A Two-Level Parallel Genetic Algorithm for the Uncapacitated Warehouse Location Problem

Jörg Homberger, Stuttgart University of Applied Sciences
joerg.homberger@hft-stuttgart.de

Hermann Gehring, University of Hagen
hermann.gehring@fernuni-hagen.de

Abstract

A new Genetic Algorithm (GA) for the Uncapacitated Warehouse Location Problem (UWLP) and its parallelization are described. The parallel method is based on two ideas. (1) The GA is using a new integer coding for the UWLP. (2) The parallelization takes advantage of a developed two-level strategy. The first level of parallelization consists of executing several subpopulations of the GA concurrently with the occasional migration of individuals between them. On the second level, the solution space is separated into several disjunctive parts. The developed method, which is the first application of a parallel metaheuristic to the UWLP, is evaluated using a large set of 717 benchmark problems available from the literature, whereas the other known solution methods are always applied to subsets of these instances. The results show, that the parallel method is competitive with the best known solution methods so far.

1. Introduction and problem formulation

The Uncapacitated Warehouse Location Problem (UWLP) belongs to the class of facility location problems. It can be described as follows ([20]): The UWLP considers a set of n warehouses W and a set of m stores S. Each warehouse has a fixed cost f_w, and the transportation cost from warehouse w to store s is given by c_{ws}. The problem is to find a subset *Open*, $Open \subseteq W$, of warehouses and an assignment of warehouses to stores in order to minimize the total costs C, as the sum of fixed and transportation costs:

$$C(Open) = \sum_{w \in Open} f_w + \sum_{s \in Stores} \min_{w \in Open} c_{ws}.$$

In the following, the number $|Open|$ of chosen warehouses is also referred to as k.

Since the UWLP belongs to the class of *NP*-hard combinatorial optimization problems [27], metaheuristics are recommended for calculating near optimal solutions for larger problem instances in reasonable computing times ([22]). In recent years, quite good results have been achieved for the UWLP with metaheuristics such as Tabu Search ([3], [30], [36], [37]), Simulated Annealing ([4], [5], [38]), Grasp ([31], [32]) and Genetic Algorithms ([24], [28]).

The solution concept underlying most metaheuristics for the UWLP is to vary decisions regarding the number k of warehouses as well as the selection of warehouses at the same time. Within another concept, called the multistart approach, different solutions are calculated independently of each other for different values of k. In each of the calculation runs, the given number of stores is kept constant. The calculation of a selection of k stores is carried out by means of a metaheuristic, e.g. a Tabu Search or a Variable Neighborhood Search ([33], [34], [20]).

GAs are powerful search techniques for optimization and learning problems ([23], [15], [18]). They manipulate a population of individuals that represent alternative solutions for a given problem, based on principles of natural selection and genetics. To improve the efficiency and effectiveness of GAs, different parallelization strategies or types of Parallel GAs (PGA) can be applied, respectively ([10], [13]): the master-slave PGA, the cellular PGA, and the multiple-deme coarse-grained PGA. Successful applications of these three types of PGA can be found in [14].

In this paper a PGA for the UWLP is presented. The method is based on a two-level parallelization strategy, which combines a master-slave strategy with a multiple-deme strategy. The rest of the paper is structured as follows. A new sequential GA for the UWLP is proposed in Section 2. The method is parallelized in Section 3 using a two-level parallelization strategy. In Section 4, the performance of the approach is demonstrated by means of a comparative test including methods from other authors as well. Finally, Section 5 contains some conclusions.

1530-1605/08 $25.00 © 2008 IEEE

2. A new sequential GA for the UWLP

2.1. ($\mu+\lambda$)-Population concept

The developed GA is based on the ($\mu+\lambda$)-Population concept, which is often used within Evolution Strategies ([1], [8], [35]). Starting from the initial population, a sequence of populations is now calculated iteratively. The population size is denoted as μ. In each iteration *iter* a number of λ offspring are generated from the current population $P(iter)$. Three steps are carried out in each case to calculate one offspring: (1) Two individuals, called parents, are selected for reproduction from the current population $P(iter)$. The selection of parents is unbiased, i.e., every individual of the current population has the same probability to be selected for reproduction. (2) Through recombination of the selected parents one or more offsprings are calculated and then subjected to a mutation. (3) The new individuals are evaluated. At the end of an iteration the μ best individuals are selected from the joined set of λ offspring and μ individuals of population $P(iter)$ to form the population $P(iter+1)$. By means of the coefficient μ/λ the global character of the search can be influenced. A high value for this quotient leads to a low selection pressure.

The developed algorithm stops as soon as no improvement of the best solution can be reached in η iterations.

2.2. Coding and calculating a start population

In the GA of [28] a binary coding, providing a bit for each possible warehouse, is used. Using genetic search operators, both the number and the concrete selection of bits and thus warehouses can be varied. In the approach used in this paper, it is assumed in contrast that the number k of warehouses to be opened is given to the GA as a parameter and left constant during the search. The aim of the evolutionary search is to calculate a cost-efficient selection of exactly k warehouses. The developed GA is consecutively denoted as GA(k). Each individual $I = \{i_1,\ldots, i_k\}$ of the start population represents a set of k selected, different warehouses i_z, $i_z \in Open$, $z = 1,\ldots, k$. For two selected warehouses i_z, $i_y \in I$ it applies that $i_z \neq i_y$.

At the beginning of the search, μ individuals are generated at random, i.e., for each individual a random selection of k different warehouses is calculated.

2.3. Genetic search operators

From two selected parents, $I_1 = \{i_1,\ldots, i_k\}$ and $I_2 = \{j_1,\ldots, j_k\}$ of the current population $P(iter)$, exactly $2\cdot k$ offspring are calculated by recombination. Therefore, each of the two parents is crossed through a k-multiple application of the one-point crossover with the other parent (respectively). The operator is described in Figure 1.

For (a = 1 to k)
  Offspring O_a is initialized: $O_a := I_1$;
  For (b = a to k)
    In Offspring O_a replace the warehouse i_b by the warehouse j_b: $i_b := j_b$;

Figure 1. Crossover

Every generated offspring O produced by crossover is mutated. Thus, one randomly selected warehouse i_z, $i_z \in O$, is replaced by a randomly selected warehouse v, $v \in W$, and $v \notin O$.

After the mutation, every generated offspring is tested with regard to reliability and repaired if necessary. A repairing is necessary if warehouses appear more than once in the generated offspring. Within the reparation, warehouses occurring repeatedly are exchanged by randomly chosen warehouses.

3. A two-level parallel search strategy

The sequential GA(k) described in Section 2 is now parallelized combining two of the well-known parallelization strategies for GAs. The resulting strategy is described by Figure 2.

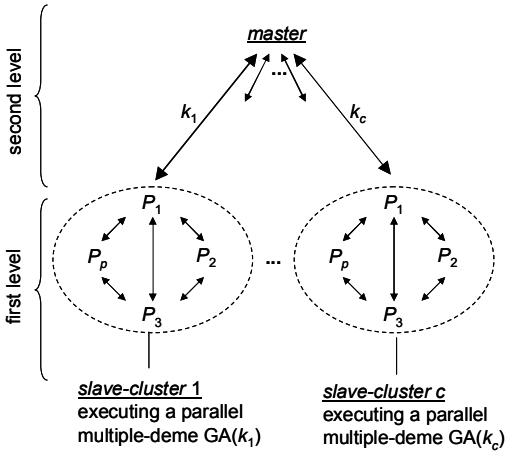

Figure 2. Two-level parallelization

On the first level, a multiple-deme strategy is used to separate the population of the GA(k) into p demes involved in parallel, P_i, $i = 1,\ldots, p$, which are assigned

to different processors. The demes have the same size of μ/p individuals.

By the use of several demes, alternative search paths can be tracked. To further increase the solution quality, the search processes cooperate by exchanging individuals between demes according to the migration model used by [11].

The model is controlled by the migration rate, the migration interval, the policy of selection, the replacement of individuals, and the topology of the connections between demes. The migration rate *mr*, $mr < \mu$, represents the number of individuals in a subpopulation who migrate each time there is a migration. The migration interval determines the frequency of migrations. Here, migration is triggered at fixed intervals of length *mi*. There are two alternatives for selecting the individuals that emigrate from a subpopulation: select randomly or select the best individuals. Here, the *mr* best individuals are selected to migrate. There are two options for replacing *mr* existing individuals in the receiving subpopulation with the incoming migrants: select randomly or replace the worst individuals. Here, the worst *mr* individuals are replaced by received individuals. The chosen policy – selecting the best individuals to migrate and replacing the worst ones in the receiving subpopulation – may cause the algorithm to converge faster ([11]). One more significant aspect of multiple-deme GAs is the migration topology describing which subpopulations send individuals to which other subpopulations. Here, all demes are connected with each other (i.e., in each migration, the selected migrants are sent to random destinations).

On the second (higher) level, a master-slave strategy is used to separate the search space into *c* several parts and to start a slave-cluster, each executed on *p* processors, for each part. The decomposition of the search space follows the multistart approach from [34], i.e., the separation of the search space in *c* disjunctive parts is based on the number *k* of warehouses to be opened. The determination of *c* alternative values for *k*, in the following denoted by $k_1,\ldots, k_c$, is done by a special process, the so called master.

The values for *k* are determined in one or more master iterations *miter*, $miter \geq 0$, depending on the number of processors available. In the following, it is assumed that exactly $c \times p$ ($c, p \in$ IN) processors or computers are provided.

In order to describe the chronological succession of the parallelization process in an unequivocal way, the interplay between the master and the slaves is visualized by Figure 3.

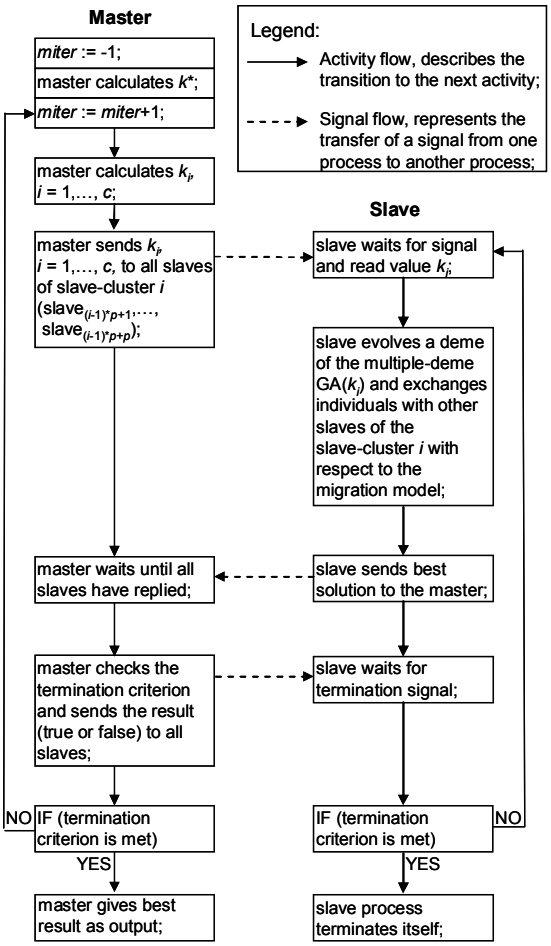

Figure 3. Flow of signals and activities

In each master iteration *miter*, the following steps are carried out:

(1) Via the number *k* of warehouses to be selected, the master determines exactly *c* different parts of the solution space; i.e., the master determines exactly *c* different values $k = k_1,\ldots, k_c$. For each slave-cluster *i*, $i = 1,\ldots, c$, a value for the number k_i, $k_i = k^* - (miter \times c + i) + 1$ of warehouses to be opened is assigned by the master. k^* is an upper approximation for the optimal number of warehouses. The value for k^* is calculated by the master within the first iteration ($miter = 0$). Therefore, a so-called start heuristic is used. The start heuristic is based on a bitstring representation of the UWLP ([28], [36]). A solution of UWLP is coded by a bitstring *B* of the length *n*. The bit $B_i = 1$ means that the warehouse *i* is chosen, whereas $B_i = 0$ means that the warehouse *i* is not chosen. At the beginning of the start heuristic, a random start solution for the UWLP is calculated. Subsequently, a simple local search procedure is performed to improve the start solution. The resulting start solution is used to calculate k^*:

$$k^* := \sum_{i=1}^{n} B_i \ .$$

(2) The master sends the value k_i to the slave-cluster i, $i = 1,..., c$, (Figure 2).

(3) The slave-cluster i, $i = 1,..., c$, executes the multiple-deme GA(k_i) and gives the best calculated solution back to the master.

(4) The master decides either to start a further master iteration *miter* or to terminate the search. If the solution value found by slave-cluster c in step (3) is the best over all slave-clusters, a further master iteration is started (*miter* := *miter* +1); i.e., the master calculates new values for k_i and starts the slave-clusters again. In each iteration, the slave-clusters are started concurrently. In this way, the determined parts of the solution space are searched in parallel and the computing time is reduced.

The resulting two-level parallel GA is referred to as Multistart Multiple-Deme GA (MMGA) in the following.

4. Computational results

4.1. Experimental setup

In order to demonstrate the performance of the developed method MMGA, a comparative test for different problem sizes has been carried out. In the following, the test problems used, the implementation environment, and the configuration of the developed method are described.

The test covers 13 classes of benchmark instances for the UWLP given in Table 1. It contains the number NOI of test instances per class and the number OPT of instances of which optimal solutions are known. Detailed descriptions of each class are given in [22], [17], and [34]. All instances are available at the UflLib ([21]), with the exception of those in class GHOSH and M**. The GHOSH instances can be generated by the instance generator provided by [17].

The M* instances were generated by [28] using a problem generator. In [34] a different class of instances were generated by the problem generator of [28] and also designed as M*. In no publication both variants of the generated M* instances are taken into consideration. In this paper, both classes are used. To differ these classes, the class generated by [34] is denoted by M** in the following.

For nine classes all optimal solutions are known. Note that the instances of the classes GAP and FPP could be solved optimally, although they were generated with the intent to be a challenge for local search ([25], [26], [20], [34]). For the other classes (GHOSH, M*, M**, MED), some of the optimal solutions are unknown. Thus, the upper bounds are used for comparison. These are the best calculated solution values so far, obtained by various metaheuristics, but have not been proved to be optimal.

Table 1. Problem classes

Class	Reference	NOI	OPT
Bilde-Krarup (BK)	[9], [21]	220	220
Finite Projective Planes (FPP)	[26], [21]	60	60
Large duality gap (GAP)	[26], [21]	90	90
Koerkel-Ghosh (GHOSH)	[17]	90	---
Galvão-Raggi (GR)	[16], [21]	50	50
M*	[28], [21]	30	20
M**	[28], [34]	22	15
K-median (MED)	[6], [21]	18	---
OR-Library (ORLIB)	[7]	15	15
Chessboard (CHESS)	[26], [21]	30	30
Perfect Codes (PCODE)	[26], [21]	32	32
UNIFORM	[25], [21]	30	30
Euclidean (EUCLID)	[26]	30	30
Total		717	592

The developed method MMGA was implemented in C. The calculations were carried out using a PC-LAN under a Linux server. Each calculation run (i.e., execution of GA(k) for a fixed number k of open warehouses), was carried out on a Dual-Core CPU (2.4 GHz, 2 GB RAM) operating under Red Hat Enterprise Linux Version 4 Update 4. The used PC-LAN comprised exactly 50 PCs. The communication between the programs took place via the PC-LAN's file system.

The MMGA was configured as follows: $c = 10$ multiple-deme GA(k) methods were started concurrently in each master iteration. For each multiple-deme GA(k) the following parameter values were chosen: $\mu = 100$, $\lambda = 10 \times 2 \times k$, $p = 5$, $mi = 5$, $mr = 1$, and $\eta = 1000$. The parameters were calculated in a parameter study. In this, 20 selected instances from the classes FPP and GAP with different parameter values were solved. The chosen values for the parameters *mi*

and *mr* follow the recommendations given in the literature. The values of *p* and *c* were chosen in dependency of the given IT infrastructure (note that $p \times c = 50$ PCs were available).

4.2. Results

In analogy to [34], each instance was solved exactly ten times independently by using the MMGA method. A new random seed was used in each run. The numerical results obtained from the test are given in Table 2.

Table 2. Average deviations A%D

Class	[34]	[20]	[19]	MMGA
BK	0.002	---	---	**0.000**
FPP	**0.000**	0.0003	---	0.014
GAP	0.820	**0.0141**	---	0.184
GHOSH	**-0.039**	---	---	0.070
GR	**0.000**	---	---	**0.000**
M*	**---**	**---**	**0.00**	**0.000**
M**	**0.000**	---	**---**	**0.000**
MED	**-0.391**	---	---	0.152
ORLIB	**0.000**	**---**	**0.00**	**0.000**
CHESS	---	---	---	**0.015**
PCODE	---	---	---	**0.006**
UNIFORM	---	---	0.03	**0.004**
EUCLID	---	---	0.03	**0.000**

The results achieved by the best known solution methods ([34], [19], and the Multistart Variable Neighborhood Algorithm described in [20]) as well as calculated by the MMGA are presented as average percentage deviation A%D of the calculated values from published reference solution values. In the case of the classes BK, FPP, GAP, GHOSH, GR, M**, MED, and ORLIB the reference solution values given in [34] are used. For M* the reference solution values (upper bounds) of [19] are taken into consideration. For CHESS, PCODE, UNIFORM, and EUCLID the reference values (lower bounds) found in [21] are used. Results obtained by other methods for the five classes BK, GR, M*, MED, and ORLIB can be found in Hoefer's comparative analysis ([22]).

All in all the following conclusions can be drawn:

MMGA is the only metaheuristic which was tested on the basis of the 717 benchmark instances.

None of the methods dominates the other methods. The best mean values could be calculated for BK with MMGA, for FPP, GHOSH, and MED with [34], and for GAP with [20]. However, it should be remarked that the method of [34] shows slight advantages, if all calculated instances are taken into consideration. Especially for the GHOSH and the MED instances, [34] calculates new best solutions (indicated by negative values in Table 2). For these instances, MMGA obtains reasonably good values, too. The average percentage deviation of the solution values obtained by MMGA from the reference solutions amount only to 0.07% (GHOSH) and 0.152% (MED). For the UNIFORM and EUCLID instances, MMGA leads to better values than [19].

The MMGA method finds optimal solutions for all instances of the classes BK, GR, ORLIB, CHESS, PCODE, and EUCLID. For M* and M** all known optimal solutions were found. For the subclasses GAPA, GAPB, GAPC, FPP11, and FPP17, a number of 21, 11, 4, 15, and 7 instances could be solved to optimality.

Following [30], the solution quality of MMGA is now considered in more detail. For the ORLIB and the M* instances, the best values (BEST), the mean values (AVG), and the standard deviations (DEV) obtained by MMGA in calculation series over 10 runs per instance, are presented in Table 3. To enable a comparison, the respective values reported by [30] are also included. These values are, however, derived in series over 100 runs per instance. From Table 3, it can be depicted that the standard deviation for MMGA are relatively low (less than 0.01%). Note that all best solutions obtained by MMGA (BEST) are equal to the best or optimal solutions reported in the literature.

In Table 4, the average computing times ACT (in seconds) of MMGA are compared with the average computing times reported in the literature.

MMGA needs only relatively short computing times. The short computing times are, however, caused by the large number of used computers (50 PCs). If the calculation times of MMGA as given in Table 4 are multiplied by the number of employed PCs then the calculation effort for MMGA is significantly higher than for the methods [34], [20], [19]. Note that a comparison of the computing times should be made with caution because of the different test environments. In [34], a SGI Challenge with a number of 28 196-MHz MIPS R10000 processors is used; each run was limited to a single processor. The algorithm was implemented in C++. In [20], a 2.1 GHz Pentium processor is employed and the algorithm was implemented in Comet. In [19], a 1 GHz Pentium III processor is used, while the algorithm was coded in Pascal.

The numerical results motivate use of the MMGA method as an alternative approach to the procedures introduced in the literature. Above all, the two-level parallelization strategy makes it possible to handle the instances in a more efficient way.

Table 3. Average deviations A%D

INST	BEST	AVG	DEV (%)
cap71	932615.75	932615.75	0.0000
cap72	977799.40	977799.40	0.0000
cap73	1010641.45	1010641.45	0.0000
cap74	1034976.97	1034976.97	0.0000
cap101	796648.44	796648.44	0.0000
cap102	854704.20	854704.20	0.0000
cap103	893782.11	893782.11	0.0000
cap104	928941.75	928941.75	0.0000
cap131	793439.56	793439.56	0.0000
cap132	851495.32	851495.32	0.0000
cap133	893076.71	893288.33	0.0004
cap134	928941.75	928941.75	0.0000
capa	17156454.48	17156454.48	0.0000
capb	12979071.58	12979071.58	0.0000
capc	11505594.33	11505594.33	0.0000
MO1	1156.91	1158.02	0.0009
MO2	1227.67	1227.67	0.0000
MO3	1286.37	1290.28	0.0035
MO4	1177.88	1177.88	0.0000
MO5	1147.60	1147.60	0.0000
MP1	2460.10	2460.10	0.0000
MP2	2419.32	2422.02	0.0024
MP3	2498.15	2503.17	0.0021
MP4	2633.56	2637.45	0.0047
MP5	2290.16	2290.16	0.0000
MQ1	3591.27	3593.21	0.0017
MQ2	3543.66	3543.66	0.0000
MQ3	3476.81	3481.14	0.0026
MQ4	3742,47	3744.49	0.0009
MQ5	3751.33	3755.62	0.0018
MR1	2349.86	2358.40	0.0038
MR2	2344.76	2349.44	0.0020
MR3	2183.24	2190.81	0.0033
MR4	2433.11	2434.04	0.0006
MR5	2344.35	2348.90	0.0025
MS1	4378.63	4388.02	0.0026
MS2	4658.35	4673.48	0.0060
MS3	4659.16	4684.43	0.0047
MS4	4536.00	4560.20	0.0024
MS5	4888.91	4899.78	0.0017
MT1	9176.51	9245.59	0.0076
MT2	9618.85	9671.54	0.0025
MT3	8781.11	8827.33	0.0041
MT4	9225.49	9274.38	0.0031
MT5	9540.67	9668.83	0.0087

Table 4. Average computing times ACT

Class	[34]	[20]	[19]	MMGA
BK	0.28	---	---	0.1
FPP	330.88	12.12	---	2.3
GAP	92.09	2.40	---	0.9
GHOSH	34.31	---	---	30.2
GR	0.32	---	---	0.4
M*	---	---	19.13	3.7
M**	7.86	---	---	3.3
MED	369.67	---	---	354.1
ORLIB	0.17	---	1.28	0.4
CHESS	---	---	---	1.3
PCODE	---	---	---	0.6
UNIFORM	---	---	2.81	0.5
EUCLID	---	---	3.77	0.9

4.3. Parallelization effects

The MMGA method is executed on $p \times c$ computers (processors) of a PC-LAN. However, it is also possible to run a sequential version of MMGA on a single computer. In the sequential version c processes, of which each evolves a panmictic GA with a population size of μ individuals, are executed one after the other. In the following, the influence of the parallelization on the solution quality and on the computing times is analyzed on the basis of a comparison of the results derived with the parallel and the sequential version of MMGA.

As to the solution quality, five subclasses of instances of high complexity are considered. These subclasses are FPP11, FPP17, GAPA, GAPB, and GAPC. On each of the respective instances, the parallel and the sequential variants of the algorithm were applied exactly once. In comparison to the sequential variant GA, the mean solution quality obtained by MMGA was 1.1%, 0.7%, 1.8%, 0.9%, and 0.5% better. To statistically prove the higher solution quality of the MMGA for the considered subclasses, the Wilcoxon Signed-Rank Test was carried out. Accordingly, the null hypothesis that both variants lead to mean results of identical quality was tested against the alternative that the MMGA obtains on average higher values of the objective function, i.e., worse solutions. The null hypothesis could be rejected in favour of the alternative at a level of 0.01. It seems therefore to be evident that the high solution quality of the MMGA is to some extent caused by the parallelization.

As to the computing times, the comparison of sequential and parallel versions of algorithms is

usually based on a speed-up analysis. Speed-up $s_{p\times c}$ computes the ratio between the mean computing time $\bar{t}$ of an algorithm on an uniprocessor and the mean execution time $\bar{t}_{p\times c}$ of the parallel version of the algorithm on $p\times c$ processors to reach a solution of a certain fixed quality ([10]):

$$s_{p\times c} = \frac{\bar{t}}{\bar{t}_{p\times c}}.$$

Analogously to [10], speed-up is analyzed by considering problem instances that can be solved optimally with the sequential variant GA as well as with the parallel variant MMGA. Here, the ORLIB instances could be solved to optimality by means of both variants GA and MMGA. Since these instances allow a comparison of computing times for the same solution quality, they are used for the speed-up analysis. Each run was stopped after the optimal solution was found. The speed-up in time of the parallel variant MMGA against its sequential variant GA in reaching the optimal solution for the ORLIB instances is on average $s_{50} = 41.5$.

5. Conclusions and recommendations for future work

A Parallel Multistart Multiple-Deme Genetic Algorithm (MMGA) for the Uncapacitated Warehouse Location Problem (UWLP) has been described. The developed algorithm is based on an integer coding and a developed two-level parallelization strategy. The strategy combines the multiple-deme approach developed by [11] and the multistart approach of [34]. The MMGA is evaluated using 717 benchmark problems from the literature. Comparison of results achieved with this new method and those methods previously established as best shows that the MMGA can be regarded as competitive in terms of the achieved solution quality. The MMGA could find 551 of 592 known optimal solutions. Parallelization allowed to speed-up the computing time needed to reach a given solution quality. The results show that the developed two-level parallelization strategy is suitable for the efficient solution of the UWLP.

Future work will extend the MMGA to solve the capacitated version of the warehouse location problem. Furthermore, alternative parallelization approaches may be used and compared with the MMGA.

6. References

1. Affenzeller, M (2002) A generic evolutionary computation approach based upon genetic algorithms and evolution strategies. Journal of Systems Science 28: 59-72
2. Alba E, Luque G (2005) Measuring the performance of parallel metaheuristics. In: Alba E (ed) Parallel metaheuristics - a new class of algorithms, pp 43-62. Wiley Series on Parallel and Distributed Computing, Hoboken, New Jersey
3. Al-Sultan KS, Al-Fawzan MA (1999) A tabu search approach to the uncapacitated facility location problem. Annals of Operations Research 86: 91-103
4. Alves ML, Almeida MT (1992) Simulated annealing algorithm for the simple plant location problem: a computational study. Revista Investigacao Operacional 12
5. Aydin ME, Yigit V, Fogarty TC (2002) Two approaches to simulated annealing for uncapactitated facility location problems. Working Paper, Napier University Edinburgh http://www.-dcs.napier.ac.uk/~benp/dream/dreampaper14.pdf - last check of address: Sep 04, 2007
6. Barahona F, Chudak F (1999) Near-optimal solutions to large scale facility location problems. Technical Report RC21606, IBM, Yorktown Heights, NY, USA
7. Beasley J (1990) OR-Library, distributing test problems by electronic mail. Journal of the Operational Research Society 41: 1069-1072
8. Beyer H-G, Schwefel H-P (2002) Evolution strategies. Natural Computing 1: 3-52
9. Bilde O, Krarup J (1977) Sharp lower bounds and efficient algorithms for the simple plant location problem. Annals of Discrete Mathematics 1: 79-97
10. Cantú-Paz E (1998) A survey of parallel genetic algorithms. Calculateurs Paralleles, Reseaux et Systems Repartis 10: 141-171
11. Cantú-Paz E (2001) Migration policies, selection pressure, and parallel evolutionary algorithms. J. of Heuristics 7**:** 311-334
12. Cantú-Paz E (2005). Theory of parallel genetic algorithms. In: Alba E (ed) Parallel metaheuristics - a new class of algorithms, pp 425-446. Wiley Series on Parallel and Distributed Computing, Hoboken, New Jersey
13. Crainic TG, Toulouse M (2003) Parallel strategies for meta-heuristics. In: Glover F, Kochenberger G (eds) State-of-the-art handbook in metaheuristics, pp 475-513. Kluwer, Norwell, MA
14. Crainic TG, Hail N (2005) Parallel meta-heuristics applications. In: Alba E (ed) Parallel metaheuristics, pp 447-494. Wiley, Hoboken, NJ
15. De Jong KA (1975) Analysis of the behaviour of a class of genetic adaptive systems. PhD Thesis, University of Michigan, Ann Arbor, MI
16. Galvao RD, Raggi LA (1989) A method for solving to optimality uncapacitated facility location problems. Annals of Operations Research 18: 225-244

17. Ghosh D (2003) Neighborhood search heuristics for the uncapacitated facility location problem. European Journal of Operational Research 150: 150-162
18. Goldberg DE (1989) Genetic Algorithms in search, optimization and machine learning. Addison-Wesley, Reading, MA
19. Greistorfer P, Rego C (2006) A simple filter-and-fan approach to the facility location problem. Computers and Operations Research 33: 2590-2601
20. Harm G, Van Hentenryck P (2005) A multistart variable neighborhood search for uncapacitated facility location. In: Proceedings of the 6th Metaheuristics International Conference (MIC), Vienna. Springer
21. Hoefer M (2002) Uncapacitated facility location problem library, http://www.mpi-inf.mpg.de/-departments/d1/projects/benchmarks/UflLib/index.html - last check of address: Jan 02, 2007
22. Hoefer M (2003) Experimental comparison of heuristic and approximation algorithms for uncapacitated facility location. In: Proceedings of the 2nd International Workshop on Experimental and Efficient Algorithms (WEA), Volume 2647 of Lecture Notes in Computer Science, pp 165-178. Springer, Berlin
23. Holland JH (1975) Adaptation in natural and artificial systems. 2nd edn. MIT Press, Cambridge, MA
24. Jaramillo JH, Bhadury J, Batta R (2002) On the use of genetic algorithms to solve location problems, Computers & Operations Research 29: 761-779
25. Kochetov Y (2003) Discrete location problems benchmark library, http://www.math.nsc.ru/AP/-benchmarks/english.html - last check of address: Jan 02, 2007
26. Kochetov Y, Ivanenko D (2005) Computationally difficult instances for the uncapacitated facility location problem. In: Ibaraki T, Nonobe K, Yagiura M (eds) Metaheuristics: progress as real solvers, Operations Research/Computer Science Interfaces Series 32, pp 351-367. Springer, Berlin
27. Krarup J, Pruzan PM (1983) The simple plant location problem: survey and synthesis. European Journal of Operational Research 12: 36-81
28. Kratica J, Tosic D, Filipovic V, Ljubic I (2001) Solving the simple plant location problem by genetic algorithm. RAIRO Operations Research 35: 127-142
29. Luque G, Alba E, Dorronsoro B (2005) Parallel genetic algorithms. In: Alba E (ed) Parallel metaheuristics - a new class of algorithms, pp 107-126. Wiley Series on Parallel and Distributed Computing, Hoboken, New Jersey
30. Michel L, Van Hentenryck P (2003) A simple tabu search for warehouse location. European Journal on Operations Research 157: 576-591
31. Resende MGC, Ribeiro CC (2003) Greedy randomized adaptive search procedures. In: Glover F, Kochenberger G (eds) Handbook of metaheuristics, pp 219-249. Kluwer, Boston, MA
32. Resende MGC, Ribeiro CC (2005) GRASP with path-relinking: Recent advances and applications. In: Ibaraki T, Nonobe K, Yagiura M (eds) Metaheuristics: progress as real problem solvers, pp 29-63. Springer, Berlin
33. Resende MGC, Werneck RF (2004) A hybrid heuristic for the pmedian problem. Journal of Heuristics 10: 59-88
34. Resende MGC, Werneck RF (2006) A hybrid multistart heuristic for the uncapacitated facility location problem. European Journal of Operational Research 174: 54-68
35. Schwefel H-P (1995) Evolution and optimum seeking. Wiley, New York
36. Sun M (2005) A tabu search procedure for the uncapacitated facility location problem. In: Rego C, Alidaee B (eds) Metaheuristic optimization via memory and evolution: tabu search and scatter search, pp 191-211. Kluwer, Boston, MA
37. Sun M (2006) Solving uncapacitated facility location problems using tabu search. Computers and Operations Research 33: 2563-2589
38. Yigit V, Aydin ME, Turkbey O (2006) Solving large-scale uncapacitated facility location problems with evolutionary simulated annealing. International Journal of Production Research 44: 4773-4791