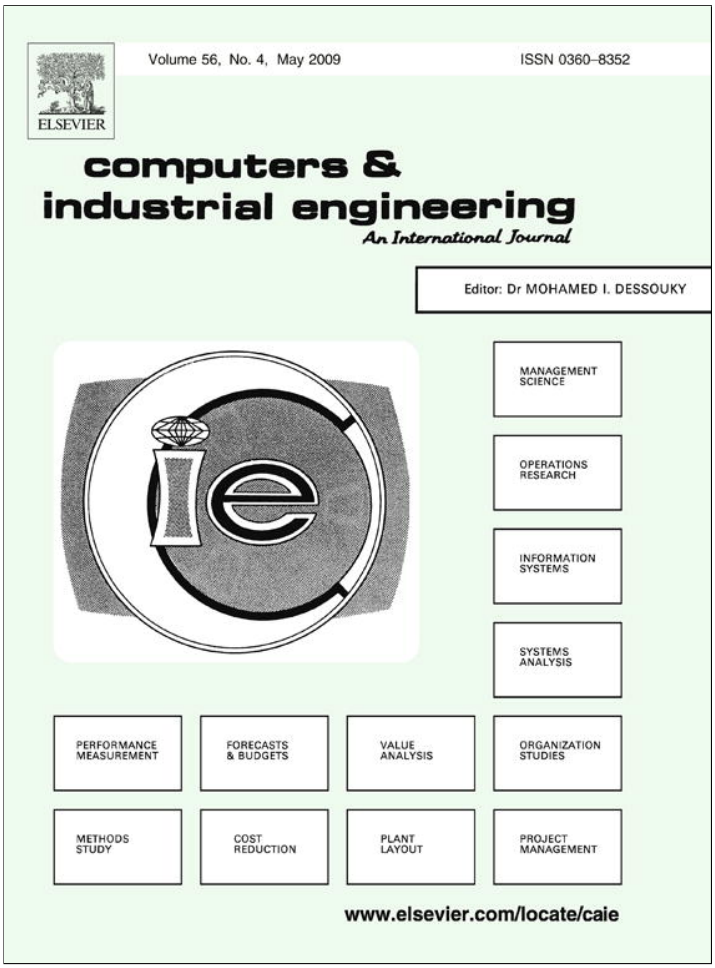

Computers & Industrial Engineering 56 (2009) 1502–1509

Contents lists available at ScienceDirect

Computers & Industrial Engineering

journal homepage: www.elsevier.com/locate/caie

A fast hybrid tabu search algorithm for the no-wait job shop problem ☆

Wojciech Bożejko *, Mariusz Makuchowski

Wrocław University of Technology, Institute of Computer Engineering, Control and Robotics, Janiszewskiego 11-17, 50-372 Wrocław, Poland

ARTICLE INFO

Article history:
Received 19 April 2008
Received in revised form 7 September 2008
Accepted 12 September 2008
Available online 20 September 2008

Keywords:
Job shop problem
No-wait constraint
Tabu search
Hybrid algorithm

ABSTRACT

This paper describes a hybrid tabu search algorithm dedicated to a job shop problem with a no-wait constraint with a makespan criterion. The proposed here algorithm complexity is that the superior algorithm based on the tabu search technique selects parameters controlling the work of a certain constructional algorithm. This approach limits the checked solutions only to a group of solutions being able to be generated by the structural algorithm in question. It bears serious consequences both positive, for example it limits the research scope for a small fraction of relatively extremely well quality of acceptable solutions, and negative that is the lack of possibility of finding the optimal solution. In this paper numerical researches of the proposed algorithm are conducted as well as a comparative analysis with reference to the literature algorithms of the algorithm in question is made.

1. Introduction

In this work there was analyzed a strongly NP-hard job shop problem with an additional no-wait constraint with an optimalization criteria constituting a finishing moment of all tasks performance (makespan). The presented here problem comes from a classical job shop problem by setting additional requirements concerning an acceptable solution. The difference between the analyzed problem and its classical job shop problem equivalent consists in the necessity to conduct the subsequent operations of a given task exactly in the moment of finishing their technological predecessors. This requirement is often encountered in real production processes where machined components over the course of time change their physical–chemical parameters. For example, during the steel elements' production materials should be heated to a proper temperature in order to come next to form ready components. The formation process has to start exactly in the moment of obtaining a hot half-finished product (Wismer, 1972). Similarly, during some foodstuff' production because of both sanitary conditions and undergoing chemical–biological processes it is essential to carry out one activity directly after another (Hall & Sriskandarajah, 1996).

With reference to the complexity computational theory the classical job shop problem is classified to a group of strongly NP-hard problems, which is proved in a paper (Lenstra & Rinnooy Kan, 1979). In the same paper it is presented that in a case of adding a no-wait constraint the problem belongs to the same class too, i.e. strongly NP-hard problems. Although the theoretical complexity of both variants of the analyzed problem is the same, according to many researchers, with the practical point of view, a problem with a no-wait constraint is much more difficult. It results from the fact that the instance size, which can be solved exactly for example by the branch and bound (B&B) method, is much smaller for a variant with no-wait constraints. For example in a paper (Mascis & Pacciarelli, 2002) there was presented an algorithm finding an optimal solution in a reasonable time for an instance of a no-wait problem, in which the size did not exceed 15 tasks executed on 5 machines or 10 tasks executed on 10 machines. Besides, different kinds of approximate algorithms, an algorithm based on a tabu search technique (Macchiaroli, Mole, & Riemma, 1999), algorithm based on a technique of simulated annealing (Raaymakers & Hoogeveen, 2000) and a genetic algorithm with elements of simulated annealing (Schuster & Framinan, 2003), provide with solutions of a rather big deviation calculated relative to the reference solutions.

In this work there is proposed a hybrid algorithm based on a tabu search technique dedicated to a job shop problem with no-wait constraints. The presented algorithm is regarded as a hybrid since the superior component – a tabu algorithm controls an entry parameter of a constructional algorithm generating an acceptable solution of a job shop problem with no-wait constraints. A proper algorithm's construction generating a solution assures that a free placement of controlling parameters obtained by a superior algorithm always generates an acceptable solution. Besides, the number of different possible placements of controlling parameters is much more smaller than the number of all possible active solutions of a main problem, which limits the research scope. Generated solutions are of a relatively high quality taking into consideration the goal function's value which results from a very calculated constructional procedure described

☆ This manuscript was processed by Area Editor Maged M. Dessouky.

* Corresponding author. Tel.: +48 603672142.
E-mail addresses: wojciech.bozejko@pwr.wroc.pl (W. Bożejko), mariusz.makuchowski@pwr.wroc.pl (M. Makuchowski).

0360-8352/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cie.2008.09.023

in a further part of this work. Unfortunately, generally this approach makes finding optimal solutions, even in a case of the best selected controlling parameters, impossible.

The presented here hybrid tabu search algorithm was examined numerically on a literature group of test examples. These examples are commonly used to examine new algorithms for the discussed here subject. The obtained results are compared to the best known results and confronted with *GASA* algorithm (Schuster & Framinan, 2003) and the latest works of Framinan and Schuster (Framinan & Schuster, 2006; Schuster, 2006) taking into consideration both the algorithms' work time and the quality of obtained solutions. The work is summarized by author's conclusions about the proposed here algorithm.

2. Problem definition

The job shop problem with a no-wait constraint can be modelled as a classical job shop model introducing an additional condition concerning the solution acceptability. The problem can be defined as follows: there is a set of tasks: $J = \{1,2,\ldots,r\}$, a set of operations: $O = \{1,2,\ldots,n\}$ and a set of machines: $M = \{1,2,\ldots,m\}$. The set of operations is divided into a r disjunctive subsets referring to individual tasks. A task $k \in J$ equates with a given sequence o_k of operations; $\sum_{k\in J} o_k = n$. This sequence establishes the required order of task's operation execution with a number k and for a notation simplification it is assumed that it looks like in the following way: $j_k + 1, j_k + 2, \ldots, j_k + o_k$, where $j_k = \sum_{i=1}^{k-1} o_i$ is a number of operations in tasks of $\{1,2,\ldots,k-1\}$ and $j_1 = 0$. Additionally, for a further notation's simplification the sequence of task's operation ki.e. H we mark as O_K and assume that a notation $h \in O_k$ means that h is a certain element occurring in a sequence O_k (classically the $\in$ operator is defined for a set of elements, not for a sequence).

Furthermore, for an each operation $h \in O$ there is determined one machine $\mu(h) \in M$, on which it is to be conducted within a period $p_h > 0$. Additionally, in this model there exist three, typical for this kind of problem, constraints:

- each machine can execute at that particular moment not more than one operation;
- simultaneously more than one operation of a given task cannot be carried out at that particular moment;
- executing an operation on a machine cannot be broken.

The accepted schedule is defined by moments of starting an execution $S(h) \geqslant 0$ of an operation $h \in O$, N such that all above constraints are satisfied. The problem consists in finding an acceptable schedule minimizing the moment of executing all operations $\max_{h\in O} C(h)$ where $C(h) = S(h) + p_h$.

Setting an extra requirement:

- no waiting – each operation (apart from the first one) of a particular task has to begin exactly at the moment of finishing the previous operation's conduction of the same task,

we obtain a model of a job shop problem with a no-wait constraint marked in the Graham's notation (Graham, Lawler, Lenstra, & Rinnooy Kan, 1979) by $J|no-wait|C_{\max}$.

Let t_k denotes the starting time of the first operation of the kth task ($t_k = S(j_k)$). Due to the no-wait constraint this also determines the starting and finishing times of all operations $h \in O_k$ of this task. For a case with no waiting the problem solution can be represented by a vector $T = (t_1, t_2, \ldots, t_r)$ of terms of all tasks' execution beginning.

3. Algorithm generating solutions

The general idea of a proposed here algorithm consists in finding a solution by a systematic tasks adding to partly made in earlier iterations schedules. The controlling parameter, which is at the same time a governing variable for a superior algorithm, is an order of tasks schedule. This schedule is further called a loading permutation and is marked by $\pi = (\pi(1), \pi(2), \ldots, \pi(r))$, $\pi(i) \in J$, $1 \leqslant i \leqslant r$. A set of all possible loading permutations is marked as Π. In ith iteration a task with a number $\pi(i)$ is submitted to a schedule. The process of $\pi(i)$ task adding to a partial schedule consists in determining all moments of $S(h)$ beginning and moments of $C(h)$ finishing a conduction of each operation of this task $h \in O_{\pi(i)}$. After an arrangement a task does not change its place in a schedule i.e. established in ith iteration moments $S(h)$ and $C(h)$, $h \in O_{\pi(i)}$ will be the moments of beginning and finishing execution of these operations in the final arrangement. In ith iteration the way of establishing the moments of beginning and finishing the operation of a task $\pi(i)$ is determined relative to the possibly smallest moment (satisfying all required constraints) $t_{\pi(i)} \geqslant 0$ of the $\pi(i)$ task execution. Notice that each π loading permutation (a permutation of the J set) defines exactly one schedule, what's more this schedule is always acceptable, which results directly from its way of construction.

Conception of an algorithm generating solution from the sequence of jobs was proposed by Macchiaroli et al. (1999), Schuster (2006) and Schuster and Framinan (2003). In the work of Schuster (2006) the best timetabling for the given sequence is looked for, where the sequence is determined by the earliest moment of beginning the job's operation. This sequence cannot be changed by the timetabling algorithm, and that is why it is regarded as NP-hard. We proposed a different approach here: to use *loading permutation* as a base for a constructive algorithm which inserts tasks one after another. An order of jobs – in understanding of Framinan and Schuster – can be changed inside a timetable.

Fig. 1 presents a skeleton of the algorithm generating solutions. A variable *taskStart* is a time of beginning of the first operation of the task. The implemented procedure possesses a computational complexity $O(r^2N^2)$ for a case of at most one operation of job is executed on each machine (as in benchmark instances), where $N = \max_{k\in J}\{o_k\}$ means the largest number of operations in arranged tasks. In this case we can check possible collisions of operations in time $O(N)$ during inserting of each task. For a general case (no limit of the number of operations of job on each machine), an upper bound of a computational complexity of the algorithm is $O(r^3N^3)$.

As it has been mentioned before even examining all possible controlling parameter's combinations cannot guarantee finding an optimal solution. Fig. 2 shows an example of the optimal solution which cannot be obtained by the proposed constructive algorithm. However, a numerical examination was made, which gives an idea of this problem's scope. It is interesting both how often it occurs in order that the best possibly solution which can be obtained, marked as $\hat{\alpha}$, it would not be the optimal solution, marked as α^*, and what is the average deviation of the goal function's value of solution $\hat{\alpha}$ relative to the goal function's value of the optimal solution α^*. To answer the question one should have a possibly wide range of instances with known goal function's values of optimal solutions α^* and know goal function's values of the best possible ones to generate a solution $\hat{\alpha}$. For this purpose one has used 27 literature examples known as La01-La05, La06-La10, La 16-La20, Orb01-Orb10, Ft06 and Ft10. For these examples there are known optimal solutions obtained by an algorithm based on branch and bound technique

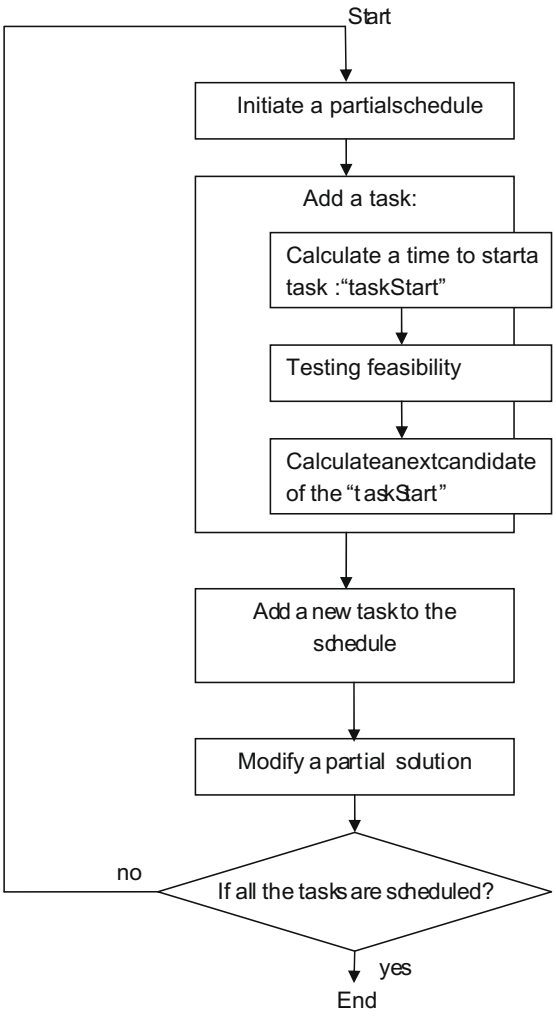

Fig. 1. Skeleton of the algorithm generating solutions.

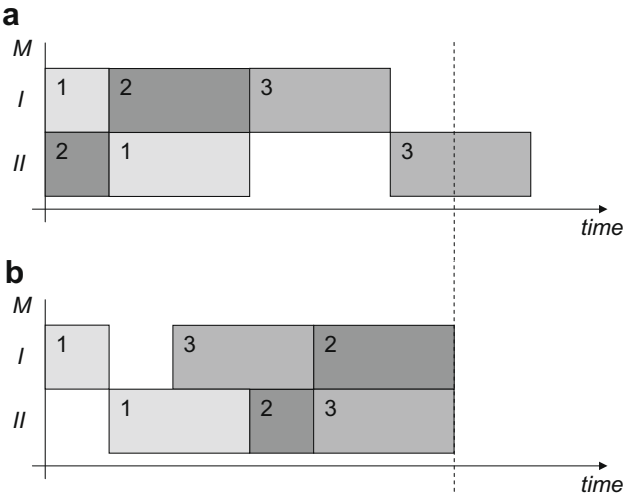

Fig. 2. Example of the optimal solution α^{**} (b), which cannot be obtained by the proposed constructive algorithm generating solutions which generates a solution α^{*} (a).

(Mascis & Pacciarelli, 2002). Simultaneously, the number of tasks in these examples does not exceed 15; $r \leqslant 15$, which allows to check all possible loading permutations which is $r!$. To achieve this aim an algorithm *HBB* was written, based on a branch and bound method, which generates a solution for the best possible loading permutation. Furthermore, for an each instance there can be generated a 'mirror' instance in which technological relations were substituted for reverse ones. Practically, it is the same instance, but an algorithm generating solutions arrange tasks in a different way than for an original instance. The obtained solution of the 'mirror' instance is certainly a reflection of a certain solution of original instance. It means that some instance 'problems' causing a lack of possibilities to generate an optimal solution cannot exist for a 'mirror' instance. Moreover, if an superior algorithm for some data has a difficulty to find a good solution it may appear that in a case of a mirror instance the difficulty does not exist. The difficulty for algorithms can be here properly understood as problems with lowering too deep or too wide the local minimum or problems with moving on a flat (in the sense of a goal function's value) solution space.

Let C^{Alg} mean a value of an optimal criterion solution obtained by *Alg* algorithm. The test examinations consisted in finding in works [4,7] C^{OPT} value of goal functions of optimal solutions, calculating C^{HBB} value of a goal function of a solution obtained by *HBB* algorithm and calculating $C^{HBB'}$ goal function of a solution obtained by a *HBB'* algorithm working on mirror instances. Next for an each instance a relative error of solutions obtained by the following algorithms was calculated:

$$\delta^{Alg} = \frac{C^{Alg} - C^{OPT}}{C^{OPT}} \cdot 100\%, \quad Alg \in \{HBB, HBB'\}. \tag{1}$$

The obtained results are included in a Table 1. Additionally, in this table there is an information concerning the instance size, the average deviation of individual instance groups; the last line contains the average values of analyzed parameters from all 32 tested examples.

Table 1
Results of *HBB* and *HBB'* algorithms

Inst.	$r \times m$	C^{OPT}	C^{HBB}	$C^{HBB'}$	δ^{HBB}	$\delta^{HBB'}$	$\min\{\delta^{HBB}, \delta^{HBB'}\}$
ft06	6×6	73	73	73	0.00	0.00	0.00
ft10	10×10	1607	1607	1609	0.00	0.12	0.00
la01		971	975	971	0.41	0.00	0.00
la02		937	961	937	2.56	0.00	0.00
la03	10×5	820	820	820	0.00	0.00	0.00
la04		887	887	887	0.00	0.00	0.00
la05		777	781	777	0.51	0.00	0.00
la16		1575	1575	1575	0.00	0.00	0.00
la17		1371	1384	1371	0.95	0.00	0.00
la18	10×10	1417	1417	1417	0.00	0.00	0.00
la19		1482	1482	1491	0.00	0.61	0.00
la20		1526	1526	1580	0.00	3.54	0.00
orb01		1615	1615	1626	0.00	0.68	0.00
orb02		1485	1485	1518	0.00	2.22	0.00
orb03	10×10	1599	1599	1599	0.00	0.00	0.00
orb04		1653	1653	1653	0.00	0.00	0.00
orb05		1365	1370	1367	0.37	0.15	0.15
orb06		1555	1555	1557	0.00	0.13	0.00
orb07		689	705	689	2.32	0.00	0.00
orb08	10×10	1319	1319	1319	0.00	0.00	0.00
orb09		1445	1445	1445	0.00	0.00	0.00
orb10		1557	1557	1564	0.00	0.45	0.00
la06		1248	1248	1248	0.00	0.00	0.00
la07		1172	1172	1172	0.00	0.00	0.00
la08	15×5	1244	1244	1256	0.00	0.96	0.00
la09		1358	1362	1358	0.29	0.00	0.00
la10		1287	1294	1287	0.54	0.00	0.00
Average					0.29	0.33	0.01

Values of δ^{HBB} and $\delta^{HBB'}$ in percents.

From the conducted research results that an optimal solution cannot be obtained in 30% examples. However, the average distance, in the sense of a goal function's value of the best solutions to the optimal solutions (percentage relative deviation, PRD) is small and equals 0.3%. For instance an *GASA* algorithm (Macchiaroli et al., 1999) generates solutions with PRD on the 8% level, the constructional algorithm presented in a further part of this work generates solutions with a PRD on the level of 18%. Additionally, applying at the same time search on both the original and the mirror instance, the number of examples, in which an optimal solution cannot be obtained, decreases to a few percent. Furthermore, the average PRD of the goal function's values of solutions obtained by the method proposed in this paper is 0.005%. Since the tests are conducted on a relatively small group of examples these values can differentiate a lot from the real ones, although they give an idea about relatively small scale of discussed here inconvenience.

4. Hybrid tabu search algorithm

The proposed here hybrid tabu search algorithm (called further as *HTS*) consists of two parts. The superior controlling element based on a tabu search technique and the other element – executive, generating a problem's solution with established by the superior element controlling parameters. The exact description of an algorithm generating solutions with an analysis of obtained results is included in a previous part of this work. For the further algorithm's description the only thing which is important is that the parameter controlling the executive is a tasks' permutation: CC which is called a loading permutation. It follows that the superior controlling is an algorithm in which the decision variable is one permutation, whereas the criterion function is the goal function's value of a solution obtained by a constructional algorithm. This approach enables to find a practical relation between a change in a loading permutation and a change of the obtained schedule length. The existence of the above phenomenon means that by adding (or taking away) a task from a partial loading permutation the obtained schedule can both lengthen and shorten.

As the general description of the tabu search algorithm can also be found in works of Glover (1989) and Glover (1990) in the following subsections the most important components of this algorithm will be presented in detail.

4.1. Neighborhood

The neighborhood of a permutation $\pi \in \Pi$ used in *HTS* algorithm is based on movements of insert type. The movement $v = (a, b)$, $1 \leqslant a \leqslant r$, $1 \leqslant b \leqslant r$, $a \neq b$, $a \neq b - 1$ of the insert type made on a permutation $\pi \in \Pi$ consists in moving an element from a position a onto a position b forming a new permutation $\pi_{(v)} \in \Pi$. The number of movements and also the neighborhood size for an r element permutation equals $(r-1)^2$. In the presented algorithm one has decided to restrict the size of the analyzed neighborhood only for solutions obtained by movements of a restricted size. The movement size $v = (a, b)$ is calculated as a number of positions about which an element is moved $|a - b|$. Finally, the analyzed neighborhood arises from movements: $v = (a, b)$, $1 \leqslant a \leqslant r$, $1 \leqslant b \leqslant r$, $a \neq b$, $a \neq b - 1$, $moveSizeMin \leqslant |a - b| \leqslant moveSizeMax$. If a few movements generate a solution identically good, a movement with a larger size is chosen. In the presented *HTS* algorithm the following parameters' values are determined: $moveSizeMin = 1$, $moveSizeMax = 10$.

Landscape of the solution space characterizes large flat regions (many solutions from a neighborhood have the same value of the goal function). During evaluation of the goal function solutions which have the same value of the goal function as it was in previous iterations are penalized. Quality of the solution obtained from π by a move v is determined as a sum of the cost function and a penalty of lack of its change comparing to the $F(\pi)$:

$$quality(\pi, v) == \begin{cases} F(\pi_{(v)}) & \text{for } F(\pi) \neq F(\pi_{(v)}), \\ F(\pi_{(v)}) + \Delta F & \text{for } F(\pi) = F(\pi_{(v)}), \end{cases} \tag{2}$$

where $F(\pi)$ is a value of solution connected with a solution obtained by the algorithm generating solutions from a permutation π and value of ΔF is a constant penalty. In the algorithm proposed here $\Delta F = 100$.

4.2. Tabu list

Elements of the tabu list T constitute some attributes of a permutation and a movement $AV(\pi, v)$ of the last tabu *tabuN* algorithm's iteration. Let an arc $(x \to y)$ denote a sequence relation of tasks x and y in a permutation. After a movement $v = (a, b)$, transforming a permutation $\pi \in \Pi$ into a permutation $\pi_{(v)} \in \Pi$, on the tabu list there is a pair (x, y) for which in a permutation π a relation $(x \to y)$ takes place, and in a permutation $\pi_{(v)}$ changes into a reverse relation $(y \to x)$. It is obvious that for an each movement can exist many such pairs (x, y). In an algorithm there was applied a method of choice of pair of tasks in such a way that it prohibits possibly the largest number of movements in successive iterations. That is on the tabu list the following attributes are written:

$$AV_{SOFT}(\pi, v) = \begin{cases} (\pi(a), \pi(b)) & \text{for } a < b, \\ (\pi(b), \pi(a)) & \text{for } a > b. \end{cases} \tag{3}$$

Movement $v = (a, b)$ is considered to be prohibited, if on the tabu list there is at least one pair of tasks (x, y) for which a relation $(x \to y)$ takes place in a created permutation $\pi_{(v)}$. In an algorithm *HTS* parallel with presented prohibitions the second tabu list *eT* also works. This idea consists in such a simple method: when a permutation π is transferred into other permutation $\pi_{(v)}$ by a move $v = (a, b)$, the number of moved element $\pi(a)$ is written down onto the *eT* list. During successive iterations if an element is on the list *eT*, it must not be moved.

4.3. Start solutions

The constructional algorithm generating start solutions, called further HNEH is a hybrid algorithm, too. The procedure generating a solution is the same as in a *HTS* algorithm, whereas the superior component is a NEH algorithm (Nawaz, Enscore, & Ham, 1983). An algorithm NEH is dedicated to a permutation flow problem, in which (by a typical permutation-graph model) a governing variable in a tasks' permutation. Since the controlling parameter of the executive component is also a tasks' permutation, an algorithm NEH without any modifications was applied to generate a starting loading permutation. Finally, the algorithm's computational complexity HNEH equals $O(r^4N^2)$, where $N = \max_{k \in J}\{o_k\}$.

4.4. Backtrack jump's method

The backtrack jump's method intensifies the search which is in the surrounding of the best found solution. A return to this solution takes place after executing *backJumplter* of idle iterations (which do not improve the best known solution). Besides, aiming at forcing another strategy and simultaneously another search's trajectory by each returning to a given solution the main algorithm's parameters are changed:

- with each return a cyclical change of written attributes on the tabu list takes place; an algorithm switches between attributes $AV_{SOFT}(\pi, v)$ and $AV_{HARD}(\pi, v)$;

- with each second return the tabu list increases by 1; $tabuN := tabuN + 1$; as well as there takes place a twice increase of acceptable lengths for executing before a backtrack jump of idle iterations: $backJumpIter := 2 \cdot backJumpIter$.

By each time of improvement of the best found solution, the start value of the controlling parameters is restored, and the tabu list's content is erased. Finally the *HTS* algorithm's parameters are:

- *timeMax* – determining the maximal time of an algorithm's work,
- *iterMax* – determining the maximal number of executed iterations,
- *tabuN* – determining the start length of a tabu list,
- *backJumpIter* – determining the starting number of idle iterations after which a return jump takes place.

5. Computational experiments

The presented here *HTS* algorithm was programmed in C++ and was started on a PC computer with a processor AMD Duron 800 MHz. The algorithm's efficiency both understood as the function's speed and the quality of delivered solutions was tested on well known in the literature test examples. These examples are diversified both in the respect of the number of machines (from 5 to 20) and in the number of tasks (from 6 to 30). For a better interpretation of obtained results they were compared in the respect of results by a hybrid algorithm *GASA* (Macchiaroli et al., 1999) which is composed of a genetic algorithm and elements from a simulated annealing algorithm. The solution of a given example by an *HTS* algorithm was conducted each time by starting the algorithm both for an original instance and a mirror one. The next solution obtained from a mirror instance underwent a simple transformation in such a way that it constituted a proper solution of the original problem. From obtained in such a way two solutions of a given instance for a further examination was taken a solution better in a sense of a goal function value. The time of obtained in such a way solution was certainly calculated as a sum of a time of finding a solution of an original instance and a time of finding a solution of mirror instance and times of starting solutions.

The computational experiments are composed of two parts. In the first part the algorithm's efficiency is examined on 'easy' test examples whereas in the second part on 'difficult' ones. 'Easy' examples are these which could be solved accurately using a branch and bound (B&B)algorithm described in paper (Mascis & Pacciarelli, 2002) with a number of tasks not exceeding 10; $r \leqslant 10$, whereas the rest are called 'difficult' examples.

For an each instance there was determined the length of C^{HTS} schedule generated by the examined *HTS* algorithm.The analogical data C^{GASA} for a *GASA* algorithm was obtained by starting 30 times the *GASA* algorithm (on a computer with an Athlon 1400 MHz processor). Also the results of TS (Schuster, 2006) have been considered. Then for an each instance an average relative percentage deviation PRD was calculated relative to the optimal solution using a formula (1); for $Alg \in \{HTS, GASA, TS\}$.

As we have already mentioned the goal function values of C^{BEST} optimal solutions were taken from a work (Mascis & Pacciarelli, 2002) where they were obtained by applying an precise algorithm of B&B. Values C^{HTS}, C^{GASA}, C^{TS} and C^{BEST} for an each easy example were put in Table 2.

Table 2
Results for 'easy' instances

Inst. name	$r \times m$	Ref	TS (Schuster)			GASA30 (Schuster)			HTS			
			C^{TS}_{max}	T^{work}	PRD	C^{GASA}_{max}	T^{work}	PRD	C^{HTS}_{max}	T^{find}	T^{work}	PRD
ft06	6×6	73	73	0	0.0	73	6	0.0	73	0	0	0.0
ft10		1607	1620	2	0.8	1620	41	0.8	1607	0	2	0.0
abz5	10×10	2150	2233	2	3.9	2235	41	4.0	2182	0	2	1.5
abz6		1718	1758	1	2.3	1758	41	2.3	1760	1	1	2.4
la01		971	1043	1	7.4	1037	23	6.8	975	0	1	0.4
la02		937	990	0	5.7	990	24	5.7	975	0	0	4.1
la03	10×5	820	832	1	1.5	832	24	1.5	820	0	1	0.0
la04		887	889	0	0.2	889	25	0.2	889	0	0	0.2
la05		777	817	0	5.1	817	24	5.1	777	0	0	0.0
orb01		1615	1663	1	3.0	1663	39	3.0	1615	1	1	0.0
orb02		1485	1555	1	4.7	1555	40	4.7	1518	0	1	2.2
orb03	10×10	1599	1603	1	0.3	1603	41	0.3	1599	1	1	0.0
orb04		1653	1653	1	0.0	1653	43	0.0	1653	0	1	0.0
orb05		1365	1415	2	3.7	1415	44	3.7	1367	0	2	0.1
orb06		1555	1555	0	0.0	1555	42	0.0	1557	0	0	0.1
orb07		689	706	1	2.5	706	43	2.5	717	0	1	4.1
orb08	10×10	1319	1319	1	0.0	1319	41	0.0	1319	0	1	0.0
orb09		1445	1535	1	6.2	1535	40	6.2	1449	0	1	0.3
orb10		1557	1618	1	3.9	1618	46	3.9	1571	0	1	0.9
la16		1575	1637	1	3.9	1637	39	3.9	1575	0	1	0.0
la17		1371	1430	1	4.3	1430	42	4.3	1384	0	1	0.9
la18	10×10	1417	1555	1	9.7	1555	42	9.7	1417	1	1	0.0
la19		1482	1610	1	8.6	1610	40	8.6	1491	0	1	0.6
la20		1526	1705	1	11.7	1693	45	10.9	1526	0	1	0.0
la06		1248	1299	2	4.1	1339	80	7.3	1248	0	2	0.0
la07		1172	1227	5	4.7	1240	70	5.8	1172	1	5	0.0
la08	15×5	1244	1305	2	4.9	1296	72	4.2	1298	0	2	4.3
la09		1358	1450	3	6.8	1447	83	6.6	1415	0	3	4.2
la10		1287	1338	4	4.0	1338	70	4.0	1345	1	4	4.5
Average				1.3	3.9		43.1	4.0		0.3	1.3	1.1

Times in seconds, PRD in percents.

The second part of the test was conducted analogically to the first part, with the main difference such as that the compared algorithms were tested on 'difficult' instances, and PRD of solutions were calculated relative to C^{REF} – goal function's values of reference solutions:

$$PRD^{Alg} = \frac{C^{Alg} - C^{REF}}{\min\{C^{REF}, C^{Alg}\}} \cdot 100\%, \quad Alg \in \{GASA, HTS, TS\}. \tag{4}$$

The goal function's values of reference solutions C^{REF} were taken from the work (Schuster & Framinan, 2003). Values C^{HTS}, C^{GASA}, C^{TS} and C^{REF} of 'difficult' instances are placed in Table 3.

We consider two kind of times of the algorithm work: T^{find} – a time of finding the best solution and T^{work} – an algorithm maximal work time. Therefore an algorithm can work also after achieving the best solution in the time T^{find}, up to the maximal assumed T^{work} time.

Table 3
Results for 'hard' instances

Inst. name	$r \times m$	Ref	TS (Schuster)			GASA30 (Schuster)			HTS			
			C_{max}	T^{work}	PRD	C_{max}	T^{work}	PRD	C_{max}	T^{find}	T^{work}	PRD
ft20		2244	1637	8	−37.1	1675	184	−34.0	1608	6	8	−39.6
la11		2821	1737	11	−62.4	1825	170	−54.6	1704	6	11	−65.6
la12	20 × 5	2434	1550	10	−57.0	1631	164	−49.2	1500	5	10	−62.3
la13		2650	1701	17	−55.8	1766	183	−50.1	1696	6	17	−56.3
la14		2662	1771	10	−50.3	1805	176	−47.5	1722	3	10	−54.6
la15		2765	1808	7	−52.9	1829	167	−51.2	1747	0	7	−58.3
la21		3574	2242	5	−59.4	2182	147	−63.8	2191	3	5	−63.1
la22		2879	2008	3	−43.4	1965	135	−46.5	1922	0	3	−49.8
la23	15 × 10	3228	2093	5	−54.2	2193	136	−47.2	2126	1	5	−51.8
la24		3075	2061	5	−49.2	2150	133	−43.0	2132	4	5	−44.2
la25		2985	2072	6	−44.1	2034	142	−46.8	2020	4	6	−47.8
la26		4268	2664	14	−60.2	2945	332	−44.9	2738	11	14	−55.9
la27		4674	2968	27	−57.5	3036	311	−54.0	2794	21	27	−67.3
la28	20 × 10	4478	2886	24	−55.2	2902	324	−54.3	2741	13	24	−63.4
la29		4117	2671	12	−54.1	2617	311	−57.3	2596	2	12	−58.6
la30		4097	2939	12	−39.4	2892	346	−41.7	2791	7	12	−46.8
la31		6519	3822	151	−70.6	4298	957	−51.7	3869	46	151	−68.5
la32		6912	4186	176	−65.1	4686	869	−47.5	4045	131	176	−70.9
la33	30 × 10	6454	3869	120	−66.8	4214	860	−53.2	3751	45	120	−72.1
la34		6380	3957	102	−61.2	4401	968	−45.0	3936	96	102	−62.1
la35		6563	3908	120	−67.9	4299	897	−52.7	3918	22	120	−67.5
la36		4526	2993	9	−51.2	2949	203	−53.5	2893	7	9	−56.4
la37		4766	3171	7	−50.3	3216	192	−48.2	3107	2	7	−53.4
la38	15 × 15	4426	2734	6	−61.9	2762	202	−60.2	2706	5	6	−63.6
la39		4295	2804	9	−53.2	2908	195	−47.7	2725	6	9	−57.6
la40		4099	2977	12	−37.7	2950	214	−38.9	2804	1	12	−46.2
abz7		2824	1820	20	−55.2	1801	288	−56.8	1775	2	20	−59.1
abz8	20 × 15	2892	1815	51	−59.3	1842	336	−57.0	1727	18	51	−67.5
abz9		2838	1781	52	−59.3	1905	273	−49.0	1705	4	52	−66.5
swv01		3824	2396	11	−59.6	2510	321	−52.4	2424	7	11	−57.8
swv02		3800	2492	16	−52.5	2605	337	−45.9	2484	6	16	−53.0
swv03	20 × 10	3797	2489	17	−52.6	2582	301	−47.1	2404	3	17	−57.9
swv04		3895	2520	23	−54.6	2609	339	−49.3	2545	17	23	−53.0
swv05		3836	2482	22	−54.6	2601	342	−47.5	2489	1	22	−54.1
swv06		5163	3502	29	−47.4	3639	437	−41.9	3463	2	29	−49.1
swv07		5076	3343	32	−51.8	3569	500	−42.2	3299	19	32	−53.9
swv08	20 × 15	5547	3611	29	−53.6	3744	450	−48.2	3567	3	29	−55.5
swv09		5117	3436	39	−48.9	3554	457	−44.0	3439	34	39	−48.8
swv10		5347	3569	23	−49.8	3773	446	−41.7	3561	17	23	−50.2
swv11		9258	5586	1736	−65.7	6786	4149	−36.4	5634	1298	1736	−64.3
swv12		9243	5358	2212	−72.5	6722	4054	−37.5	5465	757	2212	−69.1
swv13		9480	5546	2360	−70.9	6854	4158	−38.3	5807	1626	2360	−63.3
swv14		9450	5483	1602	−72.4	6614	4067	−42.9	5458	287	1602	−73.1
swv15		9247	5516	2077	−67.6	6652	4311	−39.0	5619	1384	2076	−64.6
swv16	50 × 10	10659	6337	1347	−68.2	7483	3938	−42.4	6233	698	1348	−71.0
swv17		10003	5884	1760	−70.0	7304	3595	−37.0	5900	1292	1760	−69.5
swv18		10102	6247	1430	−61.7	7246	3869	−39.4	5931	1268	1430	−70.3
swv19		10055	6308	1481	−59.4	7547	3968	−33.2	6283	1137	1481	−60.0
swv20		10663	6019	1843	−77.2	7311	3759	−45.8	5945	1653	1843	−79.4
yn1		4291	2654	68	−61.7	2751	385	−56.0	2630	33	68	−63.2
yn2	20 × 20	4025	2705	41	−48.8	2644	398	−52.2	2647	19	41	−52.1
yn3		4029	2644	134	−52.4	2704	362	−49.0	2465	92	134	−63.4
yn4		4109	2705	53	−51.9	2837	433	−44.8	2630	39	53	−56.2
Average				366.0	−56.9		1041.3	−47.2		229.5	366.0	−59.4

Times in seconds, PRD in percents.

Construction of the work time T^{work} is based on the time taken from the work (Schuster, 2006) to compare algorithms for the similar environment. Two situations have been considered:

1. if there is a time 0 of the TS (Schuster, 2006) algorithm we assume that it was executed in a time which was shorter than 0.5 of second (from a range $(0, 0.5)$). Therefore we consider 0.25 of second as work time – an expected value of a random variable of symmetric distribution (i.e. unify or normal) on a range from 0 to 0.5;
2. if there is a time $t \leqslant 0$ of the TS (Schuster, 2006) algorithm, we assume t as a work time for our *HTS* algorithm.

As mentioned above, *HTS* algorithm was tested on AMD Duron 800 MHz personal computer. Due to WinBench98, the computer from the work (Schuster, 2006) was 4154 MIPS, comparing to 2374 MPIS of our Duron 800 MHz, so our hardware was two times slower. Even without using a factor (taken from the difference of speed of computers) there was a possibility to improve almost all results of the benchmark tests with the same (or shorter) time of computations.

The average result (percentage relative deviation to the reference values) for 29 'easy' instances from Table 2 for the TS (Schuster, 2006) equals to 3.93% in an average time 1.35 s. Our *HTS* algorithm gives 1.08% in a time 0.29 s. Results of *GASA*30 algorithm (the best result from 20 execution of the GASA algorithm) were significantly worse.

For 'hard' instances, the average result for 53 instances from Table 4 for the TS (Schuster, 2006) equals to −56.94% (improvement of reference solutions) in an average time 365.96 s. The proposed *HTS* was executed with identical work time for each instance, as the time of TS (Schuster, 2006) was. The *HTS* algorithm gives −59.4% of improvement – over 2.3% better in average comparing to TS (Schuster, 2006), founding new the best known solutions for almost all 'hard' instances.

Table 4 presents results of finding the new reference solutions without the limit of computations time. The obtained results are 60% better than reference ones (i.e. 4% better than the ones taken from the paper Schuster (2006)). The newest results from Framinan and Schuster (2006) also have been considered. HTS found new the best known solutions for 51 instances.

6. Conclusion

The main achievement of this work is a very high efficiency of an algorithm. In a comparable time with a literature algorithm *GASA* and tabu search (Schuster, 2006) the presented algorithm *HTS* finds solutions much better in a sense of optimized criterion. Conception of the algorithm proposed here gives a possibility to obtain solutions which are not able to be obtained using recent algorithms from the literature. Due to this property it was possible to obtain a new, fast, two-level algorithm (level one: generating permutations, level two: construct a solution from this 'loading' permutation) and to find 51 new the best known solutions for the classic benchmark instances from literature.

Besides, an attention should be paid to a straight line of an algorithm *HTS*. This algorithm does not use in any way the essence of the problem; it only suggests the length of a schedule of the obtained result of the executive component's function. The present further research of the authors' work consists in making an attempt to show some relations between a loading permutation delivered through a superior component of an algorithm *HTS* and a length of a schedule without the necessity of its construction. According to the author finding even some statistical relations between these entities would direct a search of con-

Table 4
Old the best results C^{Old}_{max} and the new results of makespan C^{New}_{max} found by HTS for 'hard' instances. 51 new the best known solutions (marked by the bold font) have been found

Instance	$r \times m$	C^{Old}_{max}		C^{New}_{max}	Instance	$r \times m$	C^{Old}_{max}		C^{New}_{max}
ft20		1637	(2)	**1532**	abz7		1801	(1)	**1592**
la11		1714	(3)	**1621**	abz8	20 × 15	1815	(2)	**1642**
la12	20 × 5	1507	(3)	**1434**	abz9		1781	(2)	**1562**
la13		1665	(3)	**1580**	swv01		2396	(2)	**2318**
la14		1757	(3)	**1610**	swv02		2485	(3)	**2417**
la15		1771	(3)	**1686**	swv03	20 × 10	2489	(2)	**2381**
la21		2092	(3)	**2030**	swv04		2506	(3)	**2462**
la22		1928	(3)	**1852**	swv05		2482	(2)	**2333**
la23	15 × 10	2038	(3)	**2021**	swv06		3291	(3)	**3290**
la24		2061	(2)	**1972**	swv07		3331	(3)	**3188**
la25		2034	(1)	**1906**	swv08	20 × 15	3611	(2)	**3423**
la26		2664	(2)	**2506**	swv09		3378	(3)	**3270**
la27		2933	(3)	**2675**	swv10		3552	(3)	**3462**
la28	20 × 10	2789	(3)	**2552**	swv11		5586	(2)	**5564**
la29		2565	(3)	**2300**	swv12		**5358**	(2)	5441
la30		2835	(3)	**2452**	swv13		**5546**	(2)	5628
la31		3822	(2)	**3498**	swv14		5483	(2)	**5401**
la32		4186	(2)	**3882**	swv15	50 × 10	5516	(2)	**5435**
la33	30 × 10	3754	(3)	**3454**	swv16		6337	(2)	**5843**
la34		3938	(3)	**3659**	swv17		5884	(2)	**5780**
la35		3908	(2)	**3552**	swv18		6247	(2)	**5785**
la36		2810	(3)	**2685**	swv19		6308	(2)	**5997**
la37		3044	(3)	**2831**	swv20		6019	(2)	**5724**
la38	15 × 15	2726	(3)	**2525**	yn1		2654	(2)	**2360**
la39		2752	(3)	**2687**	yn2	20 × 20	2644	(1)	**2370**
la40		2838	(3)	**2580**	yn3		2644	(2)	**2320**
					yn4		2705	(2)	**2513**

(1) – found by GASA30 of Schuster and Framinan (2003).
(2) – found by TS of Schuster (2006).
(3) – from Framinan and Schuster (2006).

trolling parameters in a superior algorithm in a statistically best way. Such an *HTS* algorithm's modification could significantly increase its efficiency.

References

Framinan, J., & Schuster, C. (2006). An enhanced timetabling procedure for the no-wait job shop problem: A complete local search approach. *Computers and Operations Research, 33*(5), 1200–1213.

Glover, F. (1989). Tabu search – part I. *ORSA Journal on Computing, 1*, 190–206.

Glover, F. (1990). Tabu search – part II. *ORSA Journal on Computing, 2*, 4–32.

Graham, R., Lawler, E., Lenstra, J., & Rinnooy Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Hall, N., & Sriskandarajah, C. (1996). A survey of machine scheduling-problems with blocking and no-wait in process. *Operations Research, 44*(3), 510–525.

Lenstra, J., & Rinnooy Kan, A. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics, 4*, 121–140.

Macchiaroli, R., Mole, S., & Riemma, S. (1999). Modelling and optimization of industrial manufacturing processes subject to no-wait constrains. *International Journal of Production Research, 37*(11), 2585–2607.

Mascis, A., & Pacciarelli, D. (2002). Job shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research, 142*(3), 498–517.

Nawaz, M., Enscore, E. E., Jr., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA International Journal of Management Science, 11*, 91–95.

Raaymakers, W., & Hoogeveen, J. (2000). Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research, 126*, 131–151.

Schuster, C. (2006). No-wait job shop scheduling: Tabu search and complexity of subproblems. *Mathematical Methods of Operations Research, 63*(3), 473–491.

Schuster, C., & Framinan, J. (2003). Approximative procedures for no-wait job shop scheduling. *Operations Research Letters, 31*, 308–318.

Wismer, D. A. (1972). Solution of the flowshop scheduling-problem with no intermediate queues. *Operations Research, 20*, 689–697.