Multi-criterion Tabu Programming for Pareto-optimal Task Assignment in Distributed Computer Systems

JERZY BALICKI
Naval University of Gdynia
ul. Smidowicza 69, 81-103 Gdynia,
POLAND

Abstract: - Multi-criterion tabu programming is a new approach for a decision making support and it can be applied to determine the Pareto solutions. Similarly to rules applied in the genetic programming, tabu programming solves problems by using a general solver that is based on a tabu algorithm. In the formulated task assignment problem as a multi-criterion question, both a workload of a bottleneck computer and the cost of system are minimized; in contrast, a reliability of the distributed system is maximized. Moreover, there are constraints for the performance of the distributed systems and the probability that all tasks meet their deadlines. In addition, constraints related to memory limits and computer locations are imposed on the feasible task assignment. Finally, results of some numerical experiments have been presented.

Key-Words: - Tabu search algorithm, multi-criterion optimization, genetic programming

1 Introduction

Tabu programming is a new paradigm of artificial intelligence that can be applied for computer decision aid. Similarly to the genetic programming that applied a genetic algorithm [20], tabu programming solves problems by using a general solver that is based on a tabu algorithm. Tabu search is a combinatorial optimization technique for development in zero-one programming, non-convex non-linear programming, and general mixed integer optimization [22]. Some interesting task scheduling algorithms based on tabu search are proposed by Węglarz in [21]. This technique is applied to continuous functions by selection a discrete encoding of the problem. In a tabu search, special areas are forbidden during the seeking in a space of all possible combinations [10].

Tabu programming paradigm has been implemented as an algorithm operated on the computer program that produces the solution. Tabu search algorithm has been extended by using a computer program instead of a mathematical variable.

The first tabu programming for multi-criterion optimization has been presented by Balicki in 2007 [3]. The optimization technique called multi-criterion optimization tabu programming MOTP has been applied to the bi-criterion task assignment problem and the sub-optimal in Pareto sense solutions have been found. For solving the hierarchical solutions in the multi-objective optimization problem, MOTP was applied for three-criterion problem of robot trajectory, too [2].

In this paper, an improved MOTB for solving new three-criterion with constraints optimization problems of task assignment in the distributed computer system has been considered. The sub-effective task assignment has been obtained by development of that approach. Finally, results of some numerical experiments have been presented.

2 Convention of tabu programming

Tabu programming (TP) is based on tabu search algorithm rules. However, it is not a straightforward modification of tabu algorithm or the transformation of rules from the genetic programming. It is rather the combination of tabu search algorithm and genetic programming to create new optimization technique by avoiding some disadvantages of them. Moreover, aspects of multi-criterion optimization are respected.

The tabu programming operates on the computer program that produces an outcome that can be treated as a solution to the problem. Because in the computer program several modifications may be curried out by exchanging functions or arguments, the neighborhood of the current program can be created as a result of some adjustments of the given software procedure. TP avoids entrainment in cycles by forbidding moves which lead to points in the solution space previously visited. Number of moves and the number of programs in the neighborhood is much smaller than the number of solutions in the search space. To avoid a path already investigated a point with poor quality can be accepted from the neighborhood of the current program [11]. This insures new regions of a solution space will be explored

in with the goal of avoiding local minima and finding the global minimum [15].

To keep away from repeating the steps, recent moves are recorded in some tabu lists [18]. That lists forms the short-term memory. The memory content can vary as the search proceeds [5]. At the beginning, the target is testing the solution space, during a 'diversification' [12]. As candidate regions are identified the algorithm is more focused to find local optimal solutions in an 'intensification' process. The TP operates with the size, variability, and adaptability of the memory [7].

Special areas are forbidden during the seeking in a search space. From that neighborhood $N(x^{\text{now}})$ of the current solution x^{now} that is calculated by the given program, we can choose the next solution x^{next} to a search trajectory of TP [16]. The accepted alternative is supposed to have the best value of an objective function among the current neighborhood. In the tabu search algorithm based on the short-term memory, a basic neighborhood of a current solution may be reduced to a considered neighborhood $\boldsymbol{K}(x^{\text{now}})$ because of the maintaining a selective history of the states encountered during the exploration [13]. Some solutions, which were visited during the given last term, are excluded from the basic neighborhood according to the classification of movements [19]. If any solution satisfies an aspiration criterion, then it can be included to the considered neighborhood, only [6].

Computer programs from the neighborhood are constructed from the basic program that produces the current solution. The basic program is modeled as a tree (Fig. 1).

That tree is equivalent to the parse tree that most compilers construct internally to represent the specified computer program. A tree can be changed to create the neighborhood $N(x^{\text{now}})$ of the current program. For instance, we can remove a sub-tree with the randomly chosen node from the parent tree. Next, the randomly selected node as a terminal is required to be inserted. A functional node is an elementary procedure randomly selected from the primary defined set of functions [17]:

$$\mathsf{F} = \{f_1,..., f_n,..., f_N\} \qquad (1)$$

In the problem of finding trajectory of underwater vehicle [2], we define set of functions, as bellow:

$$\mathsf{F} = \{\textit{if\_obstacle, move, if\_end}, +, -, *, /\} \qquad (2)$$

The procedure *if_obstacle* takes two arguments. If the obstacle is recognized ahead the underwater vehicle, the first argument is performed. In the other case, the second argument is executed. The function *move* requires three arguments. It causes the movement along the given direction with the velocity equals the first argument during assumed time Δt. The time Δt is the value that is equal to the division a limited time by $M_{\max}$. The direction of the movement is changed according to the second and third arguments. The second argument is the angle of changing this direction up if it is positive or down if it is negative. Similarly, the third argument represents an angle of changing the direction to the left if it is positive or – to the right if it is negative.

The procedure *if_end* ends the path of the underwater vehicle if it is in the destination region or the expedition is continued if it is not there.

3 Function and argument sets for task assignment

Set of procedures for task assignment problems can be defined, as follows [3]:

$$\mathsf{F} = \{\theta, +, -, *, /\} \qquad (3)$$

where

θ – the procedure that converts $M=V+I(V+J)$ input real numbers called *activation levels* on M output binary numbers $\left(x_{11}^m,...,x_{1I}^m,...,x_{vi}^m,...,x_{VI}^m,x_{11}^\pi,...,x_{1J}^\pi,...,x_{ij}^\pi,...,x_{IJ}^\pi,N_1,...,N_v,...,N_V\right)$.

$$x_{ij}^\pi = \begin{cases} 1 \text{ if } \pi_j \text{ is assigned to the } w_i, \\ 0 \text{ in the other case.} \end{cases}$$

$$x_{vi}^m = \begin{cases} 1 \text{ if task } T_v \text{ is assigned to } w_i, \\ 0 \text{ in the other case,} \end{cases}$$

N_v – number of the vth module in the line for its dedicated computer,

$W = \{w_1,...,w_i,...,w_I\}$ - the set of the processing nodes,

$\boldsymbol{T}=\{T_1,...,T_v,...,T_V\}$ - the set of parallel performing tasks,

$\Pi = \{\pi_1,...,\pi_j,...,\pi_J\}$ - the set of available computer sorts.

The procedure θ is obligatory the root of the program tree and appears only one in a generated program. In that way, the formal constraints $x_m \in \mathsf{B}$, $m = \overline{1,M}$ for $\mathsf{B} = \{0, 1\}$, are satisfied. An activation level is supplied to a root from the sub-tree that is randomly generated with using arithmetic operators {+, -, *, /} and the set of terminals.

Furthermore, each procedure is supposed to be capable to allow any value and data type that may possible be assumed by any terminal selected from the following terminal set::

$$\mathsf{T} = \{a_1,...,a_m,...,a_M\} \qquad (4)$$

For finding the trajectory of the underwater vehicle, the set of arguments consists of the real numbers generated from the interval (-1; 1) [2]. However, for the task assignment the set of arguments is determined in the other way.

Let D be the set of numbers that consists of the given data for the instance of the problem. A terminal set is determined for the problem, as below [3]:

$$T = D \cup L, \quad (5)$$

where L – set of *n* random numbers, $n = \overline{D}$

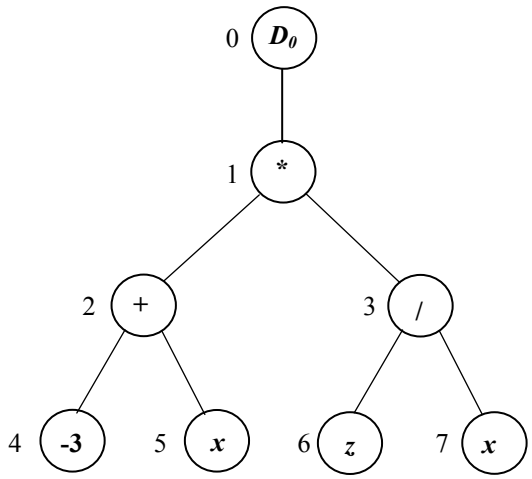

Fig. 1. The program tree for procedure *(x-3)*z/x*

4 Neighborhood and short-term memory

Some programs from the neighborhood can be created by sort of movements related to removing the randomly chosen terminal node and then adding a sub-tree with the functional node as a root. That sub-tree can be constructed from the random number of nodes.

If the node is the root of the reducing sub-tree, it can be protected against choosing it to be that root in a reducing operation until the next λ_1 movements are performed. However, that node can be selected to be the root for adding the sub-tree. Similarly, if the node is the root of the adding tree, it can be protected against choosing him to be that root in a adding operation until the next λ_2 movements is performed.

We can implement that by introducing the assignment vector of the node names to the node numbers. We insert a dummy node D_0 (Fig. 1) as the number 0, for the formal reason. The node index $l = \overline{1, L_{\max}}$, where $L_{\max}$ represents the assumed maximal number of nodes in the tree. Numbers are assigned from the dummy node to lower layers and from the left to the right at the current layer. The assignment vector of the node names to the node numbers for the tree from the Figure 1 can be represented, as below:

$$\omega = (D_0, *, +, /, -3, x, z, x) \quad (6)$$

Moreover, the vector of function and argument assignment can be defined, as follows:

$$\psi = (f, f, f, f, a, a, a, a) \quad (7)$$

The vector of the argument number can be determined, as below:

$$\chi = (1,2,2,2,0,0,0,0) \quad (8)$$

We can introduce the matrix of reducing node memory $M^- = [m_{nm}]_{L_{\max} \times L_{\max}}$, where m_{nm} represents the number of steps that can be missed after reduction the function f_m (with the parent f_n) as a root of the chosen sub-tree. After exchanging that root, $m_{nm} = \lambda_1$.

Similarly, we can define the matrix of adding node memory $M^+ = [\tilde{m}_{nm}]_{L_{\max} \times L_{\max}}$, where $\tilde{m}_{nm}$ represents the number of steps that can be missed after adding the function f_m (with the parent f_n) as a root of the created sub-tree. After exchanging that root, $\tilde{m}_{nm} = \lambda_2$.

Parameters λ_1 and λ_2 are usually equal to λ, but we can adjust their values to tune the tabu programming for the solved problem. On the other hand, the length of the short-term memory λ is supposed to be no greater than $L_{\max}$. After λ movements, the selected node may be chosen for operation once again.

5 Multi-criterion optimization tabu programming

MOTP can be used for solving an optimization problem with at least two criteria. From the set of the competitive solutions, we prefer admissible ones and coordinates of an ideal point are calculated. Then, the compromise solution x^* with the smallest distance to the ideal point is selected, as follows:

$$K(x^*, x^i) = \min_{x \in N(x^{now})} K(x, x^i) \quad (9)$$

where K – a distance function to the ideal point x^i.

The selection function W for the choosing the next solution in the search path is constructed from the criterion K and functions describing constraints [8]. Usually, the penalty function can be applied [9]. Figure 2 shows an outlook of the algorithm MOTP.

A paradigm of tabu programming gives opportunity to solve the several problems. The MOTP has been written in the Matlab language. Initial numerical experiments confirm that sub-optimal in Pareto sense solutions can be found by tabu programming for two-criterion task assignment and three-criterion underwater vehicle trajectory.

1. Initial procedure $k:=0$
- (A) Generation of the program that produces x^{now}
- (B) $x^{best} := x^{now}$, $x^{bis} := x^{now}$
- (C) $K_{min} := K(x^{now})$
- (D) Initialization of restriction matrixes M^+, M^-
- (E) Setting λ_1, λ_2

2. Solution selection and stop criterion $k:=k+1$
- (A) Finding a set of tree candidates $K(M^+, M^-, x^{now})$ from the neighborhood $N(x^{now})$
- (B) Selection of the next solution $x^{next} \in K$ (M^+, M^-, x^{now}) with the minimal value of the selection function W among solutions taken from K
- (C) **Aspiration condition.** If all solutions from the neighborhood are tabu-active and $K_{min} \geq K(x^{now})$, then $x^{best} := x^{now}$, $K_{min} := K(x^{now})$
- (D) **Re-linking of search trajectory.** If x^{next} was not changed during main iteration, then crossover procedure for parents x^{best}, x^{bis} is performed. A child with the smaller value of K is x^{next}, and another one is x^{bis}
- (E) If $k = 0.4\, K_{max}$, then $\lambda_1 := 4\lambda_1$, $\lambda_2 := 4\lambda_2$
- (F) If $k = K_{max}$ or maximal time of calculation is exceeded, then STOP.

3. Up-dating
- (A) $x^{now} := x^{next}$
- (B) If $K(x^{now}) < K_{min}$, then $x^{bis} := x^{best}$ and go to 1(B)
- (C) After reduction the procedure f_m (with the parent f_n) as a root of the chosen sub-tree $M^- := M^- - 1$, $m_{nm} = \lambda_1$.
- (D) After adding the procedure f_m (with the parent f_n) as a root of the created sub-tree $M^+ := M^+ - 1$, $\tilde{m}_{nm} = \lambda_2$.
- (E) go to 2

Fig. 2. An algorithm MOTP

6. Criteria for task assignment and scheduling

To test the ability of the MOTP, we consider a multi-criterion optimisation problem for task assignment in a distributed computer system, where three criteria are optimized. In the formulated task assignment problem as a multi-criterion question, both Z_{max} – a workload of a bottleneck computer and C – the cost of system are minimized; in contrast, R – a reliability of the distributed system is maximized. Moreover, there are constraints for the performance of the distributed systems and the probability that all tasks meet their deadlines. In addition, constraints related to memory limits and computer locations are imposed on the feasible task assignment.

It is a new approach for formulation multi-objective task assignment problems, although some three-criterion task assignment questions have been formulated yet [4]. Meta-heuristics like evolutionary algorithms, tabu algorithm and genetic programming have been applied for solving multi-criterion optimization problem. We can compare quality of obtained task assignments by MOTP to qualities produced by the other multi-criterion meta-heuristics.

Finding allocations of tasks in a distributed system may estimation of a criterion by taking a benefit of the particular properties of some workstations or an advantage of the computer load.

Let the task be executed on some computers taken from the set of available computer sorts. The overhead performing time of the task T_v by the computer π_j is represented by an item t_{vj}. A computer with the heaviest task load is the bottleneck machine and its workload is a critical value that is supposed to be minimized. The first criterion is the workload of the bottleneck computer for the allocation x, and its values are provided by the subsequent formula [4]:

$$Z_{\max}(x) = \max_{i \in 1,I} \left\{ \sum_{j=1}^{J} \sum_{v=1}^{V} t_{vj} x_{vi}^{m} x_{ij}^{\pi} + \sum_{v=1}^{V} \sum_{\substack{u=1 \\ u \neq v}}^{V} \sum_{\substack{i=1 \\ k \neq i}}^{I} \sum_{k=1}^{I} \tau_{vuik} x_{vi}^{m} x_{uk}^{m} \right. \quad \textbf{(10)}$$

where

$$x = \left(x_{11}^{m}, \ldots, x_{1I}^{m}, \ldots, x_{vi}^{m}, \ldots, x_{VI}^{m}, x_{11}^{\pi}, \ldots, x_{1J}^{\pi}, \ldots, x_{ij}^{\pi}, \ldots, x_{IJ}^{\pi}, N_1, \ldots, N_v, \ldots, N_V \right),$$

τ_{vuik} – the total communication time between the task T_v assigned to the ith node and the T_u assigned to the kth node.

Figure 3 shows the workload of the bottleneck computer in the distributed computer system for generated task assignments by an enumerative algorithm. The function Z_{max} takes value from the period [40; 110] (TU - time unit) for 256 solutions. What is more, even a small change in task assignment related to the movement of a task to another computer or a substitution of computer sort can cause a relatively big alteration of its workload. For instance, the migration of one task from the assignment with Z_{max}=40 TU may increase the workload to the 64 or even 88 TU.

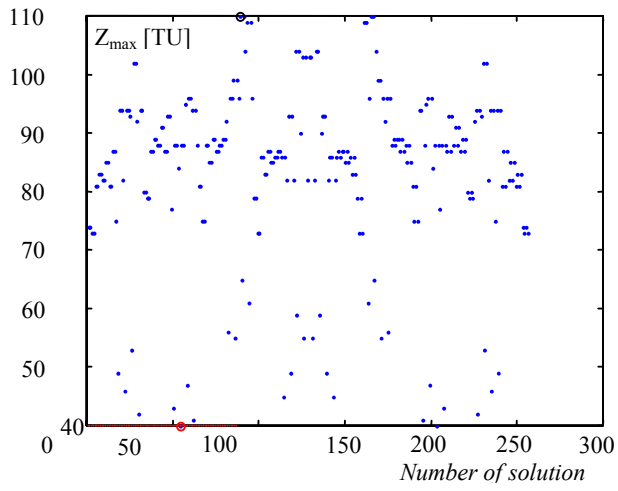

Fig. 3. Workload of the bottleneck computer for generated solutions.

The second measure of the task assignment is a cost of computers that is calculated, as below:

$$C(x) = \sum_{i=1}^{I} \sum_{j=1}^{J} \kappa_j x_{ij}^{\pi} \qquad (11)$$

where κ_j corresponds to the cost of the computer π_j.

Let π_j be failed independently due to an exponential distribution with rate $\tilde{\lambda}_j$. We do not take into account of repair and recovery times for failed computer in assessing the logical correctness of an allocation. Instead, we are supposed to allocate tasks to computers on which failures are least likely to occur during the execution of tasks. Computers and tasks can be assigned to nodes in purpose to maximize the third criterion – the reliability function R defined, as below [3]:

$$R(x) = \prod_{v=1}^{V} \prod_{i=1}^{I} \prod_{j=1}^{J} \exp(-\tilde{\lambda}_j t_{vj} x_{vi}^{m} x_{ij}^{\pi}) \qquad (12)$$

7. Constraints and decision variables

The minimal performance of the distributed systems $\Theta_{\min}$ is required that can be estimated according to the following formula:

$$\Theta(x) = \sum_{i=1}^{I} \sum_{j=1}^{J} \vartheta_j x_{ij}^{\pi} \qquad (13)$$

where ϑ_j is the numerical performance of the computer π_j for the task benchmark.

The probability that all tasks meet their deadlines is supposed to be greater than the minimal probability $P_{\min}$. The precedence constraints among tasks are figured in calculation of task release time and the timing constraints on tasks are considered. Let the distributed program P_n may begin its running after λ_n and complete before δ_n. Moreover, we assume a conditionally running task is performed with the probability q and its complementary task – with the probability (1-q). A task in the loop is performed k times for a run (k=1,2,...,$L_{\max}$), and each repetition of this task is performed with the probability p. The instance, where a loop task runs k times, can be meet with the probability $(1 - p)\, p^{k-1}$. The instance, where a conditional task appears and the loop task runs k times, occurs with the probability:

$$p_i = q(1-p)\, p^{k-1} \qquad (14)$$

Times of task completions (C_1,...,C_v,...,C_V) can be calculated for scheduled allocation modules to computers $x = (x^m, x^{\pi}, N^m)$ [4]. Let d_v represents the completion deadline for the vth task. If $C_v \le d_v$, then the time constraint is satisfied what can be written as $\xi(d_v - C_v) = 1$. If the deadline is exceeded, then $\xi(d_v - C_v) = 0$. If at least one task exceeds the deadline, then deadline constraint for the ith instance is not satisfied. Probability that all tasks meet their deadlines for K instances of the flow graph is calculated:

$$P_D(x) = \sum_{i=1}^{K} p_i \prod_{m_v \in M_i} \xi(d_v - C_v(x)) \qquad (15)$$

Constraints related to memory limits are related to the assumption that a computer is supposed to be equipped with necessary capacities of resources. Let the following memories z_1,...,z_r,...,z_R be available in the distributed system and let d_{jr} be the capacity of memory z_r in the workstation π_j. We assume the task T_v reserves c_{vr} units of memory z_r and holds it during a program execution. The memory limit R_{ir} of the rth resource in a machine cannot be exceeded in the ith node, what is written, as bellows:

$$R_{ir}(x) = \sum_{j=1}^{J} d_{jr} x_{ij}^{\pi} - \sum_{v=1}^{V} c_{vr} x_{vi}^{m},\ i = \overline{1,I},\ r = \overline{1,R}. \qquad (16)$$

8. Problem formulation and numerical experiments

Let (X, F, P) be the multi-criterion optimisation question for finding the representation of Pareto-optimal solutions [1]. It is established, as follows:

1) X - an admissible solution set

$$\mathsf{X} = \{x \in \mathsf{B}^{I(V+J)} \mid \Theta(x) \ge \Theta_{\min};\ P_D(x) \ge P_{\min};$$
$$R_{ir}(x) \ge 0,\ i = \overline{1,I},\ r = \overline{1,R};$$
$$\sum_{i=1}^{I} x_{vi}^{m} = 1,\ v = \overline{1,V};\ \sum_{j=1}^{J} x_{ij}^{\pi} = 1,\ i = \overline{1,I}\}$$

2) F - a quality vector criterion

$$F : \mathsf{X} \to \mathsf{R}^3 \qquad (17)$$

where

R – the set of real numbers,

$F(x) = [Z_{\max}(x), C(x), -R(x)]^T$ for $x \in \mathsf{X}$,

3) P - the Pareto relation [1].

Figure 4 shows the cut of the evaluation space that is explored by the most effective meta-heuristic AMEA* [4]. Evolutionary algorithm AMEA* [4], tabu algorithm MOTA [14] and genetic programming MGP [3] have been applied for solving some versions of multi-criterion task assignment. We can compare quality of obtained solutions by MOTP to qualities produced by the other multi-criterion meta-heuristics.

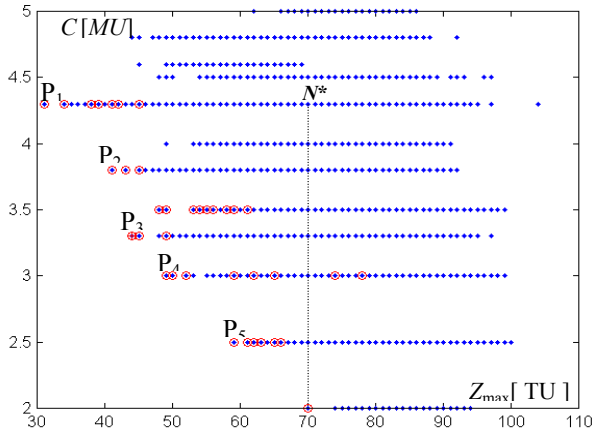

Fig. 4. Pareto front and results of AMEA*

The binary search space consisted of 1.0737×10^9 elements and included 25 600 admissible solutions. By enumerative algorithm the set of Pareto points was found. Quality of obtained solutions by the algorithms was determined by the level of the convergence to the known Pareto set [2]. An average level $\overline{S}$ was calculated for fifty runs of the algorithm. That tabu programming MOTP gives better outcomes than the genetic programming MGP for the same number of selection function or fitness function calculations. After 350 assessments of those functions, an average level of Pareto set obtaining is 1.7% for the MOTP, 3.6% for the MGP.

An average level of convergence to the Pareto set, a maximal level, and the average number of optimal solutions become worse, when the number of decision variables increase. An average level is 25.1% for the MOTP versus 37.9% for the MGP, if search space consists of 1.2396×10^{18} elements and includes 342 758 admissible solutions.

9. Concluding remarks

Tabu programming can be used for finding solution to several problems, especially some multi-criterion optimization problems. A computer program as a tree is a subject of tabu operators such as selection from neighborhood, short-term memory and re-linking of the search trajectory. The MOTP has been applied for operating on the computer procedures written in the Matlab language. Initial numerical experiments confirmed that sub-optimal in Pareto sense, task assignments could be found by tabu programming.

Our future works will focus on testing the other sets of procedures and terminals to find the Pareto-optimal solutions for distinguish criteria and constraints. Moreover, we will concern on a development the combination between tabu search and evolutionary algorithms for finding efficient solutions.

References:

1. Ameljanczyk, A.: Multicriteria Optimization, WAT Press, Warsaw (1986)
2. Balicki, J.: Hierarchical Tabu Programming for Finding the Underwater Vehicle Trajectory, International Journal of Computer Science and Network Security, 7, 32--37 (2007)
3. Balicki, J.: Tabu Programming for Multiobjective Optimization Problems, International Journal of Computer Science and Network Security, 7, 44--50 (2007)
4. Balicki, J.: Immune Systems in Multi-criterion Evolutionary Algorithm for Task Assignments in Distributed Computer System. LNCS, 3528, pp. 51--56, Springer, Heidelberg, (2005)
5. Battiti, R.: Reactive search: Toward self-tuning heuristics, In V. J. Rayward-Smith, editor, Modern Heuristic Search Methods, John Wiley and Sons Ltd, 61--83 (1996)
6. Battiti, R., Tecchiolli, G.: Simulated annealing and tabu search in the long run: a comparison on qap tasks, Computer Math. Applic., 28, 1--8 (1994)
7. Crainic, T. G., Toulouse, M., Gendreau, M.: Toward a Taxonomy of Parallel Tabu Search Heuristics, INFORMS Journal on Computing, 9, 61--72 (1997)
8. Dell'Amico, M., Trubian, M.: Applying Tabu Search to the Job-Shop Scheduling Problem, Annals of Operations Research, 41, 231--252, (1993)
9. Faigle, U., Kern, W.: Some Convergence Results for Probabilistic Tabu Search, ORSA Journal on Computing, 4, 32--38, (1992)
10. Glover, F.: Tabu Search — Part I, ORSA Journal on Computing, 1, 190--206, (1989)
11. Glover, F.: Tabu Search — Part II, ORSA Journal on Computing, 2, 4--32, (1990)
12. Glover, F.: Tabu Search: A Tutorial, Interfaces, 20, 74--94, (1990)
13. Glover, F., Laguna, M.: Tabu Search, Kluwer Academic Publishers, Boston (1997)
14. Hansen, M. P.: Tabu Search for Multicriteria Optimisation: MOTS. Proceedings of the Multi Criteria Decision Making, Cape Town, South Africa, (1997)
15. Hertz, A.: Finding a Feasible Course Schedule Using Tabu Search, Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science, 35 (1992)
16. Jaszkiewicz, A., Hapke, M., Kominek, P.: Performance of Multiple Objective Evolutionary Algorithms on a Distributed System Design Problem – Computational Experiment, LNCS, Vol. 1993, pp. 241--255, Springer, Heidelberg, (2001)
17. Koza, J.R.: Genetic programming. The MIT Press, Cambridge 1992.
18. Lokketangen, A., Jornsten, A. K., Storoy, S.: Tabu Search within a Pivot and Complement Framework, International Transactions in Operations Research, 1, 305--316, (1994)
19. Rego, C.: A Subpath Ejection Method for the Vehicle Routing Problem, Management Science, 44, 1447--1459 (1998)
20. Schaefer, R., Kołodziej, J.: Genetic Search Reinforced by the Population Hierarchy. In De Jong K. A., Poli R., Rowe J. E. (eds): Foundation of Genetic Algorithms, Morgan Kaufman Publisher (2003) 383--399.
21. Weglarz, J., Nabrzyski, J., Schopf, J.: Grid Resource Management: State of the Art and Future Trends. Kluwer Academic Publishers, Boston (2003)
22. Widmer, A. M.: The Job-shop Scheduling with Tooling Constraints: A Tabu Search Approach, J. Opt. Res. S, 42, 75--82 (1991)