Some Numerical Experiments on Multi-criterion Tabu Programming for Finding Pareto-optimal Solutions

JERZY BALICKI
Naval University of Gdynia
ul. Smidowicza 69, 81-103 Gdynia,
POLAND
J.Balicki@amw.gdynia.pl

Abstract: - Decision making for complex systems is based on multi-criterion-optimization. A decision making support can be applied to find the Pareto solutions. Multi-criterion tabu programming is a new paradigm for that task. Similarly to rules applied in the genetic programming, tabu programming solves problems by using a tabu algorithm that modifies some computer programs. We consider the multi-criterion problem of task assignment, where both a workload of a bottleneck computer and the cost of system are minimized; in contrast, a reliability of the distributed system is maximized. Furthermore, there are constraints for the performance of the distributed systems and the probability that all tasks meet their deadlines. What is more, 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

There are several artificial intelligence techniques that can be applied to solve some multi-criterion optimization problems. Genetic algorithms, artificial neural networks, simulated annealing, tabu search and artificial immunological systems are crucial paradigms for a computer aid of decision making.

Tabu programming is a relatively new paradigm of artificial intelligence that can be applied for computer decision aid. Similarly to the genetic programming that applies a genetic algorithm [28], tabu programming solves problems as 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 [15, 17]. Some efficient task scheduling algorithms that are based on tabu search are proposed by Węglarz in [37]. This optimization technique can be used to continuous functions by a selection a discrete encoding of the problem [27, 30].

We have observed that the multi-criterion tabu algorithm gave better quality results than multi-criterion evolutionary algorithm, and that fact inspired us to create new paradigm of programming based on a tabu algorithm. 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 [29, 31]. In a tabu programming, special areas for possible modification of programs are forbidden during the seeking in a space of all possible combinations [18]. In opposite to a genetic programming, tabu programming deals with one computer program at the current moment, instead of the set of procedures at the genetic approach.

The first tabu programming for multi-criterion optimization has been presented by Balicki in 2007 [3]. That 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 [4].

Then, an improved MOTB for solving multi-criterion with constraints optimization problems of task assignment in the distributed computer system has been considered [3]. The sub-effective task assignment has been obtained by development that approach.

In this paper, we consider another multi-criterion problem of task assignment, where both a workload of a bottleneck computer and the cost of system are minimized. Furthermore, there are constraints for the performance of the distributed systems and the probability that all tasks meet their deadlines. What is more, constraints related to memory limits and computer locations are imposed on the feasible task assignment as well as a reliability of the distributed system is maximized. Finally, some results of some numerical experiments have been presented.

2 Rules 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, some aspects of multi-criterion optimization are respected.

The tabu programming operates on the computer program which produces an outcome that can be treated as a solution to the problem. Because in the computer program several modifications may be carried 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 working 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 [19]. This insures new regions of a solution space will be explored in with the goal of avoiding local minima and finding the global minimum [11, 35].

To keep away from repeating the steps, recent moves are recorded in some tabu lists [5]. That lists forms the short-term memory. The memory content can vary as the search proceeds [9]. At the beginning, the target is testing the solution space, during a 'diversification' [6]. 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 [20].

Special areas are forbidden during the seeking in a search space. From that neighborhood $N(x^{now})$ 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 [7]. 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 $\mathcal{K}(x^{now})$ because of the maintaining a selective history of the states encountered during the exploration [23, 38]. Some solutions, which were visited during the given last term, are excluded from the basic neighborhood according to the classification of movements [8, 32]. If any solution satisfies an aspiration criterion, then it can be included to the considered neighborhood, only [27, 34].

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 $N(x^{now})$ 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 [12]:

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

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

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

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 set and argument sets

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

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

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 $x_m \in \mathcal{B}, m=\overline{1,M}$ for $\mathcal{B}=\{0, 1\}$, 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. So, the tree from Figure 1 could be a sub-tree that calculates one of activation levels.

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 [10]:

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

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 $\mathcal{D}$ 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]:

$$\mathsf{T}=\mathcal{D}\cup\mathcal{L}, \quad (5)$$

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

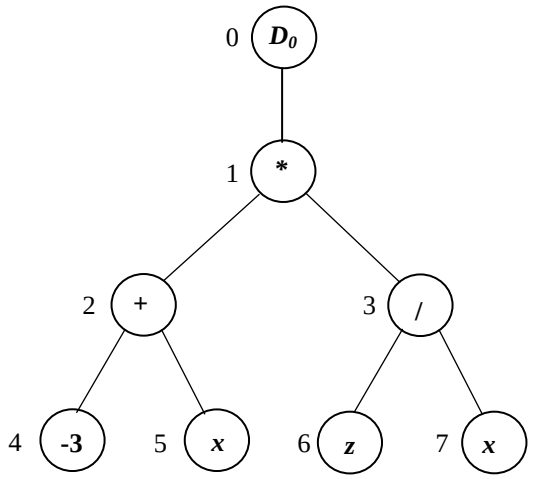

Fig. 1. The program tree for procedure *(x-3)*z/x*

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 an 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:

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

A neighborhood is generated by re-building the current program (Fig. 2). 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. However, that node can be selected to be the root for adding the sub-tree. If the node is the root of the adding tree, it can be protected against choosing it to be that root in a adding operation until the next λ_2 movements.

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, $\tilde{m}_{nm} = \lambda_2$.

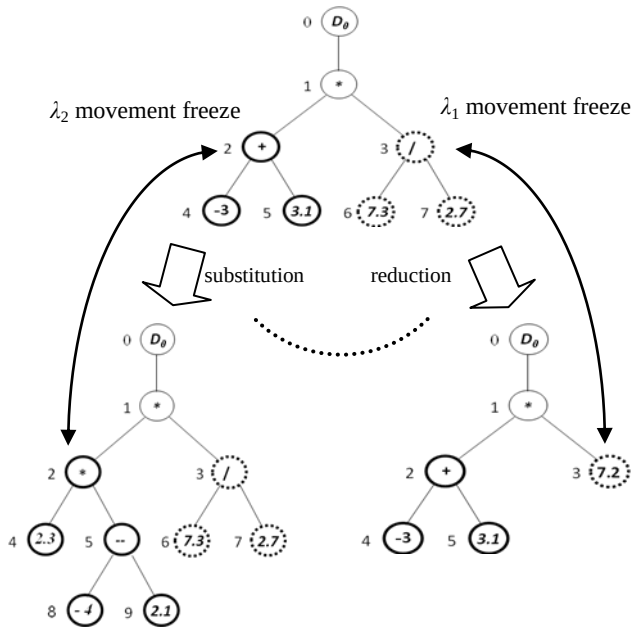

Fig. 2. A neighborhood for the current program

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 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:

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

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 [13]. Usually, the penalty function can be applied [24, 33].

Figure 3 shows an outlook of the algorithm MOTP. At the beginning, the first computer program is generated by the control program that is the implementation of the multi-criterion tabu algorithm [3]. User of the MOTP is obligated to set the input data and some parameters, only. The MOTP has been written in the Matlab language [3].

The first computer program calculates the vector of decision variables x^{now}. This program can be written as a *s-expression* in Common Lisp like:

```
(GT (* -1 x) (* v (ABS v)))
```

That s-expression can be written as a Pascal function:

```
function u(x,v:real):real;
begin
if (-x > v*abs(v)) then u:=1
                   else u:= -1;
end;
```

1. Initial procedure $k:=0$
- (A) Read some input data to the problem
- (B) Set up constraint program parameters μ_1 , μ_2
- (C) Generation of the program that produces x^{now}
- (D) $x^{best} := x^{now}$, $x^{bis} := x^{now}$
- (E) $K_{min} := K(x^{now})$
- (F) Initialization of restriction matrixes M^+, M^-
- (G) Setting the memory parameters λ_1, λ_2

2. Solution selection and stop criterion $k:=k+1$
- (A) Finding a set of tree candidates $\mathcal{X}(M^+, M^-, x^{now})$ from the neighborhood $N(x^{now})$
- (B) Selection of the next solution $x^{next} \in \mathcal{X}(M^+, M^-, x^{now})$ with the minimal value of the selection function W among solutions taken from $\mathcal{X}$
- (C) **Aspiration condition.** If all solutions from the neighborhood are tabu-active and $K_{min} < 0.8K(x^{now})$, then $x^{next} := x^{now}$
- **(D)** **Re-linking of search trajectory.** If x^{next} was not changed during main iteration, then a genetic 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 $k = 0.4\, T_{max}$, then $\lambda_1 := 4\lambda_1$, $\lambda_2 := 4\lambda_2$
- (F) If $k = T_{max}$ 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. 3. An algorithm MOTP

Because a program function is modeled by a rooted, point-labeled tree with ordered branches, then a size of program is described by μ_1 – the number of tree nodes. Figure 1 shows the tree with 8 tree nodes. Moreover, μ_2 – the number of the tree levels is another constraint parameter for the program tree. There are 4 tree levels on Figure 1. Parameters μ_1 as well as μ_2 are supposed to be

set at the beginning of tabu programming. The size, structure and contents of a program may be dynamically changed during evolution. The program size is constrained by the maximal number of tree nodes or the maximal number of the tree levels

Parameters of the short-term memory are increased after 40% of the all iterations to avoid falling in cycles.

A paradigm of tabu programming gives opportunity to solve the several problems. 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.

6. Criteria for benchmark problem

To test the ability of the MOTP, we consider a multi-criterion optimization 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]:

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

where

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

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 4 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.

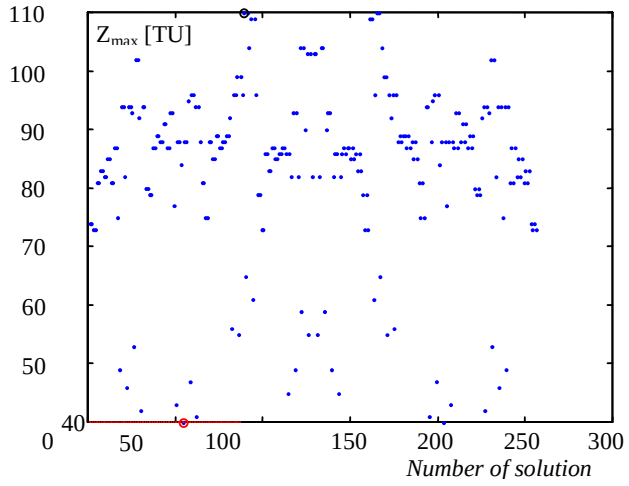

Fig 4. Workload of the bottleneck computer for generated solutions.

Figure 5 shows task assignment that minimize the workload of the bottleneck computer among four computers. The task number 7 is characterized by the larger value of the time workload than tasks *v*, 5, and 11, then the bottleneck computer could be the computer with task 7. However, we shall consider the other workloads related to the computers at the nodes 2 and 4. At the node number 4, there are more tasks than at the node 2, but the workload can be smaller. So, the bottleneck computer is determined by the values of time processing and some values of communication times.

Figure 6 shows three cuts in task assignment graph. Tasks 7, 8, and 9 generate the workload 32 TU at the node number 1. The same value of workload is assigned to the node no. 2 by tasks 1,3,4, and 10. On the other hand, tasks 2, 5, and 6 charge the computer at the node 3 by the smaller value 31 [TU]. In that case, there are two bottleneck computers that are situated at the nods 1 and 2.

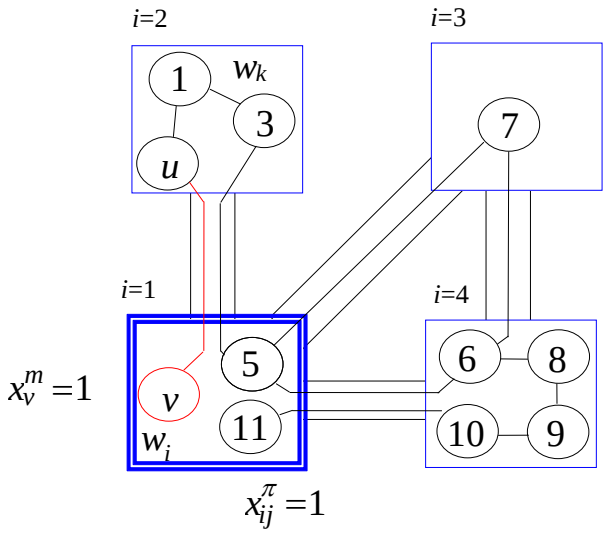

Fig. 5. Assignment of tasks to computers for the given set of tasks and the set of computer sorts

If the task number 9 is moved to the node 2, then the bottleneck computer is at that node. However, the workload of the bottleneck computer is equal to 37 TU. To sum up, we can balance workload among several processors by finding an optimal value of the bottleneck computer.

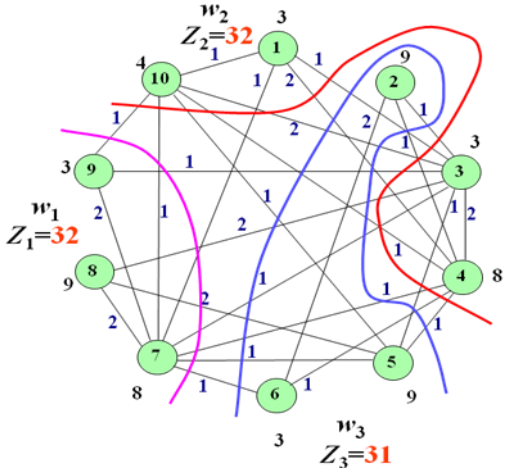

Fig. 6. Load balancing by finding an optimal task assignment

Figure 7a) shows the process of the minimization $Z_{\max}$ from the initial value equal to 62 time units to 32. Similarly, Figure 7b) shows the process of the minimization $Z_{\max}$ from the initial value equal to 170 time units to 101. The solution with the value 170 was calculated by a randomly generated program.

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]:

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

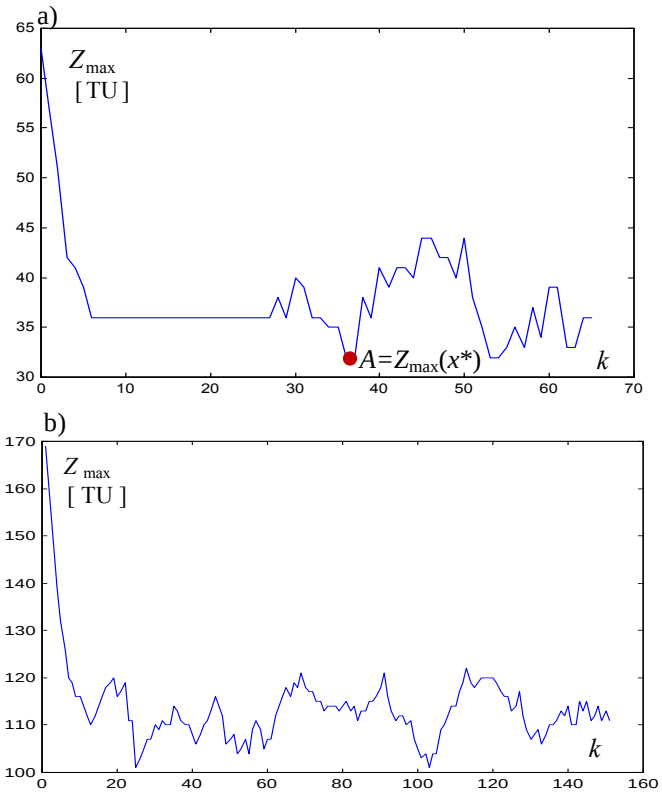

Fig. 7. Minimizations of the bottleneck computer workload:
a) finding the task assignment x^* from Figure 6
b) searching an optimal task assignment for 20 tasks

7. Constraints and decision variables

The minimal performance of the distributed systems $\Theta_{\min}$ is supposed to be smaller (Fig. 8) than the performance of the entire system 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, for instance [MFlops].

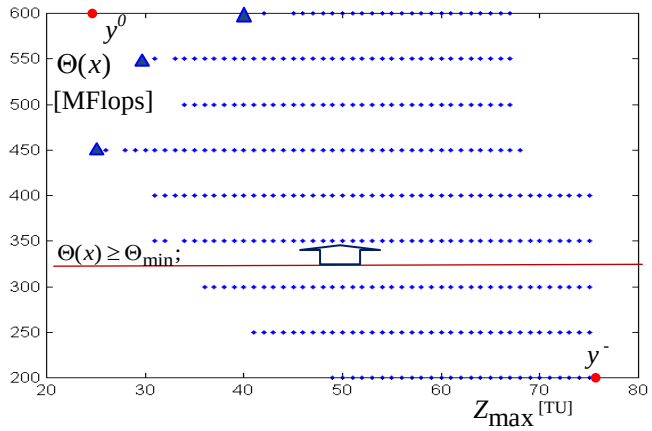

Fig. 8. Set of assignment evaluation for two criteria: the load of the bottleneck computer and the system performance

Figure 8 shows a set of assignment evaluation for two criteria: the load of the bottleneck computer and the system performance. If we consider the constraint imposed on the performance of the system, the upper part of the set is supposed to be considered. An important role plays the preferences of a decision maker. There is possible to think about maximization the benchmark performance and then points marked by triangles are preferred. Another preference situation is if we consider admissible solutions due to the performance constraint. In that case, we prefer all admissible solutions that are characterized by points from the upper part of the criterion space.

An ideal point y^o is generated by finding optimized values of separated criteria that are included as coordinates of this point [1]. In this case, we prefer a maximal value of the benchmark performance and a minimal value of the bottleneck computer load. In the opposite way, an anti-ideal point is constructed. A minimal value of the benchmark performance and a maximal value of the bottleneck computer load are taken as coordinates of y^-.

The probability that all tasks meet their deadlines is supposed to be greater than the minimal probability P_{min}. This parameter is usually set more than 0.9. 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 Π_n may begin its running after λ_n and completes before the program deadline δ_n. Moreover, we assume a conditionally running task is performed with the frequency q_v and its complementary task – with the frequency $(1\text{-}q_v)$. Those parameters are estimated by the experimental way for the representative set of the input data. So, the conditional frequency q_v is related with the task v. If the vth task is not conditional one, then q_v=1.

A task no. v can be performed in the loop k times (k=1, 2, ..., L^v_{max}), and each repetition of this task is performed with the probability p. The instance, where the loop task no. v runs k times, can be meet with the probability $(1-p_v)\,p_v^{\,k-1}$. We assume, that a continue section is performed at the end of the loop. This situation is in the instruction *while* in *C/C++/Java* language. The k-1 iterations are finished by the producing true for the continue section of the loop. The coefficient $(1-p_v)$ is the result of the output from the loop.

The instance, where the conditional task no. v appears and the loop task no. u runs k times, occurs with the probability:

$$p_l=q_v\,(1\text{-}\,p_u)\,p_u^{\,k-1}, \qquad k=1, 2, ..., L^v_{max} \qquad \textbf{(14)}$$

Similarly, the instance, where the conditional tasks no. v and w appear and the loop task no. u runs k times, occurs with the probability:

$$p_l=q_v\,q_w\,(1\text{-}\,p_u)\,p_u^{\,k-1}, \qquad k=1, 2, ..., L^v_{max}$$

The right formula for calculating probability of the instance is related to the structure of the flow graph that is a model of the program module communications.

Times of task completions $(C_1,...,C_v,...,C_V)$ can be calculated for scheduled allocation modules to computers $x=(x^m,x^\pi,N^m)$ and the preceding relation taken from the flow graph [4]. Let d_v represents the given completion deadline for the vth task. This completion deadline is known. If $C_v \leq d_v$, 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. We assume that with the flow graph instance no. l is associated a set of tasks M_l. Probability that all tasks meet their deadlines for K instances of the flow graph is calculated, as below:

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

Two main constraint types: the benchmark performance limit and also probability that all tasks meet their deadlines are supposed to be complement with some resource constraint.

Constraints related to memory capacities are related to the assumption that a computer is supposed to be equipped with some 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 run. The memory limit R_{ir} of the rth resource

in a machine cannot be exceeded in the *i*th node, what is written, as bellows:

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

To sum up, there are three sorts of limitations that should be considered. An admissible distributed task scheduling is supposed to ensure enough high value of the task performance and the probability that all tasks meet their deadlines. Moreover, some resource requirements should be guaranteed.

8. Problem formulation

Task assignment problems are usually formulated as one-criterion optimization questions. However, there are some disadvantages of those form of a problem. First of all, not all criteria are respected and more complex approach cannot be considered. If some conflict preferences are supposed to be considered, then a multi-criterion approach is much more adjusted to the decision making situation. The high-quality computers are characterized by the higher costs, and we plan to decrease that parameter. In contrast, some performance parameters are much better for rather expensive machines. Similarly, the reliability of the computer system is in conflict with the cost of high quality components. Usually, some expensive servers are characterized by the longer time between failures that can cause the breakdown of the whole system. Sometimes, the failure of a hard disc may not cause the damage of the system because the mirror element can be applied to increase the reliability of the operations.

A conflict appears between a numerical performance of the computer system and the reliability, too. Let us consider some computer clusters. They are usually deployed to improve a numerical performance or an availability of the whole distributed system. What is more, they are much more cost-effective than the other computers of comparable speed or availability.

High-availability clusters (also known as failover clusters) are implemented for improving the availability of network services. Some redundant nodes are applied to make available service when the system components fail. A high-availability cluster with two nodes is the minimum requirement to give redundancy. That cluster implementations attempt to manage the redundancy inherent in a cluster to eliminate single points of failure [25].

There are some commercial implementations of high-availability clusters for distinguish operating systems. The Linux-HA project is free software package for the Linux. They can be used for the Research on Adaptive Learning [36] or in Web Information Management Processing [39].

Load-balancing clusters operate by distributing a workload evenly over multiple nodes. Typically the cluster will be configured with multiple redundant load-balancing front ends.

Grids typically support more heterogeneous sets of processors than are commonly supported in clusters. Grid computing is optimized for tasks which consist of many independent jobs, which do not have to share data between them during the computation process. Grids serve to manage the allocation of jobs to computers which will perform the work independently of the rest of the grid cluster. Resources such as storage may be shared by all the nodes, but intermediate results of a job do not affect other jobs in progress on other nodes of the grid [4].

An interesting project of a large grid is called the *Folding@home*. That grid analyzes data that are used to find cures for diseases like a cancer or *Alzheimer's*. Another fascinating project is the *SETI@home*, which is one of the largest distributed grid. It uses more or less three million PCs all over the world to analyze data from the radio-telescope from Arecibo Observatory that searches for evidence of extraterrestrial intelligence.

A benchmark is a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it. Benchmarking is usually associated with assessing performance characteristics of computer hardware, for example, the floating point operation performance of a CPU, but there are circumstances when the technique is also applicable to software. Software benchmarks are, for example, run against compilers or database management systems. Another type of test program, namely test suites or validation suites, are intended to assess the correctness of software.

Benchmarks provide a method of comparing the performance of various subsystems across different chip/system architectures. Benchmarking is helpful in understanding how the database manager responds under varying conditions. It can be created scenarios that test deadlock handling, utility performance, different methods of loading data, transaction rate characteristics as more users are added, and even the effect on the application of using a new release of the product.

The list of the fastest computers usually includes many clusters. Clustering can provide noteworthy performance benefits versus price. Currently, the fastest computer is the IBM Roadrunner system from the Department of Energy's with performance of 1026 TFlops measured with Linpack benchmark [3]. One of supercomputers that is characterized by 12 TFlops, consists of 1100 dual-processor machines (4 GB RAM) and runs on Mac OS X. Processors are connected by the InfiniBand interconnect.

A Beowulf cluster is the cost-effective alternative to a conventional supercomputer. However it is worth noting that numerical performance expressed in FLOPs, aren't always the best metric. Clusters can have very high value of that parameter, but they cannot access all data at once. Therefore clusters are excellent for parallel computation, but much poorer at non-parallel computation [3].

JavaSpaces is a service specification supporting clustering computers via a distributed shared memory. It provides a distributed object exchange and coordination mechanism for Java objects. It is used to store the distributed system state and implement distributed algorithms. In a JavaSpace, all communication partners communicate and coordinate by sharing state.

JavaSpaces can be used to reach scalability through parallel processing. It can be applied to provides unfailing storage of objects through distributed replication. JavaSpaces are frequently used to low-latency, high performance tasks rather than reliable object caching [3].

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

1) $\mathcal{X}$ - an admissible solution set

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

2) F - a quality vector criterion

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

where

$\mathcal{R}$ – the set of real numbers,

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

3) P - the Pareto relation [1].

9. Numerical experiments

Figure 9 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 [22] 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 MOTB to qualities produced by the other multi-criterion meta-heuristics.

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 MOTB 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 MOTB, 3.6% for the MGP.

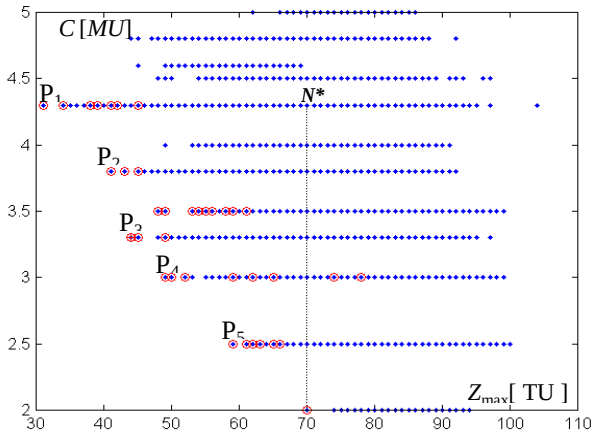

Fig. 9. Pareto front and results of AMEA*

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 MOTB versus 37.9% for the MGP, if search space consists of 1.2396×10^{18} elements and includes 342 758 admissible solutions.

Taboo search provides a promising alternative for the other problems like the job shop scheduling problem [32, 34]. However, it has to be tailored each time with respect to parameters for every instance in order to produce desirable solution. In order to improve its search efficiency, it can be proposed an approach for the job shop scheduling problem by using taboo search with fuzzy reasoning, too [14, 26]. There are two modules in this approach: taboo search module and fuzzy reasoning module that performs the function of adaptive parameter adjustment in taboo search [16, 21].

10. 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 MOTB 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. Balicka, H., Balicki J.: Effective program module assignment in the Internet banking by tabu-based evolutionary algorithm. WSEAS Transactions on Mathematics, Issue 3, Vol. 3, pp. 527-532, (2004)
3. Balicki, J,: *Tabu-based evolutionary algorithm for effective program module assignment in parallel processing.* WSEAS Transactions on Systems, Issue 1, Vol. 3, January 2004, pp. 119-124.
4. Balicki, J.: Hierarchical Tabu Programming for Finding the Underwater Vehicle Trajectory, International Journal of Computer Science and Network Security, 7, 32--37 (2007)
5. Balicki, J.: Immune Systems in Multi-criterion Evolutionary Algorithm for Task Assignments in Distributed Computer System. LNCS, 3528, pp. 51--56, Springer, Heidelberg, (2005)
6. Balicki, J.: Tabu Programming for Multiobjective Optimization Problems, International Journal of Computer Science and Network Security, 7, 44--50 (2007)
7. 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)
8. 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)
9. Castelino, D. J., Hurley, S., and Stephens, N. M.: A tabu search algorithm for frequency assignment. Annals of Operations Research, 63, 301–319, (1996).
10. Chiarandini, M., Schaerf, A., and Tiozzo, F., Solving employee timetabling problems with flexible workload using tabu search. in Proc. of the 3rd Int. Conf. on the Practice and Theory of Automated Timetabling, Konstanz, Germany, 298–302, (2000).
11. Costa, D., A tabu search algorithm for computing an operational timetable. European Journal of Operational Research, 76, 98–110, 1994.
12. Crainic, T. G., Toulouse, M., Gendreau, M.: Toward a Taxonomy of Parallel Tabu Search Heuristics, INFORMS Journal on Computing, 9, 61--72 (1997)
13. Cvijovic, D.; Klinowski, J. Taboo search - an approach to the multiple minima problem. Science, 267, 664-666, (1995)
14. Dell'Amico, M., Trubian, M.: Applying Tabu Search to the Job-Shop Scheduling Problem, Annals of Operations Research, 41, 231--252, (1993)
15. Di Gaspero, L., Schaerf A., Tabu search techniques for examination timetabling. in E. Burke and W. Erben, editors, Proc. of the 3rd Int. Conf. on the Practice and Theory of Automated Timetabling, number 2079 in Lecture Notes in Computer Science, Springer-Verlag, Berlin-Heidelberg, 104–117. 2001.
16. Dorigo, M., Maniezzo, V., and Colorni, A., The ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, B 26(1), 29–41, (1996)
17. Faigle, U., Kern, W.: Some Convergence Results for Probabilistic Tabu Search, ORSA Journal on Computing, 4, 32--38, (1992)
18. Gendreau, M., Hertz, A., Laporte, G., A tabu search heuristic for the vehicle routing problem. Management Science, 40(10), 1276–1290, (1994)
19. Glover, F. Tabu Search Fundamentals and Uses. Revised and Expanded, Technical Report, Graduate School of Business, University of Colorado, Bolder, (1995)
20. Glover, F., Laguna, M.: Tabu Search, Kluwer Academic Publishers, Boston (1997)
21. Glover, F.: Tabu Search — Part I, ORSA Journal on Computing, 1, 190--206, (1989)
22. Glover, F.: Tabu Search — Part II, ORSA Journal on Computing, 2, 4--32, (1990)
23. Glover, F.: Tabu Search: A Tutorial, Interfaces, 20, 74--94, (1990)
24. Hansen, M. P.: Tabu Search for Multicriteria Optimization: MOTS. Proceedings of the Multi Criteria Decision Making, Cape Town, South Africa, (1997)
25. Hertz, A.: Finding a Feasible Course Schedule Using Tabu Search, Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science, 35, (1992)
26. 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)
27. Korbicz, J.: Artificial intelligence in technical diagnostics. - Diagnostics, 46, pp. 7-16, (2008)
28. Koza, J.R.: Genetic programming. The MIT Press, Cambridge 1992.
29. Lokketangen, A., Jornsten, A. K., Storoy, S.: Tabu Search within a Pivot and Complement Framework, International Transactions in Operations Research, 1, 305--316, (1994)
30. Mati, Y., Rezg, N. Xie X.: Scheduling Problem of Job-Shop with Blocking: A Taboo Search Approach, Proc. on the 4th Metaheuristics International Conference, 643-648, (2001)
31. Rego, C.: A Subpath Ejection Method for the Vehicle Routing Problem, Management Science, 44, 1447--1459 (1998)
32. Sadegheih, A.: Evolutionary Algorithms and Simulated Annealing in the Topological Configuration of the Spanning Tree. WSEAS Transactions on Systems, Issue 2, Vol. 7, 31-39, (2008)
33. 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.
34. Taillard, E.: Robust taboo search for the quadratic assignment problem. Parallel Computing, 17, 433–445, (1991)
35. Tsubakitani, S., Evans, J.R.: Optimizing Tabu List Size for the Traveling Salesman Problem, Computers and Operations Research, 25, 91-97, (1998).
36. Verdu E., Regueras L., Verdu M., De Castro J, Perez M.: An analysis of the Research on Adaptive Learning: The Next Generation of e-Learning, WSEAS Transactions On Information Science & Applications, Issue 6, Volume 5, 859-868 (2008)
37. Weglarz, J., Nabrzyski, J., Schopf, J.: Grid Resource Management: State of the Art and Future Trends. Kluwer Academic Publishers, Boston (2003)
38. Widmer, A. M.: The Job-shop Scheduling with Tooling Constraints: A Tabu Search Approach, J. Opt. Res. S, 42, 75--82 (1991)
39. Yang, S. Y. How Does Ontology help Web Information Management Processing. WSEAS Transactions on Computers. Issue 9, Volume 5. 1843-1850 (2006)