Evolutionary Algorithms for Multicriteria Optimization of Program Module Allocations

Jerzy Balicki

Computer Science Laboratory, The Polish Naval Academy, 81-103 Gdynia,
ul. Smidowicza 69, Poland, e-mail: jbalicki@amw.gdynia.pl

Abstract. In this paper, three evolutionary algorithms have been discussed for solving three-criteria optimization problem of finding a set of Pareto-optimal program module assignments. An adaptive evolutionary algorithm has been recommended for solving an established multiobjective optimization problem. Moreover, a multi-criterion genetic algorithm and an evolution strategy have been considered. Some numerical results have been submitted.

Keywords. Multicriteria Optimization, Module Assignment, Evolutionary Algorithms

1. Introduction

Finding allocations of distributed program modules is a significant design problem for a distributed computer system (Chu and Lan 1987). Program component allotments may reduce the total time of a program execution by taking a benefit of the particular properties of some workstations or an advantage of the load computer states. Three criteria are utilized for a quality evaluation of the module allocation: a processing load of the bottleneck machine, the cost of computers, and the total performance of workstations.

Stone applied the efficient network flow algorithm for the minimization of the operating cost of a distributed program execution in a two-computer system (Stone 1977). If the number of computers is greater than 3 or the memory in a computer is constrained, then the problem of the program completion cost minimization is NP-hard (Bokhari 1987). But, if the structure of the module communication can be a specific graph representation such as a tree or the parallel-sequence graph, then efficient algorithms based on the shortest path procedure can be exploited (Kafil and Ahmad 1998).

In this paper, three evolutionary algorithms for solving three-criteria optimization problem of finding a set of Pareto-optimal assignments are discussed. Finally, an adaptive evolutionary algorithm is recommended for solving an established multiobjective optimization problem of program module assignments.

2. Model of parallel processing

A module can be activated several times during the program lifetime. With the program module performing are associated some processes (tasks). In results, a set of program modules $\{M_1,...,M_v,...,M_V\}$ is mapped into a set of tasks $\{T_1,...,T_m,...,T_M\}$. We assume each module is performed as one task.

Let us assume the module M_v is executed on several sorts of computers taken from the set $\Pi = \{\pi_1,...,\pi_j,...,\pi_J\}$. Workstations are dispensed to the nodes from the set $W = \{w_1,..., w_i,..., w_I\}$. The computer in the node w_i is selected from the set $\Pi = \{\pi_1,...,\pi_j,...,\pi_J\}$.

We assume one and only one computer should be allocated in each node. It implies the computer allocation constraints, as follows:

$$\sum_{j=1}^{J} x_{ij}^{\pi} = 1,\ i = \overline{1,I}, \tag{1}$$

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

Because each program module is allocated to one node, then the module allocation constraints are devised, as below:

$$\sum_{i=1}^{I} x_{vi}^{m} = 1,\ v = \overline{1,V}, \tag{2}$$

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

The below vector defines the assignment of program modules to computers:

$$\begin{aligned} x = [&x_{11}^{m},..., x_{1i}^{m},..., x_{1I}^{m},..., x_{vi}^{m},..., x_{VI}^{m}, \\ &x_{11}^{\pi},..., x_{1j}^{\pi},..., x_{1J}^{\pi},..., x_{ij}^{\pi},..., x_{I1}^{\pi},..., x_{Ij}^{\pi},..., x_{IJ}^{\pi}]^{T} \end{aligned} \tag{3}$$

Restrictions (1) and (2) reduce the number of allotments x from $2^{I(V+J)}$ to $I^{V}J^{I}$.

3. Evaluations of program module assignments

The cost of the parallel program performance is the most common used measure of an allowance evaluation. Another measure is a load of the bottleneck computer (Kafil and Ahmad 1998). The workload $Z_i^{+}(x)$ of a computer allotted to the ith node for the allocation x is provided by the subsequent formula:

$$Z_i^{+}(x) = \sum_{j=1}^{J}\sum_{v=1}^{V} t_{vj} x_{vi}^{m} x_{ij}^{\pi} + \sum_{v=1}^{V}\sum_{\substack{u=1 \\ u\neq v}}^{V}\sum_{\substack{i_2=1 \\ i_2\neq i}}^{I} \tau_{vu} x_{vi}^{m} x_{ui_2}^{m},\ i = \overline{1,I}, \tag{4}$$

where

t_{vj} – the overhead performing time of the vth module by computer π_j,

τ_{vu} – the total communication time between the *v*th and the *u*th module.

A computer with the heaviest load $Z_i^+(x)$ is the bottleneck machine in the system, and its workload is the critical value that should be minimized.

The weight of the bottleneck computer is analyzed, as below:

$$Z_{\max}(x) = \max_{i \in \overline{1,I}} \left\{ Z_i^+(x) \right\}. \tag{5}$$

The other measure of the module assignment is a cost of computers that can be calculated according to the subsequent formula:

$$F_2(x) = \sum_{i=1}^{I} \sum_{j=1}^{J} \kappa_j x_{ij}^{\pi}, \tag{6}$$

where κ_j represents the cost of the computer π_j.

The third measure of the module assignment is a total amount of computer performance that can be calculated according to the following formula:

$$\tilde{F}_2(x) = \sum_{i=1}^{I} \sum_{j=1}^{J} \vartheta_j x_{ij}^{\pi}, \tag{7}$$

where ϑ_j represents the numerical performance of the computer sort π_j.

A computer performance can be measured by an assumed benchmark for performance evaluation of modern computer systems.

4. Multiobjective problem formulation

An optimal component allocation for the cost of the parallel program performing does not assure the load balance for computers in some assignments, because the workstation with the heaviest load might have a heavier consignment than another bottleneck machine for the other program module allocation in a distributed system. The workload of the bottleneck computer can be employed as an assessment measure of an allotment quality in real-time systems or in systems, where the minimization of a response time is required, too (Balicki 1999).

The relation *P* for finding Pareto-optimal solutions is a subset of $Y \times Y$, where $Y = F(X)$. If $a \in Y$, $b \in Y$, and $a_n \leq b_n, n = \overline{1,N}$, then the pair of evaluations $(a, b) \in P$. The definition of the Pareto relation respects the minimization of all criteria. For the Pareto-optimal assignment $x^* \in X$, there is no program module allocation $a \in X$ such that $(F(a), F(x^*)) \in P$ and $F(a) \neq F(x^*)$.

Let (X, *F*, *P*) be the multiobjective optimization problem for finding the Pareto-optimal solutions. It can be established, as follows:

1) X – an admissible solution set

$$\mathsf{X} = \{x \in \mathsf{B}^{I(V+J)} \mid \sum_{i=1}^{I} x_{vi}^{m} = 1,\ v = \overline{1,V},\ \sum_{j=1}^{J} x_{ij}^{\pi} = 1,\ i = \overline{1,I}\}$$

2) *F* – a vector quality criterion

$$F : \mathsf{X} \rightarrow \mathsf{R}^{3}, \tag{8}$$

where $F(x) = [Z_{\max}(x), F_2(x), \tilde{F}_2(x)]^{\mathrm{T}}$ for $x \in \mathsf{X}$

$Z_{\max}(x)$ is calculated by (5),

$F_2(x)$ is calculated by (6),

$F_2(x)$ is calculated by (7),

3) *P* - the Pareto relation.

5. Adaptive genetic algorithm

Evolutionary algorithms are divided on genetic algorithm GA, genetic programming, evolution strategies, evolutionary programming, and classification systems (Michalewicz 1996).

A ranking procedure for non-dominated individuals was introduced to avoid the prejudice of the interior Pareto alternatives by Goldberg (1989). 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. The level is increased and the procedure is repeated until the set of admissible solutions is exhausted. Thus, all non--dominated individuals have the same reproduction fitness.

The fitness for a non-feasible solution is equal to the difference between the maximal penalty $P_{\max}$ in a population and the solution penalty. If *x* is admissible, then the fitness function value is estimated, as below:

$$f(x) = P_{\max} - r(x) + L + 1, \tag{9}$$

where $r(x)$ denotes the rank of an admissible solution in the population.

The quality of attained solutions in optimization problems with one criterion increases, if the crossover probability and the mutation rate are changed in an

adaptive way proposed by Sheble and Britting (1995). Let this approach be introduced to a multi-criterion genetic algorithm with ranking procedure (Fig. 1).

A proposed adaptive multiobjective genetic algorithm AMGA may be applied for solving a spacious class of multicriteria optimization problems. Binary vectors represent solutions in this algorithm. Let us discuss the adaptive changing of a crossover probability p_c and a mutation rate p_m. At the initial population, the crossover probability is 1. 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 generations t increases, the crossover rate decreases, according to the formula $p_c = e^{-t/T_{\max}}$. where $T_{\max}$ is a maximal number of generations

BEGIN
t:=0, set the even size of population *L*
randomly generate initial population ***P*(*t*)**
calculate ranks r(x) and fitness $f(x),\ x \in \boldsymbol{P}(t)$
finish:=FALSE
WHILE NOT *finish* DO
BEGIN /* new population */
t:= t+1, $\boldsymbol{P}(t) := \varnothing$
calculate selection probabilities $p_s(x), x \in \boldsymbol{P}(t-1)$
FOR *L*/2 DO
BEGIN /* reproduction cycle */
- $p_c := e^{-t/T_{\max}}$; $p_m := e^{0.05t/T_{\max}} - 1$;
- proportional selection of potential parent pair (**a,b**) from ***P*(*t*-1)**
- crossover of a parent pair (**a,b**) with the adaptive crossover rate p_c
- bit mutation of an offspring pair (**a',b'**) with the mutation rate p_m
- ***P*(*t*)**:=***P*(*t*)∪(**a',b'}**

END
calculate ranks r(x) and fitness $f(x), x \in \boldsymbol{P}(t)$
IF (***P*(*t*)** converges OR $t \geq T_{max}$) THEN *finish*:=TRUE
END
END

Figure 1: An adaptive multi-criterion genetic algorithm AMGA

A mutation rate is 0 at the initial generation. It is an operation that can support finding a local optimal solution. The value of p_m increases with respect to the formula $p_m = e^{0.05t/T_{\max}} - 1$, exponentially. In the final population, 5.13% bits are chosen to a bit mutation.

6. Convergence to Pareto front

The AMGA can be used for solving several multiobjective optimization problems, where the set of P-optimal solutions is searched. In particular, the AMGA can be

applied for the problem (8). Simulation results corroborate that it is capable for finding the set of Pareto-suboptimal solutions.

The quality of obtained set of solutions by the AMGA is measured by the level of convergence to the Pareto front that is a closeness measure for the obtained efficient points to the known Pareto points $\{P_1, P_2, \ldots, P_U\}$. The level S of convergence is a particular measure for module assignment problems, only. It can be use for instances for which the Pareto points may be determined by an enumerative technique.

Let an algorithm finds the point (A_{u1}, P_{u2}, A_{u3}) for the cost P_{u2}. It has the same cost of computers as the uth Pareto result (P_{u1}, P_{u2}, P_{u3}). The distance between these points is $\sqrt{(P_{u1}-A_{u1})^2+(P_{u3}-A_{u3})^2}$. If the point ($A_{u1}$, P_{u2}, A_{u3}) is not find by an algorithm, we assume the distance is $\sqrt{\left(A_{u1}^{\max}-P_{u1}\right)^2+\left(P_{u3}-A_{u3}^{\min}\right)^2}$, where $A_{u1}^{\max}$ is the maximum load of the bottleneck computer for the instance of problem (8), and $A_{u3}^{\min}$ is the minimum performance of computers for the instance of problem (8). The relative level of convergence to the Pareto front is calculated, as follows:

$$S=\sum_{u=1}^{U}\sqrt{\left(\frac{A_{u1}-P_{u1}}{A_{u1}^{\max}-P_{u1}}\right)^2+\left(\frac{P_{u3}-A_{u3}}{P_{u3}-A_{u3}^{\min}}\right)^2}. \qquad (10)$$

The smaller value of the convergence level, the better front of non-dominated module assignments. An average of these deviations over 20 runs is calculated as the measure $\overline{S}$ for comparing different algorithms. Thus, it is clear that an algorithm with the smaller average level is better, in terms of its ability to obtain Pareto-optimal solutions.

7. Adaptive evolutionary algorithm

An overview of evolutionary algorithms for multiobjective optimization problems is submitted in (Zitzler *et al.* 2000). Some specific knowledge about the considered optimization problem is respected in an evolutionary algorithm (Michalewicz 1996). Outcomes are usually much better for evolutionary algorithms.

In the adaptive multicriteria evolutionary algorithm AMEA, the preliminary population is constructed to satisfy constraints (1) and (2) by introducing integer representation of chromosomes, as follows:

$$X=(X_1^m,\ldots,X_v^m,\ldots,X_V^m,X_1^\pi,\ldots,X_i^\pi,\ldots,X_J^\pi), \qquad (11)$$

where $X_v^m = i$ for $x_{vi}^m = 1$ and $X_i^\pi = j$ for $x_{ij}^\pi = 1$. Besides, $1 \le X_v^m \le I$ and $1 \le X_i^\pi \le J$.

The crossover point is randomly chosen between coordinates in *X*. Mutation is the random swap of the integer value by another one from a feasible discrete set. If the gene X_v^m is randomly taken for mutation, then the positive integer value is taken from the set $\{1,...,I\}$. If the gene X_i^π is randomly chosen, then the value is selected from the set $\{1,...,J\}$. Mutation rate is constant and it equals to 1/*l*, where *l* – the length of chromosome.

In the AMEA, the two-weighted binary tournament has been development. In the two-weight tournament selection, the roulette rule is carried out twice. If two potential parents (**a, b**) are admissible, then a dominated individual is eliminated. If both solutions are non-dominated each other, then they are accepted. If two potential parents (**a, b**) are non--admissible, then an alternative with the smaller penalty is selected.

Let the instance be considered, where there are 10 program modules, 2 nodes, and 5 computer types. It induces 30 binary decision variables and 1 073 741 824 binary program module assignments. Z_{max} is a value from [26;75] [time unit], F_2 is from [2, 10] [money unit], and $\tilde{F}_2$ from [200, 600] [Mflops].

The AMEA gives better results than the standard multicriteria evolutionary algorithm SMEA (Balicki 1999) and much better than the AMGA (Fig. 2). After 200 generations, an average level is 1,3% for the AMEA, 3% for the SMEA, and 43% for the AMGA. 30 test initial populations were prepared, and each algorithm was started 30 times from these populations.

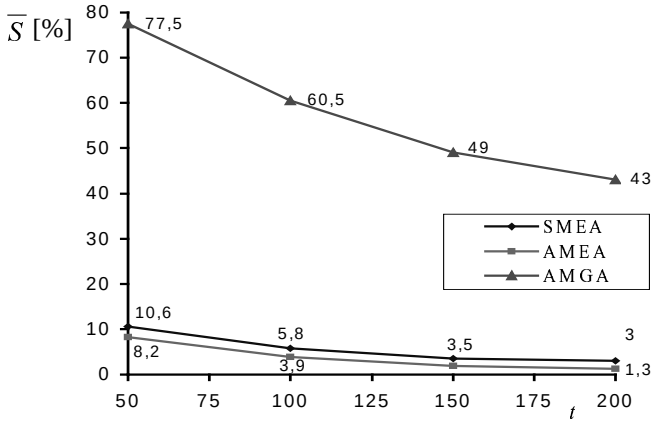

Figure 2: Convergence to Pareto front for the AMEA and the AMGA

In the AMGA, the solutions are coded as a binary vector (3). It causes the search space consists of 1 073 741 824 elements for the test instance. Standard crossover and bit mutation may result in unfeasible offspring. There are three reasons for improving quality of obtained solutions by the AMEA. For integer constrained coding of chromosomes, there are 12 decision variables in the test optimization problem. The crossover operation and the mutation ensure the search space consists of 25 600 feasible solutions, only. Moreover, the two-weighted binary tournament permits of consideration both feasible and infeasible solutions.

8. Evolution strategy

An extension of evolution strategy towards multiobjective optimization has been introduced by Kursawe (1991). A chromosome in the multicriteria evolution strategy MES for problem (8) consists of two main parts, as follows:

$$\overline{X} = (X, \sigma), \tag{12}$$

where

X – the integer decision variable vector given by (11),

σ – the standard deviation vector for X.

The new population is created from the μ individuals in the current generation by 3 steps. In the step 1, λ individuals are randomly chosen from the current population to the temporary parent set. In the step 2, crossover is carried out by the gene recombination between randomly chosen module assignments in a pair from the parent pool. Each *m*th gene is or *m*th gene from the parent *A* or *m*th gene from the parent *B*.

The temporary offspring set is transformed by mutation, in the step 3. It changes a value of each decision variable X_m of each offspring by adding the random value Δx_m that represents a random variable with a normal distribution $N(0,\sigma_m)$. Δx_m is rounded to the integer number. Each new decision variable X_m is bounded by one of the equations $1 \le X_v^m \le I$ and $1 \le X_i^\pi \le J$. Next, the standard deviations are changed by the similar way. An extended set of μ individuals from the old population and λ mutated offspring is narrowed to the μ individuals by an *elitist* selection according to the values of a fitness function. For the benchmark problem, this evolution strategy gives a bit worse products as the AMEA (Fig. 3).

For the benchmark instance, a maximal level of convergence to Pareto set was 6,4% for the MES versus 5,3% for the AMEA, but the average number of proper optimal solutions was 45,3% for the MES versus 41,6% for the AMEA. An average level of convergence, an maximal level, and the average number of proper optimal solutions became worse, if the number of modules, number of nodes, and number of computer types increase. An average level of convergence to Pareto set was 38,6% for the MES versus 37,3% for the AMEA, if the instance includes 50 program modules, 4 nodes, 5 computer types, and 220 binary decision variables.

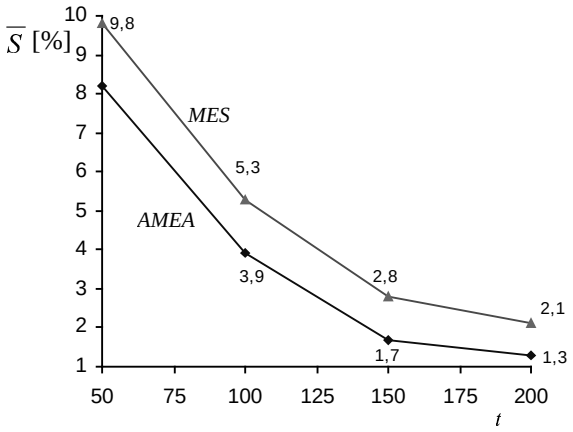

Figure 3: Minimization of the average level of convergence to the Pareto front by the MES and the AMEA

9. Concluding remarks

The adaptive evolutionary algorithm AMEA and the evolution strategy MES are capable techniques for finding program module allocations that minimize a workload of the bottleneck computer, the cost of machines and maximize a performance of system.

Introducing two additional criteria can modify an established problem: an overhead cost of program execution and the probability of program performing. In this extended approach, the scheduling of tasks for each computer should be considered (Weglarz 1998).

References

Balicki, J. (1999): Evolutionary neural networks for solving multiobjective optimization problems. In *Computational Intelligence and Applications.* Szczepaniak, S. (ed). Heidelberg-New York, A Springer Verlag Company, 108-118.

Bokhari, S. H. (1987): Assignment problems in parallel and distributed computing, Boston, Kluwer Academic Publishers.

Chu, W. W. and Lan, L. M. T. (1987): Task allocation and precedence relations for distributed real-time systems. *IEEE Transactions on Computers* **36**(6):667-679.

Goldberg, D. E. (1989): *Genetic algorithms in search, optimization, and machine learning*. Massachusetts, Addison-Wesley Publishing Company, Inc. Reading.

Kafil, M. and Ahmad, I. (1998): Optimal task assignment in heterogeneous distributed computing systems. *IEEE Concurrency* **6**(3):42-51.

Kursawe, E.: A variant of evolution strategies for vector optimization. In *Parallel problem solving from nature.* Schwefel, H.-P. and Manner, R. (1991) eds. *Lecture Notes in Computer Science* **496**:193-197.

Michalewicz, Z. (1996): *Genetic Algorithms + Data Structures = Evolutionary Programs*, Springer Verlag.

Sheble, G. B. and Britting K. (1995): Refined genetic algorithm – economic dispatch example. *IEEE Transactions on Power Systems* **10**:(2):117-124.

Stone, H. S. (1977): Multiprocessor scheduling with the aid of network flow algorithms. *IEEE Transactions on Software Engineering* **3**(3):85-93.

Weglarz, J. (1998) (ed.): *Recent advances in project scheduling.* Dordrecht, Kluwer Academic Publishers.

Zitzler, E., Deb, K. and Thiele L. (2000): Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation* **8**(2):173-195.