A Neuro-tabu Search Algorithm for the Job Shop Problem*

Wojciech Bożejko and Mariusz Uchroński

Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology, Janiszewskiego 11-17, 50-372 Wrocław, Poland
{wojciech.bozejko,mariusz.uchronski}@pwr.wroc.pl

Abstract. This paper deals with tabu search with neural network instead of classic tabu list applied for solving the classic job shop scheduling problem with makespan criterion. Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that the proposed algorithm solves the job shop instances with high accuracy in a very short time. Presented ideas can be applied for many scheduling problems.

1 Introduction

The paper deals with the job shop problem which can be briefly presented as follows. There is a set of jobs and a set of machines. Each job consists of a number of operations which have to be processed in a given order, each one on a specified machine during a fixed time. The processing of an operation cannot be interrupted. Each machine can process at most one operation at a time. We want to find a schedule (the assignment of operations to time intervals on machines) that minimizes the *makespan*.

The job shop scheduling problem, although relatively easily stated, is NP-hard and it is considered as one of the hardest problems in the area of combinatorial optimization. Many various methods have been proposed, ranging from simple and fast dispatching rules to sophisticated branch-and bound and metaheuristic algorithms. For the literature see Balas and Vazacopoulos [1] (guided local search method with shifting bottleneck), Morton and Pentico [5] (heuristic local search), Nowicki and Smutnicki [6] (tabu search with representatives and block properties), Vaessens et al. [9] (local search methods), and their references.

In a classic tabu search method a move is chosen in each iteration of the algorithm. This move is remembered on the list of the length *maxt* called the *tabu list* and it is forbidden for *maxt* number of iteration. After executing *maxt* iterations by the algorithm this move is removed from the list and it can be executed again. One can say that this move looses its status of being forbidden 'suddenly'. Here we present a mechanism in which a status a forbidden move is changing in the exponential way. Such an approach was successfully applied for the quadratic assignment problem [4] and for the flow shop scheduling problem [7].

* The work was supported by MNiSW Poland, within the grant No. N N514 232237.

2 Problem Formulation and Preliminaries

The job shop problem can be formally defined as follows, using the notation by Nowicki and Smutnicki [6]. There are: a set of jobs $J = \{1, 2, ..., n\}$, a set of machines $M = \{1, 2, ..., m\}$, and a set of operations $O = \{1, 2, ..., o\}$. Set O decomposes into subsets (chains) corresponding to the jobs. Each job j consists of a sequence of o_j operations indexed consecutively by $(l_{j-1} + 1, ..., l_{j-1} + o_j)$, which are to be processed in order, where $l_j = \sum_{i=1}^{j} o_i$, is the total number of operations of the first j jobs, $j = 1, 2, ..., n$, $(l_0 = 0)$, and $o = \sum_{i=1}^{n} o_i$. Operation x is to be processed on machine $\mu_x \in M$ during processing time p_x, $x \in O$. The set of operations O can be decomposed into subsets $M_k = \{x \in O | \mu_x = k\}$, each containing the operations to be processed on machine k, and $m_k = |M_k|$, $k \in M$. Let permutation π_k define the processing order of operations from the set M_k on machine k, and let Π_k be the set of all permutations on M_k. The processing order of all operations on machines is determined by m-tuple $\pi = (\pi_1, \pi_2, ..., \pi_m)$, where $\pi \in \Pi_1 \times \Pi_2 \times ... \times \Pi_m$.

It is useful to present the job shop problem by using a graph. For the given processing order π, we create the graph $G(\pi) = (N, R \cup E(\pi))$ with a set of nodes N and a set of arcs $R \cup E(\pi)$, $N = O \cup \{s, c\}$, where s and c are two fictitious operations representing dummy 'start' and 'completion' operations, respectively. The weight of node $x \in N$ is given by the processing time p_x, $(p_s = p_c = 0)$. The set R contains arcs connecting consecutive operations of the same job, as well as arcs from node s to the first operation of each job and from the last operation of each job to node c. Arcs in $E(\pi)$ connect operations to be processed by the same machine. Arcs from set R represent the processing order of operations in jobs, whereas arcs from set $E(\pi)$ represent the processing order of operations on machines. The processing order π is feasible if and only if graph $G(\pi)$ does not contain a cycle.

Let $C(x, y)$ and $L(x, y)$ denote the longest (critical) path and length of this path, respectively, from node x to y in $G(\pi)$. It is well-known that makespan $C_{max}(\pi)$ for π is equal to length $L(s, c)$ of critical path $C(s, c)$ in $G(\pi)$. Now, we can rephrase the job shop problem as that of finding a feasible processing order $\pi \in \Pi$ that minimizes $C_{max}(\pi)$ in the resulting graph.

We use a notation similar to the paper of Balas and Vazacopoulos [1]. For any operation $x \in O$, we will denote by $\alpha(x)$ and $\gamma(x)$ the job-predecessor and job-successor (if it exists), respectively, of x, i.e. $(\alpha(x), x)$ and $(x, \gamma(x))$ are arcs from R. Further, for the given processing order π, and for any operation $x \in O$, we will denote by $\beta(x)$ and $\delta(x)$ the machine-predecessor and machine-successor (if it exists), respectively, of x, i.e. the operation that precedes x, and succeeds x, respectively, on the machine processing operation x. In other words, $(\beta(x), x)$ and $(x, \delta(x))$ are arcs from $E(\pi)$.

Denote the critical path in $G(\pi)$ by $C(s, c) = (s, u_1, u_2, \ldots, u_w, c)$, where $u_i \in O$, $1 \leq i \leq w$, and w is the number of nodes (except fictitious s and c) in this path. The critical path $C(s, c)$ depends on π, but for simplicity in notation we will not express it explicitly. The critical path is decomposed into subsequences $B_1, B_2, \ldots, B_r$ called *blocks* in π on $C(s,c)$ (Grabowski et al. [3]), where

1. $B_k = (u_{f_k}, u_{f_k+1}, ..., u_{l_k-1}, u_{l_k}), \quad 1 \leq f_k \leq l_k \leq w, \quad k = 1, 2, ..., r.$
2. B_k contains operations processed on the same machine, $k = 1, 2, \ldots, r.$
3. two consecutive blocks contain operations processed on different machines.

In other words, the block is a maximal subsequence of $C(s,c)$ and contains successive operations from the critical path processed consecutively on the same machine. In the further considerations, we will be interested only in *non-empty* block, i.e. such that $|B_k| > 1$, or alternatively $f_k < l_k$. Operations u_{f_k} and u_{l_k} in B_k are called the *first* and *last* ones, respectively. The k-th block, exclusive of the first and last operations, is called the k-th *internal block*.

A block has advantageous so-called *elimination properties*, introduced originally in the form of the following theorem (Grabowski et al. [3]).

Theorem 1. *Let $G(\pi)$ be an acyclic graph with blocks B_k, $k = 1, 2, ..., r$. If acyclic graph $G(\omega)$ has been obtained from $G(\pi)$ through the modifications of π so that $C_{max}(\omega) < C_{max}(\pi)$, then in $G(\omega)$*

(i) at least one operation $x \in B_k$ precedes job u_{f_k}, for some $k \in \{1, 2, ..., r\}$, or
(ii) at least one operation $x \in B_k$ succeeds job u_{l_k}, for some $k \in \{1, 2, ..., r\}$.

Example 1. There are three jobs, 9 operations and three machines, $n = 3$, $m = 2$, $o = 9$. The job 1 consist of the sequence of three operations (1,2,3), the job 2 consist of a sequence of three operations (4,5,6) and the job 3 consist of a sequence of three operations (7,8,9). Operations have to be processed on machines $\mu_2 = \mu_6 = \mu_7 = 1$, $\mu_1 = \mu_5 = \mu_9 = 2$, $\mu_3 = \mu_4 = \mu_8 = 3$. A feasible processing order is $\pi = (\pi_1, \pi_2, \pi_3)$, where $\pi_1 = (7, 2, 6)$, $\pi_2 = (1, 5, 9)$ and $\pi_3 = (4, 8, 3)$. The graph $G(\pi)$ is shown in the Figure 2 and the Gantt chart – in the Figure 1.

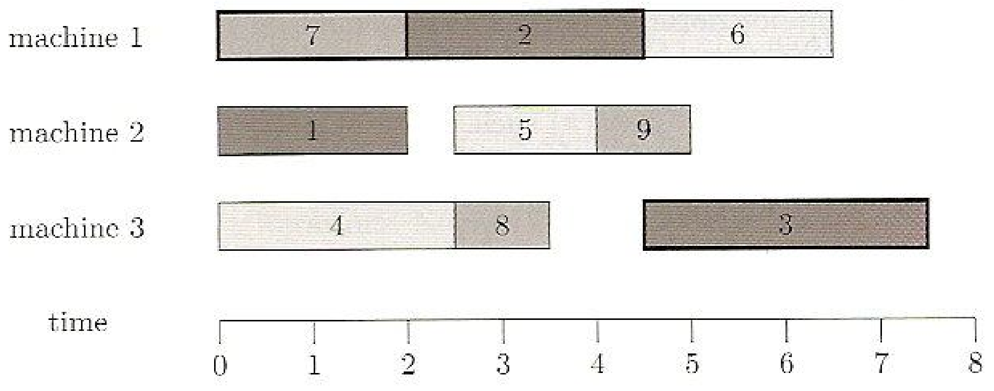

Fig. 1. The Gantt chart for the Example 1

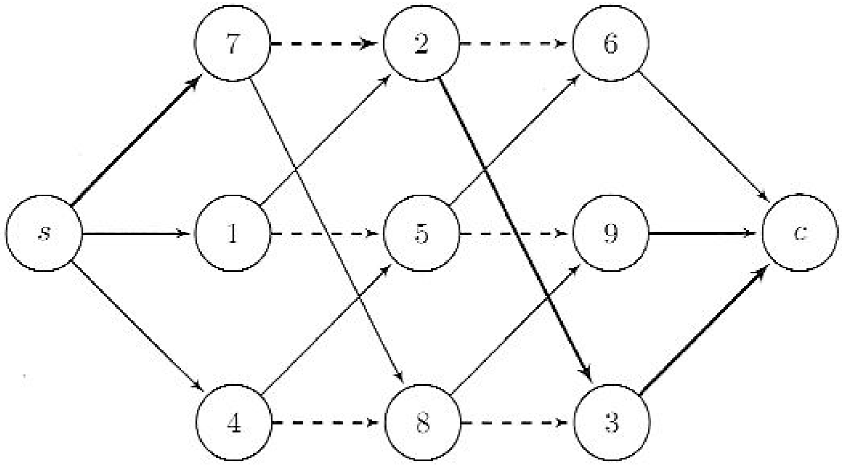

Fig. 2. The graph $G(\pi)$ for the Example 1

3 Tabu Search Method

The tabu search method was proposed by Glover [2]. Generally, it consists of improving the starting solution's value π^*. An algorithm generates the neighborhood of the current solution and seeks the solution which has the minimal value of $C_{\max}(\beta)$, $\beta \in \mathcal{N}(\pi^*)$. This solution β is the starting solution in the next iteration of the algorithm. Such a procedure allows the possibility of increasing the current solution's value (when a new starting solution is sought), but it increases the chance of finding the global minimum. To prevent generating of recently considered solutions (making cycles), those solutions are recorded on a list of prohibited solutions, the so-called tabu list T (short-term memory). A standard tabu search algorithm can be written as follows.

Algorithm 1. Standard tabu search algorithm

```
Let $\pi \in \Pi$ be an initial solution;
    $\pi^*$ – the best known solution; $\pi^* \leftarrow \pi$ – starting solution;
Step 1. Generate the neighborhood $\mathcal{N}(\pi)$ of the current solution $\pi$.
        Exclude from $\mathcal{N}(\pi)$ elements from the $T$ list except $\beta \in \mathcal{N}(\pi)$ such that
            $C_{\max}(\beta) < C_{\max}(\pi^*)$;
Step 2. Find the solution $\delta \in \mathcal{N}(\pi)$ such, that
            $C_{\max}(\delta) = min(C_{\max}(\beta), \beta \in \mathcal{N}(\pi))$;
Step 3. if $C_{\max}(\delta) < C_{\max}(\pi^*)$ then
            $\pi^* \leftarrow \delta$;
        Include $\delta$ in the list $T$; $\pi \leftarrow \delta$;
Step 4. if (Stop condition is true) then Stop; else go to Step 1;
```

Let B_k ($k = 1, 2, \ldots, m$) be the k-th block in a solution π, B_k^f and B_k^l the subblocks. For job $j \in B_k^f$ by $N_k^f(j)$ let us denote a set of solutions created by moving job j to the beginning of block B_k (before the first job in block $\pi(f_k)$). Analogously, for job $j \in B_k^f$ by $N_k^l(j)$ let us denote a set of solutions created by moving job j to the end of block B_k (after the last job in block $\pi(l_k)$). The neighborhood of the solution π: $\mathcal{N}(\pi) = \bigcup_{j \in B_k} (N_k^f(j) \cup N_k^l(j))$.

Additionally, there is a backtracking mechanism applied in the algorithm (long-term memory). A certain number of good solutions are recorded on backtracking list. Good solution – this means that the relative difference between this solution β and the best known (current) solution $\pi^*$is small or negative – less then ϵ parameter ($\frac{C_{\max}(\beta)-C_{\max}(\pi^*)}{C_{\max}(\pi^*)} < \epsilon$). If there is no improvement of the best solution's objective function value after some number of iterations, the algorithm jumps to the latest solution obtained from the backtracking list (so the current solution π is overwritten by the solution from the list). The current tabu list is also overwritten – the algorithm receives the tabu list connected with the backtracked solution from the backtracking list.

4 Tabu Search Mechanism with Neural Network Application

In the considered tabu search algorithm each move is represented by its neuron. For the neighborhood considered in [6] a network of neurons formed of $o - 1$ neurons. Let i-th neuron represents a move consisting in swap of two adjacent elements on the positions i and $i+1$ in a solution π. In a proposed neural network architecture a history of each neuron is stored as its internal state (*tabu effect*). If in an iteration neuron is activated, then the value 1 is fixed on its output and values 0 are fixed on the outputs of other neurons. The neuron activated in an iteration must not be activated once again for the next s iterations. Each neuron is defined by the following equations:

$$\eta_i(t+1) = \alpha \Delta_i(t), \ \Delta_i(t) = \frac{C_{max}(\pi_v^{(t)}) - C_{max}^*}{C_{max}^*}, \ \gamma_i(t+1) = \sum_{d=0}^{s-1} k^d x_i(t-d), \quad (1)$$

where $x_i(t)$ is an output of the neuron i in the iteration t. Symbol $C_{max}(\pi_v^{(t)})$ means the value of the goal function for the permutation obtained after executing a move v in the iteration t, i.e. $\pi^{(t)}$. Symbol $\Delta_i(t)$ means a normalized, current value of the goal function, and C_{max}^* is the value of the best solution found so far. Parameters α i k are scale factors. A symbol $\eta_i(t+1)$ (*gain effect*) defines quality of a move v. A variable $\gamma_i(t+1)$ (*tabu effect*) stores a history of the neuron i for the last s iterations. Neuron is activated if it has a low value of the tabu effect and it gives a better reduction of the C_{max}. More detailed a neuron i is activated if it has the lowest $\{\eta_i(t+1) + \gamma_i(t+1)\}$ value of all the neurons.

If $0 < k < 1$ i $s = t$ then the formula of γ_i from the equation (1) takes the form of:

$$\gamma_i(t+1) = k\gamma_i(t) + x_i(t), \quad (2)$$

where $\gamma_i(0) = 0$ and $x_i(0) = 0$ for each i. From the equation (2) it follows, that the value of $\gamma_i(t)$ of each neuron decreases exponentially (see Figure 3).

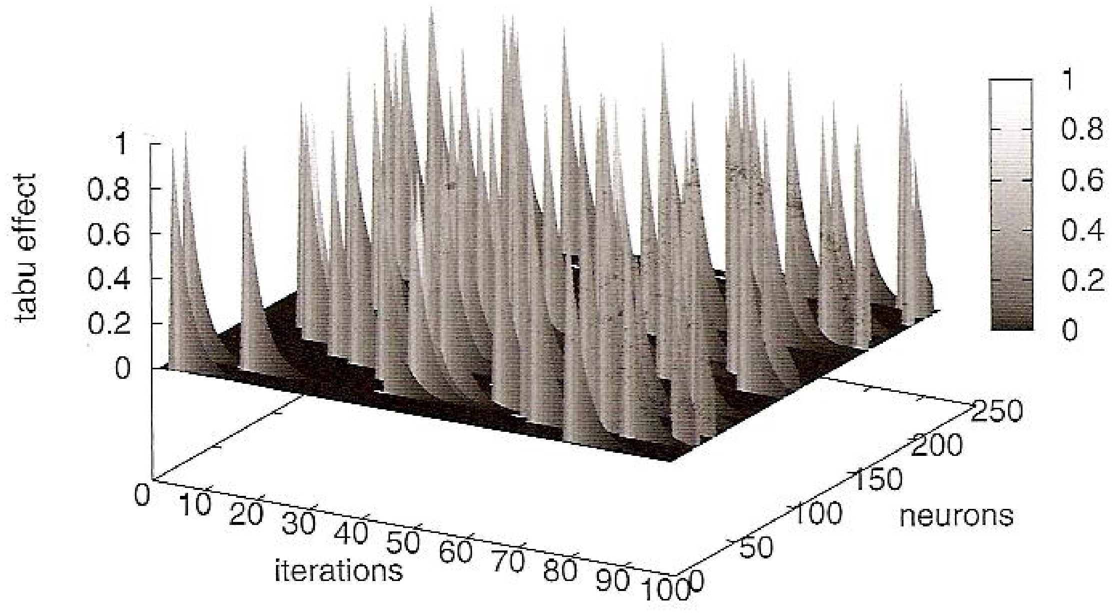

Fig. 3. Changes of the $\gamma(t)$ value

In many algorithms proposed in the literature which are based on the tabu search method a so-called aspiration criterion is implemented. It consists in executing of the forbidden move if it follows to the base solution with the goal function value lower than the best found so far.

In the proposed neuro-tabu search such a function can be implemented by ignoring tabu effect for a move v for which $\Delta_i < 0$. However during computational experiments we have observed that it does not give good effects. Therefore a proposed neuro-tabu search has not got such a function.

5 Computational Experiments

A neuro-tabu search (NTS) algorithm for the job shop scheduling problem was implemented in C++ language and executed on PC with processor with $1.7 GHz$ clock. Computational experiments have been provided for the Taillard [8] benchmark instances. Starting solutions of the NTS were determined by using INSA (*INSertion Algorithm*)[6]. The NTS algorithm was terminated after performing 100000 iterations, the value of tuning parameter *max_iter* was taken from [6], $max\_iter = 10000$ if makespan has been improved and $max\_iter = 6000$ if the backtracking jump has been performed.

The time per single iteration was approximately $3.04 \cdot 10^{-7} \cdot o$ seconds on PC with Intel Celeron 1.7 GHz processor. TSAB was run on PC 386DX which is 430 times slower than PC with Intel Celeron 1.7 GHz processor. The computing power of PC with Intel Celeron 1, 7 GHz was measured using SiSoftware Sandra (the System ANalyser, Diagnostic and Reporting Assistant)[10]. Value of computing power of PC 386DX was taken from [11]. Therefore, in order to normalize

Table 1. Percentage relative deviations to the best known solutions

			NTS		
problem	$n \times m$	INSA	$k = 0.5$	$k = 0.6$	$k = 0.7$
TA01-10	15×15	14.62	1.30	1.41	1.30
TA11-20	20×15	18.29	2.50	2.65	2.44
TA21-30	20×20	17.17	2.36	2.17	2.43
TA31-40	30×15	21.13	2.65	1.92	2.29
TA41-50	30×20	23.01	3.72	3.70	3.73
TA51-60	50×15	16.43	0.09	0.09	0.09
TA61-70	50×20	20.07	0.29	0.36	0.22
TA71-80	100×20	15.21	0.01	0.01	0.01
average		18.24	1.62	1.54	1.56

Table 2. A percentage relative deviations to the reference solutions given from Taillard [8]

				NTS		
problem	$n \times m$	INSA	TSAB	$k = 0.5$	$k = 0.6$	$k = 0.7$
TA01-10	15×15	13.93	0.8	0.70	0.81	0.72
TA11-20	20×15	16.50	0.9	0.95	1.10	0.90
TA21-30	20×20	15.75	1.2	1.11	0.93	1.18
TA31-40	30×15	18.78	0.6	0.68	-0.03	0.33
TA41-50	30×20	20.35	1.9	1.48	1.45	1.49
TA51-60	50×15	16.38	0.0	0.05	0.05	0.05
TA61-70	50×20	17.12	-2.0	-2.16	-2.09	-2.22
TA71-80	100×20	15.13	-0.1	-0.05	-0.05	-0.05
average		16.74	0.41	0.35	0.27	0.30

time per single iteration, the algorithm execution time for TSAB was divided by the transformation factor 430. After the normalization the time per single iteration for TSAB equals $0.62 \cdot 10^{-7} \cdot o$ seconds.

The best known solutions, as well as solutions from [8], were taken as reference solutions. The NTS algorithm was executed for various values of the scaling parameter k ($k = 0.5, 0.6, 0.7$) and compared with the best up to now algorithm TSAB of Nowicki and Smutnicki [6]. The value of the k parameter has got a great influence onto obtained results. The best results was obtained when the value of scaling parameter $k = 0.6$ (see Table 1 and Table 2). The table 2 shows the differences between results of TSAB and NTS algorithms – results obtained by NTS algorithms are better than results obtained by TSAB. For instances of the size 30×15 the difference between results obtained by NTS and TSAB is the most significant (see Table 2). Percentage relative deviation to the reference solutions for NTS equals -0.03% and for TSAB 0.6%. Change scaling parameter k for instances of the size 50×15 and 100×20 (Table 1 and Table 2) does not give any effect.

6 Conclusions

We present a fast algorithm based on the neuro-tabu search approach. Computational experiments are provided and compared with the results yielded by the best algorithms discussed in the literature. Results obtained by the proposed algorithm are comparable with the results of the state-of-the-art algorithm TSAB algorithm after a small number of iterations.

As a future work it is possible to adapt the proposed approach for the job shop problem with different types of neighborhood. a neural network with $(o-1)^2$ neurons can be applied for neighborhood which is generated by insertion moves. In this case a neuron (i, j) represents the move of inserting operation $\pi(i)$ in position j. In the traditional tabu search algorithm a move has the tabu status or it has not. The proposed neural network can be also modified by introduce moves with different degrees of tabu.

References

1. Balas, E., Vazacopoulosi, A.: Guided local search with shifting bottleneck for job shop scheduling. Management Science 44(2), 262–275 (1998)
2. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
3. Grabowski, J., Nowicki, E., Smutnicki, C.: Block algorithm for scheduling of operations in job shop system. Przeglad Statystyczny 35, 67–80 (1988) (in Polish)
4. Hasegawa, M., Ikeguchi, T., Aihara, K.: Exponential and chaotic neuro-dynamical tabu searches for quadratic assignment problems. Control and Cybernetics 29, 773–788 (2000)
5. Morton, T., Pentico, D.: Heuristic scheduling systems. Wiley, New York (1993)
6. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Management Science 42, 797–813 (1996)
7. Solimanpur, M., Vrat, P., Shankar, R.: A neuro-tabu search heuristic for the flow shop scheduling problem. Computers and Operations Research 31, 2151–2164 (2004)
8. Taillard, E.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64, 278–285 (1993)
9. Vaessens, R., Aarts, E., Lenstra, J.K.: Job shop scheduling by local search. INFORMS Journal of Computing 8, 303–317 (1996)
10. `http://www.sisoftware.net`
11. `http://www.roylongbottom.org.uk/mips.htm#anchorAltos`