Abstract

In this paper we propose a solution of a multiprocessor task scheduling problem with use of a new meta-heuristic inspired by a model of natural evolution called Generalized Extremal Optimization (GEO). It is inspired by a simple co-evolutionary model based on a Bak-Sneppen model. One of advantages of the model is a simple implementation of potential optimization problems and only one free parameter to adjust. The idea of GEO metaheuristic and the way of applying it to the multiprocessor scheduling problem are presented in the paper. In this problem the tasks of a program graph are allocated into multiprocessor system graph where the program completion time is minimized. The problem is known to be a NP-complete problem. In this paper we show that GEO is able to solve this problem with better performance than genetic algorithm.

Keywords: multiprocessor task scheduling problem, Generalized Extremal Optimization, GEO, genetic algorithm

1 Introduction

In the present-day many optimization problems in science and engineering (Pardalos and Romeijn, 2002) are difficult to solve. These problems are often NP-complete problems (Garey and Johnson, 1979). The problems belong to the class of computational problems for which no efficient solution algorithm has been found. NP-complete problems have been only solved approximately by existing techniques like randomization, parameterization or using heuristics (meta-heuristics). The most methods based on local search algorithms in a complex problem with multiple local optima often converge to the local minimum (Eldred, 1998).

A more general approach is to use a global search algorithm. In this case we can find global optimum, but it requires more computational cost e.g. time for solving optimization problem. One of class of the global optimization algorithms is particularly worth to consider – algorithms based on natural phenomena. This motivation is based on observation of natural processes which are fre-
Figure 1: Program and system graphs: the graph of four-processor system in FULL4 architecture (a), an example of a program graph (b).

The multiprocessor task scheduling problem is one of NP-complete problems. The objective of scheduling is to minimize the completion time of parallel application divided into tasks by properly allocating the tasks to the processors (Kwok and Ahmad, 1999). In this problem the tasks organized in a graph need to be allocated into a multiprocessor system graph. The optimal solution of this problem is very difficult to obtain. This problem have been often solved by algorithms based on a natural phenomena mentioned above.

In this paper we have proposed a relatively new metaheuristic called GEO to solve the multiprocessor scheduling problem. Experiments show that this algorithm is very efficient for this problem and provides better results than GA.

The paper is organized as follows. In the next section we describe the multiprocessor task scheduling problem. Section 3 presents GEO algorithm applied for the scheduling problem. Next, in Section 4 experimental results are given. Last section concludes the paper.

2 Multiprocessor Task Scheduling Problem

The problem is defined as follows (see, e.g. (Seredynski and Zomaya, 2002)). For a given number of processors $m$ and a given topology of connections between them, and a parallel program graph consisting of $n$ tasks with precedence relations and the run times on individual processors, we have to find schedule which has the shortest run time. The schedule must define for each task a time of execution and the processor to be used for execution of the particular task. A topology of multiprocessor system is represented by weighted and directed graph $G_s = (V_s, E_s)$ with $m$ vertices $V_s = \{v_0, ..., v_{m-1}\}$ and a number of edges $e_{ij} \in E_s$. The vertices represent processors $P_s = \{P_0, ..., P_{m-1}\}$. Edges are connections between nodes of
program graph representing processors. Edge $e_{ij}$ is the direct connection between nodes representing processor $P_i$ and processor $P_j$. All connections (channels of data exchange) are bi-directional. Graph $G_s$ is called a system graph. Fig. 1a presents a four-processors system in architecture called FULL4.

Let’s consider program $Z$ consisting of $n$ indivisible tasks $Z = (z_0, ..., z_{n-1})$. If the results of realization of problem $z_i$ are input data of task $z_j$ (task $z_i$ must be completed before task $z_j$ is started), then these tasks are in precedence relations: $z_i \prec z_j$. If program $Z$ runs on a multiprocessor system, then tasks which are not in a precedence relation can be run simultaneously on different processors. With every task $z_i$ is related processing time $b_i$ (computational cost). Additionally, transfer of task $z_i$ results to task $z_j$ may be related with noticeable data transfer time through the communication channel between processors, if tasks are run on different processors. For example, computational cost may be proportional to the size of these results. Therefore, for every precedence relation between tasks $z_i$ and $z_j$ in program $Z$ communication cost of sending the results from $z_i$ to $z_j$ is defined if they are run on neighboring processors. For example, computational cost may be proportional to the size of these results. Therefore, for every precedence relation between tasks $z_i$ and $z_j$ in program $Z$ communication cost of sending the results from $z_i$ to $z_j$ is defined if they are run on neighboring processors. For example, computational cost may be proportional to the size of these results. Therefore, for every precedence relation between tasks $z_i$ and $z_j$ in program $Z$ communication cost of sending the results from $z_i$ to $z_j$ is defined if they are run on neighboring processors. For example, computational cost may be proportional to the size of these results. Therefore, for every precedence relation between tasks $z_i$ and $z_j$ in program $Z$ communication cost of sending the results from $z_i$ to $z_j$ is defined if they are run on neighboring processors. For example, computational cost may be proportional to the size of these results. Therefore, for every precedence relation between tasks $z_i$ and $z_j$ in program $Z$ communication cost of sending the results from $z_i$ to $z_j$ is defined if they are run on neighboring processors. For example, computational cost may be proportional to the size of these results. Therefore, for every precedence relation between tasks $z_i$ and $z_j$ in program $Z$ communication cost of sending the results from $z_i$ to $z_j$ is defined if they are run on neighboring processors.

The purpose of scheduling is to distribute task among processors in such a way to minimize the total execution of time $T$.

3 The Generalized Extremal Optimization algorithm

3.1 Idea of the algorithm

The idea of this algorithm is based on Bak-Sneppen model (Bak and Sneppen, 1993). Evolution in this model is driven by a process in which the weakest species in the population, together with its nearest neighbors, is always forced to mutate. The dynamics of this extremal process show characteristics of Self-Organized Criticality (SOC), such as punctuated equilibrium, that are also observed in natural ecosystems. Punctuated equilibrium is a theory in evolutionary biology. It states that in evolution there are periods of stability punctuated by a change in environment that forces relatively rapid adaptation by generating "avalanches". The probability distribution of these avalanches is described by a power law in the form $p_i = k^{-\tau_i}$, where $p$ is a probability of mutation of $i$-th individual, $k$ is position of individual in a rank, $\tau$ is a positive parameter. If $\tau \to 0$ the algorithm search a solution randomly, while $\tau \to \infty$ the algorithm provides deterministic searching. This idea is used in this algorithm. Bak and Sneppen developed a simplified model of an ecosystem in which $N$ species are placed side by side on a line. Fig. 2 shows the population of species in the Bak-Sneppen model and the idea of GEO algorithm.

In the GEO approach, a population of species is a string of bits that encodes the design variables of the optimization problem, and each bit corresponds to one
species. In Fig. 2 two variable function $F(x_1, x_2)$ is optimized. Each variable is coded using seven bits, so the population of the algorithm consisting of 14 bits (upper part of Fig. 2). Each bit of the string is considered as the species (lower part of Fig. 2) of the Bak-Sneppen model. The each bit representing species $e_i (i = 1, N)$ has a value 0 or 1. A number of bits per variable depends on the type of the problem. In contrast to GA, in GEO there is not a population of strings, but one population of bits represented by one string. In this algorithm each bit is forced to mutate with a probability proportional to its fitness. The fitness is a number assigned to each bit of this string that indicates the level of adaptability of each bit of the population, according to the gain or loss to value of the fitness function if the bit is mutated.

3.2 Presentation of the algorithm

According to the paper by (Sousa et al., 2004) the GEO algorithm can be described as follows:

1. Initialize randomly a binary string of length $L$ that encodes $N$ design variables of bit length $L/N$.

2. For the current configuration $C$ of bits, calculate the objective function value $V$ and set $C_{best} = C$ and $V_{best} = V$.

3. For each bit $i$ do,

   (a) flip the bit (from 0 to 1 or 1 to 0) and calculate the objective function value $V_i$ of the string configuration $C_i$,

   (b) set the bit fitness $F_i$ as $(V_i - R)$, where $R$ is a constant. It serves only as a reference number and can assume any value. The bit fitness indicates the relative gain (or loss) that one has in mutating the bit.

   (c) return the bit to its original value.
4. Rank the $N$ bits according to their fitness values, from $k = 1$ for the least adapted bit to $k = N$ for the best adapted. In a minimization problem higher values of $F_i$ will have higher ranking. Otherwise for maximization problems, if two or more bits have the same fitness, rank them in random order, but following the general ranking rule.

5. Choose a bit $i$ to mutate according to the probability distribution $P_i = k^{-\tau}$, where $\tau$ is an adjustable parameter.

6. Set $C = C_i$ and $V = V_i$.

7. If $F_i < F_{best}$ ($F_i > F_{best}$, for a maximization problem) then set $F_{best} = F_i$ and $C_{best} = C_i$.

8. Repeat steps 3 to 7 until a given stopping criteria is reached.

9. Return $C_{best}$ and $F_{best}$.

4 Experiments

4.1 Representation of the individual

We have implemented the GEO algorithm to a multiprocessor task scheduling problem. The population consists of one binary string. The number of bits in the string is equal to $N_t * l_{\text{bits}}^{\text{processor}}$, where $N_t$ – a number of tasks in a program graph and $l_{\text{bits}}^{\text{processor}}$ – a number of bits used to code a processor number. For example, for eight processors (from 0 to 7) we need three bits to represent each processor, so $l_{\text{bits}}^{\text{processor}} = 3$ and for a program graph consisting of $N_t = 8$ tasks the total number of binary string is equal to 24.

4.2 Adjustment of a $\tau$ parameter

We have implemented GEO with the representation of a population described above. Firstly, we have observed the average value $V$ of fitness function with respect to the $\tau$ parameter.

In the Fig. 3 we compare two values of the $\tau$ parameter: 0.5 and 2. For a $\tau = 0.5$ (see, Fig. 3a), solution is searched more randomly than for $\tau = 2.0$ (see, Fig. 3b). Speed of convergence of the algorithm to the optimal value is faster for $\tau = 2.0$ than $\tau = 0.5$.

The $\tau$ parameter influences substantially on the results. In the Fig. 4 we can notice influence of this parameter for obtained results. For small program graphs ($N_t \leq 20$) (see, Fig. 4a) the best value of tau parameter is 1.0. For larger program graphs (see, Fig. 4b) the tau parameter should be increased from 1.5 to 3. Above of these values the algorithm gives only worse results. For the most carried out experiments the $\tau$ parameter in the range from 1.5 to 2.5 was an optimal value.

4.3 Two-processor scheduling with GEO

A number of experiments with deterministic program graphs known in the literature (see, e.g., (Seredynski and Zomaya, 2002)) and random program graphs (Bollobas, 1985) has been conducted. The results were compared with those obtained with use of GA. We used the following parameters in the experiment: $\tau$
Random program graph g100_1; system graph FULL4
(a)

Random program graph g100_1; system graph FULL4
(b)

Figure 3: Speed of convergence of the algorithm to the optimal value for (a) $\tau = 0, 5$, (b) $\tau = 2, 0$. Comparison of the $V$ – actual value of fitness function and $V_{\text{best}}$ – the best value of fitness function.
Figure 4: Influence of the $\tau$ parameter on the results for (a) a small program graph, (b) a large program graph (averaged on 10 runs)
Table 1: Comparison of the algorithms: GEO, GA, SA, TS for several program graphs. The best and average (in rounded brackets) values for 10 runs. The optimal solutions are in bold.

<table>
<thead>
<tr>
<th>Program graph</th>
<th>GEO</th>
<th>GA</th>
<th>SA</th>
<th>TS</th>
</tr>
</thead>
<tbody>
<tr>
<td>tree15</td>
<td>9 (9)</td>
<td>9 (9)</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>g18</td>
<td>9 (9)</td>
<td>46 (46)</td>
<td>46 (46)</td>
<td>46 (46)</td>
</tr>
<tr>
<td>g40</td>
<td>80 (80)</td>
<td>80 (80)</td>
<td>80 (80)</td>
<td>80 (80)</td>
</tr>
<tr>
<td>gauss18</td>
<td>44 (44)</td>
<td>44 (44)</td>
<td>44 (44)</td>
<td>49 (49)</td>
</tr>
<tr>
<td>g25</td>
<td>495 (495)</td>
<td>495 (495)</td>
<td>495 (495)</td>
<td>495 (495)</td>
</tr>
<tr>
<td>g25_5</td>
<td>94 (95)</td>
<td>94 (95)</td>
<td>94 (95)</td>
<td>100 (100)</td>
</tr>
<tr>
<td>g25_10</td>
<td>62 (62)</td>
<td>62 (62)</td>
<td>62 (62)</td>
<td>62 (62)</td>
</tr>
<tr>
<td>g100_1</td>
<td>1481 (1481)</td>
<td>1481 (1481)</td>
<td>1481 (1481)</td>
<td>-</td>
</tr>
<tr>
<td>g100_5</td>
<td>395 (398)</td>
<td>402 (412)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>g100_10</td>
<td>174 (177)</td>
<td>174 (177)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>g200_1</td>
<td>3025 (3025)</td>
<td>3025 (3025)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>g200_5</td>
<td>558 (558)</td>
<td>570 (574)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>g200_10</td>
<td>484 (492)</td>
<td>516 (519)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

parameter = 1.5 - 2.0 (GEO algorithm), population size = 200, mutation probability = 0.03, crossover probability = 0.5 (GA algorithm). We also compare the GEO and GA algorithm to another algorithms described in (Swiecicka et al., 2006): TS – tabu search, SA – simulated annealing.

The first program graph used in experiments is tree15. It is a binary tree consisting of 15 tasks. All computational and communication costs are the same and equal to 1. The optimal response time T for tree15 in the two-processor system equals 9. Experiments (Tab. 1) have shown that for this program graph all algorithms have found an optimal solutions. The next graphs (g18 and g40) are also simple example of program graphs. The Gauss program graph is more difficult because of its nonregular structure. Not all algorithms found optimal solution. GEO found optimal solution in every run in opposite to GA, where optimal solution was found in only one run for 10 runs.

In the next experiments random program graphs has been used. In the experiments we changed $\tau$ parameter to 2.0 for GEO algorithm. Experiments with a relative small graph (g25) shows that not all algorithms found solutions. In the larger random graphs (g100 and g200) we carried out experiments with GEO and GA algorithms. Both algorithms found optimal solutions only for simple variant of these graphs: g100_1 and g200_1, where an average communication cost equal to an average computational cost. In the other variants experiments shows that GEO is better than GA.
Table 2: Comparison of the algorithms: GEO and GA for several program graphs and k=4 processors or k=8 processors. The best values for 10 runs.

<table>
<thead>
<tr>
<th>Program graph</th>
<th>k=4</th>
<th>k=8</th>
<th>k=4</th>
<th>k=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEO</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>GA</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>tree15</td>
<td>26</td>
<td>26</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>g18</td>
<td>45</td>
<td>45</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>gauss18</td>
<td>44</td>
<td>44</td>
<td>44</td>
<td>44</td>
</tr>
<tr>
<td>g100</td>
<td>198</td>
<td>208</td>
<td>199</td>
<td>202</td>
</tr>
<tr>
<td>g200</td>
<td>483</td>
<td>498</td>
<td>479</td>
<td>482</td>
</tr>
</tbody>
</table>

4.4 Multiprocessor scheduling with GEO

In this section we present results of GEO algorithm for more than two processors. This variant is more complicated, because the individual in GEO is two times (for four processors) or three times (for eight processors) longer than for two processors.

For small program graphs either GEO and GA found the same values (Tab. 2). In random graphs GEO is better than GA especially for g200_10. This result prove that GEO can be used in task scheduling problem for solving huge program graphs in multiprocessor graphs.

5 Conclusions

Applying the GEO algorithm to the task scheduling problem has confirmed that this algorithm is useful for this problem. The results of the experiments show advantages of GEO. Simplicity of implementation of the algorithm is one of the advantages. In the opposite to GA we have only one parameter to adjust. Results of the experiments for GEO algorithm are better than GA for optimal parameters of both algorithms.

References


