Meta-heuristics: theory & applications
Kluwer academic publishers, MA, USA, 1996, pp. 237–248

Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search

Takeshi Yamada and Ryohei Nakano
NTT Communication Science Laboratories
2 Hikaridai Seika-cho Soraku-gun Kyoto 619-02 JAPAN
E-mail: {yamada,nakano}@cslab.kecl.ntt.jp

Abstract:
The Job-Shop Scheduling Problem (JSSP) is one of the most difficult $\mathcal{NP}$-hard combinatorial optimization problems. This paper proposes a new method for solving JSSPs based on simulated annealing (SA), a stochastic local search, enhanced by shifting bottleneck (SB), a problem specific deterministic local search. In our method new schedules are generated by a variant of Giffler and Thompson's active scheduler with operation permutations on the critical path. SA selects a new schedule and probabilistically accepts or rejects it. The modified SB is applied to repair the rejected schedule; the new schedule is accepted if an improvement is made. Experimental results showed the proposed method found near optimal schedules for the difficult benchmark problems and outperformed other existing local search algorithms.

Key Words: Simulated annealing, shifting bottleneck, job-shop scheduling, heuristics, local search

1. Background

Scheduling is allocating shared resources over time to competing activities. It has been the subject of a significant amount of literature in the operations research area. Emphasis has been on investigating *machine scheduling problems* where *jobs* represent activities and *machines* represent resources; each machine can process at most one job at a time.

The $n \times m$ *minimum-makespan* general job-shop scheduling problem, hereafter referred to as JSSP, can be described by a set of n jobs that is to be processed on a set of m machines. Each job has a technological sequence of machines to be processed. Each *operation* requires the exclusive use of each machine for an uninterrupted duration called *processing time*. The time required to complete all jobs is called *makespan*. The objective when solving or optimizing this general problem is to determine the processing order of all operations on each machine that minimizes the makespan. The JSSP is not only $\mathcal{NP}$-hard , but is extremely difficult to solve optimally. An indication of this is given by the fact that one 10×10 problem formulated by Muth and Thompson (1963) remained unsolved for over 20 years.

Besides exhaustive search algorithms based on branch and bound methods, several approximation algorithms have been developed. The most popular ones in practice are based on priority rules and active schedule generation. Adams, Balas and Zawack (1988) developed a more sophisticated method called *shifting bottleneck* (SB) which has been shown to be very successful. Stochastic approaches such as simulated annealing (SA), genetic algorithms and tabu search have been recently applied with good success.

This paper proposes a powerful method to solve the JSSP. The method is based on a combination of critical block simulated annealing (CBSA): a stochastic local search method

proposed by Yamada, Rosen and Nakano (1994), and shifting bottleneck: a deterministic local search method.
CBSA possessed the simplicity and flexibility of SA, worked well for difficult benchmark problems, and outperformed the existing SA approaches. However more recent approaches perform better than CBSA. In the present method, a modified SB is applied to repair the rejected schedule of CBSA and the new schedule is accepted if it is improved. By adding a long jump of SB, the new CBSA can omit many time-consuming transitions to make its search much more efficient.

2. Neighborhood Structure

2.1. Critical blocks

A JSSP is often described by a disjunctive graph $G = (V, C \cup D)$, where

- V is a set of nodes representing operations of the jobs together with two special nodes, a *source* (0) and a *sink* $\star$, representing the beginning and end of the schedule, respectively.
- C is a set of conjunctive arcs representing technological sequences of the operations.
- D is a set of disjunctive arcs representing pairs of operations that must be performed on the same machines.

The processing time for each operation is the weighted value attached to the corresponding nodes.

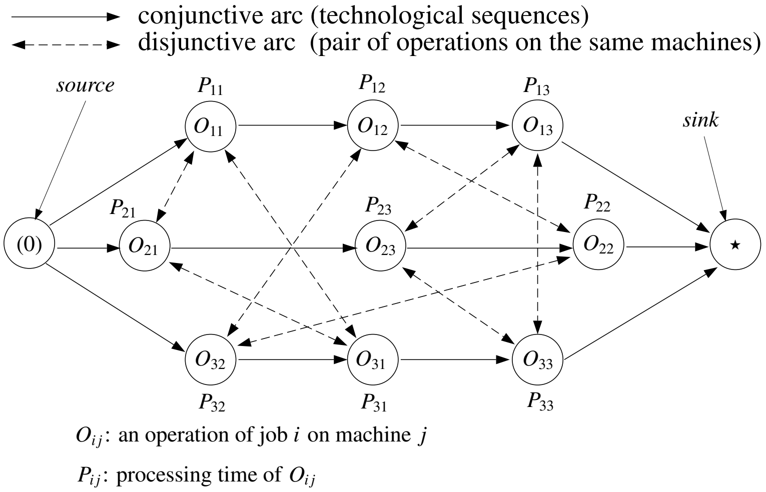

Figure 1: The disjunctive graph G of a 3×3 problem

Scheduling is to define ordering between all operations that must be processed on the same machine, i.e. to fix precedences between these operations. In the disjunctive graph model, this is done by turning all undirected (disjunctive) arcs into directed ones. The set of all directed arcs selected from disjunctive arcs is called *selection*. A selection S defines a feasible schedule if and only if the resulting directed graph is acyclic. For such a case, S is called a *complete selection*. A complete selection and the corresponding feasible schedule can be used interchangeably and represented by the same symbol S. *Makespan* is given by the length of the longest weighted path from source to sink in this graph. This path $\mathcal{P}$ is called *critical path* and is composed of a sequence of *critical operations*. A sequence of consecutive critical operations on the same machine is called a *critical block*.

2.2. Critical block neighborhood

For a JSSP, a neighborhood $N(S)$ of a schedule S can be defined as the set of feasible schedules that can be reached from S by exactly one transition (a single perturbation of S). For example, a transition operator that exchanges a pair of consecutive operations in a critical block and forms a neighborhood has been used in Laarhoven, Aarts and Lenstra (1992) and in Taillard (1994). The transition operator was originally defined by Balas (1969) in his branch and bound approach. Another very powerful transition operator was used in Brucker, Jurisch and Sievers (1994) and in Dell'Amico and Trubian (1993). The transition operator permutes the order of operations in a critical block by moving an operation to the beginning or end of the critical block, thus forming a *CB neighborhood*.

A schedule obtained from S by moving an operation within a block to the front of the block is called a *before candidate*, and a schedule moving an operation to the rear of the block is called an *after candidate*. A set of all before and after candidates $N'^{C}(S)$ may contain infeasible schedules. The CB neighborhood is given as:

$$N^{C}(S) = \{S' \in N'^{C}(S) \mid S' \text{ is a feasible schedule}\}.$$

It has been experimentally shown by Yamada, Rosen and Nakano (1994) that the CB neighborhood is more powerful than the former one.

A schedule's makespan may often be reduced by shifting an operation to left without delaying other jobs. When no such shifting can be applied to a schedule, it is called an *active schedule*. An optimal schedule is clearly active so it is safe and efficient to limit search space to the set of all active schedules. An active schedule is generated by the *GT algorithm* proposed by Giffler and Thompson (1960). The outline is described in Algorithm 2.1. In the algorithm, the *earliest completion time* $EC(O)$ of an operation O means its completion time when processed with highest priority. An active schedule is obtained by repeating the algorithm until all operations are processed. In step 3 , if all possible choices are considered, all active schedules will be generated, but the total number will still be very large.

Algorithm 2.1 GT algorithm

1. Let O^* be an operation with the minimum among the earliest completion times of unscheduled operations and M^* be the machine which processes O^*:
   $EC(O^*) = \min\{EC(O) \mid O : \text{unscheduled}\}$.
2. Assume $j-1$ operations have been processed on M^*. A *conflict set* $C[M^*, j]$ means a set of unscheduled operations on M^* each of whose processing overlaps with O^*.
3. Select an operation O from $C[M^*, j]$ and schedule it as the j-th operation on M^* with its completion time equals to $EC(O)$.

As explained above, a before or after candidate is not necessarily executable. In the following, we propose a new neighborhood similar to CB neighborhood; its element is not only executable, but also active and close to the original. Let S be an active schedule and $B_{k,h,M}$ be a block of S on a machine M, where the front and the rear operations of $B_{k,h,M}$ are the k-th and the h-th operations on M respectively. Let $O_{p,M}$ be an operation in $B_{k,h,M}$ that is the p-th operation on M. Algorithm 2.2 generates an active schedule $S_{M,p,k}$ (or $S_{M,p,h}$) by modifying S such that $O_{p,M}$ at the position p is moved to the position as close to the front position k (or the rear position h) of $B_{k,h,M}$ as possible. Parts of the algorithm are due to Dell'Amico and Trubian (1993). The new neighborhood $AN^C(S)$ is now defined as a set of all $S_{M,p,k}$'s and $S_{M,p,h}$'s over all critical blocks:

$$AN^C(S) = \bigcup_{B_{k,h,M}} \{S' \in \{S_{M,p,k}\}_{k<p<h} \cup \{S_{M,p,h}\}_{k<p<h},\ S' \neq S\}.$$

Algorithm 2.2 Modified GT algorithm generating $S_{M,p,k}$ or $S_{M,p,h}$

1. *Same as step 1 of* Algorithm 2.1.
2. *Same as step 2 of* Algorithm 2.1.
3. **CASE 1:** $S_{M,p,k}$ generation
   - If $k \leq j \leq p$ and $O_{p,M} \in C[M^*, j]$, then select and schedule $O_{p,M}$.
   - Otherwise, select and schedule from $C[M^*, j]$ the earliest operation in S.

   CASE 2: $S_{M,p,h}$ generation
   - If $p \leq j \leq h$ and $C[M^*, j]$ contains an operation other than $O_{p,M}$, then select and schedule from $C[M^*, j] \setminus \{O_{p,M}\}$ the earliest operation in S.
   - If $j = h$ or $C[M^*, j] = \{O_{p,M}\}$, then select and schedule $O_{p,M}$.
   - Otherwise, select and schedule from $C[M^*, j]$ the earliest operation in S.

3. Critical Block Simulated Annealing

Critical block simulated annealing (CBSA) proposed by Yamada, Rosen and Nakano (1994) is a method of searching job shop scheduling space by using simulated annealing with $N^C(S)$ as the neighborhood of a schedule S. CBSA consists of three parts: the main part is the iteration loop shown in Algorithm 3.1; the *warmup* is a reverse annealing process which adaptively determines initial and final temperatures after starting at a sufficiently low temperature; the *reintensification strategy* makes the system jump from the current state to the best one obtained so far if no better solution is found after a large number R of acceptances (R is called the reintensification frequency). In this paper, CBSA is extended to use $AN^C(S)$ instead of $N^C(S)$.

Algorithm 3.1 CBSA main loop

1. Set $S = S_0$, a randomly generated initial schedule with makespan $L(S_0)$, the initial step counter $k = 0$, and $T_{k=0} = T_0$, the initial temperature.
2. Repeat the following steps:
   (a) Pick S' from $AN^C(S)$ randomly.
   (b) Accept S' probabilistically according to the Metropolis Criterion, i.e. choose S' with probability one if $L(S') \leq L(S)$, and with $e^{-(L(S')-L(S))/T_k}$ otherwise.
   (c) If S' is accepted, set $S = S'$.
3. Set $k = k+1$ and decrease the current temperature T_k according to the annealing schedule.
4. If $T_k > T_f$, the final temperature, go to step 2; otherwise stop.

In Algorithm 3.1, if the acceptance probabilities are low for all members in $AN^C(S)$, the system will remain trapped in a local minimum S and it will take a long time to move to a new state. The algorithm may stay in S even after all members are selected in step 2a and evaluated in step 2b. To avoid this, relative acceptance probability $\overline{P}(S_i) = P(S_i)/\sum_{S_j \in AN^C(S)} P(S_j)$ for each $S_i \in AN^C(S)$ will be introduced after all members in $AN^C(S)$ are visited. The member S_i in $AN^C(S)$ is then randomly selected in proportion to $\overline{P}(S_i)$, and the system moves unconditionally to S_i. This modification is effective because the neighborhood size is limited. Therefore, Algorithm 3.1 is modified as in Algorithm 3.2.

Algorithm 3.2 Modified version of CBSA main loop

1. Set $S = S_0$, $k = 0$, and $T_{k=0} = T_0$. Set the number of $AN^C(S)$ members already visited $n = 0$.
2. Repeat the following steps:
   (a) Select S_i from $AN^C(S)$ randomly. Calculate $L(S_i)$ and set $n = n + 1$ if S_i is not yet visited.
   (b) Accept S_i with probability one if $L(S_i) \leq L(S)$, and with $P(S_i) = e^{-(L(S_i)-L(S))/T_k}$ otherwise.
   (c) If S_i is accepted, then set $S = S_i$, $n = 0$.
   Or if $n = N$, select S' from $AN^C(S)$ in proportion to the probability $\overline{P}(S') = P(S')/\sum_{S_j \in AN^C(S)} P(S_j)$, and set $S = S'$, $n = 0$.
3. Set $k = k + 1$ and decrease T_k according to the annealing schedule.
4. If $T_k > T_f$, go to step 2; otherwise stop.

4. CBSA enhanced by Shifting Bottleneck

Shifting bottleneck (SB) proposed by Adams, Balas and Zawack (1988) is a powerful method for solving a JSSP. In the method, a one machine scheduling problem (a relaxation of the original JSSP) is solved for each machine not yet sequenced, and the outcome is used to find a bottleneck machine, a machine of the longest makespan. Every time a new machine has been sequenced, the sequence of each previously sequenced machine is subject to reoptimization. SB consists of two subroutines: the first one (SBI) repeatedly solves one machine scheduling problems; the second one (SBII) builds a partial enumeration tree where each path from the root to a leaf is similar to an application of SBI.

SBI is a constructive method that generates a complete schedule from scratch. Modifying the method is necessary in order to refine a certain complete schedule for improvement. The *BottleRepair* shown in Algorithm 4.2 describes an iterative version of the basic SB. The reoptimization process used here is the same as used in Algorithm 4.1. The basic idea of *BottleRepair* comes from the original paper of SB (1988) where the last α noncritical machines are temporarily removed for the reoptimization.

Algorithm 4.1 SBI

1. Set $S = \emptyset$ and make all machines unsequenced.
2. Solve a one machine scheduling problem for each unsequenced machine. Among the unsequenced machines, find the bottleneck machine and add its schedule to S. Make the machine sequenced.
3. Reoptimize all sequenced machines in S.
4. Go to step 2 unless S is completed; otherwise stop.

As shown in Algorithm 3.2, S_i is selected from $AN^C(S)$ and is probabilistically accepted. *BottleRepair* is applied to S_i only when S_i is rejected. The resulting schedule $S_i{}^*$ is accepted if its makespan is shorter than that of S. To summarize, step **2*b*-1** as defined in Algorithm4.3 is added to Algorithm 3.2 just after step 2b.

Algorithm 4.2 *BottleRepair*: Iterative SB

---

1. A certain complete schedule S is given. Reset all sequences of all the non-critical machines (machines which do not include any part of critical path in S) and make the machines unsequenced.
2. Reoptimize all sequenced machines.
3. Solve a one machine scheduling problem for each unsequenced machine. The machines are ranked by the makespans. The next two steps are applied in the descending order of rank.
4. Solve a one machine scheduling problem and add its schedule to S. Make the machine sequenced.
5. Reoptimize all the sequenced machines.

BottleRepair gives a systematic way to inspect the schedule's critical path and permutes operations again and again by repeatedly solving one machine problems in a deterministic manner. If it generates an improved schedule S' from S, the critical path of S' becomes different from S and the difference is much greater than that between S and its CBSA neighbor. On the other hand, CBSA gives a stochastic more focused local search around the current critical path. The proposed integration of CBSA and SB is expected to have the synergistic effect as: SB gives a long jump to CBSA so that it can omit many time-consuming inferior transitions and CBSA adds stochastic perturbations to SB so that it can escape from the local minima.

Algorithm 4.3 SB enhancement for CBSA

2*b*-1 If S_i is rejected, apply *BottleRepair* to S_i and generate $S_i{}^*$.
Accept $S_i{}^*$ and set $S_i = S_i{}^*$ if $L(S_i{}^*) < L(S)$.

---

5. Experimental Results

5.1. Muth and Thompson's Benchmark

A 10×10 problem (mt10) and 20×5 problem (mt20) formulated by Muth and Thompson (1963) (*MT benchmarks*) are well known benchmark JSSPs. CBSA with and without SB modification was evaluated using these problems. Table 1 shows the results of 20 trials with different random number seeds on a SUN SPARC station 10. All programs are written in the C language.

Results for the mt10 problem using CBSA without SB show that the optimal solutions of $L = 930$ were found in 11 trials. The average cpu time was 3 min. 10 sec., and the fastest was 1 min. 21 sec. Although the solutions of $L = 930$ were found in only half of the trials, the cpu time in successful runs were satisfactorily short. If the temperature is more slowly lowered, though it takes longer, the rate of finding optimal solutions will become higher as in Yamada, Rosen and Nakano (1994). CBSA without SB could not find

Table 1: Comparisons between CBSA and CBSA+SB using MT benchmarks

Prob	$n \times m$	CBSA ($R = 6,000$)				CBSA+SB			
		best	mean	std	BT	best	mean	std	BT
mt10	10×10	930	933.65	4.04	190	930	932.45	3.01	786
mt20	20×5	1178	1179.45	1.94	235	1165	1165.00	0.00	449

std: standard deviation

BT: average cpu time (sec.) to find the best solution

any optimal solution for mt20 problem. In most cases, solutions of $L = 1178$ were found instead of the optimal $L = 1165$.

The results for the mt10 problem using CBSA with SB modification show that the number of trials finding optimum solutions increased slightly, but the average cpu time increased about four times. This fact indicates that CBSA without SB is powerful enough to solve mt10 problem. On the other hand, all 20 trials with SB modification for mt20 problem found the optimal solutions of $L = 1165$ in an average cpu time of 7 min. 29 sec., and 1 min. 22 sec. was the best time. The effect of SB enhancement is obvious from this problem. Reintensification did not work well because the optimal or near optimal solutions were obtained at an early stage of the search.

5.2. Other Benchmarks

Results in the previous section indicate that if CBSA without SB can solve a problem skillfully, applying CBSA+SB has no advantage. However, if CBSA fails to work well, CBSA+SB may improve solution quality and compensate for the extra cpu time needed with SB enhancement. A set of benchmark problems has been established to evaluate different algorithms for JSSPs. Table 2 shows the makespan performances of CBSA+SB and various other algorithms for the ten difficult benchmark JSSPs. All experiments of CBSA+SB runs were done on a HP 730 (HP 730 is about 1.5 times faster than SUN SPARC station 10).

The LB column indicates the theoretical lower bound of the problem if the optimal makespan is unknown. The CBSA+SB column indicates the best makespans found from ten trials. Each trial used a reintensification frequency of $R = 1,000$ and ran for three hours or until the known optimal makespan or lower bound was found. The column headings Aart, Matt, Appl and Tail indicate the best performances of those in Aarts *et al.* (1994), Mattfeld, Kopfer and Bierwirth (1994), Applegate and Cook (1991) and Taillard (1994) respectively.

Table 2: Results of 10 tough JSSPs

Prob	$n \times m$	LB	CBSA+SB				Aart	Matt	Appl	Tail
			best	mean	std.	BT				
abz7	20×15	654	665	671.0	3.92	7814	668	672	668	665
abz8	20×15	635	675	680.0	3.13	8775	670	683	687	676
abz9	20×15	656	686	698.6	7.42	8749	691	703	707	691
la21	15×10	1040	1046	1049.3	3.32	361	1053	1053	1053	1047
la24	15×10	–	935	939.2	1.99	6226	935	938	935	
la25	20×10	–	977	979.3	1.62	4117	983	977	977	
la27	20×10	1235	1235	1242.4	6.15	7805	1249	1236	1269	1240
la29	20×10	1120	1154	1162.4	7.10	5434	1185	1184	1195	1170
la38	15×15	1184	1198	1206.8	4.53	3479	1208	1201	1209	1202
la40	15×15	–	1228	1230.2	2.32	3331	1225	1228	1222	

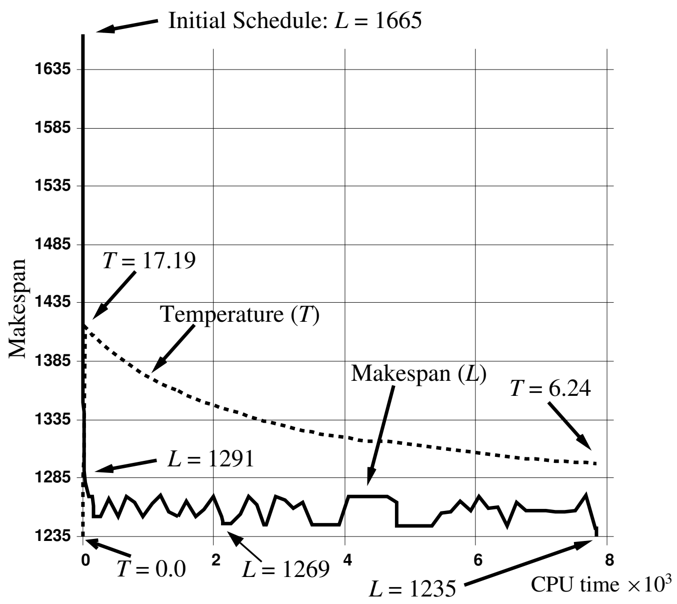

Figure 2: The time evolution of CBSA+SB trial for the la27 problem

Figure 2 shows the time evolution of the makespan (L) and temperature (T) of the best trial of CBSA+SB for the la27 problem. The abscissa shows the cpu time in seconds, and the solid and dotted lines show the makespan and the temperature respectively. Starting from the makespan value $L = 1665$, it rapidly decreases to $L = 1291$ during the first warmup interval. After twelve times of reintensification, it finally reached the optimal value $L = 1235$. In this experiment, 56059 schedules were generated and 8850 schedules accepted. About 25% of the accepted schedules were accepted by *BottleRepair* to add CBSA long jumps and the rest served as stochastic perturbations to *BottleRepair*. The oscillatory behavior is due to reintensification.

Although CBSA+SB outperformed other methods most of the cases in Table 2, the required computational time is much longer. For example, Vaessens, Aarts and Lenstra (1994) reported that Applegate's method in the Appl colum found a schedule of $L = 1269$ in 604.2 sec. on SUN SPARC station ELC which is about 10 times slower than HP 730.

Table 3: An optimal solution of la27 problem

m	Job sequences on each machine																			
1	18	9	4	2	11	3	12	8	10	7	13	15	20	1	6	19	5	14	17	16
2	14	9	8	12	3	17	15	4	10	13	6	7	11	19	1	2	18	16	20	5
3	11	14	3	19	10	4	9	5	6	2	16	17	13	15	7	20	18	12	8	1
4	6	11	10	1	19	3	8	13	15	18	2	12	14	17	4	7	9	16	20	5
5	12	3	5	6	20	13	11	8	1	17	9	19	7	14	2	16	18	15	10	4
6	19	18	11	8	14	7	3	16	2	17	1	10	9	5	13	6	20	4	12	15
7	17	2	13	15	9	8	19	7	6	20	10	18	14	16	5	4	3	1	11	12
8	2	12	16	14	11	10	3	5	19	20	6	7	13	17	8	18	15	1	4	9
9	18	12	3	14	11	10	16	8	19	13	4	6	15	2	17	20	1	9	7	5
10	13	20	18	9	4	5	11	6	3	8	2	19	10	14	1	17	7	15	16	12

m: machine

This is because each CBSA+SB experiment includes a lot of unsuccessful *BottleRepair* trials. But this gap can be filled to some extent by the fact that in the same experiment, CBSA+SB passed a point $L = 1269$ in 164 sec.

6. Conclusion

The proposed method CBSA+SB is an improved CBSA enhanced by integrating with a problem specific method called shifting bottleneck. The performance of CBSA+SB was evaluated using difficult benchmark problems. The results show that for eight problems of ten difficult benchmark problems, CBSA+SB could find schedules better than or equal to the best schedules published so far in the literature, when enough computational time is given. A new solution of $L = 1235$ was found for the la27 problem; it is optimal because the value equals the theoretical lower bound. Further research is necessary to reduce the computational time.

7. References

E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra and N.L.J. Ulder, A computational study of local search algorithms for job shop scheduling, *ORSA J. on Comput.*, 6 (1994) 118.

J. Adams, E. Balas and D. Zawack, The Shifting Bottleneck Procedure for Job Shop Scheduling, *Mgmt. Sci.*, 34 (1988) 391.

D. Applegate and W. Cook, A Computational Study of the Job-Shop Scheduling Problem, *ORSA J. on Comput.*, 3 (1991) 149.

E. Balas, Machine sequencing via disjunctive graphs: an implicit enumeration algorithm, *Oper. Res.*, 17 (1969) 941.

P. Brucker, B. Jurisch and B. Sievers, A Branch & Bound Algorithm for the Job-Shop Scheduling Problem, *Discrete Applied Mathematics*, 49 (1994) 107.

D.C. Mattfeld, H. Kopfer, and C. Bierwirth, Control of Parallel population dynamics by social-like behavior of ga-individuals, In: *Proc. of PPSN 3rd* (Springer-Verlag, 1994) p. 16.

M. Dell'Amico and M. Trubian, Applying tabu search to the job-shop scheduling problem, *Annals of Oper. Res.*, 41 (1993) 231.

B. Giffler and G.L. Thompson, Algorithms for solving production scheduling problems. *Oper. Res.*, 8 (1960) 487.

P.J.M. van Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job Shop Scheduling by Simulated Annealing, *Oper. Res.*, 40 (1992) 113.

J.F. Muth and G.L. Thompson, *Industrial Scheduling*, (Prentice-Hall, New Jersey, 1963).

E.D. Taillard, Parallel taboo search techniques for the job-shop scheduling problem, *ORSA J. on Comput.*, 6 (1994) 108.

R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra, Job shop scheduling by local search, Tech. rept., COSOR 94-05, Eindhoven University of Technology, Dpt. of Math. and CS, NL (1994).

T. Yamada, B.E. Rosen and R. Nakano, A Simulated Annealing Approach to Job Shop Scheduling using Critical Block Transition Operators, In: *Proc. of IEEE ICNN* (IEEE, Florida, 1994).