8. Genetic Programming for Workload Balancing in the Comcute Grid System

Jerzy Balicki, Waldemar Korłub, Jacek Paluszak, Artur Zacniewski*

Gdańsk University of Technology

Faculty of Electronics, Telecommunications and Informatics

Computer System Architecture Department

e-mail: balicki@eti.pg.gda.pl, stawrul@gmail.com, jacekpaluszak@gmail.com

**The Naval University of Gdynia*

e-mail: a.zacniewski@amw.gdynia.pl

Abstract

In this chapter, a genetic programming paradigm is implemented for reliability optimization in the Comcute grid system design. Chromosomes are generated as the program functions and then genetic operators are applied for finding Pareto-suboptimal task assignment and scheduling. Results are compared with outcomes obtained by an adaptive evolutionary algorithm.

Keywords: *grid computing, parallel programming, volunteer computing, genetic algorithm, workload balancing*

8.1. Introduction

In the Comcute system, several tasks are performed concurrently by three ordered levels of the support software modules. This middleware consists of some modules that are dedicated to different system nodes. So, the question is how assign system modules to nodes to obtain some advantages for the whole grid [7].

A workload balancing and the total cost of a program run in a grid computer like the Comcute may be reduced by the task assignment and the scheduling [1]. Moreover, it is possible to decrease the cost of computers if a selection of the computer sort is carried out. A total amount of the system performance is another measure that can be minimized by task distribution and scheduling [16].

The probability that all computers remain fault-free during the execution of the modules assigned to computers is important criterion of evaluation task

assignments. In real-time systems that are preferred for some anti-crisis dilemmas, a required time of response is supposed to be guaranteed. In this case, time precedence between concurrent tasks play important role. Therefore, the second important criterion is the probability that tasks are completed on time.

A problem of task allocation can be formulated as a multiobjective combinatorial optimization question, which is solved by an approach based on genetic programming. It is applied for finding the subset of Pareto-optimal solutions.

Genetic algorithms, evolutionary algorithms, evolution strategies and genetic programming are the alternative evolutionary approaches to the modern metaheuristic multicriteria optimization methods such as simulated annealing, tabu search or Hopfield models of neural networks. Evolutionary calculations deal simultaneously with a population of possible solutions, which allows finding a subset of Pareto optimal solutions by one algorithm run, instead of having to perform a number of separate runs of standard multiobjective optimization techniques. Moreover, evolutionary approaches give outcomes with good quality for different instances of multiobjective problems and can be considered as a robust optimization technique. From these reasons evolution approaches are convenient if we look for the subset of Pareto-optimal solutions.

8.2. Genetic programming rules

Genetic programming is a remarkable paradigm of an artificial intelligence. A theory of genetic programming has been created by John R. Koza of the Computer Science Department of Stanford University [14]. Solutions to several problems have been found for instances from different areas like optimal control, planning, sequence induction, symbolic regression, automatic programming or discovering a game playing strategy. Furthermore, problems related to empirical discovering and forecasting, symbolic integration or differentiation, discovering mathematical identities, classification and decision tree induction, evolution of emergent behavior and also automatic programming of cellular automata are on the list of problems that have been solved successfully by genetic programming [17].

Although, several different problems have been taken into account, finding task assignment by genetic programming is a new scientific challenge [18]. Fig. 8.1 shows an example of a tree of the computer program performance.

This tree corresponds to the program written in the LISP language, as follows:

$$(\mathrm{LT}\ (+\ -3\ x)\ (*\ v\ (\mathrm{SQRT}\ v)))$$

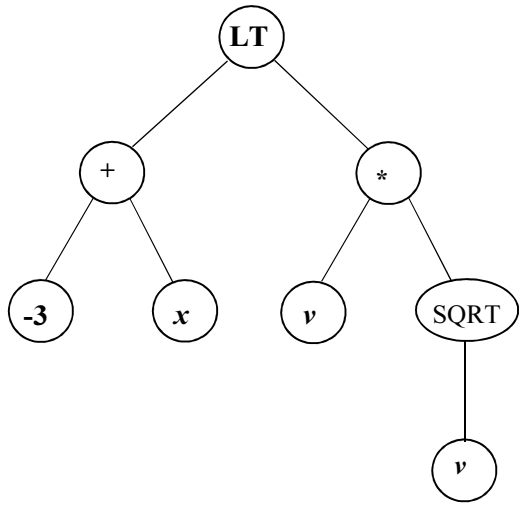

Fig. 8.1. Tree as a model of the computer program

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. If a computer program was represented by any algorithm diagram, genetic operators like reproduction, crossover or mutation would be difficult to implementation. That is, a tree is a relevant model of chromosome that can be transformed by removing a sub-tree or exchanging two sub-trees.

Despite the data structure represents a chromosome in an evolutionary algorithm [11], a chromosome for genetic programming is the tree of a computer program. The simplest procedure differs from a complex data structure significantly. The procedure can be used to calculation that gives ability to represent not only knowledge about a problem and also it gives possibility to draw conclusions or to process data in the way difficult to discover. That is, a computer program may find a solution to the problem and a genetic algorithm without an aid of computer programmer can be able to construct this procedure.

Generation of the trees is an important step for finding Pareto-optimal task assignments. 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. A functional node represents an

elementary procedure randomly chosen from the primary defined set of functions:

$$\mathcal{F} = \{f_1, ..., f_n, ..., f_N\} \quad (1)$$

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:

$$\mathcal{T} = \{a_1, ..., a_m, ..., a_M\} \quad (2)$$

An above property of the procedure is called a closure property because each function should be well defined and closed for any arrangement of arguments that it may come across.

Another property of a set of procedures, called the sufficiency property, requires that the solution to the problem should 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 $\mathcal{F} = \{AND, OR, NOT\}$ 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 realizing any Boolean function. In addition, a sufficient set is $\{AND, NOT\}$ as well.

8.3. Computer and channel failures in a grid system

We assume that computers and communication channels may failure during data processing. The fault-tolerant system is able to deal with failures of its elements during the execution of task modules assigned to computers and for the generated communication. Each computer and each link between them are assumed to fail independently with exponential rates. We do not take into account of repair and recovery times for failed computers in assessing the logical correctness of an allocation. Instead, we shall allocate modules to computers on which failures are least likely to occur during the execution of task modules. That is, we are supposed to assign modules with maximum reliability and thus eliminate the need of on-line repair and recovery.

To guard against the unlikely failures of these computers, one can assign copies of a module to multiple computers, but this subject is not the scope of this paper. The rationale behind the above assumption is that repair and recovery times are largely implementation-dependent. Moreover, repair and recovery

routines usually introduce too high time overheads to be used on-line for time-critical applications.

A set of program modules $\{M_1,...,M_m,...,M_M\}$ communicated to each other's is considered among the coherent computer network with computers located at the processing nodes from the set $W=\{w_1,...,w_i,...,w_I\}$. A program module can be activated several times during the program lifetime and with the program module runs are associated some processes (tasks). In results, a set of program modules is mapped into the set of parallel performing tasks $\{T_1,...,T_v,...,T_V\}$.

Let the task T_v be executed on computers taken from the set of available computer sorts $\Pi=\{\pi_1,...,\pi_j,...,\pi_J\}$. The overhead performing time of the task T_v by the computer π_j is represented by t_{vj}. Let a computer π_j be failed independently due to an exponential distribution with rate λ_j. The longer time of task execution, the higher probability of computer failure. Computers can be allocated to nodes and also tasks can be assigned to them in purpose to maximize the reliability function R defined, as below [2]:

$$R(x)=\prod_{v=1}^{V}\prod_{i=1}^{I}\prod_{j=1}^{J}\exp(-\lambda_j t_{vj} x_{vi}^{m} x_{ij}^{\pi}), \tag{3}$$

where

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

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

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

A computer can be chosen several times from the set Π to be assigned to the node w_i and one computer is allocated to each node. On the other hand, each task is allocated to any node.

Fig. 8.2 shows a relation between the reliability R_j of computer and parameter λ_j.=0.001 [TU^{-1}, TU – time unit].

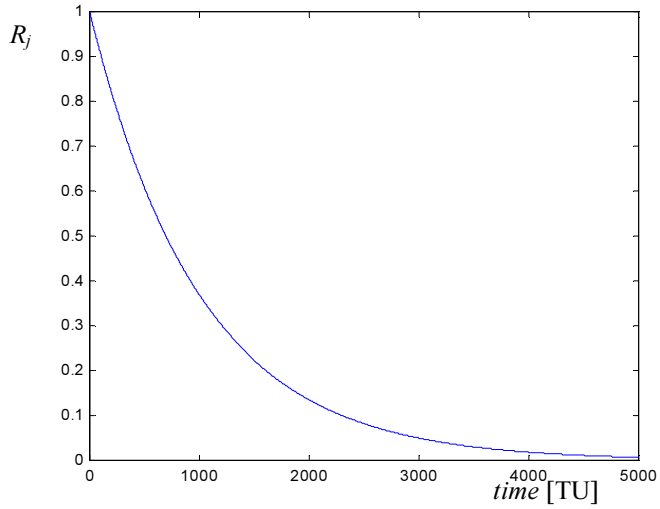

Fig. 8.2. Reliability of an individual computer node

If there are two computer nodes with λ_1.=0.001 [TU^{-1}] and λ_2.=0.002 [TU^{-1}], the reliability of the two-computer system decreases faster than the reliabilities for each of them. Fig. 8.3 shows the relation between the measure of system reliability *R* and time of using this system for the chosen two-computer system.

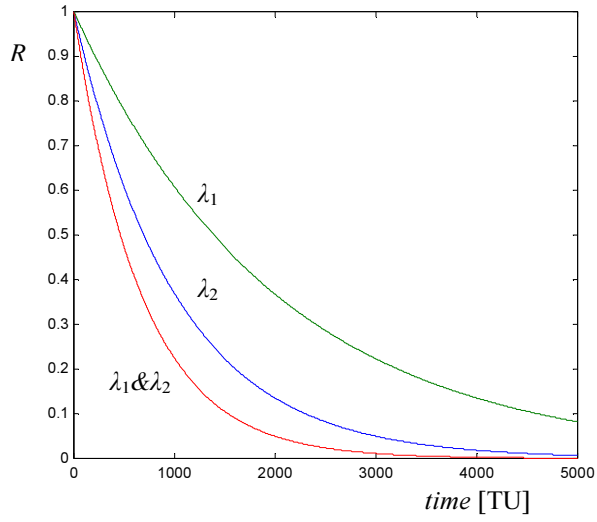

Fig. 8.3. The reliability of the two-computer system

8.4. Probability of deadline meetings for the tasks

The precedence constraints among modules are figured in calculation of module release time and the timing constraints on modules are considered then probability P_D that all tasks meet their deadlines is evaluated. If a task misses its deadline under the given allocation modules, then this probability is equal to zero.

Let the distributed program $\mathcal{P}_n$ may begin its running after λ_n and complete before δ_n. A task flow graph characterizes the logical structure of program performance. The precedence constraints among modules and the timing constraints can be presented on the task flow graph. Fig. 8.4 shows an example of the task flow graph for two programs divided on five modules.

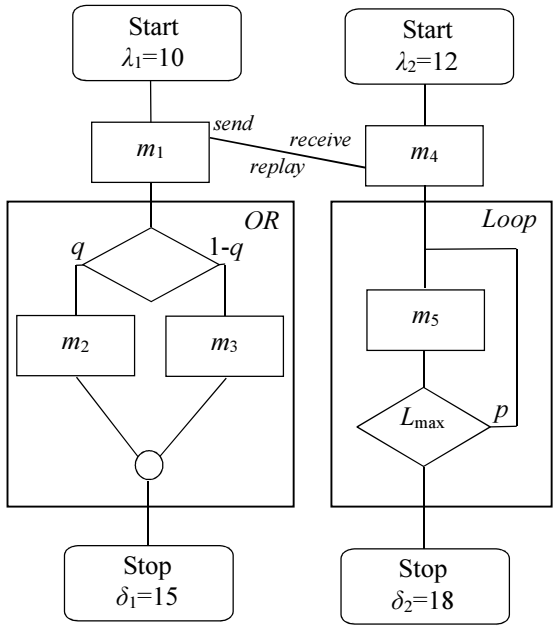

Fig. 8.4. The task flow graph for two programs divided on five modules

Task m_2 is performed with the probability q in a sub-graph denoted as *OR* (Fig. 4) and task m_3 – with the probability (1-q). Task may be performed at the most L_{max} times in a sub-graph denoted as *Loop*, and each repetition of this module is performed with the probability p.

The task flow graph is split on some instances to schedule tasks if the sub-graph *OR* appears. For example, the first instance incorporates the module m_2 instead the sub-graph *OR* and this instance emerges with frequency q. The second instance incorporates the module m_3 instead the sub-graph OR and it appears with frequency (1-q). For the sub-graph *Loop*, L_{max} instances are designed, and

module is run k times for each instance (k=1,2,...,L_{max}). The instance, where module runs k times, can be met with the probability $(1 - p)\, p^{k-1}$. There are $2L_{max}$ instances for the task graph from Figure 4. The instance, where task m_2 appears and task m_5 runs k times, occurs with the probability:

$$p_i = q(1- p)\, p^{k-1} \tag{4}$$

An allocation modules to computers (x^m, x^π) creates possibility to schedule tasks for each computer. Let this distributed schedule be determined, as follows:

$$N^m = [N_1, ..., N_v, ..., N_V]^T, \tag{5}$$

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

Times of task completions (C_1,...,C_v,...,C_V) can be calculated for scheduled allocation modules to computers $x=(x^m, x^\pi, N^m)$ [4]. Let d_v represents the completion deadline for the vth task. If $C_v \le d_v$, then the time constraint is satisfied what can be written as $\xi(d_v - C_v) = 1$. If the deadline is exceeded, then $\xi(d_v - C_v) = 0$. The state of deadline constraints regarding the ith instance of the flow graph with the set of tasks marked M_i is determined, as below:

$$S_i = \prod_{m_v \in M_i} \xi(d_v - C_v(x)) \tag{6}$$

.

If at least one task exceeds the deadline, then deadline constraint for the ith instance is not satisfied. Probability that all tasks meet their deadlines for K instances of the flow graph is calculated:

$$P_D(x) = \sum_{i=1}^{K} p_i \prod_{m_v \in M_i} \xi(d_v - C_v(x)) \tag{7}$$

.

8.5. Bottleneck of limited computer workload and specific constrains

Chu and Lan have introduced the workload of the bottleneck computer as the criterion for the evaluation of an allocation quality [4]. A computer with the heaviest task load is the bottleneck machine in the system, and its workload is

a critical value that should be minimized [12]. The workload $Z_i^+(x)$ of a computer allotted to the ith node for the allocation x is provided by the subsequent formula:

$$Z_i^+(x) = \sum_{v=1}^{V}\sum_{i=1}^{I} t_{vi} x_{vi}^m x_{ij}^\pi + \sum_{\substack{v=1 \\ v\neq u}}^{V}\sum_{u=1}^{V}\sum_{i=1}^{I} \tau_{vuii_2} x_{vi}^m x_{ij}^\pi, \tag{8}$$

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

A computer with the heaviest load $Z_i^+(x)$ is the bottleneck machine in the system, and its workload is the critical value that is supposed to be limited:

$$\min_{i=1\overline{I}} \left\{ Z_i^+(x) \right\} \leq Z_{\max}^{\lim}. \tag{9}$$

The second constraint is related to the limited capacities of resources. Each computer is supposed to be equipped with necessary capacities of resources for a program execution. Let the following memories $z_1,...,z_r,...,z_R$ be available in an entire system and let d_{jr} be denote the capacity of memory z_r in the workstation π_j. We assume the module m_v reserves c_{vr} units of memory z_r and holds it during a program execution. Both values c_{vr} and d_{jr} are nonnegative and limited. The memory limit in any machine cannot be exceeded in the ith node, what is written, as bellows:

$$\sum_{v=1}^{V} c_{vr} x_{vi}^m \leq \sum_{j=1}^{J} d_{jr} x_{ij}^\pi, \;\; i = \overline{1,I}, \;\; r = \overline{1,R}. \tag{10}$$

The third constraint is related to the limit cost of computers [1]:

$$\sum_{i=1}^{I}\sum_{j=1}^{J} \kappa_j x_{ij}^\pi \leq \kappa_{\max}, \tag{11}$$

where κ_j corresponds to the cost of the computer π_j.

The fourth constraint is a requirement that total amount of computer performance is supposed to be greater than minimal value:

$$\sum_{i=1}^{I}\sum_{j=1}^{J} \vartheta_j x_{ij}^\pi \geq \vartheta_{\max} \tag{12}$$

where ϑ_j is the numerical performance of the computer π_J for the given benchmark.

8.6. Two-criterion optimization problem and evolutionary approach

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

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

$$\mathcal{X} = \{x \in \mathcal{B}^{I(V+J)} \times \mathcal{V}^{V} \mid \min_{i=\overline{1,I}} \left\{ \sum_{v=1}^{V} \sum_{i=1}^{I} t_{vi} x_{vi}^{m} x_{ij}^{\pi} + \sum_{v=1}^{V} \sum_{\substack{u=1 \\ v \neq u}}^{V} \sum_{i=1}^{I} \tau_{vuii_2} x_{vi}^{m} x_{ij}^{\pi} \right\} \le Z_{\max}^{\lim};$$

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

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

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

where $\mathcal{B} = \{0, 1\}$, $\mathcal{V} = \{1, 2, \ldots, V\}$

2) F - a vector superiority criterion

$$F : \mathcal{X} \to \mathcal{R}^2 \;, \tag{13}$$

where

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

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

$R(x)$, $P_D(x)$ are calculated by (3) and (7), respectively

3) P - the Pareto relation [3, 5].

There is no task allocation $a \in \mathcal{X}$ such that $(F(a), F(x^*)) \in P$ for the Pareto-optimal assignment $x^* \in \mathcal{X}$ and $a \neq x^*$.

An overview of evolutionary algorithms for multiobjective optimisation problems is submitted in [6, 13]. Some specific knowledge about the considered

optimization problem is applied in an evolutionary algorithm. Consequently, a standard genetic algorithm can be used for solving the wide class of optimization problems, but an evolutionary algorithm is rather focused on the special class of problems [15]. However, outcomes are regularly much better for evolutionary algorithms than for genetic algorithms [15].

In the first multi-criterion genetic algorithm called a vector evaluated genetic algorithm VEGA, a population of solutions is divided on N subpopulations, where N is the number of partial criteria [19]. For each nth subpopulation, the criterion F_n is a fitness function. However, a crossover and a mutation are carried out for the entire population. This method for a fitness evaluation has a weakness related to the discrimination of Pareto solutions located in an interior of the Pareto front.

Fourman has considered the selection with binary tournaments, where two randomly chosen solutions have been compared [8]. The hierarchical alternative is chosen and it is included to a mating pool of potential parents. A selection probability is calculated for the most significant aim. A random choice is carried out twice according to the roulette rule. Hierarchical tournaments push the population towards lexicographical solutions likewise the VEGA approach. Another selection is based on a random choice of a goal that is taken to comparison of selected solutions. Selection probabilities are constant or they can depend on the chosen purpose for the other tournament selection.

A ranking idea for non-dominated individuals has been introduced to avoid the prejudice of the interior Pareto alternatives by Goldberg [10]. Srinivas and Deb [21] have built a non-dominated sorting genetic algorithm NSGA on the ideas mentioned by Goldberg. If some admissible solutions are in a population, then the Pareto-optimal individuals are determined, and after that they get the rank 1. Subsequently, they are temporary eliminated from the population. Next, the new Pareto-optimal alternatives are found from the reduced population 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 individuals have the same reproduction fitness because of the equivalent rank.

To maintain the diversity of the population and to preclude premature convergence, fitness-sharing techniques have been developed [6]. A mating restriction assumes that individuals from a criteria space neighbourhood are similar, so that they can form stable niches. If a non-dominated evaluation for the current population has a long distance to the nearest non-dominated evaluation and there is a niche of non-dominated results, then the separated individual is supposed to be preferred by increasing its fitness before individuals from the niche.

Above multi-criterion techniques are based on a genetic algorithm. Another approach is an extension of evolution strategy. Binh and Korn have developed a multicriteria evolution strategy for combinatorial optimization problems [3]. A new Pareto archived evolution strategy called PAES was proposed by Knowles and Corne [13].

Zietzler and co-authors have suggested an elitist selection in their strength Pareto evolutionary algorithm SPEA [22]. At each generation, a combined population with the external and the current population is constructed. In an external population, all non-dominated solutions discovered so far are archived. An elitist selection is applied in the other elitist non-dominated sorting genetic algorithms.

8.7. Adaptive multi-criterion evolutionary algorithm

An approach based on the genetic programming for solving multicriteria optimization problem produces a population of algorithms that adapt themselves to the problem. However, the name "adaptive evolutionary algorithm" for this evolutionary algorithm is related to the changing of some parameters as a crossover probability, a mutation rate, a population size, and the others during the searching. According to another meaning, it is related to the operators change as a result of the search process (i.e., population diversity, etc), not as a function of generation. For considered algorithm, the crossover probability is decreased due to the number of new generations.

Figure 5 shows a scheme of the adaptive multi-criterion evolutionary algorithm called AMEA/GP that operates on the population of programs. This algorithm permits on achieving better results for task assignment than the other tested multiobjective evolutionary algorithms [2].

The preliminary population of programs is created in a specific manner (Fig. 5, line 3). Each generated program consists of set of procedures and set of attributes. Set of procedures is defined, as follows:

$$\mathcal{F} = \{list, +, -, *, /\} \tag{14}$$

where: *list* is a procedure that convert $I(V+J)+V$ input real numbers called *activation levels* on $I(V+J)$ output binary numbers:

$$x^m_{1,1},\ldots x^m_{1J},\ldots x^m_{vi},\ldots x^m_{VI},x^\pi_{1,1},\ldots x^\pi_{1J},\ldots x^\pi_{ij},\ldots x^\pi_{I1},\ldots x^\pi_{Ij},\ldots x^\pi_{IJ}$$

and V are 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 $\mathcal{D}$ 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:

$$\mathcal{T} = \mathcal{D} \cup \mathcal{L}, \tag{15}$$

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

If x calculated by the program is admissible, then the fitness function value (Lst. 8.1, line 4) is estimated, as below:

$$f(x) = r_{\max} - r(x) + P_{\max} + 1, \tag{16}$$

where $r(x)$ denotes the rank of an admissible solution, $1 \le r(x) \le r_{\max}$.

Lst. 8.1. Adaptive multicriteria evolutionary algorithm for genetic programming

```
1. BEGIN
2. t:=0, set the even size of population L, p_m:=1/M, M – the length of solution
3. generate initial population of programs P(t), t – the number of population
4. run programs, calculate ranks r(x) and fitness f(x), x ∈ P(t)
5. finish:=FALSE
6. WHILE NOT finish DO
7.    BEGIN /* new population */
8.    t:= t+1, P(t) := ∅
9.    calculate selection probabilities p_s(x), x∈P(t–1)
10.   FOR L/2 DO
11.   BEGIN /* reproduction cycle */
12.      2WT-selection of a potential parent pair (a,b) from the population P(t-
1)
13.      S-crossover of a parent pair (a,b) with the adaptive crossover rate p_c,
p_c := e^(–t/T_max)
14.      S-mutation of an offspring pair (a',b') with the mutation rate p_m
15.      P(t):=P(t)∪(a',b'}
16.   END
17.   calculate ranks r(x) and fitness f(x),x∈P(t)
18.   IF (P(t) converges OR t≥T_max) THEN finish:=TRUE
19.   END
20. END
```

Another ranking procedure has been introduced by Fonseca and Fleming [6]. It assigns each individual a rank based on the number of other individuals by which it is dominated. A niching procedure modifies these ranks. The surface

region of the Pareto front is divided by the size of the population. The number of other member's falling within the sub-area of any individual is taken to establish the niching penalty for it [6].

In the two-weight tournament selection (Fig. 5, line 12), the roulette rule is carried out twice. If two potential parents (**a**, **b**) are admissible, then a dominated individual is eliminated. If two solutions non-dominate each other, then they are accepted. If potential parents (**a**, **b**) are non-admissible, then an alternative with the smaller penalty is selected.

The fitness sharing technique can be substituted by the adaptive changing of main parameters. The quality of attained solutions increases in optimization problems with one criterion, if the crossover probability and the mutation rate are changed in an adaptive way proposed by Sheble and Britting [20]. The crossover point is randomly chosen for the chromosome X in the S-crossover operator (Fig. 1, line 13). The crossover probability is 1 at the initial population and each pair of potential parents is obligatory taken for the crossover procedure.

A crossover operation supports the finding of a high-quality solution area in the search space. It is important in the early search stage. If the number of generation t increases, the crossover probability decreases according to the formula $p_C = e^{-t/T_{\max}}$. The search region or some search areas are identified after several crossover operations on parent pairs. That is why, value p_c is smaller and it is equal to 0.6065, if t =100 for maximum number of population $T_{\max}$=200. The final smallest value p_c is 0.3679. A crossover probability decreases from 1 to $exp(-1)$, exponentially.

In S-mutation (Fig. 5, line 14), the random swap of the integer value by another one from a feasible discrete set is applied. If the gene X_v^m is randomly taken for mutation, the value is taken from the set $\{1,..., I\}$. If the gene X_i^π is randomly chosen, the value is selected from the set $\{1,..., J\}$. A mutation rate is constant in the AMEA and it is equal to $1/M$, where M represents the number of decision variables.

8.8. Level of convergence to Pareto front and tabu mutation

The AMEA/GP is able to find task assignment representation for several numerical instances of multiobjective optimization problem (13) that was confirmed by extended simulations. Quality of obtained solutions can be assessed by a level of convergence to the Pareto front [1].

Let the Pareto points $\{P_1, P_2,..., P_U\}$ be given for any instance of the task assignment problem (13). 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 $\left|P_{u1} - A_{u1}\right|$. If the point ($A_{u1}$, P_{u2}) is not discovered by the algorithm, we assume the distance is $\left|P_{u1} - A_{u1}^{\min}\right|$, where $A_{u1}^{\min}$ is the minimal reliability of the system for the instance of problem (13).

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

$$S = \sum_{u=1}^{U} \left|P_{u1} - A_{u1}\right|. \tag{17}$$

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. We suggest reducing this disadvantage by an introduction a tabu algorithm [9] (Lst. 8.2) as an advanced additional mutation operator to decrease the minimum value of a workload of the bottleneck computer. According to this concept, to the line 14 (Lst. 8.1) should be added a new procedure, as follows:

14 b) Tabu-mutation of an offspring pair (**a',b'**) with the constant tabu-mutation probability p_{tabu}

The adaptive multicriteria evolutionary algorithm AMEA/GP with a tabu mutation is a hybrid optimization technique that combines advantages of a programming genetic search with a tabu search. In *a tabu search* special areas are forbidden during the seeking in a space of all possible combinations. 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.

Tabu search uses memory structures by reference to dimensions consisting of recency, frequency, quality and influence [11]. It inherits from a simple descent method an idea of a neighborhood $N(x^{\text{now}})$ of a current solution x^{now}. In our hybrid method, the initial task assignment is randomly chosen from the population with the small rate $p_{\text{tabu}} = 0.01 p_{\text{m}}$. From this neighborhood, we can choose the next solution x^{next} to a search trajectory. The accepted alternative is supposed to have the best value of an objective function among the current neighborhood. However, the descent method terminates its searching, when the chosen candidate is worse than the best one from the searching trajectory.

Lst. 8.2. Tabu search algorithm for minimization of the bottleneck computer load programming

1. Initial procedure $k:=0$

(A) Random selection x^{now} from the current population $P(t)$

(B) $x^{best} := x^{now}$, $x^{bis} := x^{now}$

(C) $best\_Z_{max} := Z_{max}(x^{now})$

(D) Initialization of restriction matrixes H^R, G^R

(E) $\lambda_1 := \lceil V(I-1)/4 \rceil$, $\lambda_2 := \lceil I(J-1)/4 \rceil$

2. Task assignment selection and stop criterion (main iteration) $k:=k+1$

(A) Finding a set of candidates $\mathcal{X}(H^R, x^{now})$ from the neighborhood $N(x^{now})$

(B) Selection of the next solution $x^{next} \in \mathcal{X}(H4^R, x^{now})$ with the minimal value of the selection function W among solutions taken from $\mathcal{X}$

(C) **Aspiration condition.** If all solutions from the neighbourhood are tabu-active and $best\_Z_{max} \geq Z_{max}(x^{now})$, then $x^{best} := x^{now}$, $best\_Z_{max} := Z_{max}(x^{now})$

(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 Z_{max} is x^{next}, and another one is x^{bis}

(E) If $k = 0.4\, K_{max}$, then $\lambda_1 = \lceil 1{,}25\, V(I-1) \rceil$, $\lambda_1 = \lceil 1{,}25\, I(J-1) \rceil$

(F) If $k = K_{max}$ or assumed time of calculation is exceeded, then STOP.

3. Updating

(A) $x^{now} := x^{next}$

(B) If $Z_{max}(x^{now}) < best\_Z_{max}$, then $x^{bis} := x^{best}$ and go to 1(B)

(C) If vth task was moved at the kth iteration from the nth node to the ith, then $H^R := H^R - 1$, $h_{vi} := \lambda_1$, $h_{vn} := \lceil \lambda_1/2 \rceil$

(D) If the qth computer sort is exchanged on the jth one at the ith node, then $G^R := G^R - 1$, $g_{ij} := \lambda_2$, $g_{iq} := \lceil \lambda_2/2 \rceil$

(E) go to 2

In the tabu search algorithm based on the short-term memory, a basic neighborhood $N(x^{now})$ of a current solution may be reduced to a considered neighborhood $\mathcal{X}(x^{now})$ 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 neighborhood according to the tabu classification of movements. If any solutions performs aspiration criterion, then it can be included to the considered neighborhood, only.

Hansen has proposed a multiobjective optimization tabu search MOTS [11] to generate non-dominated alternatives. The MOTS works with a population of solutions, which, through manipulation of weights, are moved towards the Pareto front [11]. However, the MOTS do not cooperate with an evolutionary algorithm.

A tabu-mutation is implemented as the tabu algorithm TSZmax [2] that has been designed to find the task assignment with the minimum value of the function Z_{max} . Fig. 8.5 shows the process of the minimization Z_{max} from the initial value equal to 62 time units to 32. The task assignment with the value 62 was randomly taken from the current population with the probability p_{tabu} . An outcome is inserted to the new population.

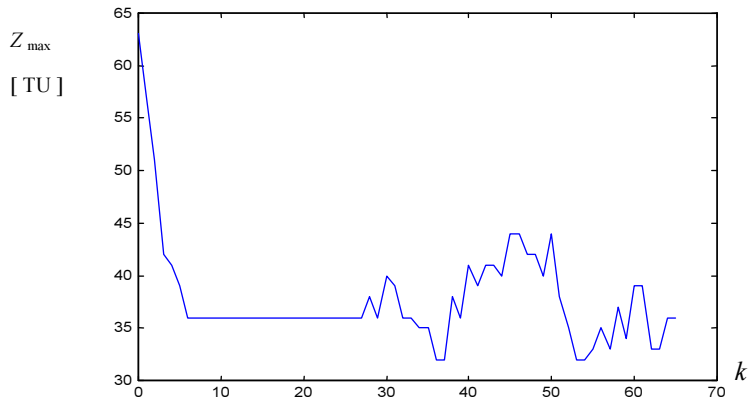

Fig. 8.5. Minimization of the bottleneck computer workload by the tabu mutation

Better outcomes from the tabu mutation are transformed into improving of solution quality obtained by the adaptive multicriteria evolutionary algorithm with tabu mutation AMEA/GP+. This adaptive evolutionary algorithm gives better results than the AMEA/GP (Fig. 8.6). After 200 generations, an average level of Pareto set obtaining is 1.8% for the AMEA/GP+, 3.4% 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.

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 16.7% for the AMEA/GP+ and 18.4% for the AMEA/GP. A maximal level is 28.5% for the AMEA/GP+ and 29.6% for the AMEA/GP. For this instance the average number of optimal solutions is 19.5% for the AMEA/GP+ 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 34.6% for the AMEA/GP+ versus 35,7% for the AMEA/GP, if the instance includes 50 tasks, 4 nodes, 5 computer types and also 220 binary decision variables.

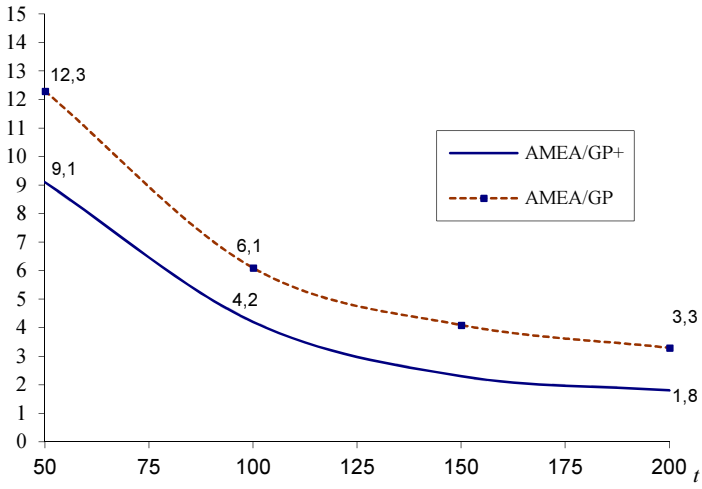

Fig. 8.6. Outcome convergence for the AMEA/GP+ and the AMEA/GP

8.9. Concluding remarks

Genetic programming is relatively new paradigm of artificial intelligence that can be used for finding task assignment and scheduling for the Comcute grid system. A computer program as a chromosome is a subject of genetic operators such as recombination, crossover and mutation. 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.

Our initial numerical experiments confirm that feasible, sub-optimal in Pareto sense, task assignments can be found by genetic programming. Although, the quality of obtained task assignment is a little better than the solution determined by an evolutionary algorithm, a paradigm of genetic 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. Balicki J.: Immune systems in multi-criterion evolutionary algorithm for task assignments in distributed computer system. LNCS, Vol. 3528, 2005, pp. 51-56.

2. Balicki J.: An adaptive quantum-based multi-objective evolutionary algorithm for efficient task assignment in distributed systems, Proc. of The WSEAS Int. Conf. on Computers, July 22-26, 2009, Rodos Island, Greece, WSEAS Press, pp. 417-422.
3. Binh, T. T., Korn, U.: Multiobjective Evolution Strategy for Constrained Optimisation Problems. Proceedings of the 15th IMACS World Congress on Scientific Computation, Modelling and Applied Mathematics, Berlin (1997), 357-362
4. Chu, W. W., Lan, L. M. T.: Task Allocation and Precedence Relations for Distributed Real-Time Systems. IEEE Transactions on Computers, Vol. C-36, No. 6 (1987) 667-679
5. Coello Coello C. A.: A Comprehensive Survey of Evolutionary-Based Multiobjective Optimisation Techniques. Knowledge and Information Systems. An International Journal, Vol. 1 (1999) 269-308
6. Fonseca, C. M., Fleming, P. J.: An Overview of Evolutionary Algorithms in Multiobjective Optimisation, Evolutionary Computation, Vol. 3, No. 1 (1995) 1-16
7. Foster I., Kesselman C, Tuecke S., *The anatomy of the grid: Enabling scalable virtual organizations*, Int. J. High Perform. Comput. Appl., 15(3), August 2001, pp. 200-222.
8. Fourman, M. P.: Compaction of Symbolic Layout Using Genetic Algorithms. Proceedings of the First International Conference on Genetic Algorithms, Hillsdale (1985) 141-153
9. Glover F., Laguna M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
10. Goldberg, D. E.: Genetic Algorithms in Search, Optimisation, and Machine Learning. Addison-Wesley Publishing Company, Massachusetts (1989)
11. Hansen M. P.: Tabu Search for Multicriteria Optimisation: MOTS. Proceedings of the Multi Criteria Decision Making, Cape Town, South Africa (1997)
12. Kafil, M. Ahmad, I.: Optimal Task Assignment in Heterogeneous Distributed Computing Systems. IEEE Concurrency, Vol. 6, No. 3 (1998) 42 - 51
13. Knowles, J., Corne, D. W.: Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. Evolutionary Computation, Vol. 8, No. 2 (2000) 149-172
14. Koza J.R.: Genetic programming. The MIT Press, Cambridge (1992)
15. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer Verlag, Berlin Heidelberg New York (1996)

16. Murthy, I., Seo, P. K.: A Dual-Descent Procedure for The File Allocation and Join Site Selection Problem on a Telecommunications Network. An International Journal Networks, Vol. 33, No. 2 (1999) 109-123
17. Nedjah N., A. Abraham, Luiza de Macedo Mourelle, *Genetic Systems Programming: Theory and Experiences,* Springer Verlag, New York 2009.
18. Russell S. J., P. Norvig, *Artificial Intelligence a modern approach*, Prentice Hall, Upper Saddle River, 2nd edition, New York 2003.
19. Schaffer, J. D.: Multiple Objective Optimisation with Vector Evaluated Genetic Algorithm. Proceedings of the First International Conference on Genetic Algorithms, Hillsdale, (1985) 93-100
20. Sheble, G. B., Britting, K.: Refined Genetic Algorithm – Economic Dispatch Example. IEEE Transactions on Power Systems, Vol. 10, No. 2 (1995) 117-124
21. Srinivas N., Deb K.: Multiobjective optimisation using nondominated sorting in genetic algorithms. Evolutionary Computation, Vol. 2, No. 3, 1994, pp. 221-248.
22. Zitzler, E., Deb, K., and Thiele, L.: Comparison of multiobjective evolutionary algorithms: empirical results. Evolutionary Computation, Vol. 8, No. 2, pp. 2000, pp. 173-195.