A CROSS-ENTROPY BASED POPULATION LEARNING ALGORITHM FOR DISCRETE-CONTINUOUS SCHEDULING WITH CONTINUOUS RESOURCE DISCRETISATION

A problem of scheduling nonpreemptable tasks on parallel identical machines under constraint on discrete resource and requiring, additionally, renewable continuous resource to minimize the schedule length is considered in the paper. A continuous resource is divisible continuously and is allocated to tasks from given intervals in amounts unknown in advance. Task processing rate depends on the allocated amount of the continuous resource. The considered problem can be solved in two steps. The first step involves generating all possible task schedules and second - finding an optimal schedule among all schedules with optimal continuous resource allocation. To eliminate time consuming optimal continuous resource allocation, a problem $\Theta_Z$ with continuous resource discretisation is introduced. Because $\Theta_Z$ is NP-hard a population-learning algorithm (PLA2) is proposed to tackle the problem. PLA2 belongs to the class of the population-based methods. Experiment results proved that PLA2 excels known algorithms for solving the considered problem.

1. PROBLEM FORMULATION

We define a problem $\Theta_Z$ in same way as in [17]. Namely, let $J = \{J_1, J_2, \ldots, J_n\}$ be a set of nonpreemptable tasks, with precedence relations and ready times $r_i = 0, i = 1, 2, \ldots, n$, and $P = \{P_1, P_2, \ldots, P_m\}$ be a set of parallel and identical machines, and there is one additional renewable discrete resource in amount $U = 1$ available. A task $J_i$ can be processed in one of the modes $l_i = 1, 2, \ldots, D_i$, for which $J_i$ requires a processor from $P$ and amount of the additional resource known in advance. The processing mode of $J_i$ cannot change during the processing. For each task two vectors are defined:
a processing times vector \( \tau_i = \{\tau_{i1}, \tau_{i2}, \ldots, \tau_{iD_i}\} \), where \( \tau_{il} \) is the processing time of task \( J_i \) in mode \( l = 1, 2, \ldots, D_i \) and a vector of additional resource quantities allocated in each processing mode \( u_i = \{u_{i1}, u_{i2}, \ldots, u_{iD_i}\} \). The problem is to find processing modes for tasks from \( J \) and their sequence on processors from \( P \) such that schedule length \( Q = \max\{C_i\}, i = 1, \ldots, n \) is minimized.

2. POPULATION LEARNING ALGORITHM

General idea of the present implementation of Population learning algorithm first proposed in [8] is shown in the following pseudo code:
Procedure PLA2
Begin
Create an initial population \( P_0 \) of size \( x_0 - 1 \) using procedure CE.
Learn an individual – TSI with aid of procedure TS.
Create population \( P_1 = P_0 + TSI \).
Learn all individuals in \( P_1 \) with aid of procedure IBEA.
Output the best solution to the problem.
End.

All used in PLA2 procedures learn individuals (solutions) concerning which the following assumptions are made:
– an individual (a solution) is represented by an \( n \)-element vector \( S = \{c_i\} \ 1 \leq i \leq n \},
– all processing modes of all tasks are numbered consecutively. Thus processing mode \( l_b \) of task \( J_b \) has the number \( c_b = \sum_{i=1}^{b-1} D_i + l_b \), where \( D_i \) is the number of processing modes of task \( J_i \),
– all \( S \) representing feasible solutions are potential individuals;
– each individual can be transformed into a schedule by applying LSG, which is a specially designed list-scheduling algorithm for discrete-continuous scheduling;
– each schedule produced by the LSG can be directly evaluated in terms of its fitness.

2.1. A Cross-Entropy Algorithm

A procedure CE using cross-entropy method for combinatorial optimization described in [3] and modified for solving \( \Theta_Z \) problem is shown in the following pseudo code:
Procedure CE

Begin
Set $ic := 1$ ($ic$ – iteration counter), $ic^{stop}$ – maximal number of iterations, $a := 1$.
In $\hat{p}$ set $\hat{p}_j = \{ p_{ji} = 1/n \mid 1 \leq i \leq n \}, 1 \leq j \leq n$.
Set $\hat{p}'_{ji} = \{ p_{ji} = 1/D_j \mid 1 \leq l \leq D_j \}, 1 \leq j \leq n, 1 \leq i \leq n$.
While $ic \leq ic^{stop}$ do
Generate a sample $S_1, S_2, \ldots, S_s, \ldots, S_n$ of solutions with success probability vectors $\hat{p}$ and $\hat{p}'$.
Order $S_1, S_2, \ldots, S_s, \ldots, S_n$ by nondecreasing values of their fitness function.
Set $\gamma = \lceil \rho \cdot N \rceil$, $\rho \in (0, 1)$.
In $\hat{p}$ set $\hat{p}_j = \left\{ p_{ji} = \frac{\sum_{s=1}^{\gamma} I(S_s(j) = i)}{\gamma} \mid 1 \leq i \leq n \right\}, 1 \leq j \leq n$, (1)
$I(S_s(j) = i) = 1, I(S_s(j) \neq i) = 0$, where $S_s(j)$ – number of the task located on $j$-th place in $s$-th solution $S$.
Set $\hat{p}'_{ji} = \left\{ p_{ji} = \frac{\sum_{s=1}^{\gamma} I(S_s(ji) = l)}{\gamma} \mid 1 \leq l \leq D_j \right\}, 1 \leq j \leq n, 1 \leq i \leq n$, (2)
$I(S_s(ji) = l) = 1, I(S_s(ji) \neq l) = 0$, where $S_s(ji)$ – an execution mode of task $i$ located on $j$-th place in $s$-th solution $S$.
Save the first $h = \lceil K \cdot PS/ic^{stop} \rceil$ best solutions from the ordered sample into $P_0$ under address $a$. Set $a := a + h$.
Set $ic := ic + 1$.
EndWhile
EndProc

In the presented pseudo code parameters $K$ – the number of islands and $PS$ – the population size are explicitly explained in the description of IBEA.

2.2. Tabu Search Algorithm

Tabu search is a metaheuristic used in PLA2 (see [5]). The considered tabu search procedure is shown in the following pseudo code:

Procedure TS

Begin
Set $S_0 = \text{initial solution } TSI (l_i = 1, 1 \leq i \leq n)$.
Set the best solution $S_{\text{best}} = S_0$. 

47
Set Tabu List \( TL = \{\emptyset\} \).
Set \( N_t = \{S_i\} \) and \( N_{md} = \{\emptyset\} \).
Repeat the following \textit{max\_number\_of\_iterations} times:

- Find the best legal neighbour \( S_{bln} \) of \( S_0 \), i.e. the best across \( N_t \) and \( N_{md} \) neighbour which is not on TL.
- Set \( S_0 = S_{bln} \).
- If \( S_{bln} \) is more fit than \( S_{best} \) then \( S_{best} = S_{bln} \).
- Put \( S_{bln} \) on the Tabu list.

If the fitness of \( S_0 \) has not improved after \( nit \) number of iterations construct a new solution by moving a task \( J_i \) in \( S_0 \) to one of the chosen randomly less frequently visited places on the task list and assigning to it one of the chosen randomly less frequently assigned execution modes.

EndRepeat
EndProcedure.

\subsection*{2.3. An Island-Based Evolution Algorithm}

The following pseudo-code shows main stages of the IBEA algorithm:

\textbf{Procedure IBEA}
Begin
- Set number of islands \( K \), number of generations \( PN \) for each island, size of the population for each island \( PS \). For each island \( I_k \), generate an initial population \( PP_0 \).
- While no stopping criteria is met do
  - For each island \( I_k \) do
    - Evolve \( PN \) generations using PBEA.
    - Send the best solution to \( I_{(k \mod K) + 1} \).
    - Incorporate the best solution from \( I_{((K+k-2) \mod K) + 1} \) instead of the best one.
  - EndWhile
- Find the best solution \( S_{best} \) across all islands and save it as the final one.
End.

\textbf{PBEA algorithm} is shown in the following pseudo-code:

\textbf{Procedure PBEA}
Begin
- Set population size \( PS \), generate a set of \( PS \) individuals to form an initial population \( PP_0 \);
- Set \( ic := 0 \); (\( ic \) - iteration counter);
- While no stopping criteria is met do
  - Set \( ic := ic + 1 \)
End.
Calculate fitness factor for each individual in $PP_{ic-1}$ using LSG;

Form new population $PP_{ic}$:
- Select randomly a quarter of PS of individuals from $PP_{ic-1}$ (probability of selection depends on fitness of an individual);
- Produce a quarter of $PS$ of individuals by applying crossover operator to previously selected individuals from $PP_{ic-1}$;
- Produce a quarter of $PS$ of individuals by applying mutation operators to previously selected individuals from $PP_{ic-1}$;
- Generate half of a quarter of $PS$ of individuals from set of potential individuals (random task processing mode, and task order);
- Generate half of a quarter of $PS$ of individuals from set of potential individuals (random task processing mode, and ascending order of the task numbers).

EndWhile
End.

LSG algorithm used within PBEA is carried in the three steps:
Procedure LSG
Begin
Step 1. Construct a list of tasks from the code representing individuals. Set loop over tasks on the list.
Step 2. Within the loop, allocate current task to a processor considering the amount of a continuous resource allotted to the task, and minimizing the beginning time of its processing. Continue with tasks until all have been allocated.
Step 3. If the resulting schedule has task delays, a fitness of the individual $S$ is calculated as $Q_u = -\max\{C_i\}$, $i = 1, \ldots, n$. Otherwise, $Q_u = \max\{C_i\}$, $i = 1, \ldots, n$.
End.

3. COMPUTATIONAL EXPERIMENTS

The proposed Cross-Entropy Based population learning algorithm for solving discrete-continuous scheduling problems with continuous resource discretisation (PLA2) was implemented and tested. For testing purposes three combinations of $n \times m$ were considered ($n$ – the number of tasks and $m$ – the number of machines): 10x2, 10x3, and 20x2. For each combination $n \times m$ 100 instances of a problem $\Theta_Z$ were generated and three discretisation levels were considered: 10, 20, and 50, which makes 900 instances of the problem. Each
instance was tested 24 times. Value of the RE calculated according to the formulae \( RE = \frac{Q_{PLA2} - Q_{best-known}}{Q_{best-known}} \) for each instance was used to find mean and maximum relative errors. \( RE_{\text{mean}} \) and \( RE_{\text{max}} \) of the solutions found by PLA2, PLA1, and \( G_{\text{dskr}} \) are presented in Table 1.

### Table 1

<table>
<thead>
<tr>
<th>Nxm</th>
<th>D=10</th>
<th>D=20</th>
<th>D=50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PLA2</td>
<td>PLA1</td>
<td>PLA2</td>
</tr>
<tr>
<td>10x2</td>
<td>2.75%</td>
<td>1.82%</td>
<td>7.59%</td>
</tr>
<tr>
<td></td>
<td>9.29%</td>
<td>9.39%</td>
<td>15.34%</td>
</tr>
<tr>
<td>10x3</td>
<td>2.99%</td>
<td>3.76%</td>
<td>14.58%</td>
</tr>
<tr>
<td></td>
<td>10.66%</td>
<td>11.58%</td>
<td>25.18%</td>
</tr>
<tr>
<td>20x2</td>
<td>2.70%</td>
<td>2.49%</td>
<td>13.25%</td>
</tr>
<tr>
<td></td>
<td>9.21%</td>
<td>9.48%</td>
<td>19.40%</td>
</tr>
</tbody>
</table>

In our tests PLA2 improved 80.33% of 300 the best known solutions for combinations 10x2, 10x3, and 20x2, and reduced average of \( RE_{\text{mean}} \) compared to PLA1 and \( G_{\text{dskr}} \). Table 2 shows how many percents on average the solutions found by PLA2 were better than the solutions found by PLA1 and \( G_{\text{dskr}} \).

### Table 2

<table>
<thead>
<tr>
<th>nxm</th>
<th>PLA1</th>
<th>( G_{\text{dskr}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>10x2</td>
<td>0.25%</td>
<td>6.89%</td>
</tr>
<tr>
<td>10x3</td>
<td>1.22%</td>
<td>13.36%</td>
</tr>
<tr>
<td>20x2</td>
<td>1.37%</td>
<td>13.80%</td>
</tr>
</tbody>
</table>

Mean time required by PLA2 and PLA1 to find a solution for 10x2 on Pentium (R) 4 CPU 3.00GHz compiled with aid of Borland Delphi Personal v.7.0 was 5s. Mean time required by \( G_{\text{dskr}} \) to find a solution for 10x2 on supercomputer Silicon Graphics Power Challenge XL with twelve RISC MIPS R8000 processors was 33s. Such results make PLA2 quite an effective algorithm for solving problem \( \Theta_z \). As the future work a modification of PLA2 for solving discrete-continuous problems without continuous resource discretisation should be considered.

### REFERENCES

1. Alba E., Troya J., *Analysis of Synchronous and Asynchronous Parallel Distributed Genetic Algorithms with Structured and Panmictic Islands*, [In:] Jose Rolim et al., Eds., *Proceedings of the 10th Symposium on Parallel and Distributed Processing*, San Juan, Puerto Rico, 12-16 April 1999, USA.


ALGORYTM UCZENIA POPULACJI Z WYKORZYSTANIEM ENTROPII SKROŚNEJ DO ROZWIĄZYWANIA DYSKRETNO-CIĄGŁYCH PROBLEMÓW SZEREGOWANIA Z DYSKRETYZACJĄ ZASOBU CIĄGŁEGO

(Streszczenie)