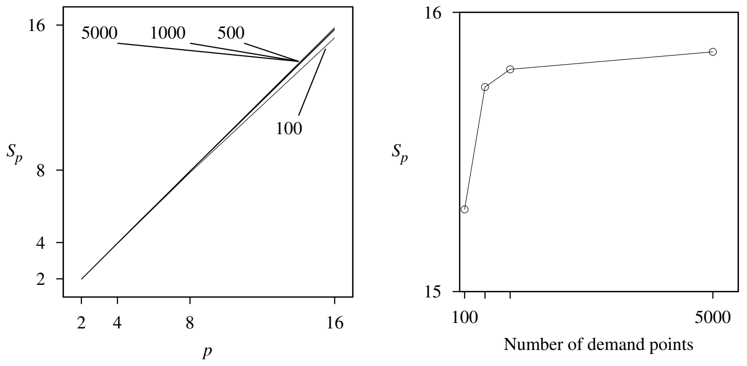

Figure 9. The speed-up of ParMOSS/OMP, obtained using different numbers of demand points, versus the number of processing units (on the left) and the speed-up, obtained using 16 processing units, versus the number of demand points (on the right).

The speed-up of ParMOSS/OMP, obtained using 2, 4, 8, and 16 shared-memory processing units is given in the left image of Figure 9, where the horizontal axis corresponds to the number of processing units, and the vertical one – to the speed-up of the algorithm. One can see from the figure, the speed-up of the algorithm is almost linear for all experiments except for the case of using 16 processing units to solve the smallest problem with 100 demand points.

The dependence of the speed-up of the algorithm, obtained using 16 processing units, on the number of demand points is illustrated in the right image of Figure 9, where the horizontal axis corresponds to the number of demand points, and the vertical axis – to the speed-up from the range from 15 to 16.

The obtained results show that ParMOSS/OMP has almost linear speed-up, since the further reduction of the demand points is not reasonable in practical CFLPs, and vice versa – further increment of the scope of the problem cannot reduce the speed-up, but rather increase, as the time required for function evaluations will be increased, thus increasing the fully parallel part of the algorithm.

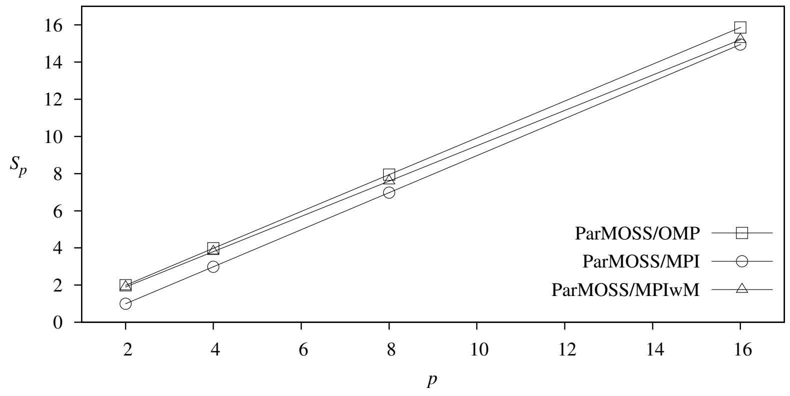

Figure 10. Comparison of speed-up of ParMOSS/MPI with speed-up of ParMOSS/OMP using different numbers of processing units.

Similar experiment has been performed using distributed-memory parallel algorithms ParMOSS/MPI and ParMOSS/MPIwM. The Pareto front of CFLP with 1000 demand points has been approximated by using 2, 4, 8, and 16 processing units. The obtained results are presented in Figure 10 with the context of results obtained by ParMOSS/OMP. One can see from the figure, that speed-up of ParMOSS/MPI increases linearly with increment of the number of processing units. The figure also shows that speed-up of ParMOSS/MPI is lower than speed-up of ParMOSS/OMP exactly by one independent on the number of processing units; this can be explained by an idle time of the master processing unit which has no computational work, but is responsible for the management of the computational process and communication. The speed-up of ParMOSS/MPIwM, where the master is employed for function evaluations, is better than speed-up of ParMOSS/MPIwM. On the other hand speed-up of both MPI algorithms became similar using 16 processing units.

These results show that the shared-memory algorithm has notable advantage against the distributed-memory algorithms – the speed-up of the shared-memory algorithm is almost linear on up to 16 processing units whereas the speed-up of the distributed-memory algorithms is around 15 when 16 processing units are used. On the other hand the shared-memory computing systems have hardware limitations in the sense of number of shared-memory processing units, whereas the distributed-memory algorithm can be executed on a significantly larger number of processing units.

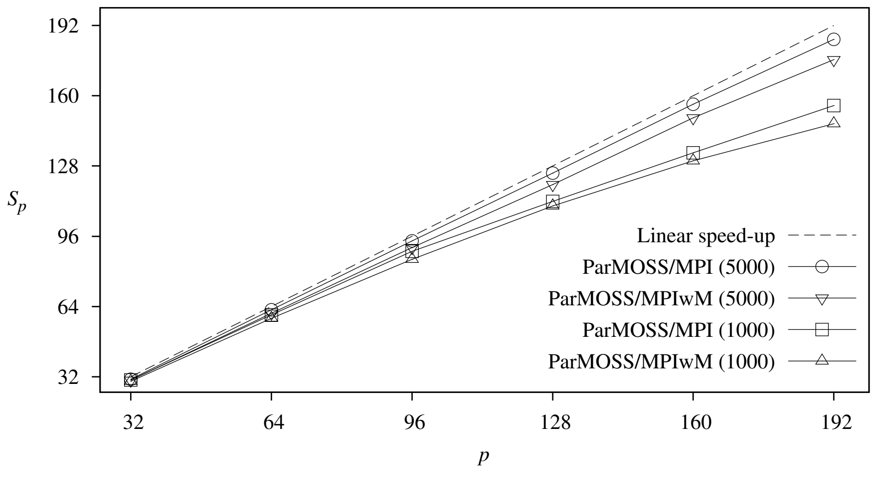

Figure 11. Speed-up of ParMOSS/MPI using different number of processing units and different number of demand points.

The performance of the proposed distributed-memory parallel algorithms has been investigated using a larger number of processing units – from 32 to 192. The Pareto front of two instances of the CFLP – with 5000 and 1000 demand points – has been approximated by 25000 function evaluations as in previous experiments. The average speed-up of the algorithm versus the number of processing units is presented in Figure 11.

One can see from the figure that the approximation of the Pareto front of the problem with 5000 demand points has been performed with almost linear speed-up of the algorithm – the speed-up on 192 processing units is around 186.

The approximation of the Pareto front of the problem with 1000 demand points has been performed with notably lower speed-up, comparing with previous instance – the speed-up on 192 processing units is around 155 which corresponds to 80% of effectiveness of the processing units when performing the computations. On the other hand the speed-up of the algorithm on 96 processing units is around 89 which corresponds to 93% of effectiveness of the processing units when performing the computations; further increment of the number of processing units is not reasonable for approximation of the Pareto front of a real-world CFLP as the approximation on 128 processing units has been performed within 2 seconds.

ParMOSS/MPIwM graphs in Figure 11 show that assignment of computational work for the master processing unit is not reasonable using more than 32 processing units as it causes a bottleneck effect. The master processing units performs 12686 function evaluations when 2 processing units are used in total (for the CFLP with 5000 demand points); this number corresponds to 98% of function evaluations performed by the slave. If the number of processing

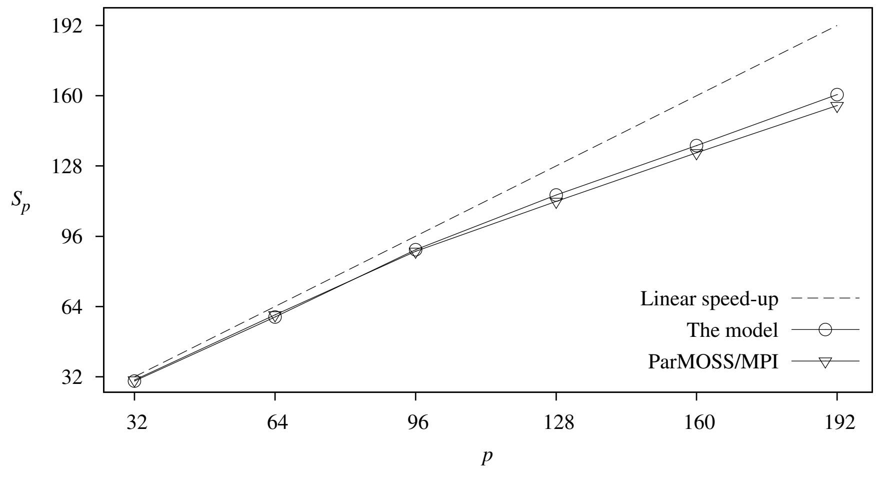

Figure 12. Comparison of the evaluated and the actual speed-up of ParMOSS/MPI on different number of processing units.

units is increased to 32, the master performs 538 function evaluations and it corresponds to 67% of function evaluations performed by a single slave. If 192 processing units are used, the master performs 15 function evaluations and it equals to 11% of function evaluations performed by a slave. These results lead to a conclusion that the master processing unit is busy enough by management of computational work for the slaves and assignment of additional computational work can raise the bottleneck effect in communication between large number of processing units.

The results obtained by experimental investigation approximately coincides with those evaluated by the model 3.1 with $\alpha = 0.999$ (for the CFLP with 1000 demand points). Such a ratio of the time required for the function evaluations in the sequential algorithm has been experimentally determined by solving the CFLP by the MOSS algorithm.

The comparison of the evaluated and the actual speed-up of ParMOSS/MPI when solving CFLP with 1000 demand points is presented in Figure 12. One can see from the figure, that the graph of the evaluation is slightly above the graph representing the actual speed-up of the algorithm. The difference is due to the ratio of time required for communication between processors, which was considered as unknown parameter in the model.

5 Conclusions

The shared- and distributed-memory parallel algorithms for multi-objective optimization have been developed and experimentally investigated by solving the CFLP/FE using up to 16 shared-memory and up to 192 distributed-memory processing units.

The obtained results showed that the modifications made to the sequential multi-objective single agent stochastic search algorithm, improve the precision of the approximation of the Pareto front, measured by the hyper-volume metric, and make the algorithm more suitable for parallel computing environment.

The performance results of the parallel algorithm showed that the shared-memory parallel algorithm has linear speed-up on up to 16 processing units. The distributed-memory parallel algorithm has almost linear speed on up to 16 processing units when the master processing unit has workload for function evaluations. Further increment of processing units require to leave the master processing unit without function evaluations in order to avoid a bottleneck effect.

In general the developed distributed-memory parallel algorithm has close to linear speed-up using 192 processing units to solve competitive CFLP/FE with 5000 demand points – the speed-up is 176.

Acknowledgments

This research was funded by a grant (No. MIP-051/2014) from the Research Council of Lithuania.

References

[1] A. Abraham and L. Jain. Evolutionary multiobjective optimization. In A. Abraham, L. Jain and R. Goldberg(Eds.), *Evolutionary Multiobjective Optimization*, Advanced Information and Knowledge Processing, pp. 1–6. Springer London, 2005. http://dx.doi.org/10.1007/1-84628-137-7_1.

[2] G.E. Blelloch and K. Tangwongsan. Parallel approximation algorithms for facility-location problems. In *Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA'10, pp. 315–324, New York, NY, USA, 2010. ACM. http://dx.doi.org/10.1145/1810479.1810535.

[3] E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, **10**(2):141–171, 1998.

[4] A. de Silva and D. Abramson. A parallel interior point method and its application to facility location problems. *Computational Optimization and Applications*, **9**(3):249–273, 1998. http://dx.doi.org/10.1023/A:1018302308154.

[5] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, **6**(2):182–197, 2002. http://dx.doi.org/10.1109/4235.996017.

[6] Z. Drezner, K. Klamroth, A. Schöbel and G.O. Wesolowsky. The Weber problem. In Z. Drezner and H.W. Hamacher(Eds.), *Facility Location: Applications and Theory*, pp. 1–36. Springer Berlin Heidelberg, 2001.

[7] J.J. Durillo, A.J. Nebro, F. Luna and E. Alba. A study of master-slave approaches to parallelize NSGA-II. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pp. 1–8, 2008. http://dx.doi.org/10.1109/IPDPS.2008.4536375.

[8] R.Z. Farahani, S. Rezapour, T. Drezner and S. Fallah. Competitive supply chain network design: An overview of classifications, models, solution techniques and applications. *Omega*, **45**:92–118, 2014. http://dx.doi.org/10.1016/j.omega.2013.08.006.

[9] J. Fernández, B. Pelegrín, F. Plastria and B. Tóth. Planar location and design of a new facility with inner and outer competition: An interval lexicographical-like solution procedure. *Networks and Spatial Economics*, **7**(1):19–44, 2007. http://dx.doi.org/10.1007/s11067-006-9005-4.

[10] T.L. Friesz, T. Miller and R.L. Tobin. Competitive networks facility location models: a survey. *Papers in Regional Science*, **65**(1):47–57, 1998. http://dx.doi.org/10.1111/j.1435-5597.1988.tb01157.x.

[11] L. Huapu and W. Jifeng. Study on the location of distribution centers: A bi-level multi-objective approach. In *Logistics*, chapter 447, pp. 3038–3043. American Society of Civil Engineers, 2009. http://dx.doi.org/10.1061/40996(330)448.

[12] V. Kumar. *Introduction to Parallel Computing.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2002.

[13] A. Lančinskas, P.M. Ortigosa and J. Žilinskas. Multi-objective single agent stochastic search in non-dominated sorting genetic algorithm. *Nonlinear Analysis: Modelling and Control*, **18**(3):293–313, 2013.

[14] A. Lančinskas and J. Žilinskas. Approaches to parallelize Pareto ranking in NSGA-II algorithm. In R. Wyrzykowski, J. Dongarra, K. Karczewski and J. Waśniewski(Eds.), *Parallel Processing and Applied Mathematics*, volume 7204 of *Lecture Notes in Computer Science*, pp. 371–380. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-31499-5. http://dx.doi.org/10.1007/978-3-642-31500-8_38.

[15] A. Lančinskas and J. Žilinskas. Solution of multi-objective competitive facility location problems using parallel NSGA-II on large scale computing systems. In P. Manninen and P. Öster(Eds.), *Applied Parallel and Scientific Computing*, volume 7782 of *Lecture Notes in Computer Science*, pp. 422–433. Springer Berlin Heidelberg, 2013. http://dx.doi.org/10.1007/978-3-642-36803-5_31.

[16] A. Lančinskas and J. Žilinskas. Parallel multi-objective memetic algorithm for competitive facility location. In R. Wyrzykowski, J. Dongarra, K. Karczewski and J. Waśniewski(Eds.), *Parallel Processing and Applied Mathematics*, volume 8385 of *Lecture Notes in Computer Science*, pp. 354–363. Springer Berlin Heidelberg, 2014. http://dx.doi.org/10.1007/978-3-642-55195-6_33.

[17] A.L. Medaglia, J.G. Villegas and D.M. Rodríguez-Coca. Hybrid bi-objective evolutionary algorithms for the design of a hospital waste management network. *Journal of Heuristics*, **15**(2):153–176, 2009. http://dx.doi.org/10.1007/s10732-008-9070-6.

[18] F. Plastria. Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research*, **129**(3):461–470, 2001. http://dx.doi.org/10.1016/S0377-2217(00)00169-7.

[19] J.L. Redondo, J. Fernández, J.D. Álvarez, A.G. Arrondo and P.M. Ortigosa. Approximating the Pareto-front of continuous bi-objective problems: Application to a competitive facility location problem. In J. Casillas, F.J. Martínez-López and J.M. Corchado Rodríguez(Eds.), *Management Intelligent Systems*, volume 171 of *Advances in Intelligent Systems and Computing*, pp. 207–216. Springer Berlin Heidelberg, 2012. http://dx.doi.org/10.1007/978-3-642-30864-2_20.

[20] C.S. ReVelle, H.A. Eiselt and M.S. Daskin. A bibliography for some fundamental problem categories in discrete location science. *European Journal of Operational Research*, **184**(3):817–848, 2008. http://dx.doi.org/10.1016/j.ejor.2006.12.044.

[21] V. Starikovičius, R. Čiegis and O. Iliev. A parallel solver for the design of oil filters. *Mathematical Modelling and Analysis*, **16**(2):326–341, 2011. http://dx.doi.org/10.3846/13926292.2011.582591.

[22] J.G. Villegas, F. Palacios and A.L. Medaglia. Solution methods for the bi-objective (cost-coverage) unconstrained facility location problem with an illustrative example. *Annals of Operations Research*, **147**:109–141, 2006. http://dx.doi.org/10.1007/s10479-006-0061-4.

[23] A. Weber. *Theory of the Location of Industries.* Materials for the study of business. University of Chicago Press, 1929.

[24] E. Zitzler, M. Laumanns and L. Thiele. SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou and T. Fogarty(Eds.), *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems: Proceedings of the EUROGEN2001 Conference, Athens, Greece, September 19-21, 2001*, pp. 95–100, 2001.

[25] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms – A comparative case study. In A.E. Eiben, T. Bäck, M. Schoenauer and H.-P. Schwefel(Eds.), *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, volume 1498 of *PPSN V*, pp. 292–304. Springer-Verlag, London, UK, 1998.