CMST 3(1) 25-37 (1997) DOI:10.12921/cmst.1997.03.01.25-37

COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY 3,25-37 (1997)

DISCRETE-CONTINUOUS SCHEDULING TO MINIMIZE THE MEAN FLOW TIME — COMPUTATIONAL EXPERIMENTS

Joanna Józefowska*), Marek Mika*), Rafał Różycki *), Grzegorz Waligóra*), Jan Węglarz*)**)**

*)*Institute of Computing Science, Poznań University of Technology, Piotrowo 3a, 60-965 Poznań, Poland, e-mail: office_ics@sol.put.poznan.pl*

****)*Poznań Supercomputing and Networking Centre, Wieniawskiego 17/19, Poznań, Poland*

Abstract

Problems of scheduling nonpreemptable jobs which require simultaneously a machine from a set of parallel, identical machines and a continuous, renewable resource to minimize the mean flow time are considered. For each job there are known: its processing speed as a continuous, concave function of a continuous resource allotted at a time and its processing demand. The problem can be decomposed into two interrelated subproblems: (i) to sequence jobs on machines, and (ii) to find an optimal (continuous) resource allocation among jobs already sequenced. For some special cases the problem can be solved in polynomial time. For the general case we propose to use heuristic search methods defined on the space of feasible sequences. Three metaheuristics: Tabu Search, Simulated Annealing and Genetic Algorithm have been implemented and compared computationally. The computational experiment has been carried out on a SGI PowerChallenge XL computer with 12 RISC R8000 processors. Some directions for further research have been pointed out.

1. Problem formulation

The discrete-continuous scheduling problem is defined as follows (Józefowska. Węglarz, 1997). Consider n independent, nonpreemptable jobs which simultaneously require for their processing discrete (i.e. discretely-divisible) and continuous (i.e. continuously-divisible) resources. The discrete resource is a set of m parallel, identical machines. The total amount of the renewable, continuous resource available at a time is limited. Jobs are all available at the start of the process. Processing rate of job i, $i=1,2,\ldots,n$, depends on the amount of the continuous resource allotted to this job at time t and is described by the equation:

$$\dot{x}_i(t) = \frac{dx_i(t)}{dt} = f_i[u_i(t)],\ x_i(0) = 0,\ x_i(C_i) = \tilde{x}_i \qquad (1)$$

where

$x_i(t)$ is the state of job i at time t,

$u_i(t)$ is the amount of the continuous resource allotted to job i at time t,

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

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

$\tilde{x}_i$ is the final state (or the processing demand) of job i.

Assume that $0 \le u_i(t) \le 1$, $\sum_{i=1}^{n} u_i(t) \le 1$ for every t. The problem is to find a sequence of jobs on machines and, simultaneously, a continuous resource allocation which minimize the mean flow time $\overline{F} = \frac{1}{n}\sum_{i=1}^{n} C_i$.

In many practical situations additional resources can be allotted to jobs in amounts (unknown in advance) from given intervals. These are, for example, the situations when jobs are assigned to parallel processors driven by a common (electric, hydraulic, pneumatic) power source, e.g. commonly supplied grinding or mixing machines, electrolytic tanks or refuelling terminals. As another example one can consider manpower and money which is a common continuous resource. Also in computer systems multiple processors may share a common primary memory. If it is a paged-virtual memory system and the number of pages goes into hundreds, it is purposeful to treat primary memory as a continuous resource (see Węglarz, 1980).

Notice that the defined problem can be decomposed into two interrelated subproblems: (i) to sequence jobs on machines, and (ii) to allocate the continuous resource among jobs already sequenced.

Let us first analyse the representation of a feasible sequence of jobs. Observe that a feasible schedule (i.e. a feasible solution of a discrete-continuous problem) can be divided into $p \le n$ intervals of length M_k, $k = 1,2,\ldots,p$, defined by completion times of consecutive jobs. Let Z_k denote the combination of jobs processed in parallel in the k-th interval. Thus, in general, a feasible sequence S of combinations Z_k, $k=1,2,\ldots,p$, can be associated with each feasible schedule. Feasibility of such a sequence requires, in addition to the number of elements in each combination restricted by m, that each job appears in at least one combination and that nonpreemptability of each job is guaranteed. The last condition means that each job appears exactly in one or in consecutive combinations in S.

For a given feasible sequence of jobs on machines one can find an optimal division of processing demands of jobs, $\widetilde{x}_i$, i=l,2,...,n, among combinations in S, i.e. a division which leads to a schedule with the minimum mean flow time from among all feasible schedules generated by S. To that end the following convex mathematical programming problem has to be solved in the general case (see Józefowska, Węglarz, 1996):

Minimize:
$$\overline{F} = \frac{1}{n}\sum_{k=1}^{n}(n-k+1)M_k^*\left(\{\widetilde{x}_{ik}\}_{i\in Z_k}\right) \tag{2}$$

subject to:
$$\sum_{k\in K_i}\widetilde{x}_{ik} = \widetilde{x}_i, \qquad i = 1, 2, \ldots, n \tag{3}$$

$$\widetilde{x}_{ik} \geq 0, \qquad i = 1, 2, \ldots, n; \quad k \in K_i \tag{4}$$

where $M_k^*\left(\{\widetilde{x}_{ik}\}_{i\in Z_k}\right)$, k=1, 2, ..., n, is the unique positive root of the following equation:

$$\sum_{i\in Z_k} f_i^{-1}\left(\frac{\widetilde{x}_{ik}}{M_k^*}\right) = 1. \tag{5}$$

In consequence, an optimal schedule can be found by solving the continuous resource allocation problem optimally for all feasible sequences from a set (so-called Potentially Optimal Set - POS) containing at least one feasible sequence corresponding to an optimal schedule. Unfortunately, in general, the cardinality of a POS grows exponentially with the number of jobs. Therefore it is justified to apply local search metaheuristics such as Simulated Annealing (SA), Tabu Search (TS) or Genetic Algorithms (GA) operating on a POS. It has been proved (Józefowska, Węglarz, 1996) that for concave functions f_i, i = 1, 2, ..., n, a set containing all feasible sequences composed of n combinations of jobs, such that first n - m + 1 combinations contain m elements and the consecutive ones m - 1, m - 2, ..., 1 element respectively, is a POS. It will also constitute the search space of the developed algorithms.

In the following sections applications of the three above metaheuristics for discrete-continuous scheduling problems with concave functions f_i, i = 1, 2, ..., n, for the mean flow time criterion are presented. In all the applications the POS defined above is the search space for the relevant metaheuristic.

2. Tabu Search

2.1. Representation of a feasible solution

A feasible solution for TS is a feasible sequence described in Section 1 (n combinations of jobs). It is worth noticing that in such a feasible solution every combination Z_k, $k = 2,..., n - m + 1$, differs from the previous one by exactly one job. Otherwise repetitions of combinations would appear in the solution. Such solutions with repeated combinations are redundant and are not taken into consideration. Of course, every combination Z_k, $k = n - m + 2,..., n$, differs from the previous one in such a way that one job is simply eliminated.

2.2. Starting solution

A starting solution is generated in two steps. In the first step successive jobs in particular combinations are generated randomly but a job is accepted only if it does not violate feasibility of a sequence. Since successive combinations differ by one job it can be done as follows:

- the first combination is generated randomly as a m-element combination without repetitions from the n-element set of jobs,
- every next combination Z_k, $k = 2,..., n - m + 1$, is created from the previous one by generating randomly a position in the combination (from 1 to m) and inserting into that position a job randomly generated which has not appeared in the sequence so far,
- every next combination Z_k, $k = n - m + 2,..., n$, is created from the previous one by generating randomly a position in the combination and removing a job in that position from the combination.

In the second step a feasible solution generated randomly is transformed according to the vector of processing demands in such a way that the job which is completed in the last combination is replaced by the job with the largest processing demand and so on. Intuitively, it should lead to better schedules.

2.3. Objective function

The value of the objective function of a feasible solution is defined as the minimal mean flow time for the corresponding feasible sequence.

2.4. Neighbourhood generation mechanism

Consider a feasible solution consisting of successive positions. The total number of positions is $m(n - m + 1) + m(m - 1)/2$ ($m(n - m + 1)$ positions in the first $n - m + 1$ combinations plus $m(m - 1)/2$ positions in the last $(m - 1)$ combinations). Each job

occurs in at least one position. A neighbour of a current solution is obtained by replacing a job in a chosen position by another job. A job may be replaced only if it occurs more than once in the feasible solution (every job has to be executed) and only in the first or last combination in that it occurs (nonpreemptability). It is easy to notice that in order to avoid repetitions of combinations in a sequence a job in its first combination has to be replaced by a job from the next combination and a job in its last combination by a job from the previous one. Moreover, in combinations $n - m + 2,\ldots, n$, exactly one job is completed.

Assume that job i occurs in combination Z_k and also in either Z_{k-1} or Z_{k+1}. Then:

- if Z_k is the last (but not the only one) combination which i belongs to, then i in Z_k is replaced by the only job from the combination Z_{k-1} which does not belong to Z_k;
- if Z_k is the first (but not the only one) combination which i belongs to and $k<n - m + 1$, then i in Z_k is replaced by the only job from Z_{k+1} which does not belong to Z_k.

A neighbourhood consists of feasible solutions obtained by performing all such replacements.

Example. Let n=5, m=3 and S={ 1,2,3},{2,3,4},{3,4,5},{4,5},{5}. The neighbouring solutions are:

{1,4,3},{2,3,4},{3,4,5},{4,5},{5}
{1,2,4},{2,3,4},{3,4,5},{4,5},{5}
{1,2,3},{1,3,4},{3,4,5},{4,5},{5}
{1,2,3},{2,3,5},{3,4,5},{4,5},{5}
{1,2,3},{2,3,4},{2,4,5},{4,5},{5}
{1,2,3},{2,3,4},{3,4,5},{3,5},{5}
{1,2,3},{2,3,4},{3,4,5},{4,5},{4}.

Observe that exactly one job is completed in the first combination of the feasible sequence. Thus, this job occurs in exactly one combination and can not be replaced. Moreover, exactly one job started in combination $n - m + 1$ can be replaced only in its last combination. In consequence, at most $n - 2$ jobs may occur in multiple combinations. Each of them may be replaced in exactly two positions (in its first and its last occurrence). As a result, the maximum possible number of neighbours for a given feasible solution is independent from the number of machines and is equal to $2(n-2) + 1 = 2n - 3$.

2.5. Tabu list management

The tabu list is managed according to the Reverse Elimination Method - REM (Glover. 1990). Considering the neighbourhood generation mechanism described earlier a move leading from a feasible solution to a neighbouring one may be defined as a 3-tuple: (index of a combination, job replaced, job introduced). A single element of the tabu list

(i.e. so-called Residual Cancellation Sequence - RCS) is the minimal set of moves leading back from the current solution to a solution already visited. If the starting solution is denoted by #1 then RCS #*j* (the *j*-th element of the tabu list) is the minimal set of moves by which the current solution differs from solution #*j* (i.e. visited in the *j*-th iteration).

REM takes advantage of the fact that a solution can be revisited in the next iteration only if it is a neighbour of the current solution (i .e. if the set of moves leading back to it from the current solution consists of one element only). In consequence, every move occurring on the tabu list as a one-element RCS is forbidden in the next iteration. Thus, in order to define the status of a move, it is sufficient to check only those elements of the tabu list which contain exactly one move.

The way of updating the tabu list using REM has been described in detail in (Józefowska, Waligóra, and Węglarz, 1996)

The tabu list length has been set at 10. The decision has been made on a basis of computational experiments which have shown that for the considered problems this value both prevents cycles and leads to good solutions.

3. Simulated Annealing

3.1. Representation of a solution

A solution is a feasible sequence described in Section 1. It is represented by two *n*-element sequences. The first one is a permutation of jobs and defines the order in which they are started. The second one consists of machines on which the corresponding jobs from the first one will be executed. These two sequences allow to create the first $n - m + 1$ combinations of a feasible sequence. In order to define the last $m - 1$ combinations we take the jobs from combination $n - m + 1$ and assume that the job on the first machine is completed first, the job on the second machine is completed second and so on.

3.2. Initial solution

The initial sequence of jobs is generated by setting all jobs in an ascending order. The initial sequence of machines is created according to the following rule:

$$\text{machine} = ((\text{job} - 1) \bmod m) + 1.$$

3.3. Neighbourhood generation mechanism

The current solution is represented by two sequences. Therefore a neighbour may be generated in two ways. Either by a small perturbation in the sequence of jobs, or by a small change in the sequence of machines. We will firstly describe how to get the next

solution by a change in the sequence of machines. A neighbour of the current solution is obtained in the following way. One element from the sequence of machines is randomly chosen and replaced by another random integer from the interval [1, *m*]. However, the mean flow time does not depend on the assignment of jobs to machines only. It also depends on the order in which jobs are processed. So, we must combine changes in the sequence of machines with changes in the sequence of jobs. To that end we apply the shift neighbourhood mechanism which is executed by removing a job randomly chosen from one position and putting it into another position also randomly chosen.

3.4. Objective function

The objective function is defined as the minimum mean flow time for a feasible solution.

3.5. Initial value of the control parameter T_0

In our implementation of SA we have applied a simple cooling strategy (see Aarts and Van Laarhoven, 1987).
The initial value of the control parameter T_0 is calculated from the following equation:

$$T_0 = \frac{\overline{\Delta c}^{(+)}}{\ln\left(c_0^{-1}\right)}$$

where $\overline{\Delta c}^{(+)}$ is the average increase in cost for a number of random transitions, and χ_0 is the initial acceptance ratio defined as the number of accepted transitions divided by the number of proposed transitions. In our implementation we assume $\chi_0 = 0.9$, in other words, we demand that the ratio of accepted moves at the start of the algorithm will be not less than 90%. $\overline{\Delta c}^{(+)}$ is calculated for 20 randomly generated neighbours of the initial solution.

3.6. Decrement of the control parameter

The control parameter decreases according to the following function:

$$T_{k+1} = 0.95 \quad T_k, \quad k = 0,1,2,\ldots,.$$

3.8. Length of Markov chains L_k

The length of Markov chains determines how many moves are generated for a fixed value of the control parameter and when the present temperature is to be decreased. Our

choice for the length of Markov chains is based on an intuitive argument that for each value T_k of the control parameter a minimum number of transitions should be accepted. In other words, L_k is determined so that at least a fixed number η_{min} of transitions are accepted. However, since transitions are accepted with decreasing probability, one would obtain $L_k \to \infty$ for $T_k \to 0$. Consequently, to avoid extremely long Markov chains for small values of the control parameter, L_k is bounded by some constant $\overline{L}$. We assume $\eta_{min} = 20$ and $\overline{L} = 80$.

4. Genetic Algorithm

Genetic algorithms operate on a population of individuals, each one representing a feasible solution of the considered problem. In our case a single individual represents a feasible sequence ofjobs on machines.

In GA there is not an evident definition of a solution neighbourhood. The process of looking over new solutions is made by using recombination operators. Random nature of these operators, assuming accidental changes of randomly selected individuals from the current population, results in the difficulty in maintaining feasibility of individuals during the evolution process. Particularly, for the considered class of problems, a crossover operator seems to be hard to implement.

4.1. Representation

In our implementation of GA the representation of a feasible sequence is not so evident as in the case of TS. Here each individual consists oftwo chromosomes.

The first one is a permutation of n jobs defining the order in which the jobs are started. Since all jobs are available at the start of the process, the first m jobs start concurrently on machines 1, 2, ... , m.

The second chromosome consists of $n - m$ elements. It defines machines on which jobs $m + 1$, $m + 2$, ... , n are executed. In combinations 2,..., $n - m + 1$ a job executed on a released machine is replaced by the first job from the first chromosome which has not been started yet. In combinations $n-m + 2,..., n$ jobs are completed in the order of the numbers of machines on which they are executed.

4.2. Creating an initial population

An initial population is a starting point for the multidirectional evolution search process. The simplest method of creating an initial population - the random creating - is used in our implementation. The only assumption that has to be held during a random creating of the individuals is maintaining a proper form of the first chromosome. In fact, the first chromosome is a permutation of n unique numbers. In the second chromosome, every integer number from the range [1,...,m] is feasible in every position.

4.3. Fitness of an individual

The fitness of an individual depends on the minimum mean flow time for the corresponding feasible sequence. The shorter mean flow time, the higher fitness of the individual.

4.4. Genetic operators

The main genetic operator is a selection operator. At the stage of the selection, a temporal population of individuals is created. This population contains *pop_size* (where *pop_size* is the size of the population) relatively good, in a sense of the fitness, individuals which are potential „parents" for recombination operators. As our selection operator, the linear ranking with elitism was selected. This strategy does not need any scaling method, which is necessary in a case of roulette wheel for minimization problems, and is easy to implement. The individuals were ranked in a nonincreasing order of their corresponding values of the mean flow time. Probability p_i of the selection of individual i $(i=1,2,...,pop\_size)$ to the temporal population depends on its position a_i in the ranking and is equal to:

$$p_i = \frac{(pop\_size - a_i + 1)}{pop\_size * (pop\_size + 1)/2}.$$

In order to assure a better convergence of our GA, the elitism property was added to the selection operator. The best individual from the previous population is selected to the new one independently of the ranking strategy.

The recombination operators replace randomly chosen individuals (so-called parent individuals) by offspring individuals. The number of individuals which undergo recombination depends on the setting of the recombination parameters. After the recombination process the temporal population containing some new offspring individuals and some individuals which do not undergo recombination becomes a new population of individuals. In this way, a single iteration of GA (so-called generation) is finished. Then a new evolution cycle is started. After a number of such cycles the algorithm converges - the best individual hopefully represents a solution near to the optimal one.

There are four recombination operators especially suited for the assumed representation of a feasible sequence of jobs on machines. Two of them operate on a single parent individual (mutation type) and the remaining two work on pairs of parent individuals (crossover type). The first mutation type operator we call *mutation I.* By a mutation in this case we understand an elementary change made in the second chromosome of an individual. A machine in a randomly chosen position of the second chromosome is replaced by another one.

The second mutation type operator is *mutation II.* This operator exchanges a random number of randomly selected jobs. In other words, a number of jobs which will be exchanged (from the range [2,...,*n*]), and then these jobs, have to be chosen randomly. Of course, this operator concerns only the first chromosome in the selected individual. As a result of mutation II we get a new order of jobs.

The first crossover operator, the *head crossover,* operates on the first chromosome only. This operator has to preserve the form of the first chromosome (a permutation of jobs) in both the offspring individuals. Because of a strong similarity of the first chromosome to the path representation of the solution in Travelling Salesman Problem one of known crossover operators (namely OX (Michalewicz, 1984)) is used as the head crossover. This operator uses two crossing points. First, the segments between crossing points are copied into offspring individuals. Next, starting from the second crossing point of one parent, the jobs from the other parent are copied in the same order, omitting symbols already present. Reaching the end of the string, copying is continued from the first place of the string.

The second crossover operator, the *tail crossover,* concerns the second chromosome only. The feasibility conditions of this chromosome allow to use its simplest version. The tail crossover cuts the second chromosomes of both parent individuals at the same randomly chosen cut point and exchanges the resulting segments.

Note that the above recombination operators have to incorporate the problem-specific knowledge. It allows to maintain the feasibility of the individuals during the evolution search process.

4.5. Other parameters of the genetic algorithm

The population size was set at 50. The stop criterion was adapted to the computational experiment requirements and is described in Section 5. Other parameters (mutation I rate, mutation II rate, tail crossover rate, head crossover rate) were set in preliminary experiments.

5. Computational experiments

A number of computational experiments have been carried out to compare the performance of the considered metaheuristics on a randomly generated set of instances. In this section we present the results of the computational experiments.

All the heuristics have been implemented in C++ and have run on SGI PowerChallenge XL with 12 RISC 8000 processors. As we have mentioned before, for each solution visited in the solution space, the corresponding schedule has been found by solving a convex mathematical programming problem. In this step specially adopted solver CFSQP (A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequalrty Constraints) 2.3 (Lawrence. Zhou, Tits, 1995) has been applied. The solver stopped when the absolute

difference in consecutive values of the objective function was less than or equal to 10^{-5}
In order to ensure a comparable computational effort devoted to each heuristic, the stop criterion has been defined as a number of solutions visited. This number has been set at 2000. It is clear that neither tabu search nor genetic algorithm can stop exactly after the required number of solutions visited, so they stop after visiting at least the required number of solutions. In fact, tabu search performed the smallest number of iterations in which the total number of visited solutions exceeded 2000 and the genetic algorithm generated the smallest number of generations in which the total number of individuals exceeded 2000.

The experiments have been carried out for processing rates of jobs of the form $f_i = u_i^{1/\alpha_i}$. Processing demands have been generated from the interval [1,100] with a uniform distribution. Values of α_i from the set {1,2} have been generated randomly with equal probability. For each problem size 100 instances have been generated.

The first experiment has been performed for three groups of instances with two machines and 10, 15 and 20 jobs.

The second experiment has been carried out for $n=10$ jobs and the number of machines m = 2, 3 and 4. For each problem size 100 instances have been generated.

The comparison among heuristics is presented in Tables 1 and 2 on the basis of three values: the number of instances for which the relevant heuristic found the best solution over all the three heuristics, the average relative deviation from the best solution and the maximum relative deviation from the best solution. For each problem size # is the number of instances for which the solution found by the considered algorithm was the best solution found.

Table 1. Results of the computational experiments for m = 2.

n		Simulated Annealing	Genetic Algorithm	Tabu Search
10	#	1	6	94
	average relative deviation	1,025361	1,015818	1,004919
	maximum relative deviation	1,059911	1,065863	1,006167
15	#	0	1	99
	average relative deviation	1,048011	1,041335	1.000004
	maximum relative deviation	1,142909	1,101520	1.000411
20	#	0	0	100
	average relative deviation	1,076401	1,071523	1
	maximum relative deviation	1,218169	1,143513	1

Comparing the performance of metaheuristics tested in the experiment, it is clear that TS performs best, finding the largest number of best solutions and showing smallest deviation from optimum for all the problem sizes.
Genetic Algorithm shows slightly better performance in terms of the average and maximum relative deviation from the best solution found.
The relative deviation from the best solution found increases with the number of jobs for Tabu Search. For the other algorithms the deviation practically does not depend on the number of jobs.
For $m > 2$ neither Simulated Annealing nor Genetic Algorithm found a better solution than Tabu Search for any instance tested. The realtive devition from the best solution increases with the number of machines for both the algorithms: Simulated Annealing and Genetic Algorithm.

Table 2. Results of the computational experiment for n=10.

m		Simulated Annealing	Genetic Algorithm	Tabu Search
2	#	1	6	94
	average relative deviation	1,025361	1,015818	1,004919
	maximum relative deviation	1,059911	1,065863	1,006167
3	#	1	4	95
	average relative deviation	1.035275	1.019903	1.022071
	maximum relative deviation	1.102449	1.073357	1.047822
4	#	3	43	77
	average relative deviation	1. 035666	1.019104	1.16719
	maximum relative deviation	1.117811	1.091474	1.076304

As we have mentioned earlier, the experiment required large computational effort. Thus, further development of the heuristics aims at improving their computational times. This goal may be achieved in two ways. Firstly, a heuristic evaluation of feasible sequences visited in the search space will be introduced. Secondly, parallelization of the search process will be implemented.

6. Final remarks

In this paper applications of three local search metaheuristics to some discrete-continuous scheduling problems are presented. All the described algorithms have been designed, adjusted and applied to the considered class of scheduling problems. The parameters of each algorithm have been selected on a basis of computational experiments in order to make the algorithm as effective as possible.
A computational experiment has been carried out and the results obtained by all the heuristics have been compared with each other. Some conclusions and remarks as well as suggestions for further research have been presented.

Acknowledgement
The computational experiments were performed at the Poznań Supercomputing and Networking Center. This work was supported by research grant BW 43-242.

References

Aarts E.H.L., and van Laarhoven P.J.M. (1987), *Simulated Annealing: Theory and Applications,* Reidel, Dordrecht.

Davis L. (1985), „Applying adaptive algorithms to epistatic domains", *Proc. Internat. Joint Conf. on Artificial Intelligence*, 162-164.

Glover F. (1989), „Tabu Search - part 1", *ORSA J. Computing*, 1, 190-206.

Glover F. (1990), „Tabu Search - part 2", *ORSA J. Computing*, 2, 4-32.

Józefowska J., Dyskretno-ciągłe problemy szeregowania zadań", rozprawa habilitacyjna. Wydawnictwo Politechniki Poznańskiej, Poznań 1997.

Józefowska J., Waligóra G., and Węglarz J. (1996), „A Tabu Search algorithm for some discrete-continuous scheduling problems", in: V.J. Rayward-Smith, (ed.). *Modern Heuristics Search Methods*, Wiley, 169-182.

Józefowska J., and Węglarz J. (1996), „Discrete-continuous scheduling problems - Mean completion time results", *European Journal of Operational Research* 94 (1996), pp.302-309.

Józefowska J., and Węglarz J. (1997), „On a methodology for discrete-continuous scheduling", *European Journal of Operational Research.*

Lawrence C., Zhou J.L., Tits A.L.: *Users guide for C.FSQP Version 2.3* (Released August 1995), available by e-mail: andre@eng.umd.edu.

Michalewicz Z. (1984), *Genetic Algorithms + Data Structures = Evolution Programs,* Springer Verlag.

Osman I.H., Potts C.N. (1989), „Simulated annealing for permutation flow shop scheduling problem", *Omega* 16, 7, 551-557.

Węglarz J. (1982), Modelling and control of dynamic resource allocation project scheduling systems, in: Tzafestas S.G. (ed.) *Optimization and Control of Dynamic Operational Research Models. North Holland*, 105-140.