Multi-criterion Decision Making by Artificial Intelligence Techniques

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.

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.

On that background, 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 [16, 22], tabu programming solves problems as a general solver that is based on a tabu algorithm. Some advanced tabu-based algorithms for scheduling problems are proposed by Węglarz in [23]. This optimization technique can be used to continuous functions by a selection a discrete encoding of the problem [5, 7]. 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 [6, 9]. In a tabu programming, special areas for possible modification of programs are forbidden during the seeking in a space of all possible combinations [8, 10]. 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 [2]. 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 [3].

Then, an improved MOTB for solving multi-criterion with constraints optimization problems of task assignment in the distributed computer system has been considered [4]. 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, 22]. To keep away from repeating the steps, recent moves are recorded in some tabu lists [14]. That lists forms the short-term memory. The memory content can vary as the search proceeds [12]. At the beginning, the target is testing the solution space, during a ‘diversification’ [13]. 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 \( \mathcal{N}(x^{\text{now}}) \) of the current solution \( x^{\text{now}} \) that is calculated by the given program, we can choose the next solution \( x^{\text{new}} \) to a search trajectory of TP [20]. 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 \( \mathcal{X}(x^{\text{now}}) \) because of the maintaining a selective history of the states encountered during the exploration [15, 19]. Some solutions, which were visited during the given last term, are excluded from the basic neighborhood according to the classification of movements [17]. If any solution satisfies an aspiration criterion, then it can be included to the considered neighborhood, only [18].

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 \( \mathcal{N}(x^{\text{now}}) \) 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 [21]:

\[
\{ f_1, \ldots, f_n, \ldots, f_N \} \tag{1}
\]

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

\[
\{ \text{if\_obstacle}, \text{move}, \text{if\_end}, +, -, \ast, / \} \tag{2}
\]

The procedure \( \text{if\_obstacle} \) 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 \( \text{move} \) requires three arguments. It causes the movement along the given direction with the velocity equals the first argument during assumed time \( \Delta t \). The time \( \Delta t \) is the value that is equal to the division a limited time by \( M_{\text{max}} \). 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 \( \text{if\_end} \) 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]:

\[
\{ \varnothing, +, -, \ast, / \} \tag{3}
\]

where

- \( \varnothing \) – the procedure that converts \( M=V+I(V+J) \) input real numbers called activation levels on \( M \) output binary numbers

\[
\{ \{ x^{m}_{ij}, x^{m}_{ij}, \ldots, x^{m}_{ij}, \ldots, x^{m}_{ij}, \ldots, x^{m}_{ij} \} \}
\]

- \( x^{\pi}_{ij} = \begin{cases} 1 & \text{if } \pi_j \text{ is assigned to } \omega_i, \\ 0 & \text{in the other case}. \end{cases} \)

- \( x^{\pi}_{vi} = \begin{cases} 1 & \text{if task } T_v \text{ is assigned to } \omega_i, \\ 0 & \text{in the other case}, \end{cases} \)

- \( N_v \) – number of the \( v \)th module in the line for its dedicated computer,

\( W = \{ w_1, \ldots, w_i, \ldots, w_j \} \) - the set of the processing nodes,

\( T = \{ T_1, \ldots, T_v, \ldots, T_v \} \) - the set of parallel performing tasks,

\( \Pi = \{ \pi_1, \pi_j, \ldots, \pi_j \} \) - the set of available computer sorts.

The procedure \( \varnothing \) is obligatory the root of the program tree and appears only one in a generated program. In that way, the formal constraints
The set of arguments for the task assignment is determined in another 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]:

\[
T = D \cup \mathcal{L}.
\]  

(5)

where \( \mathcal{L} \) – set of \( n \) random numbers, \( n = \max L \).

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 \( \lambda_1 \) movements are performed. However, that node may be selected to be the root for adding a 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 \( \lambda_2 \) 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 \( l = 1, L_{\text{max}} \), where \( L_{\text{max}} \) 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:

\[
\omega = (D_0, *, +, /, -3, x, z, x)
\]  

(6)

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

\[
\psi = (f, f, f, f, a, a, a, a, a)
\]  

(7)

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

\[
\chi = (1, 2, 2, 0, 0, 0, 0)
\]  

(8)

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 \( \lambda_1 \) movements are performed. However, that node can be selected to be the
We can introduce the matrix of reducing node memory \( M^- = [m_{nm}]_{L_{\text{max}} \times L_{\text{max}}} \), 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 \( M^+ = [\tilde{m}_{nm}]_{L_{\text{max}} \times L_{\text{max}}} \), 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 \( \lambda_1 \) and \( \lambda_2 \) are usually equal to \( \lambda \), 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 \( \lambda \) is supposed to be no greater than \( L_{\text{max}} \). After \( \lambda \) 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:

\[
K(x^*, x^i) = \min_{x \in N(x_{\text{now}})} K(x, x^i) \tag{9}
\]

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

The selection function \( W \) for choosing the next solution in the search path is constructed from the criterion \( K \) and functions describing constraints [11]. Usually, the penalty function can be applied [12].

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 calculate the vector of decision variables \( x_{\text{now}} \). This program can be written as a s-expression in Common Lisp like:

\[
\text{(GT (* -1 x) (* v (ABS v)))}
\]

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

```pascal
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 \( \mu_1, \mu_2 \)
   (C) Generation of the program that produces \( x_{\text{best}} \)
   (D) \( x_{\text{best}} := x_{\text{now}} \)
   (E) \( K_{\text{min}} := K(x_{\text{now}}) \)
   (F) Initialization of restriction matrices \( M^+, M^- \)
   (G) Setting the memory parameters \( \lambda_1, \lambda_2 \)

2. Solution selection and stop criterion \( k:=k+1 \)
   (A) Finding a set of tree candidates \( x_{\text{next}} \) from the neighborhood \( N(x_{\text{now}}) \)
   (B) Selection of the next solution \( x_{\text{next}} := x_{\text{next}} \) with the minimal value of the selection function \( W \) among solutions taken from \( x_{\text{best}} \)
   (C) Aspiration condition. If all solutions from the neighborhood are tabu-active and \( K_{\text{min}} < 0.8 K(x_{\text{now}}) \), then \( x_{\text{next}} := x_{\text{best}} \).
   (D) Re-linking of search trajectory. If \( x_{\text{next}} \) was not changed during main iteration, then a genetic crossover procedure for parents \( x_{\text{best}}, x_{\text{now}} \) is performed. A child with the smaller value of \( K \) is \( x_{\text{next}} \), and another one is \( x_{\text{best}} \).
   (E) If \( k = 0.4 T_{\text{max}} \) then \( \lambda_1 := 4 \lambda_1, \lambda_2 := 4 \lambda_2 \)
   (F) If \( k = T_{\text{max}} \) or maximal time of calculation is exceeded, then STOP.

3. Up-dating
   (A) \( x_{\text{now}} := x_{\text{next}} \)
   (B) If \( K(x_{\text{now}}) < K_{\text{min}} \) then \( x_{\text{best}} := x_{\text{next}} \) and go to 1(B)
   (C) After reduction the procedure \( f_{\text{next}} \) (with the parent \( f_{\text{now}} \)) as a root of the chosen sub-tree \( M := M' + 1 \), \( m_{nm} := \lambda_1 \).
   (D) After adding the procedure \( f_{\text{now}} \) (with the parent \( f_{\text{now}} \)) 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 \( \mu_1 \) – the number of tree nodes. Figure 1 shows the tree with 8 tree nodes. Moreover, \( \mu_2 \) – the number of the tree levels is another constraint parameter for the program tree. There are 4 tree levels on Figure 1. Parameters \( \mu_1 \) as well as \( \mu_2 \) 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. Results 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 $Z_{\text{max}}$ – 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.

The first criterion is the workload of the bottleneck computer for the allocation $x$, and its values are provided by the subsequent formula [4]:

$$Z_{\text{max}}(x) = \max_{x \in X} \left( \sum_{i,j} x_{ij} \lambda_{ij} + \sum_{v=1}^{V} \sum_{i,k} \tau_{vuk} x_{vk} \right)$$

where

$$x = \{x_{ij}, x_{ik}, x_{kj}, x_{ij}, x_{ik}, x_{kj}, x_{ij}, x_{ik}, x_{kj}, N_1, N_2, N_3, N_4, N_5\},$$

$\tau_{vuk}$ – the total communication time between the task $T_i$ assigned to the $i$th node and the $T_v$ assigned to the $k$th node.

Figure 4 shows three cuts in task assignment graph. We can balance workload among several processors by finding an optimal value of the bottleneck computer.

![Fig. 4. Load balancing by finding an optimal task assignment](image)

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

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

where $\kappa_j$ corresponds to the cost of the computer $\pi_j$.

Let $\pi_j$ be failed independently due to an exponential distribution with rate $\lambda_j$. 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]:

$$R(x) = \prod_{i=1}^{V} \prod_{j=1}^{J} \prod_{k=1}^{K} \exp(-\tilde{\lambda}_{ij} x_{ij})$$

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:

$$\Theta(x) = \sum_{j=1}^{J} \sum_{i=1}^{I} \gamma_j x_{ij}$$

where $\gamma_j$ is the numerical performance of the computer $\pi_j$ for the task benchmark, for instance [MFlops].

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.

$$P_D(x) = \sum_{i=1}^{K} \prod_{j=1}^{J} (d_j - C_v(x))$$

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.

Figure 5 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 [17] 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.

![Fig. 5. Pareto front and results of AMEA*](image)
The binary search space consisted of $1.0737 \times 10^9$ 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 $\bar{S}$ 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.

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 $1.2396 \times 10^{18}$ elements and includes 342,758 admissible solutions.

Taboo search provides a promising alternative for the other problems like the job shop scheduling problem [23]. 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]. 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].

7. 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 taboo search and evolutionary algorithms for finding efficient solutions.

References:


