Tabu Programming for Multiobjective Optimization Problems

Jerzy Balicki [†],

Naval University of Gdynia, ul. Smidowicza 69, Gdynia, Poland

Summary
In this paper, tabu programming for solving multiobjective optimization problems has been considered. Tabu search algorithm has been extended by using a computer program instead of a mathematical variable. For finding Pareto-optimal solutions, the ranking procedure in the neighborhood of the current solutions has been applied. Moreover, the multiobjective optimization problem of task assignment in a distributed computer system has been studied. Finally, results of some numerical experiments have been presented.

Key words:
Tabu algorithm, efficient solutions, multi-criterion optimization.

1. Introduction

Tabu search algorithm is the alternative approaches to the modern meta-heuristic optimization techniques such as genetic algorithms, evolutionary algorithms, evolution strategies, genetic programming, simulated annealing or Hopfield models of neural networks [3]. In a tabu search, special areas are forbidden during the seeking in a space of all possible combinations [9]. Tabu search can be treated as a general combinatorial optimization technique for using in zero-one programming, non-convex non-linear programming, and general mixed integer optimization [27].

The tabu search algorithm has been applied on combinatorial optimization problems [22]. This technique is basically used to continuous functions by selection a discrete encoding of the problem. A lot of the applications involve traveling salesman, scheduling, and routing problem [16].

Hansen has proposed a multiobjective optimisation tabu search MOTS [17] to generate non-dominated alternatives. The MOTS works with a population of solutions, which, through manipulation of weights, are moved towards the Pareto front [19]. Tabu search can cooperate with a multi-criterion evolutionary algorithm as an additional mutation [2]. During extensive numerical experiments, we notice that tabu search algorithm with distance function from the current solution to the ideal point is capable to find some Pareto-suboptimal solutions [2].

Tabu programming paradigm is implemented as a tabu search algorithm operated on the computer program that produces the current solution. A solution is generated as the program function and then tabu search procedures are applied for finding Pareto-suboptimal solutions.

In this paper, tabu programming for solving multiobjective optimization problems has been considered. Tabu search algorithm has been extended by using a computer program instead of a mathematical variable. For finding Pareto-optimal solutions, the ranking procedure in the neighborhood of the current solutions has been applied. Moreover, the multiobjective optimization problem of task assignment in a distributed computer system has been studied. Finally, results of some numerical experiments have been presented.

2. Tabu search algorithm

The basic concept of tabu search is a meta-heuristic applied to the other heuristic [4]. The overall approach is to avoid entrainment in cycles by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited (hence "tabu"). The method is still actively researched, and is continuing to evolve and improve [13]. The tabu method was partly motivated by the observation that human behavior appears to operate with a random element that leads to inconsistent behavior given similar circumstances. The resulting tendency to deviate from a charted course, might be regretted as a source of error but can also prove to be source of gain [28]. The tabu method operates in this way with the exception that new courses are not chosen randomly. Instead the tabu search proceeds according to the supposition that there is no point in accepting a new (poor) solution unless it is to avoid a path already investigated [23]. This insures new regions of a problems solution space will be investigated in with the goal of avoiding local minima and ultimately finding the desired solution.

The tabu search begins by marching to a local minima. To avoid retracing the steps used, the method records recent moves in one or more tabu lists [20]. The original intent of the list was not to prevent a previous move from being repeated, but rather to insure it was not reversed [12]. The

Manuscript received October 16, 2007
Manuscript revised October 25, 2007

tabu lists are historical in nature and form the tabu search memory. The role of the memory can change as the algorithm proceeds [6]. At initialization the goal is make a coarse examination of the solution space, known as 'diversification', but as candidate locations are identified the search is more focused to produce local optimal solutions in a process of 'intensification'. In many cases the differences between the various implementations of the tabu method have to do with the size, variability, and adaptability of the tabu memory to a particular problem domain [25].

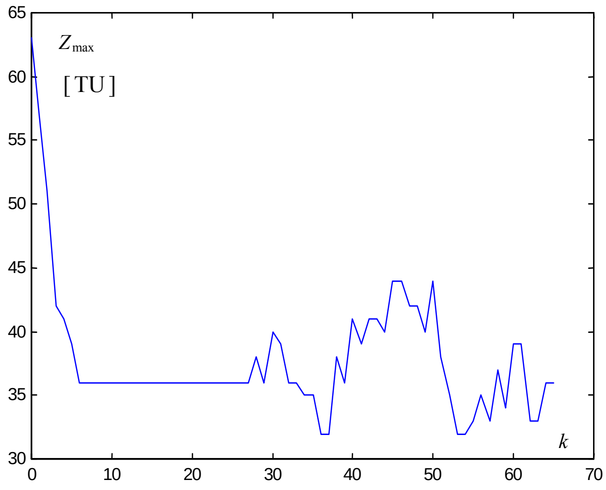
In *a tabu search* special areas are forbidden during the seeking in a space of all possible combinations. Tabu search uses memory structures by reference to dimensions consisting of recency, frequency, quality and influence [21]. It inherits from a simple descent method an idea of a neighbourhood $N(x^{\text{now}})$ of a current solution x^{now}. From this neighbourhood of the current solution, we can choose the next solution x^{next} to a search trajectory [8]. The accepted alternative is supposed to have the best value of an objective function among the current neighbourhood. However, the descent method terminates its searching, when the chosen candidate is worse than the best one from the search trajectory [18].

In the tabu search algorithm based on the short-term memory, a basic neighbourhood $N(x^{\text{now}})$ of a current solution may be reduced to a considered neighbourhood $\boldsymbol{K}(x^{\text{now}})$ because of the maintaining a selective history of the states encountered during the exploration. Some solutions, which were visited during the given last term, are excluded from the basic neighbourhood according to the tabu classification of movements [24]. If any solutions performs aspiration criterion, then it can be included to the considered neighbourhood, only [11].

Let consider the tabu algorithm version called TSZmax [2] that has been designed to find the task assignment with the minimum value of the workload of the bottleneck computer $Z_{\max}$. Figure 1 shows the process of the minimization $Z_{\max}$ from the initial value equal to 62 time units to 32.

2. Tabu programming

Tabu programming is the tabu search algorithm that operates on the dedicated population of computer program. Computer programs are constructed from the basic program that produces the current solution. The basic program can be modeled as a tree (Fig. 2).



by the tabu search algorithm

The size of the generated tree is supposed to be limited by the number of nodes or by the number of the tree levels. The tree nodes are divided on functional nodes and terminal ones. This tree corresponds to the program written in the LISP language, as follows:

$$(LT\ (+\ -2\ x)\ (*\ v\ (SQRT\ v)))$$

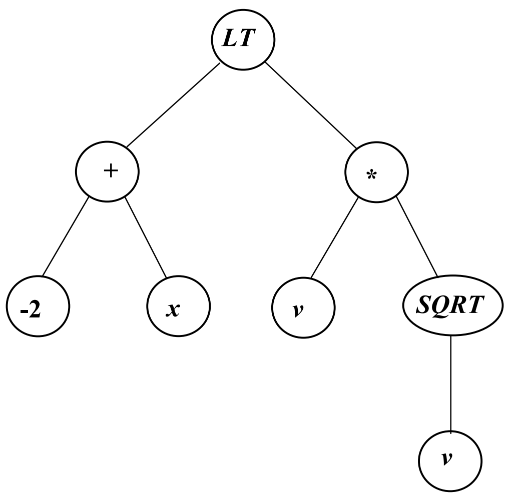


Fig. 2. Tree as a model of the computer program

Above program calculates both the value $-2x$ and $v\sqrt{v}$, and then compare $-2x$ to $v\sqrt{v}$. If $-2x$ is smaller than $v\sqrt{v}$, then an outcome of the LISP procedure is equal to 1. In the other case, the result is –1, because the function LT is defined in such a way.

This tree is equivalent to the parse tree that most compilers construct internally to represent the given computer program. A current solution x^{now} can be produced by this program. A tree can be changed to create the neighborhood $N(x^{now})$ of the current program. We can use the move *in tabu sense* that is related to removing a sub-tree with the randomly chosen node from the parent tree (Fig. 3). Next, the randomly selected node as a terminal is required to be inserted. There are L_1 moves of removing a sub-tree, where L_1 is the number of functional nodes in the tree.

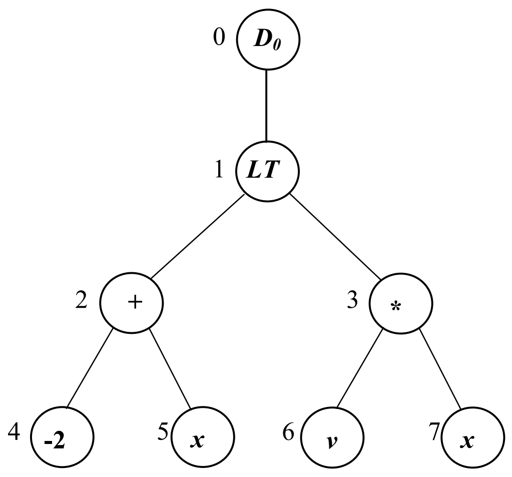


Fig. 3. The computer program tree with the dummy node D_0 after substitution the sub-tree with the root *SQRT* by a node *x*

A functional node represents an elementary procedure randomly chosen from the primary defined set of functions:

$$\mathsf{F} = \{f_1,..., f_n,..., f_N\} \tag{1}$$

Each function should be able to accept, as its arguments, any value and data type that may possible be returned by the other procedure [2]. Because a procedure is randomly chosen from the set, and then it is returned, each function ought to be able to accept, as its arguments, any value and data type that may possible be returned by itself, too. Moreover, 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:

$$\mathsf{T} = \{a_1,..., a_m,..., a_M\} \tag{2}$$

An above condition for the procedures can be called a compatibility requirement because each function should be well defined and closed for any arrangement of arguments that it may come across.

Another condition of a set of procedures, called the sufficiency requirement, postulates that the solution to the problem is supposed to be expressed by the any combination of the procedures from the set of functions and the arguments from the set of terminals. For example, the set of functions $\mathsf{F} = \{AND, OR, NOT\}$ is sufficient to articulate any Boolean function. If the logical operator *AND* is removed from this set, the remaining procedure set is still satisfactory for implementation any Boolean function. In addition, a sufficient set is $\{AND, NOT\}$ as well.

Another sort of movements in tabu programming is related to removing the randomly chosen terminal node and then adding a sub-tree with the functional node as a root. There are L_2 movements of adding a sub-tree. That sub-tree can be constructed from the random number of nodes.

The basic neighborhood of the current solution consists of $L= L_1+ L_2$ solutions obtained from L programs. We introduce some limitation of nodes in the parse tree L_{max} that ensures the reasonable number of nodes in the program tree and the number of solutions in the neighborhood.

3. Short-term memory

From this neighborhood of the current solution, we can choose the next solution x^{next} to a search trajectory of a tabu programming [15]. The short-term memory may reduce a basic neighborhood $N(x^{now})$ to a considered neighborhood $K(x^{now})$ because of the maintaining a selective history of the states encountered during the exploration [14]. Some solutions, which were visited during the given last term, are excluded from the basic neighborhood according to the tabu classification of movements.

If the node is the root of the reducing sub-tree for the current program, it can be protected against choosing its to be that root in a reducing operation until the next λ_1 movements is 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 consider a dummy node D_0 (Fig. 3) 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 3 can be represented as below:

$$\omega = \left(D_0, LT, +, *, -2, x, v, x\right) \tag{3}$$

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

$$\psi = \left(f, f, f, f, a, a, a, a\right) \tag{4}$$

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

$$\chi = \left(1,2,2,2,0,0,0,0\right) \tag{5}$$

Now, we can introduce the matrix of reducing node memory $M^{-} = \left[m_{nm}\right]_{L_{\max} \times L_{\max}}$, 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 $M^{+} = \left[\widetilde{m}_{nm}\right]_{L_{\max} \times L_{\max}}$, where $\widetilde{m}_{nm}$ 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, $\widetilde{m}_{nm} = \lambda_2$.

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.

If any solutions performs aspiration criterion, then it can be included to the considered neighborhood [15]. The accepted alternative is supposed to have the best value of an objective function among the current neighborhood.

3. Algorithm of the tabu programming

Tabu programming rules can be implemented as an algorithm ATP (Fig. 4) that can be used for optimisation.

```
1. Initial procedure   k:=0
    (A)  Generation of the program that produces $x^{now}$
    (B)  $x^{best} := x^{now}$ , $x^{bis} := x^{now}$
    (C)  $F_{\min} := F(x^{now})$
    (D)  Initialization of restriction matrixes $M^{+}$, $M^{-}$
    (E)  Setting $\lambda_1$, $\lambda_2$
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
         neighbourhood are tabu-active and $F_{\min} \geq F(x^{now})$,
         then $x^{best} := x^{now}$, $F_{\min} := F(x^{now})$
    (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 F 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 $F(x^{now}) < F_{\min}$, 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 $M^{+} := M^{+} - 1$,
         $\widetilde{m}_{nm} = \lambda_2$.
    (E)  go to 2
```

Fig. 4. An algorithm ATP of tabu search programming

ATP can be used for solving an optimization problem with one criterion, as follows:

$$F_{\min} = \min_{x \in X} F(x) \tag{6}$$

where

F – criterion f the problem,

X – set of admissible solutions.

The selection function W is constructed from the criterion F and functions describing constraints [7]. Usually, the penalty function can be applied.

3. Tabu programming for multi-criterion optimization

A ranking idea for non-dominated individuals has been introduced to avoid the prejudice of the interior Pareto alternatives [2]. It was developed for genetic algorithm. To adjust ATM for solving multi-criterion problems a ranking procedure can be applied for sorting solutions in the neighborhood of the current solution.

If some admissible solutions are in the neighborhood, then the Pareto-optimal solutions are determined, and after that they get the rank 1. Subsequently, they are temporary eliminated from the neighborhood. Next, the new Pareto-optimal alternatives are found from the reduced neighborhood and they get the rank 2. In this procedure, the level is increased and it is repeated until the set of admissible solutions is exhausted. All non-dominated alternatives have the same value of the selection function *W* because of the equivalent rank. Then, from the solutions with the rank 1, the best one is accepted according to the distance criterion to an ideal point with the optimal values of coordinates.

Tabu search uses memory structures by reference to dimensions consisting of recency, not only [5]. Moreover frequency, quality and influence aspects can be considered and the capabilities of the ATM can be extended [10].

Let (X, F, P) be the multi-criterion optimisation problem for finding a Pareto-optimal task assignment in the distributed computer system [26]. That benchmark problem can be established, as follows:

1) X - an admissible solution set

$$\mathsf{X} = \{x \in \mathsf{B}^{I(V+J)} \times \mathsf{V}^{V} \mid$$

$$\min_{i=\overline{1,I}} \left\{ \sum_{v=1}^{V}\sum_{i=1}^{I} t_{vi} x_{vi}^{m} x_{ij}^{\pi} + \sum_{v=1}^{V}\sum_{\substack{u=1 \\ v\neq u}}^{V} \tau_{vu} x_{vi}^{m} x_{ij}^{\pi} \right\} \le Z_{\max}^{\lim};$$

$$\sum_{v=1}^{V} c_{vr} x_{vi}^{m} \le \sum_{j=1}^{J} d_{jr} x_{ij}^{\pi}, \quad i=\overline{1,I}, \ r=\overline{1,R};$$

$$\sum_{i=1}^{I}\sum_{j=1}^{J} \kappa_j x_{ij}^{\pi} \le \kappa_{\max}; \ \sum_{i=1}^{I}\sum_{j=1}^{J} \vartheta_j x_{ij}^{\pi} \ge \vartheta_{\max}; \tag{7}$$

$$1 \le N_v \le V, v=\overline{1,V};$$

$$\sum_{i=1}^{I} x_{vi}^{m} = 1, v=\overline{1,V}; \ \sum_{j=1}^{J} x_{ij}^{\pi} = 1, \ i=\overline{1,I} \ \}$$

where

$\mathsf{B}=\{0, 1\}$,

$\mathsf{V}=\{1, 2,\ldots,V\}$

$$x = (x^{m}, x^{\pi}, N^{m})$$

$$(x^{m},x^{\pi}) = [x_{11}^{m},\ldots,x_{1I}^{m},\ldots,x_{vi}^{m},\ldots,x_{VI}^{m},x_{11}^{\pi},\ldots,x_{1J}^{\pi},\ldots,x_{ij}^{\pi},\ldots,x_{I1}^{\pi},\ldots,x_{Ij}^{\pi},\ldots,x_{IJ}^{\pi}]^{T}.$$

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

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

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

$\{T_1,\ldots,T_v,\ldots,T_V\}$ - the set of parallel performing tasks,

$W = \{w_1,\ldots,w_i,\ldots,w_I\}$ - the se of the processing nodes,

$N^{m} = [N_1,\ldots,N_v,\ldots,N_V]^{T}$,

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

t_{vj} - the overhead performing time of the task T_v by the computer π_j.

τ_{vu} – the total communication time between the task T_v and the T_u,

$z_1,\ldots,z_r,\ldots,z_R$ - memories available in the system,

d_{jr} - the capacity of memory z_r in the workstation π_j,

κ_j - the cost of the computer π_j,

ϑ_j - the numerical performance of the computer π_j for the given benchmark,

2) F - a vector superiority criterion

$$F : \mathsf{X} \to \mathsf{R}^{2}, \tag{8}$$

where

R – the set of real numbers,

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

$$R(x) = \prod_{v=1}^{V}\prod_{i=1}^{I}\prod_{j=1}^{J} \exp(-\lambda_j t_{vj} x_{vi}^{m} x_{ij}^{\pi}),$$

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

λ_j - rate of failing for the computer π_j that can be failed independently due to an exponential distribution,

d_v - the completion deadline for the vth task,

C_v - the completion time for the vth task,

If $C_v \le d_v$, $\xi(d_v - C_v) = 1$.

If $C_v > d_v$, $\xi(d_v - C_v) = 0$.

K – number of instances in the flow graph

3) P - the Pareto relation [1].

The relationship P is a subset of the product $Y \times Y$, where an evaluation set $Y=F(X)$. If $a \in Y$, $b \in Y$, and $a_n \le b_n, n = \overline{1,N}$, then the pair of evaluations $(a,b) \in P$. The meaning of the Pareto relationship respects the minimization of all criteria. That is why, criteria for maximization are written with minus. There is no task allocation $a \in X$ such that $(F(a),F(x^*)) \in P$ for the Pareto-optimal assignment $x^* \in X$ and $a \neq x^*$.

4. Implementation and numerical experiments

Each program consists of set of procedures and set of attributes. Set of procedures is defined, as follows:

$$\mathsf{F} = \{list, +, -, *, /\} \quad (9)$$

where

list – the procedure that convert $I(V+J)+V$ input real numbers called *activation levels* on $I(V+J)$ output binary numbers $x_{11}^m, \ldots, x_{1I}^m, \ldots, x_{vi}^m, \ldots, x_{VI}^m, x_{11}^\pi, \ldots, x_{1J}^\pi, \ldots, x_{ij}^\pi, \ldots, x_{I1}^\pi, \ldots, x_{Ij}^\pi, \ldots, x_{IJ}^\pi$ and V output integer numbers $N_1, \ldots, N_v, \ldots, N_V$.

The procedure *list* is obligatory the root of the program tree and appears only one in a generated program. An activation level is a root for the sub-tree that is randomly generated with using arithmetic operators {+, -, *, /} and the set of terminals.

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

$$\mathsf{T} = \mathsf{D} \cup \mathsf{L}, \quad (9)$$

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

If x calculated by the program is admissible, then the selection function value is estimated, as below:

$$f(x) = r_{\max} - r(x) + P_{\max} + 1, \quad (10)$$

where $r(x)$ denotes the rank of an admissible solution,

Let the Pareto points $\{P_1, P_2, \ldots, P_U\}$ be given for any instance of the task assignment problem. If the AMEA/GP finds the efficient point (A_{u1}, P_{u2}) for the probability that tasks meet deadlines P_{u2}, this point is associated to the uth Pareto result (P_{u1}, P_{u2}) with the same value of probability.

The distance between points (A_{u1}, P_{u2}) and (P_{u1}, P_{u2}) is calculated according to an expression $|P_{u1} - A_{u1}|$. If the point (A_{u1}, P_{u2}) is not discovered by the algorithm, we assume the distance is $|P_{u1} - A_{u1}^{\min}|$, where $A_{u1}^{\min}$ is the minimal reliability of the system for the instance of problem.

The level of convergence to the Pareto front is calculated, as follows:

$$S = \sum_{u=1}^{U} |P_{u1} - A_{u1}|. \quad (11)$$

An average level $\overline{S}$ is calculated for several runs of the evolutionary algorithm. Initial numerical examples indicated that obtained task assignments had higher value of the workload of the bottleneck computer than the limit for some instances with the number of tasks larger than 15.

This tabu programming ATM gives better results than the genetic programming AMEA/GP (Fig. 5). After 200 selections, an average level of Pareto set obtaining is 1.9% for the ATM, 3.3% for the AMEA/GP. 30 test preliminary populations were prepared, and each algorithm starts 30 times from these populations. For integer constrained coding of chromosomes there are 12 decision variables in the test optimization problem. The search space consists of 25 600 solutions.

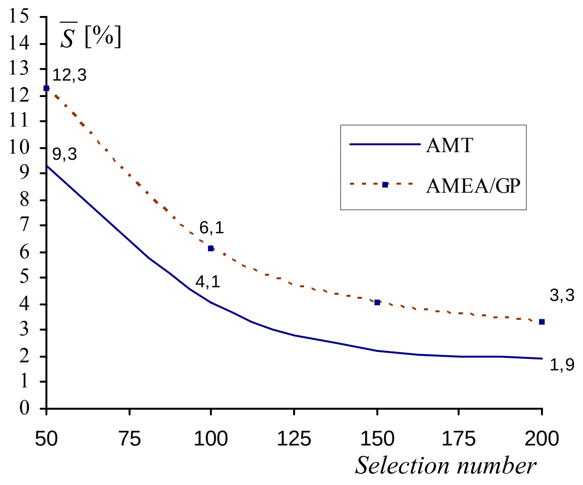


Fig. 5. Outcome convergence for the AMT and the AMEA/GP

For the other instance with 15 tasks, 4 nodes, and 5 computer sorts there are 80 binary decision variables. An average level of convergence to the Pareto set is 16.2% for the AMT and 18.4% for the AMEA/GP. A maximal level is 26.5% for the AMT and 29.6% for the AMEA/GP. For this instance the average number of optimal solutions is 19.2% for the AMT and 21.1% for the AMEA/GP.

An average level of convergence to the Pareto set, an maximal level, and the average number of optimal solutions become worse, when the number of task, number of nodes, and number of computer types increase. An average level is 33.6% for the AMT versus 35,7% for the AMEA/GP, if the instance includes 50 tasks, 4 nodes, 5 computer types and also 220 binary decision variables.

8. Concluding remarks

Tabu programming is new paradigm of artificial intelligence that can be used for finding solution to several 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. It gives possibility to represent knowledge that is specific to the problem in more intelligent way than for the data structure. That is, we process the potential ways of finding solution not the possible solutions. A tabu programming has been applied for operating on the computer procedures written in the Matlab language.

Our initial numerical experiments confirm that feasible, sub-optimal in Pareto sense, task assignments can be found by tabu programming. A paradigm of tabu programming gives opportunity to solve this problem for changeable environment.

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

References

[1] A. Ameljańczyk, *Multicriteria optimization,* WAT, Warsaw 1986.

[2] J. Balicki, *Negative selection with ranking procedure in tabu-based multi-criterion evolutionary algorithm for task assignment*, Lecture Notes in Computer Science. Vol. 3993, 2006, pp. 863-870.

[3] R. Battiti, *Reactive search: Toward self-tuning heuristics*, in V. J. Rayward-Smith, editor, *Modern Heuristic Search Methods*, John Wiley and Sons Ltd, 1996, pp. 61-83.

[4] R. Battiti, G. Tecchiolli, *The Reactive Tabu Search*, *ORSA Journal on Computing*, Vol. 6, No. 2, 1994, pp. 126-140.

[5] R. Battiti, G. Tecchiolli, *Simulated annealing and tabu search in the long run: a comparison on qap tasks*, Computer Math. Applic., Vol. 28, No. 6, 1994, pp. 1-8.

[6] T. G. Crainic, M. Toulouse and M. Gendreau, *Toward a Taxonomy of Parallel Tabu Search Heuristics*, INFORMS Journal on Computing, Vol. 9, No. 1, 1997, pp. 61-72.

[7] M. Dell'Amico, M. Trubian, *Applying Tabu Search to the Job-Shop Scheduling Problem*, Annals of Operations Research, Vol. 41, 1993, pp. 231-252.

[8] U. Faigle, W. Kern, *Some Convergence Results for Probabilistic Tabu Search*, *ORSA Journal on Computing,* Vol. 4, No. 1, 1992, pp. 32-38.

[9] F. Glover, *Heuristics for Integer Programming Using Surrogate Constraints*, *Decision Sciences*, Vol. 8, No. 1, 1977, pp. 156-166.

[10] F. Glover, *Tabu Search — Part I*, *ORSA Journal on Computing*, Vol. 1, No. 3, 1989, pp. 190-206.

[11] F. Glover, *Tabu Search — Part II*, ORSA Journal on Computing, Vol. 2, No. 1, 1990, pp. 4-32.

[12] F. Glover, *Tabu Thresholding: Improved Search by Nonmonotonic Trajectories*, *ORSA Journal on Computing*, Vol. 7, No. 4, 1995, pp. 426-442.

[13] F. Glover, *Future paths for Integer Programming and Links to Artificial Intelligence*, Computers and Operations Research*,* Vol. 5, pp. 533-549, 1986.

[14] F. Glover, *Tabu Search: A Tutorial*, *Interfaces*, Vol. 20, No. 4, 1990, pp. 74-94.

[15] F. Glover, M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston 1997

[16] F. Glover, Laguna M., Tabu Search, in *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves, editor, John Wiley & Sons, Inc, 1993

[17] M. P. Hansen, *Tabu Search for Multicriteria Optimisation: MOTS*. Proceedings of the Multi Criteria Decision Making, Cape Town, South Africa, 1997

[18] A. Hertz, *Finding a Feasible Course Schedule Using Tabu Search*, Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*,* Vol. 35, 1992.

[19] A. Jaszkiewicz, M. Hapke, P. Kominek, *Performance of Multiple Objective Evolutionary Algorithms on a Distributed System Design Problem – Computational Experiment.* Lectures Notes in Computer Science, Vol. 1993, 2001, pp. 241-255.

[20] A. M. Laguna, A J. W. Barnes, A F. Glover, *Tabu Search Methodology for a Single Machine Scheduling Problem*, J. of Int. Manufacturing, Vol. 2, 1991, pp. 63-74.

[21] A. M. Laguna, A J. L. Gonzalez-Velarde, *A Search Heuristic for Just-in-time Scheduling in Parallel Machines*, *J. of Int. Manu.*, Vol. 2, 1991, pp. 253-260,

[22] A. Lokketangen, A. K. Jornsten and S. Storoy, *Tabu Search within a Pivot and Complement Framework*, International Transactions in Operations Research, Vol. 1, No. 3, 1994, pp. 305-316.
[23] A S. C. S. Porto, A C.C. Ribeiro, *Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling under Precedence Constraints*, Journal of Heuristics, Vol. 1, 1995
[24] A. S. C. S. Porto, A C.C. Ribeiro, *A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints*, International Journal of High-Speed Computing, Vol. 7, No. 2, 1995
[25] C. Rego, *A Subpath Ejection Method for the Vehicle Routing Problem*, Management Science, Vol. 44, No. 10, 1998, pp. 1447-1459.
[26] J. Węglarz, *Recent Advances in Project Scheduling.* Kluwer Academic Publishers, Dordrecht 1998.
[27] A. M. Widmer, *The Job-shop Scheduling with Tooling Constraints: A Tabu Search Approach*, J. Opt. Res. S, Vol. 42, 1991, pp. 75-82
[28] A. M. Widmer, A A. Hertz, *A New Heuristic Method for the Flow Shop Sequencing Problem*, Euro. J. Opt. Res., Vol. 41, 1989, pp. 186-193

Jerzy Balicki received the M.S. and Ph.D. degrees in Computer Science from Warsaw University of Technology in 1982 and 1987, respectively. During 1982-1997, he stayed in Computer Center of Maritime High School of Gdynia to study management systems, mobile systems, and decision support systems. Then, he achieved habilitation from Technical University of Poznan in 2001. He was admitted as a university professor at Naval University of Gdynia in 2002.