Serdica J. Computing **2** (2008), 267–276

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

GENETIC ALGORITHM APPROACH FOR SOLVING THE TASK ASSIGNMENT PROBLEM[*]

Aleksandar Savić, Dušan Tošić, Miroslav Marić, Jozef Kratica

Abstract. In this paper a genetic algorithm (GA) for the task assignment problem (TAP) is considered.An integer representation with standard genetic operators is used. Computational results are presented for instances from the literature, and compared to optimal solutions obtained by the CPLEX solver. It can be seen that the proposed GA approach reaches 17 of 20 optimal solutions. The GA solutions are obtained in a quite a short amount of computational time.

1. Introduction. In recent time, with the growth of computerization, some problems have arisen traced in earlier times, but nowadays mostly associated with the optimization of CPU time. In earlier times, the task assignment

ACM Computing Classification System (1998): G.1.6.
Key words: evolutionary approach; genetic algorithms; assignment problems, multiprocessor systems, combinatorial optimization.

[*]This research was partially supported by the Serbian Ministry of Science and Ecology under project 144007. The authors are grateful to Ivana Ljubić for help in testing and to Vladimir Filipović for useful suggestions and comments.

was solved manually by the overseer and the distribution was done manually between working parties on construction and other working sites. The idea was to assign tasks to the working parties minimizing jobs interferences and making the total working time as short as possible. Nowadays computers replace working parties and the task assignment is bounded with minimally spent CPU time with the additional requirement that communication costs between parties should be minimal. Some similar problems can be seen in [3, 6, 12].

In this paper a special Task Assignment Problem (TAP), sometimes called TAS with non-uniform communication costs, is studied. The problem is present from the beginning of computer era, but only now the new techniques capable of solving it are being developed.

The problem of task assignment with non-uniform communication costs can be modeled as a nonlinear integer 0-1 minimization problem.

In this paper the application based on genetic algorithms for solving TAP is described. Because this problem is fully 0-1 integer, it is in the area of research in recent developments of genetic algorithms (for example, see [1, 9, 10]). There are well many documented applications considering large optimization problems [5, 8, 13, 14], and the ideas from these applications could be applied for solving TAP.

Let us first describe the mathematical formulation of TAP, then in brief explain GA and finally present experimental results.

2. Mathematical formulation. The problem of task assignment with non-uniform communication costs is related to finding an assignment of N tasks to M processors providing that sum of:

- total cost of execution for given tasks,
- total cost of all communications between processors, while they executing allocated tasks,

is minimal.

We will use quadratic integer programming formulation from [2].

Let there be N tasks, M processors, and e_{ik} be the cost of executing task i on a processor k. Let c_{ijkl} be the communication cost between tasks i and j if they are respectively assigned to processors k and l. Let us denote with 0–1 integer variable x_{ik} which has value 1 if task i is assigned to processor k.

Now we can formulate the quadratic integer programming model for TAP as follows:

$$\min \sum_{i=1}^{N}\sum_{k=1}^{M} e_{ik}x_{ik} + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N}\sum_{k=1}^{M}\sum_{l=1}^{M} c_{ijkl}x_{ik}x_{jl} \tag{1}$$

subject to

$$\sum_{k=1}^{M} x_{ik} = 1, \qquad i = 1,\ldots,N \tag{2}$$

$$x_{ik} \in \{0,1\} \qquad i = 1,\ldots,N,\;\; k = 1,\ldots,M \tag{3}$$

As can be seen, the objective function is non-linear and this optimization problem cannot be easily solved by using an exact algorithm. The constraints (2) are natural and reflect the fact that any task can be executed only on one processor. Now we will describe the properties of the genetic algorithm and some of its parts for solving the presented problem.

3. Proposed GA method.

The GA algorithms are stochastic methods for searching and finding best solutions. They are motivated by processes in natural world trying to emulate them. Similar to nature, GA works with individuals constituting a population. Each individual represents some solution to the problem. As in nature, the individuals better suited for survival are at the same time individuals that are better attain the optimal solution. These individuals are favored for passing into the next generation. The passing on of good qualities is accomplished by using genetic operators of crossover and mutation. The decision what individual has better quality for passing into next generation is obtained by evaluation of the fitness function. This process of betterment of individuals in the population is iteratively continued until an optimum is achieved or some other stopping criterion is applied. The detailed description of GAs is out of this paper's scope and can be found in [11].

The encoding of individuals in our case is integer. The genetic code has a length of N, where every gene corresponds to one particular task. This means that every gene represents the ordinal numeral of the processor on which the task is executed. For example, if the ith gene has value k this means that $x_{ik} = 1$ and $x_{ip} = 0$ for every p, $p \neq k$.

The very important parts of proposed GA are: Fitness Function, Selection, Crossover and Mutation. In the following paragraphs, their use and importance to the whole GA application will be explained.

The function f_{ind} that decides the best suitability of an individual to pass into the next generation is the fitness function. The values of this function are computed by scaling objective values obj_{ind} of all individuals into the interval [0,1], so that the best suitable individual ind_{max} gets value 1 and the worst ind_{min} gets 0. Explicitly, $f_{ind} = \dfrac{obj_{ind_{min}} - obj_{ind}}{obj_{ind_{min}} - obj_{ind_{max}}}$. Now the individuals are arranged in a non-increasing order by their best fitness: $f_1 \geq f_2 \geq \cdots \geq f_{N_{pop}}$, where N_{pop} is number of individuals in population.

All elite individuals (their number being N_{elite}) are automatically passed into the next generation. All non-elite individuals (their number being $N_{nnel} = N_{pop} - N_{elite}$) are subject to genetic operators. In this way the computational time is reduced because the objective function of elite individuals is same in the next generation and need to be calculated only once, in the first generation.

Individuals with the same genetic code in the population must be avoided, so their fitness is set to 0 on all occurrences, except the first one. Also, the number of individuals with the same objective function, but different genetic code, must be limited by some constant N_{rv}. To avoid this problem, the fitness of all individuals with the same value of objective function, but different genetic material, will be set to 0 except for the first N_{rv} of them.

Selection operators are applied on all non-elite individuals, so they choose which of these individuals will have offspring in the next generation. This is done through tournaments. From the whole population an a priori set number of individuals is chosen to participate in the tournament. The number of participants is called tournament size. The choosing of individuals is done randomly. The winner of the tournament is the individual with highest value of the objective function. The number of tournaments is equal to the number of non-elite individuals N_{nnel}, so that exactly N_{nnel} parents can be chosen for crossover. The same individual from the current generation can participate in more than one tournament. In the standard tournament selection, the tournament size is integer and this choice can reduce the efficiency of the algorithm.

Because of this, for selection an improved tournament selection operator, Fine-grained tournament selection – FGTS [4, 5] is implemented. Here, the tournament size is a real parameter F_{tour}, representing the preferable average tournament size. In this procedure, there are two types of tournaments. One is held k_1 times, with tournament size $\lfloor F_{tour} \rfloor$, and the other type is held k_2 times, with tournament size $\lceil F_{tour} \rceil$. From here $F_{tour} \approx \dfrac{k_1 \cdot \lfloor F_{tour} \rfloor + k_2 \cdot \lceil F_{tour} \rceil}{N_{nnel}}$.

To get satisfactory results of GA, a good ratio between the number of elite and the number of non-elite individuals is necessary. For example, for $N_{pop} = 150$ an adequate proportion is $N_{elite} = 100$ and $N_{nnel} = 50$. The corresponding k_1 and k_2 parameters for deciding the tournament size are 30 and 20, respectively. Typically, for $N_{pop} = 150$ an adequate maximum of individuals with the same fitness (with the same value of objective function), is $N_{rv} = 40$.

As can be seen in [4, 5, 13], FGTS performs best with value of F_{tour} set on 5.4. Leaning on experience presented in the cited works, the same value is used in this paper.

In the crossover operator, non-elite individuals are randomly paired in $\lfloor N_{nnel}/2 \rfloor$ pairs for exchange of genes with the intention to produce offspring with potentially better suitability.The applying of the crossover operator on chosen pair of parents produces two offsprings. In this paper a standard one-point crossover operator is used. This operator exchanges all genes between genetic codes of parents to produce offspring.The probability of applying the crossover operator is 85%. This means that approximately 85% of the pairs of individuals will exchange genes.

In genetic algorithm the simple mutation operator with frozen genes is used. This operator changes a randomly selected gene in the genetic code of an individual with some mutation rate. For the improvement of GA, a modification is included dealing with so-called frozen genes. Sometimes it happens that all individuals in the population have the same gene in a certain position. These genes are called frozen. With the frozen genes a problem could arise because they reduce the search space and increase the possibility of premature convergence. Selection and crossover operators cannot change the frozen genes, because all individuals in the population have them in the same position. The basic mutation rate is $0.4/N$.

The mentioned improvement of GA consists in increasing the mutation rate only on frozen genes. In each generation it is determined in advance which genes are frozen. Then, the mutation rate for these genes is increased. In the proposed GA the increase is 2.5 times greater $(1./N)$ than the basic mutation rate on unfrozen genes.

The initialization of population is done in a random way. So, the population in the first generation is the most heterogeneous and diversified genetic pool.

The performance of the proposed GA is improved by using a caching technique. The main idea behind this technique is to avoid the evaluation of the objective function for individuals with the same genetic code. The values of individuals for which the objective function was already computed are stored by the least recently used (LRU) caching technique into the hash-queue data structure. Because of this, whenever an individual with the same genetic code is generated, the value of its objective function is not computed, but is found in cache memory. This procedure can result in significant time saving. The number of calculated values of the objective function in this implementation is limited to 5000. If the cache memory is full, then we remove the least recently used cash memory block. Detailed information about caching GA can be found in [7].

4. Experimental results. All computations were executed on a Quad Core 2.5 GHz PC computer with 4 GB RAM. The genetic algorithm was coded in C. For experimental testings in our implementation the instances described at [2] were used. These instances include different numbers of tasks (N = 10, 15) and different numbers of processors (M = 3, 5). For each pair of task-processor (N, M), there is a set of ten instances.

The finishing criterion of GA is the maximal number of generations N_{gen} = 5000. The algorithm also stops if the best individual or best objective value remains unchanged through N_{rep} = 2000 successive generations. Since the results of GA are nondeterministic, our GA was run on a single processor 20 times for each of the instances.

Table 1 summarizes the results of the executions of our application on these instances. In the first column the names of instances are given. The name of the instance carries information about the number of tasks N, the number of processors M and the number of generated cases with same N and M. (For example, the instance tassnu_10_3_1 is an instance which has N = 10 tasks on M = 3 processors and it is the first case generated for this N and M). The second and third columns contain optimal solutions and running times of the CPLEX solver. The best GA values GA_{sol} are given in the following column. The mark *opt* is given if an optimal solution is reached and there is no difference between that solution and the solution obtained by CPLEX.

Average running times needed to detect the best GA values are given in the t column. On average, GA finished after *gen* generations. The quality of the solution in all 20 executions is evaluated as a percentage gap named *agap*, with respect to the optimal solution sol_{opt}, by using standard deviation σ of the

Table 1. GA results on TAP instances

	CPLEX		*GA*						
Instance name	*sol*	*t*	*sol*	*t*	*gen*	*agap*	σ	*eval*	*cache*
	(*opt*)	(sec)		(sec)		(%)	(%)		(%)
tassnu_10_3_1	−719	< 1	opt	0.162	2014	0.000	0.000	12778	87.3
tassnu_10_3_2	−790	< 1	opt	0.165	2023	0.000	0.000	14800	85.4
tassnu_10_3_3	−624	< 1	opt	0.210	2624	0.641	1.208	28130	78.5
tassnu_10_3_4	−734	< 1	opt	0.172	2128	0.000	0.000	18161	83.0
tassnu_10_3_5	−871	< 1	opt	0.162	2021	0.000	0.000	15209	85.0
tassnu_10_3_6	−677	< 1	opt	0.165	2018	0.214	0.958	14270	85.9
tassnu_10_3_7	−613	< 1	opt	0.162	2026	0.000	0.000	12294	87.9
tassnu_10_3_8	−495	< 1	opt	0.164	2016	0.000	0.000	11409	88.7
tassnu_10_3_9	−750	< 1	opt	0.163	2020	0.000	0.000	14501	85.7
tassnu_10_3_10	−486	< 1	opt	0.164	2017	0.021	0.092	20510	79.7
tassnu_15_5_1	−1985	51832	opt	0.254	2285	3.131	6.743	65830	42.4
tassnu_15_5_2	−1568	129840	−1539	0.328	2900	4.827	1.579	89965	37.9
tassnu_15_5_3	−1892	52955	−1856	0.257	2223	9.905	2.110	75729	32.0
tassnu_15_5_4	−1806	91146	opt	0.292	2545	1.462	2.754	85165	32.9
tassnu_15_5_5	−1881	78795	opt	0.248	2192	2.818	2.967	69018	37.1
tassnu_15_5_6	−1950	79872	opt	0.254	2256	5.285	5.333	73777	34.8
tassnu_15_5_7	−1893	51547	opt	0.236	2072	4.691	3.892	70383	32.2
tassnu_15_5_8	−1733	108092	opt	0.241	2100	2.796	1.056	73421	30.2
tassnu_15_5_9	−1798	107982	−1780	0.246	2183	2.406	2.355	72580	33.8
tassnu_15_5_10	−1763	109963	opt	0.246	2159	4.169	2.869	71855	33.3

average gap. The percentage gap *agap* is defined as $agap = \frac{1}{20}\sum_{i=1}^{20} gap_i$, where $gap_i = 100 * \frac{GA_i - sol_{opt}}{sol_{opt}}$ and GA_i represents the GA solution obtained in the i-th running, while σ is the standard deviation of gap_i, $i = 1, 2, \ldots, 20$, obtained by the formula $\sigma = \sqrt{\frac{1}{20}\sum_{i=1}^{20}(gap_i - agap)^2}$. The last two columns are related to the caching: *eval* represents the average number of evaluations, while *cache* displays the savings (in percent) achieved by using caching technique.

As can be seen in Table 1, GA reached 17 of 20 optimal solutions and the running time on all instances is really small. The average execution on the biggest instance is less than half a second. The CPLEX solver also runs very fast on instances with 10 tasks, but on larger instances, with 15 tasks, the running times are huge ($> 50\,000$ seconds). Although the GA algorithm did not reach the optimal solution for 3 of 20 instances, this fact is overwhelmed by its speed. It is obvious that CPLEX cannot handle larger instances with more than 15 tasks, but GA will quickly give solutions of acceptable quality even in these instances.

5. Conclusions. The GA metaheuristic for solving TAP is presented. The integer representation of the task assignment was used based on the non-linear integer 0-1 optimization problem. The fine-grained tournament selection, one-point crossover and simple mutation with frozen genes were used. The computational performance of GA was additionally improved by caching GA technique. For almost all instances, except three, GA calculates solutions matching optimal ones and obtained in a very small running time.

Based on the presented results, we can conclude that GA has the potential of being a useful metaheuristic for solving other similar problems, while TAP could be considered with additional constraints. Parallelization of the GA and its hybridization with exact methods are most promising directions of future work.

REFERENCES

[1] Djurić B., J. Kratica, D. Tošić, V. Filipović. Solving the maximally balanced connected partition problem in graphs by using genetic algorithm. *Computing and Informatics*, **27**, No. 3, 2008, 341–354.

[2] Elloumi S. The task assignment problem, a library of instances. `http://cedric.cnam.fr/oc/TAP/TAP.html`, 2004.

[3] Fidanova S., M. Durchova. Ant Algorithm for Grid Scheduling Problem. Large Scale computing, Lecture Notes in Computer Science, vol. **3743**, Springer, Germany, 2006, 405–412.

[4] Filipović V. Fine-grained Tournament Selection Operator in Genetic Algorithms. *Computing and Informatics*, **22** (2003), 143–161.

[5] Filipović V. Selection and migration operators and Web services in parallel evolutionary algorithms. PhD thesis, University of Belgrade, Faculty of Mathematics, 2006 (in Serbian).

[6] Gong L., X.H. Sun, E. Waston. Performance Modeling and Prediction of Non-Distributed Network Computing. *IEEE Transaction on Computers*, **51**, No 9 (2002), 1041–1055.

[7] Kratica J. Improving Performances of the Genetic Algorithm by Caching. *Computers and Artificial Intelligence*, **18** (1999), 271–283.

[8] Kratica J., V. Kovačević-Vujčić, M. Čangalović. Computing strong metric dimension of some special classes of graphs by genetic algorithms. *Yugoslav Journal of Operations Research*, **18**, No. 2 (2008), 143–151.

[9] Kovačević J. Hybrid Genetic Algorithm For Solving The Low-Autocorrelation Binary Sequence Problem. *Yugoslav Journal of Operations Research*, in press.

[10] Marić M. An efficient genetic algorithm for solving the multi-level uncapacitated facility location problem. *Computing and Informatics*, in press.

[11] Mitchell M. Introduction to genetic algorithms. Cambridge, Massachusetts, MIT Press, 1999.

[12] Schopf M. J. General Architecture for Scheduling on the Grid. Specia issue of JPDC on Grid Computing, 2002, `http://www.mcs.anl.gov/~schopf/Pubs/sched.arch.2002.pdf`, 2008.

[13] Stanimirović Z. Genetic algorithms for solving some NP-hard hub location problems. Ph. D. thesis, University of Belgr. Faculty of Mathematics, 2007.

[14] Stanimirović Z. A genetic algorithm approach for the capacitated single allocation p-hub median problem. *Computing and Informatics*, **27** (2008), in press.

Aleksandar Savić
Dušan Tošić
Miroslav Marić
Faculty of Mathematics
University of Belgrade,
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: `aleks3rd@eunet.yu`
`dtosic@matf.bg.ac.yu`
`maricm@matf.bg.ac.yu`

Jozef Kratica
Mathematical Institute
Serbian Academy of Sciences and Arts
Kneza Mihaila 36/III
11000 Belgrade, Serbia
e-mail: `jkratica@mi.sanu.ac.yu`

Received September 25, 2008
Final Accepted October 15, 2008