CMST 2(1) 73-85 (1996) DOI:10.12921/cmst.1996.02.01.73-85

A SIMULATED ANNEALING ALGORITHM FOR SOME CLASS OF DISCRETE-CONTINUOUS SCHEDULING PROBLEMS

Joanna Józefowska, Marek Mika and Jan Węglarz

Poznań University of Technology, Institute of Computing Science, Piotrowo 3A, 60-965 Poznań, Poland
e-mail: mika@man.poznan.pl

Abstract

In this paper a Simulated Annealing algorithm is proposed and applied to the n-job and m-machme discrete-continuous scheduling problem with the objective to minimize the makespan. This problem can be divided into two interrelated subproblems (i) constructing a feasible sequence of jobs on machines and (ii) allocating the continuos resource among jobs already sequenced. The application of the Simulated Annealing algorithm operating on a space of feasible sequences is presented. By computational experiment on randomly generated test problems, the proposed algorithm is compared with two other heuristics, namely Multi-start Iterative Improvement algorithm and Random Sampling Technique.

1. Introduction

We consider a problem of scheduling n independent, nonpreemptable jobs on m identical parallel machines under additional resources. In the classical model of scheduling the additional resource can be allotted to jobs in amounts from a given finite set only. For this model a number of results are known in the literature, concerning the computational complexity analysis, as well as optimization and approximation algorithms [2], But in many practical situations the additional resource can be allotted to a job in an amount (unknown in advance) from a given interval (a continuous resource). In the problem considered in this paper jobs require for their processing simultaneously both: discrete and continuous resources. This problem will be called a discrete-continuous scheduling problem [3], Such problems arise, for example, when jobs are assigned to parallel processors driven by a common electric, hydraulic or pneumatic power source. The processors correspond to the discrete resource and power corresponds to the continuous resource. The processing rate of a job depends on the amount of the continuous resource allotted to it at a time.

The problem is to find a sequence of n jobs on m machines and a continuous resource allocation among jobs already sequenced, that optimize the makespan. Of course, other criteria can be examined, as well.

2. Problem Formulation

Let us consider a discrete-continuous scheduling problem with *n* nonpreemptable and independent jobs, m identical, parallel machines and a single continuous renewable resource. We assume that all jobs and machines are simultaneously available at the start of the process, and that each machine can process one job at a time only. The total amount of the continuous resource available at a time is limited (without loss of generality this limit can be fixed at 1). The processing rate of job *i* depends on the amount of the continuous resource allotted to this job at time t and is described by the equation:

$$\dot{x}_i(t) = \frac{dx_i(t)}{dt} = f_i\left[u_i(t)\right], \qquad x_i(0) = 0, \quad x_i\left(C_i\right) = \widetilde{x}_i$$

where $x_i(t)$ is the state of job *i* at time *t;*

f_i is a continuous nondecreasing function, $f_i(0) = 0$;

$u_i(t)$ is the resource amount allotted to job i at a time *t.*

we assume that $\forall t, \quad 0 \le u_i(t) \le 1$ and $\forall t \sum_{i=1}^{n} u_i(t) = 1$;

C_i is (unknown in advance) completion time of job i;

$\widetilde{x}_i$ is the final state (processing demand) of job i;

The problem is to find a sequence of jobs on machines and, simultaneously, a continuous resource allocation, which minimize the schedule length (makespan):

$$M = \max_i \left\{C_i\right\}$$

The continuous resource allocation is defined by a piece-wise continuous, non-negative vector function:

$$\underline{u}^*(t) = \left[u_1^*(t), u_2^*(t), \ldots, u_n^*(t)\right]$$

which values $\underline{u}^* = \left(u_1^*, u_2^*, \ldots, u_n^*\right)$ are continuous resource allocations corresponding to the minimal value M* of M.

It has been proved [3] that for the problem with uniform machines and functions $f_i \le c_i u_i$ where $c_i = f_i(1)$ and i = 1,2, ..., n, a schedule with minimal makespan can be obtained by scheduling all jobs on the fastest machine, and processing them using the total available amount of the continuous resource.

Let us consider concave functions f_i. For this case the parallel configuration is optimal.

A feasible schedule, which is a solution of a discrete-continuous problem, can be divided into $p \leq n$ intervals of length M_k $(k = 1,2,\ldots,p)$, defined by the completion times of consecutive jobs. Let S denote a sequence $Z_1,\ldots,Z_p$ associated with each feasible schedule, where Z_k $k = 1, 2, \ldots, p$ is a combination of jobs processed in the interval M_k. Sequence $Z_1,\ldots,Z_p$, is feasible if the number of elements in each combination is at most m, and each job appears in at least one combination in S. If job appears in more than one Z_ks these combinations must be consecutive ones (jobs are nonpreemptable).

Example 1

Consider the following feasible schedule of 10 jobs on 3 machines. For simplicity it is assumed $u_i(t) = \frac{1}{3}$ for every t (Fig 1).

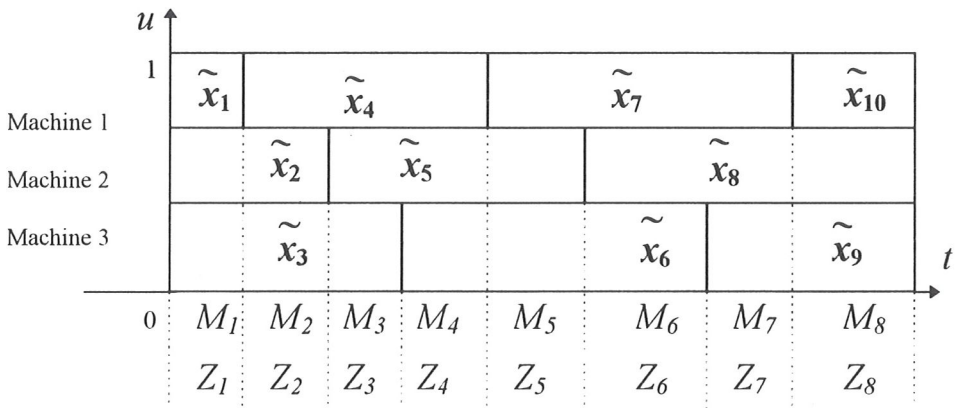

Fig 1. An example of a schedule for discrete-continuous scheduling

The corresponding feasible sequence of combinations $Z_1, Z_2, \ldots, Z_8$ is:

S= {1,2,3}, {4,2,3}, {4,5,3}, {4,5,6}, {7,5,6}, {7,8,6}, {7,8,9}, {10,8,9}

Further, for each job i its processing demand $\tilde{x}_i$, $i = 1,2, \ldots,n$, can be divided into parts $\tilde{x}_{ik} \geq 0$, $k = 1, \ldots, p$, processed in particular time intervals (combinations), as it is illustrated in Fig. 2.

For a given feasible sequence S one can find a division of processing demands of jobs $\tilde{x}_i$ among combinations in S, which leads to a schedule of the minimum length from among all feasible schedules generated by S. Such a division is an optimal one. To this end a nonlinear programming problem can be formulated in which the sum of the minimum-length intervals generated by consecutive combinations in S, as functions of $\tilde{x}_{ik}$'s is minimized subject to obvious constrains.

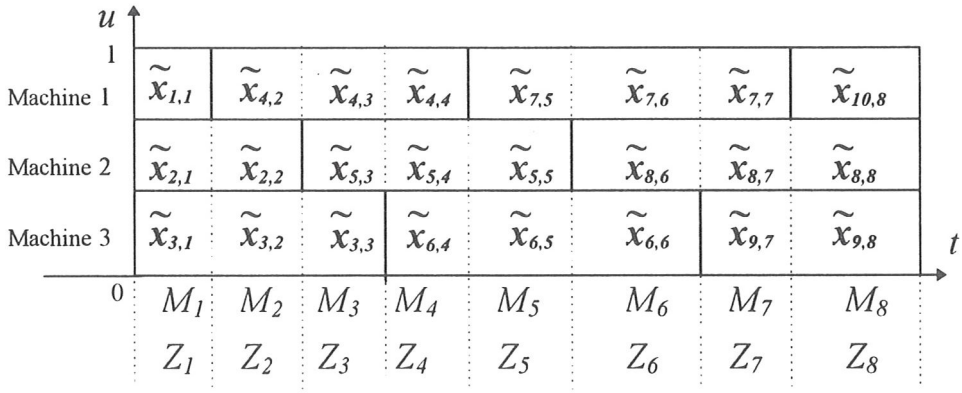

Fig 2. A division of processing demands of jobs in discrete-continuous scheduling

Notice that for identical concave functions f_i it is sufficient to consider feasible schedules in which the resource allocation among jobs remains constant in each interval M_k, $k = 1,2. \ldots , p.$ [3], The optimization problem obtained is always a convex one, because a sum of convex functions is also convex.

Of course, knowing a division of $\tilde{x}_i$'s one can easily calculate corresponding resource allocations.

The following mathematical programming problem can be defined to find an optimal demand division for a given feasible sequence S .

Problem P

Minimize $$C = \overline{M}^*\left(\left\{\tilde{x}_{-k}\right\}_{k=1}^{p}\right) = \sum_{k=1}^{p} \overline{M}_k^*\left(\tilde{x}_{-k}\right)$$

subject to

$$\sum_{k \in K_i} \tilde{x}_{ik} = \tilde{x}_i \qquad i = 1,\ldots,n$$

$$\tilde{x}_{ik} \geq 0 \qquad i = 1,\ldots,n; \quad k \in K_i$$

where $\overline{M}_k^*\left(\tilde{x}_{-k}\right)$ for concave functions f_i is the unique positive root of the equation $\sum_{i \in Z_k} f_i^{-1}\left(\frac{\tilde{x}_{ik}}{\overline{M}_k^*}\right) = 1$;

K_i is the set of all indices Z_k's such that $i \in Z_k$;

$\tilde{x}_{ik}$ is a part of job i processed in combination Z_k ;

Thus our problem is to minimize a convex function subject to linear constraints.

Let us define a potentially optimal set ***POS*** for a given instance *1*, as a set of feasible sequences S, which contains at least one sequence corresponding to an optimal schedule. ***A POS*** which consists of all feasible sequences of *n-m+1 m*-element combinations will be called a general ***POS.*** If the number of sequences in a ***POS*** is not very large one can solve problem P for each ***S*** from the ***POS*** and chose a schedule with the minimum length as an optimal solution. However, in the general case, the general ***POS*** has to be considered, the cardinality of which grows exponentially with the number of jobs *n*. Thus, in general, we are left in a situation, in which the problem cannot be solved to optimality, because the search for an optimum requires prohibitive amount of computation time. In these cases one may use e.g. local search metaheuristics.

In this paper we propose a Simulated Annealing algorithm for a special case of *n*-jobs *m*-machines discrete-continuous scheduling problem with an objective to minimize the makespan. in which functions f_i are of the form:

$$f_i = c_i u_i^{\frac{1}{\alpha}} \qquad \alpha > 1 \ ; \quad i = 1,2,\ldots,n$$

The proposed algorithm is compared with two other approximation algorithms: Multi-start Iterative Improvement algorithm and a Random Sampling Technique.

3. Simulated Annealing algorithm

A homogeneous Simulated Annealing algorithm [1] can be written as follows.

Step 1. Let $k := 0$; Generate an initial solution *a ;*
Calculate initial value of the control parameter T_0 ;
Set initial length of Markov chain L_0 ;

Step 2. Let $i := 1$;

Step 3. Randomly generate a solution *b* from the neighborhood *N(a)* of the current solution a; Let $i := i + 1$;

Step 4. Replace *a* by *b* with the probability

$$P(a \to b) = \min\left\{1, \exp\left(\frac{C(a) - C(b)}{T_k}\right)\right\}$$

where *C* is the cost function.

Step 5. If $i \leq L_k$, then go to step 2 ;

Step 6. Let $k := k + 1$; Update L_k;Set new value of T_k ;

Step 7. If stop criterion is fulfilled then stop else go to step 2

In order to apply the above described algorithm, a number of decisions have to be made. One must choose:

1. representation of a solution;
2. evaluation function *C(a):*
3. mechanism of neighbourhood generation;
4. initial solution a_0;
5. initial value of control parameter T_0 ;
6. number of iterations (the Markov chains length) ;
7. $T_{k+1} = f(T_k)$ - the temperature function;
8. stop criterion;

These decisions can be divided into two groups. The first one contains the problem specific decisions (1-4), which are closely related to the considered problem, and the second one, the generic decisions (5-8) which are usually called the cooling schedule.

3.1. The problem specific decisions

1. Representation of a solution.

   A solution is represented by a feasible sequence consisting of (n - m + 1) element sequence consisting of m-element sets of job indices, which occur in corresponding Z_k 's . The solution space is a general ***POS*.**

2. Evaluation function.

   For the considered class of functions ($f_i = c_i u_i^{\frac{1}{\alpha}}$) the evaluation of the cost function is computationally easy and consist in solving the following system of equations:

$$\begin{cases} u_1 + u_2 + \ldots + u_m = 1 \\ u_1^{\frac{1}{\alpha}} \cdot C_{\max} = \sum\limits_{i \in M_1} \dfrac{\widetilde{x}_i}{c_i} \\ \vdots \\ u_m^{\frac{1}{\alpha}} \cdot C_{\max} = \sum\limits_{i \in M_m} \dfrac{\widetilde{x}_i}{c_i} \end{cases}$$

Thus our (minimized) cost function, is the following one:

$$C_{\max} = \sqrt[\alpha]{\left(\sum_{i \in M_{i_1}} \frac{\tilde{x}_i}{c_i}\right)^{\alpha} + \ldots + \left(\sum_{i \in M_m} \frac{\tilde{x}_i}{c_i}\right)^{\alpha}}$$

3. Mechanism of neighbourhood generation.

New solutions are generated in the following way. One element is randomly choosen from the current configuration. The chosen element must fulfill the following feasibility conditions:

(i) it must represent a job. which occurs in at least two Z_k's;

(ii) if the number of combinations Z_k's in which the chosen job occurs is greater than two. this Z_k must be the first one or the last one from the combinations in which this job occurs.

If the randomly chosen job doesn't fullfil (i) or (ii) we must chose another job. The job chosen from combination Z_v can be replaced only by a job which:

- doesn't occur in Z_v and does occur in Z_{v-1} (if Z_v is the last combination in the feasible sequence in which the chosen job occurs);
- doesn't occur in Z_v and does occur in Z_{v+1} (if Z_v is the first combination in the feasible sequence in which the chosen job occurs);

  For example, for the n-jobs m-machines problem, where n - 10 and *m*. = 4, and current solution is represented by the sequence:

S = {1,**2**,**3**,**4**}, {**2**,3,4,**5**}, {**3**,4,5,**6**}, {**4**,5,6,**7**}, {**5**,6,7,**8**}, {**6**,7,8,**9**}, {**7**,**8**,**9**,10}

Elements which fulfill the conditions (i) and (ii) are displayed as a bold text. Let us assume that v = 4, then we can chose either job number 4 or job number 7. Job number 4 can be replaced only by job number 3 and then job 7 can be replaced only by job number 8.

4. Initial solution

The initial solution has the following form:

S = {1,2, ... , m},{2,3, ... , m + 1},...,{n - m + n - m + *2*, ... , n -1, n)

We choosethis kind of solution, because the neighborhood which can be obtained from this configuration is greater then neighborhoods which can be obtained from other kinds of configurations.

3.2. Simple Cooling Schedule

1. The initial value of the control parameter T_0 is obtained from the equation:

$$T_0 = \frac{\overline{\Delta C}^{(+)}}{\ln\left(\chi_0^{-1}\right)}$$

where χ_0 is an initial acceptance ratio, which is defined as the number of accepted transitions divided by the number of proposed transitions and $\overline{\Delta C}^{(+)}$ is the average difference in cost. We chose $\chi_0 = 0.95$ and the number of proposed transition equal to 50.

2. A decrement function decreasing the value of the control parameter is given by:

$$T_{k+1} = \alpha \cdot T_k$$

where $k = 0,1,2,\ldots,$ and $\alpha = 0.95$.

3. Stop criterion is the computation time, which depends on the size of the problem.

4. The length of the Markov chains L_k is determined so that the number of the rejected transition is not less than the number of jobs n .

4. Multi-start Iterative Improvement algorithm and Random Sampling Technique

A Multi-start Iterative Improvement algorithm can be written as follows.

Step 1. Randomly choose an initial solution *a ;*

Step 2. Generate a solution *b* from the neighborhood *N(a)* of the current solution *a:*

Step 3. Calculate the value of the cost function *C.* If *C(b) < C(a)* then replace *a* with *b* and go to step 4. else go to step 5.

Step 4. If stop criterion is fulfilled then stop else go to step 2.

Step 5. If there is no solution that improves *a* go to step 1. else go to step 4.

In this algorithm, the current solution *a* is replaced by the first solution *b* that improves *a.* If there is not such solution in *N(a),* i.e., if a is a local optimum, the iterative improvement procedure restarts at another initial solution. This procedure is repeated until the stopping condition in step 4 is satisfied. The stop criterion and the neighborhood generation mechanism are identical as in the Simulated Annealing algorithm and therefore we can compare results obtained by using both algorithms.

The Random Sampling Technique is an algorithm in which we randomly generate the next solution from the current one in order to find a configuration with the smallest value of the cost function. In this algorithm, the next solution is generated according to the mechanism described in the simulated annealing algorithm. The stop criterion is identical with this of Simulated Annealing algorithm.

5. Computational experiments

5.1. Test problems

As the test problems, we randomly generated 1000 instances for 13 problems of different sizes (for $\alpha = 2:10$ jobs x 2 machines, 10 x 3, 20 x 2, 20 x 3, 50 x 2, 50 x 3,100 x 10 and for $\alpha = 3$: 10 x 2, 10 x 3, 20 x 2, 20 x 3, 50 x 2, 50 x 3). A processing demand of each job is given as a random integer from the interval [1,1000]. All algorithms have been written in C++ language, and the experiment has been carried out on SGI PowerChallenge computer.

The numerical results are summarized in Table 1 ÷ Table 4. For problems with the number of jobs $n = 10$ the results obtained by all three algorithms has been compared with optimal solutions. For other problems the results are compared among the applied algorithms.

Table 1 shows the number of instances in which an optimal solution has been obtained by proposed algorithms. In Table 2 and Table 4 we summarize the average relative deviation from the reference makespan over 1000 randomly generated instances for each problem. The relative deviation of each solution *a* was calculated from the equation:

$$\text{relative deviation} = \frac{C(a) - \text{reference makespan}}{\text{reference makespan}} \times 100\ \%$$

where C(*a*) is the makespan obtained by an algorithm. The reference makespan is chosen as an optimal one for problems with $n = 10$ (Table 2) and the best from among those obtained by Simulated Annealing, Random Sampling Technique and Multi-start Iterative Improvement algorithm, for $n > 10$ (Table 4).

Table 3 shows the number of instances in which solutions obtained by a given algorithm is not worse than results obtained by two others.

Table 1. The number of instances whose optimal solutions are obtained by the proposed algorithms.

Problem size		Value of α	Stop criterion [s]	Simulated Annealing	Multi-start Iterative Improvement	Random Sampling Technique
jobs (n)	machines (m)					
10	2	2	0.1	839	731	837
10	2	3	0.1	871	706	813
10	3	2	0.2	269	270	177
10	3	3	0.2	266	258	157

Table 2. Average relative deviation from optimum obtained by the proposed algorithms.

Problem size		Value of α	Stop criterion [s]	Simulated Annealing	Multi-start Iterative Improvement	Random Sampling Technique
jobs (n)	machines (m)					
10	2	2	0.1	0.00006	0.00039	0.00026
10	2	3	0.1	0.00047	0.00080	0.00079
10	3	2	0.2	0.00617	0.00777	0.00916
10	3	3	0.2	0.01362	0.01623	0.03680

Table 3. The number of instances whose best results are obtained by the proposed algorithms.

Problem size		Value of α	Stop criterion [s]	Simulated Annealing	Multi-start Iterative Improvement	Random Sampling Technique
jobs (n)	machines (m)					
20	2	2	0.30	893	612	800
20	2	3	0.30	821	515	696
20	3	2	0.75	334	478	201
20	3	3	0.75	342	458	205
50	2	2	2.00	979	706	858
50	2	3	2.00	990	768	868
50	3	2	6.00	449	438	206
50	3	3	6.00	356	514	168
100	10	2	200.00	793	207	0

Table 4. Average relative deviation from the best known solution obtained by the proposed algorithms.

Problem size		Value of α	Stop criterion [s]	Simulated Annealing	Multi-start Iterative Improvement	Random Sampling Technique
jobs (n)	machines (m)					
20	2	2	0.30	0.00000	0.00002	0.00000
20	2	3	0.30	0.00000	0.00004	0.00001
20	3	2	0.75	0.00076	0.00073	0.00143
20	3	3	0.75	0.00150	0.00140	0.00289
50	2	2	2.00	0.00000	0.00001	0.00000
50	2	3	2.00	0.00000	0.00001	0.00000
50	3	2	6.00	0.00005	0.00726	0.00015
50	3	3	6.00	0.00012	0.00545	0.00032
100	10	2	200.0	0.00642	0.04223	0.23676

We can draw the following conclusions from the results of the computational experiment:

1. In almost all cases the Simulated Annealing algorithm is better than two other heuristics.
2. The approach based on the Random Sampling Technique for problems with two machines is comparable to the Simulated Annealing. But for other problems sizes this algorithm is much worse than two other heuristics.
3. The proposed Simulated Annealing is the best one for large problems sizes, where the Random Sampling Technique disappoints completely.
4. In cases where the number of solutions obtained by the proposed Simulated Annealing algorithm is less than the one obtained by Multi-start Iterative Improvement the average relative deviation shows that results obtained by Simulated Annealing are very close to those obtained by Iterative Improvement.
5. The proposed stop criterion is not very good, because the number of generated solutions is slightly different for each technique. The Simulated Annealing algorithm generates less configurations than two other techniques.
6. The proposed neighborhood generation mechanism is good for small numbers of machines, but it is worse for problems with greater values of m, because the size of the neighborhood, which consist of only feasible solutions is different for different current solutions. For example, for the n-jobs m-machines problem, where n = 10 and m = 5, in the best case, when current solution is represented by sequence

$$S = \{1,\mathbf{2},\mathbf{3},\mathbf{4},\mathbf{5}\}, \{\mathbf{2},3,4,5,\mathbf{6}\}, \{\mathbf{3},4,5,6,\mathbf{7}\}, \{\mathbf{4},5,6,7,\mathbf{8}\}, \{\mathbf{5},6,7,8,\mathbf{9}\}, \{\mathbf{6},\mathbf{7},\mathbf{8},\mathbf{9},10\}$$

we can replace all elements which fulfill the feasibility conditions and are displayed as bold text. The number of these elements is equal to $2(n - 2)$ and the probability that we randomly choose such an element is represented by expression

$$\frac{2 \cdot (n-2)}{m \cdot (n-m+1)}$$

In the worst case, when the current solution is represented by the sequence

$$5 = \{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},5\},\{1,2,3,4,6\},\{1,2,3,4,7\},\{1,2,3,4,8\},\{1,2,3,4,9\},\{\mathbf{1},\mathbf{2},\mathbf{3},\mathbf{4},10\}$$

The number of elements which fulfills the feasibility conditions is equal to $2(m. - 1)$ and the corresponding probability is equal to

$$\frac{2 \cdot (m-1)}{m \cdot (n-m+1)}$$

It influences the algorithms, where the neighbour solution is generated in random way like in Simulated Annealing or Random Sampling Technique and have smaller effect in algorithms where the neighborhood is generated in deterministic way like in Iterative Improvement algorithm.

6. Conclusions

In this paper, we proposed a Simulated Annealing algorithm for some class of discrete-continuous scheduling problems with the objective to minimize the makespan. By computer tests on various problem sizes it was shown that the proposed algorithm is better than two other heuristics (Multi-start Iterative Improvement and Random Sampling Technique) and that for small numbers of machines the Random Sampling Technique gives better solutions than the Multi-start Iterative Improvement algorithm, but for greater values of m it is inferior to the Iterative Improvement. However, these two algorithms are still not better than the proposed Simulated Annealing algorithm. In the future it is planned to apply this algorithm to a wide class of functions f_i . Also, we will improve the neighbourhood generation mechanism. Moreover, we plan to parallelize this algorithm to make the searching procedure faster and more efficient.

Acknowledgments

This research has been supported by the KBN Grant 8T11F 010 08p02. The numerical experiments were carried out on a Silicon Graphics PowerChallenge computer with eight RISC TFP 75 MHz processors in the Poznań Supercomputing and Networking Center.

References

[1] Aarts. E.H.L., and Korst. J., *Simulated Annealing and Boltzman Machines*, Wiley. Chichester. 1989.

[2] Błażewicz. J., Ecker. K.H.. Schmidt. G., Węglarz. J., *Scheduling in Computer and Manufacturing Systems.* 2nd edition. Springer Verlag, Berlin (1994).

[3] Józefowska. J., and Węglarz. J., *On a methodology for discrete-continuous scheduling* Research Report of the Institute of Computing Science. Poznań University of Technology . RA-004/95 (1995).

[4] van Laarhoven. P.J.M., and Aarts, E.H.L., *Simulated Annealing: Theory and Applications.* Kluwer Academic Publishers, Dordrecht. 1987.

[5] Węglarz. J., *Multiprocessor scheduling with memory allocation - a deterministic approach* . IEEE Trans. Computers C-29/8, 703-710 (1980).