Comput Optim Appl (2010) 47: 61–76
DOI 10.1007/s10589-008-9206-5

Solving job shop scheduling problems utilizing the properties of backbone and "big valley"

Panos M. Pardalos · Oleg V. Shylo · Alkis Vazacopoulos

Received: 14 March 2007 / Revised: 3 September 2008 / Published online: 20 September 2008

Abstract In this paper, a new metaheuristic for the job shop scheduling problem is proposed. Our approach uses the backbone and "big valley" properties of the job shop scheduling problem. The results of the computational experiments have demonstrated the high efficiency of our approach. New upper bounds have been obtained for many problems.

Keywords Job shop scheduling problem · Makespan · Backbone · Big valley · Metaheuristic

1 Introduction

The goal of this paper is to introduce a new metaheuristic approach for the minimum makespan problem of the job shop scheduling problem (JSP), which uses properties of the search space such as backbone and "big valley" to accelerate the search. The algorithm was proposed not in order to beat previous highly sophisticated procedures, but to investigate the potential of certain search principles (i.e. algorithm with two phases: intensification and diversification, an accumulation of the search

Research partially supported by NSF and AirForce grants.

P.M. Pardalos · O.V. Shylo (✉)
Dept. of Industrial and Systems Engineering, University of Florida, 303 Weil Hall, Gainesville, FL, 32611, USA
e-mail: shylo@ufl.edu

P.M. Pardalos
e-mail: pardalos@ufl.edu

A. Vazacopoulos
Dash Optimization Inc., 560 Sylvan Avenue, Englewood Cliffs, NJ, 07632, USA
e-mail: av@dashoptimization.com

space properties and utilization of those properties in the algorithm dynamics). In our opinion, such principles are crucial for an effective optimization technique and the computational results support this assertion.

The JSP minimizes the completion time needed for processing all jobs J on a set of machines M subject to the following constraints: (i) each job has a predefined processing order through all machines (precedence constraints), and (ii) each machine can process at most one job at a time (resource constraints). If a machine starts processing a job it must finish it without any interruption (non-preemptive). This problem is known to be NP-hard [12]. Exact methods were successfully applied to job shop problems of small dimensions, but for problems with more than 15 machines and 15 jobs high quality solutions cannot be found with reasonable computational effort. For large dimensions there is a need for good approximation algorithms. Local search based methods were successfully applied to such problems. These methods include tabu search methods [15, 23, 26], greedy randomized adaptive search [3], simulated annealing [24], global equilibrium search [19], and guided local search with shifting bottleneck procedure [2]. A detailed review of the methods for the JSP can be found in [9] and [13].

Among these approaches, the tabu search and global equilibrium search method dominate all other approaches. However, the metaheuristic presented in this paper is able to achieve the same level of performance on a wide set of benchmark instances.

2 Mathematical model and notations

Let $M = \{M_1, M_2, \ldots, M_m\}$ be a set of machines and $O = \{\sigma_0, \sigma_1, \ldots, \sigma_{N+1}\}$ denote the set of operations, where σ_0 and σ_{N+1} denote fictitious "start" and "finish" operations. Each operation $\sigma \in O$ has two properties: a machine $M(\sigma) \in M$ that must process it and its processing time $p(\sigma)$ ($p(\sigma_0) = p(\sigma_{N+1}) = 0$). Additionally, let A denote the set of ordered pairs of operations which are tied by the precedence constraints, and let E_k denote the set of all pairs of operations which require machine M_k for their processing ($k = 1, \ldots, m$).

Further, let $s(\sigma)$ denote the start time of the operation σ, and let $s(\sigma_0) = 0$. The problem is to find a starting time for each operation $\sigma \in O$ (schedule) such that

$$\max_{\sigma \in O}[s(\sigma) + p(\sigma)] \tag{1}$$

is minimized subject to

$$s(\sigma) \geq 0, \quad \forall \sigma \in O \tag{2}$$

$$s(\omega) - s(\sigma) \geq p(\sigma), \quad \forall (\sigma, \omega) \in A \tag{3}$$

$$s(\omega) - s(\sigma) \geq p(\sigma) \vee s(\sigma) - s(\omega) \geq p(\omega)$$
$$\forall (\omega, \sigma) \in E_k, k = 1, \ldots, m \tag{4}$$

The value given in (1) is often referred to as the makespan of the schedule represented by starting times of operations from O.

This problem can be described using the disjunctive graph model of Roy and Sussmann [20]. Let $G = (O, A, E)$ denote the disjunctive graph, where O is a set of nodes (a node for each operation), A is a set of directed arcs $((\sigma_0, \sigma) \in A$ and $(\sigma, \sigma_{N+1}) \in A$ for all $\sigma \in O)$, and $E = \bigcup_{k=1}^{m} E_k$ is a set of undirected arcs. The weight of each node is given by the processing time of the corresponding operation. Let S (selection) denote a set of directed arcs which is obtained from E by choosing the direction for each of its arcs. Then let $G_S = (O, A \cup S)$ be a directed graph that corresponds to the selection S. If G_S is an acyclic graph, then it defines a set of feasible schedules for the problem (1)–(4). The best solution from this set can be obtained by setting $s(\sigma)$ equal to the longest path from σ_0 to σ in G_S.

The following notation is useful for our discussion. Let $JP(\sigma)$ $(JS(\sigma))$ denote an operation such that $(JP(\sigma), \sigma) \in A$ $((\sigma, JS(\sigma)) \in A)$. In other words, $JP(\sigma)$ $(JS(\sigma))$ is an operation that belong to the same job and has to be processed directly before (after) σ. Assuming S is a feasible selection, let $MP(\sigma)$ denote an operation such that $(MP(\sigma), \sigma) \in S$, and for any operation $\omega \neq \sigma$ if $(MP(\sigma), \omega) \in S$ then $(\sigma, \omega) \in S$. Let $MS(\sigma)$ denote an operation such that $MP(MS(\sigma)) = \sigma$. In other words, $MP(\sigma)$ $(MS(\sigma))$ is an operation that is processed on the same machine and directly before (after) σ.

Let $q(\sigma)$ be the length of the longest path from σ to σ_{N+1} (tail), and $s(\sigma)$ be the length of the longest path from σ_0 to σ (head). The longest path in G_S is referred to as the critical path, and the operations which belong to this path are called critical operations. Thus, the critical operations possess the property that any delay of a critical operation will result in an delay of the whole schedule. We can decompose any critical path into a set of critical blocks. We say that critical operations σ $(\sigma \neq \sigma_0)$ and ω $(\omega \neq \sigma_{N+1}$ and $s(\sigma) < s(\omega))$ belong to the same critical block if there is a path of length l between them, such that all its arcs belong to S and $s(\sigma) + p(\sigma) + l = s(\omega)$.

3 Implementation of the local search

In contrast to most recent local search based methods for the JSP, our local search is based on the N4 move operator proposed by Grabowski et al. [7] (we use the neighborhood notation introduced in [4]). The N4-neighborhood consists of solutions obtained by moving a critical operation either to the beginning or to the end of its critical block.

In many applications the priority has been given to the highly restricted N5 move operator. The solutions from the N5-neighborhood are obtained by changing the processing order of two consecutive critical operations such that at least one of them is either the first or the last operation of its critical block. The main drawback of such a restriction is that the resulting search space becomes disconnected, i.e. the existence of the path, with respect to the N5, from any arbitrary solution to the optimum is not guaranteed. On the contrary, the N4 move operator induces the connected search space. Additionally, as the N5 move operator is just a restriction of the N4 move operator, it is clear that the latter provides a more thorough search.

Nowicki and Smutnicki proposed an effective method of the makespan calculation for the N5 move operator [17], which significantly decreases the overall time spent on

move evaluations. Based on their approach, we use the similar accelerator for the N4 move operator. The details of such acceleration are simple, but quite cumbersome, thus we refer the reader to [19] for the details. As a result of such acceleration, the move evaluations are performed up to 3 times faster.

In order to provide an even more thorough search, we introduce a procedure, which has some common features with the iterated local search technique [14, 25].

Whenever an operation is moved either to the beginning or the end of the block, the reason that such move does not improve the solution is because of a constraint coming from the job predecessor of some operation which belonged to the same critical block as the moved operation.

Therefore, after identifying such an operation, we push its job predecessors back in the processing order to remove this specific constraint. After the transformation is finished, the local search starts from the obtained solution (see Fig. 1). This procedure was discussed in [19] and is used in our studies as an enhancement of the local search. The investigation of its impact on the algorithm performance was provided in [19].

Whenever two solutions have the same makespan value, we consider the total processing time of their critical operations. The solution with smaller total processing time is considered an improving solution. This rule allows us to take into account cases where we encounter multiple critical paths (Fig. 1: line 27).

4 Description of an algorithm

The algorithm described below uses ideas and concepts introduced by the simulated annealing technique [11] and the global equilibrium search method [21].

The general framework of the algorithm is presented in the Fig. 2. This algorithm will be referred to as Distance Based Search (DBS). The main cycle of the DBS is performed until some stopping criteria is satisfied. In our implementation we stopped DBS after a certain amount of computational time. The main cycle consists of a series of temperature cycles (Fig. 2: line 4). During a temperature cycle the DBS generates a set of random solutions (Fig. 2: line 16). These random solutions are generated based on the rejection probability value, that changes during the cycle (Fig. 2: line 6). Instead of generating the solutions from scratch, a number of simple moves is applied using the N4 and the N1 move operators to the previously encountered solution (the N1-neighborhood consists of solutions obtained by swapping two consecutive critical operations). The acceptance or rejection of a certain move is based on the solutions from the set P (the pool of solutions). If the move leads to a solution that is further from P than the current solution, then it is rejected with probability given by a rejection value. In our studies, 40 temperature cycles were used and 100 solutions were generated at each temperature stage.

After the new solution is generated, we apply local search and improvement procedures (see Fig. 1) in order to find better solutions in its vicinity (Fig. 2: line 17). The best solution found by the improvement procedures is used to update all memory structures. Lines 8–14 and 24–29 (Fig. 2) allow us to intensify the search in the vicinity of solutions from P (in our tests we used the pool of size 5). Whenever the

Input: x—a local minimum
Let $\mathcal{M} = \mathrm{N4}(x)$ (Grabowski's neighborhood)
while $\mathcal{M} \neq \emptyset$ **do**
  choose randomly $y \in \mathcal{M}$
  $\mathcal{M} = \mathcal{M} - y$
  let $\mathcal{B}$ be a set of operations σ, ω for which
    $y(\sigma, \omega) \neq x(\sigma, \omega)$
  find an operation $\sigma \in \mathcal{B}$ such that
    $end(JP(\sigma)) = head(\sigma)$
  if $\sigma \neq \emptyset$ **then**
    $\omega = JP(\sigma)$
    while $\sigma \neq \omega$ **do**
      if $head(JS(\omega)) \neq end(\omega)$ **then**
        $\omega = JS(\omega)$
      else if $JP(\omega) \neq \emptyset$ and $end(JP(\omega)) = head(\omega)$ **then**
        $\omega = JP(\omega)$
      else if $MP(\omega) \neq \emptyset$ **then**
        Switch $MP(\omega)$ and ω in y
        if makespan of the resulting solution is worse **then**
          break
        end if
      else
        break
      end if
    end while
  end if
  $y = localSearch\mathrm{N4}(y)$ {*Starting in* y}
  if $f(y) < f(x)$ or $f(y) = f(x)$ and y is an improving solution **then**
    RETURN y {*Improvement!*}
  end if
end while
RETURN x {*No improvement*}

Fig. 1 Improvement procedure

temperature is larger than T', instead of using the whole set P' to determine the dynamics of the transformation procedure, we use a single solution from the set P'. At this stage the behavior of the algorithm is similar to the path-relinking procedure.

4.1 "Big valleys" and backbones

The theoretical and empirical aspects of the backbone properties are getting more attention in literature [5, 18, 22]. Our algorithm uses information about the backbones to organize a more effective search.

```
Initialize algorithm's parameters
while Stopping criteria is not satisfied do
   x = GenerateRandomSolution()
   for cycle = 0 to maxnfail do
      for temp = 0 to maximal temperature do
         Calculate rejection probabilities
         Define the percentage of edges to fix, and define B^f
         if temp > T' then
            Choose random solution S from P''
            P = {S}
            P'' = P'' − {S}
         else
            P = P'
         end if
         for gen = 1 to number of generations do
            x = TransformSolution(x, B^f, P, temp, maxmoves)
            x = ImproveSolution(x)
            if f(x) < f* then
               cycle = 0
               f* = f(x)
            end if
            Include x to the pool of solution P', if eligible
         end for
         if temp > T' AND |P''| > 0 then
            temp = temp − 1 (repeat with the same temperature)
         end if
         if temp > T' AND |P''| = 0 then
            |P'' = P'|
         end if
      end for
   end for
   f* = ∞
end while
RETURN Best solution found
```

Fig. 2 Pseudo-code of the algorithm

For the JSP, the backbone is a set B of directed arcs which is obtained by fixing the orientation of some arcs from E, such that for any globally optimal selection S: $S \cap B = B$. In other words, it is the set of solution features common to all global optima.

The selection is called ρ-optimal, if the corresponding makespan is less than or equal to the ρ percentage of the optimal makespan value. We can define the ρ-backbone $B(\rho)$ as a set of directed arcs which is obtained by fixing the ori-

entation of some arcs from E, such that for any ρ-optimal selection S, we have $S \cap B(\rho) = B(\rho)$.

In our algorithm we use another interesting property of the JSP: the distribution of local optima has the structure, which is referred to as "big valley" (i.e. the solutions of high quality tend to be clustered together with respect to the Hamming distance between solutions). This property has been successfully used in the algorithm of Nowicki and Smutnicki [17]. In [22] the authors provide a formal definition of this concept.

4.2 Use of backbone property

Let Ω^* be some subset of feasible solutions, and let S^* be the best solution from this set. Then Ω^* determines the approximation of backbone:

$$B^*(\rho) = \{\cap S | S : f(S) < \rho \cdot f(S^*), S \in \Omega^*\}$$

One can set Ω^* equal to the set of local optima encountered by a single run of the algorithm. This approximation is used in our algorithm to fix a certain number of edge orientations.

Through computational experiments we found that such limitation of the search space allows us to increase the efficiency of the search.

One of the parameters used in our studies is q, which is the percentage of edges to fix. Using this value, we define a set of fixed edges. Formally, this procedure consists of the following steps:

1. Find ρ_1, ρ_2 such that $|B^*(\rho_1)| \leq q \cdot |S| \leq |B^*(\rho_2)|$,
   $\rho_1 \leq \rho$ for any ρ such that $|B^*(\rho)| \leq q \cdot |S|$,
   $\rho_2 \geq \rho$ for any ρ such that $|B^*(\rho)| \geq q \cdot |S|$.
2. Set $B^f = B^*(\rho_1)$.
3. Select randomly $\lceil q \cdot |S| - |B^*(\rho_1)| \rceil$ elements from the set given by $B^*(\rho_2) - B^*(\rho_1)$ and add them to B^f.

As the result of the above steps, the set B^f contains $\lceil q \cdot |S| \rceil$ edge orientations to be fixed.

4.3 Use of the "big valley" property

In order to use the "big valley" property of the JSP, we use a mechanism similar to the path-relinking procedure [1].

In our approach we use the value of the Hamming distance between some subset of the solutions as the main decision criteria, and the cost of the solutions is used to define such a subset (given two selections S_1 and S_2 representing two schedules, the Hamming distance is defined as $|S_1 \backslash S_2|$). In this paper we refer to this subset as the *pool of solutions*.

The pool of solutions is used to keep track of the best solution encountered by the algorithm and to intensify search in its vicinity. The main criteria for an inclusion into the pool is the solution quality and the distance to other solutions in the pool. In

contrast to the elite set in path-relinking applications, the pool of solutions is more volatile, i.e. it changes more frequently.

In order to determine whether to include a solution into the pool or not, we introduce the notion of the quality of the pool, which is given by a *quality function*. If the size of the pool is less than a maximal size, then all solutions are inserted into the pool. When the pool reaches its maximal size, then we try to substitute each solution in the pool with the new solution, and if the quality of the pool increases after such a substitution, then the pool is changed (i.e. the new solution substitutes the solution from the pool).

Let P denote a set of solutions which belong to the pool. The quality function is defined as:

$$Q(P) = \sum_{S'' \in P} \min\{|S'' \backslash S'| : S' \in P, S' \neq S''\}$$

A pool which consists of more diverse solutions has a higher quality function value. Thus, the choice of this function was made in order to encourage the algorithm to keep the diverse set of solutions in the pool.

In order to avoid situations in which the pool is populated with low-quality solutions, we use two threshold values θ_1 and θ_2, which are changed dynamically. A solution can enter the pool only if its makespan value is smaller than θ_1. A parameter θ_2 is used to identify the solutions which should be removed from the pool, i.e. after each update of the threshold values, the solutions with makespan larger than θ_2 are excluded from the pool.

These threshold values are reset every three temperature cycles. In our implementation, θ_1 is set to $1.01 * \mathrm{CMax}_{best}$ and θ_2 is set to $1.015 * \mathrm{CMax}_{best}$, where CMax_{best} is the best makespan value found by the algorithm since the last reset of the threshold.

4.4 Rejection probabilities

In order to control the generation of new solutions, we use the so-called temperature values, which are integers from the interval [0, *maximal temperature*]. At the first temperature stage, the value of the temperature is zero, at the beginning of every subsequent stage the temperature value increases by one. When the value of the temperature reaches the *maximal value*, the temperature cycle is finished. The rejection probability for a temperature t is given by:

$$\Pr\{rejection\}(t) = 1 - 0.5 \exp(-alpha * t)$$

The parameter *alpha* controls the speed of the convergence. We use a simple rule to adjust it in order to make a smooth convergence for any problem solved. If during the previous temperature stage, more than 10% of the solutions were the same as the previous solutions, then the new value of *alpha* is set to $0.9 * alpha$. On the other hand, if the algorithm was not able to find the best solution from the pool during the last five stages of the temperature cycle, then the new value of *alpha* is set to $1.1 * alpha$. In order to determine, whether or not the solution was previously visited, we use the hash table with capacity to store 10000 keys. Each key is a sum of the weighted elements of the solution vector (in a binary form). All entries are removed from the hash table after every three temperature cycles.

Input: S—initial solution, B^f—set of fixed edges, P—current pool of solutions, t—current temperature, *maxmoves*—maximal # of moves
Function:
1: $N = \mathrm{N4}(S) \cup \mathrm{N1}(S)$
2: $nmoves = 0$
3: **while** $|N| \neq 0$ and $nmoves < maxmoves$ **do**
4: Choose randomly $S' \in N$
5: $N = N - S'$
6: $accept = false$
7: **if** $|S' \cap B^f| \geq |S \cap B^f|$ **then**
8: $accept = true$
9: **end if**
10: **if** $\min\{|S' - S^e| : S^e \in P\} > \min\{|S - S^e| : S^e \in P\}$ **then**
11: $rand$ =uniform(0,1)
12: **if** $rand > p(t)$ **then**
13: $accept = true$
14: **end if**
15: **else**
16: $accept = true$
17: **end if**
18: **if** $accept = true$ **then**
19: $nmoves = nmoves + 1$
20: $S = S'$
21: $N = \mathrm{N4}(S) \cup \mathrm{N1}(S)$
22: **end if**
23: **end while**

Fig. 3 Transformation procedure

4.5 Transformation procedure

The transformation procedure allows the algorithm to escape from the local optima by subsequently applying the N4 and the N1 move operators. Additionally, it allows us to move towards solutions with some specific properties. As discussed earlier, we are exploiting the properties of backbone and "big valley" to organize an effective search. To do this, we simply encourage moves that lead to the solutions with edge orientations common to backbone and which are close to the solutions from the pool. The pseudo-code of this procedure is presented in Fig. 3.

5 Computational results

The algorithm described in this paper was implemented in C language. All computational experiments were conducted using a personal computer Pentium 2.8 GHz with 512 Mb of RAM.

The parameters of the algorithm were the same for all computational experiments: *maxnfail* = 3, *maximal temperature* $K = 40$, $T' = 35$, *number of generations* $ngen = 100$, initial value of α is set to 2.0. There was no sophisticated tuning involved in the definition of the algorithm's parameters. The choice of these parameters was made in such a way that algorithm returns high quality solution within a reasonable ammount of time (i.e. one cycle in 60 seconds for "small" sized problems). The choice of initial α is not significant, since the DBS updates this parameter based on the search results. The value of parameter $maxmoves$ is set to the number of jobs of the problem solved, the main criteria for such choice was to allow an escape from the local minimum when the trasformation is applied. The proposed algorithm does not involve a large number of parameters and those parameter do not require any sophisticated tuning procedures. The specific choice of the parameters is not important for our investigation. We wanted to show that the techniques that involve the ideas introduced by the SA can be efficiently used in scheduling (without complicated tuning procedures) as long as the proper search organization is used (i.e. an intensification stage and a diversification stage, an accumulation of the search space properties and utilization of those properties in the algorithm dynamics).

The proposed algorithm has been tested on Taillard's benchmark problems taken from the OR-library. This set contains 80 problems denoted by (TA1–TA80) due to Taillard [23]. Optimal solutions are known for 47 problems from this class. The problems TA51–TA80 are the easiest problems, therefore we limited our testing to the problems TA01–TA50.

The other set of problems we used for computational experiments was proposed in [6]. This set contains 80 problems, denoted as DMU1–DMU80. The optimal solutions are only known for 22 of these problems.

The computational results are given in Tables 1 and 2 below. For the problems with an unknown optimal solution, we provide its lower bound (LB) and upper bound (UB), taken from [8] and [10]. "Alg5" corresponds to the DBS algorithm with the pool of size 5. There was no fixed edges for these version of the algorithm. ($B^f = \emptyset$). As for the algorithm "AlgFix", we fixed 15% of edges after 5000 seconds of running time. For every algorithm we set up a time limit of 10000 seconds. The results for shorter running times are provided for problems TA01–Ta10 (see 5.1 Comparison with the algorithm of Nowicki and Smutnicki). Most of the algorithms proposed for the job shop scheduling have a very significant drawback. The increase of computational effort used to solve the problem (i.e. long computational times: sometime weeks of computations) do not provide any improvement in terms of solution quality. Thus, the long time runs where reported in order to provide an evidence of the algorithm's capability of avoiding stagnation in low quality solutions (the reported new UBs prove this assertion). In our opinion, the ability of an approximate algorithm to provide better solution when long run times are involved is very crucial, especially when parallelization of such techniques is considered.

The data presented in the tables corresponds to the best solutions found by the algorithm. The "GES" column corresponds to the result obtained by the GES based algorithm in [19]. The best solutions found by the algorithm described in the current paper were put into brackets (i.e. DMU6—new upper bound found by DMS is 3244). These solutions are available at [10]. For a majority of other test instances the algorithm finds the best known solutions. It is interesting to note the performance of the

Table 1 Results for TA1–TA50

Prob	$\|J\| \times \|M\|$	(LB, UB)	Alg5	AlgFix	GES
ta01	15×15	1231	1231	1231	1231
ta02	15×15	1244	1244	1244	1244
ta03	15×15	1218	1218	1218	1218
ta04	15×15	1175	1175	1175	1175
ta05	15×15	1224	1224	1224	1224
ta06	15×15	1238	1238	1238	1238
ta07	15×15	1227	1228	1228	1228
ta08	15×15	1217	1217	1217	1217
ta09	15×15	1274	1274	1274	1274
ta10	15×15	1241	1241	1241	1241
ta11	20×15	(1323, 1357)	1359	1358	1357
ta12	20×15	(1351, 1367)	1367	1367	1367
ta13	20×15	(1282, 1342)	1342	1342	1344
ta14	20×15	1345	1345	1345	1345
ta15	20×15	(1304, 1339)	1339	1339	1339
ta16	20×15	(1302, 1360)	1360	1360	1360
ta17	20×15	1462	1473	1473	1469
ta18	20×15	(1369, 1396)	1396	1396	1401
ta19	20×15	(1297, 1332)	1332	1332	1332
ta20	20×15	(1318, 1348)	1348	1348	1348
ta21	20×20	(1539, 1644)	1647	[1643]	1647
ta22	20×20	(1511, 1600)	1600	1600	1602
ta23	20×20	(1472, 1557)	1557	1557	1558
ta24	20×20	(1602, 1646)	1647	1646	1653
ta25	20×20	(1504, 1595)	1595	1595	1596
ta26	20×20	(1539, 1645)	1649	1647	1647
ta27	20×20	(1616, 1680)	1685	1686	1685
ta28	20×20	(1591, 1603)	1603	1613	1614
ta29	20×20	(1514, 1625)	1625	1625	1625
ta30	20×20	(1472, 1584)	1584	1584	1584
ta31	30×15	1764	1766	1766	1764
ta32	30×15	(1774, 1793)	1803	[1790]	1793
ta33	30×15	(1778, 1791)	1798	1791	1799
ta34	30×15	(1828, 1829)	1832	1832	1832
ta35	30×15	2007	2007	2007	2007
ta36	30×15	1819	1819	1819	1819
ta37	30×15	1771	1778	1784	1779
ta38	30×15	1673	1677	1673	1673
ta39	30×15	1795	1798	1795	1795
ta40	30×15	(1631, 1674)	1682	1979	1680
ta41	30×20	(1859, 2014)	2036	2022	2022
ta42	30×20	(1867, 1949)	1963	1953	1956

Table 1 (*Continued*)

Prob	$\|J\| \times \|M\|$	(LB, UB)	Alg5	AlgFix	GES
ta43	30×20	(1809, 1858)	1868	1869	1870
ta44	30×20	(1927, 1983)	1990	1992	1991
ta45	30×20	(1997, 2000)	2005	2000	2004
ta46	30×20	(1940, 2011)	2023	2011	2011
ta47	30×20	(1789, 1900)	1911	1902	1903
ta48	30×20	(1912, 1949)	1964	1962	1962
ta49	30×20	(1915, 1967)	1974	1974	1969
ta50	30×20	(1807, 1926)	1927	1927	1931

algorithm "AlgFix" on the larger and "tougher" instances from the set DMU. As it can be seen from Table 2, "AlgFix" obtained better solutions with respect to "Alg5" in 23 cases, and "Alg5" was able to get better solution in 18 cases. Therefore, the idea of fixing the orientations of some edges in order to intensify the local search in the vicinity of good solutions, appears to be promising for the larger problem instances, but not very effective for small sized instances.

5.1 Comparison with the algorithm of Nowicki and Smutnicki

To our knowledge, one of the best algorithms to date for job shop scheduling was proposed by Nowicki and Smutnicki [16, 17]. One type of computational results given in their paper is based on average performance on a subset of equally sized problems.

A test on the problems Ta01–Ta10 was performed to provide a comparison of short running times. We run Alg5 for 100 seconds on each problem from this set. For every problem we calculated the following value, $\mathrm{RE}[i] = 100\%(\mathrm{CMax}[i] - \mathrm{CMax}^*[i])/\mathrm{CMax}^*[i]$, where $\mathrm{CMax}[i]$ is the best makespan value obtained by our algorithm for the ith problem in the set, $\mathrm{CMax}^*[i]$—is the optimal makespan value for ith problem. For the problems Ta01–Ta10, our algorithm provides an average of $\mathrm{RE}[i]$ equal to 0.08%, and the average time to the best solution is equal to 24.6 seconds. These values are comparable to the values provided in the paper of Nowicki and Smutnicki [16]: 0.11% and 26 seconds, respectively. Hence, one can see, that the relationship between running times versus the quality of solutions is comparable.

The other data, provided in [16, 17] are the new upper bounds obtained by the tabu search algorithm. These bounds were obtained during numerous time consuming experiments with different parameters of the tabu search algorithm. Therefore, this data does not represent the results for a single set of parameters, and there is no data available on the computational times spent to acquire those bounds. Nevertheless, our approach allowed us to improve most of these upper bounds in a single run of our algorithm. For the problems DMU1–DMU8, there are 60 new upper bounds reported in [16], but the single run of Alg5 together with single run of AlgFix was able to improve 44 bounds, and for 3 problems we found the same bounds. The fact that all of our improvements were found in a two runs of algorithm with no tuning for each particular instance, suggests the high effectiveness of the proposed approach.

Table 2 Results for DMU1–DMU80

Prob	$\|J\| \times \|M\|$	(LB, UB)	Alg5	AlgFix
DMU1	20×15	(2501, 2563)	2563	2563
DMU2	20×15	(2651, 2706)	2706	2706
DMU3	20×15	2731	2731	2731
DMU4	20×15	(2601, 2669)	2669	2669
DMU5	20×15	2749	2749	2749
DMU6	20×20	(2834, 3250)	[3244]	[3244]
DMU7	20×20	(2677, 3053)	[3046]	[3046]
DMU8	20×20	(2901, 3197)	[3188]	[3188]
DMU9	20×20	(2739, 3092)	3092	3096
DMU10	20×20	(2716, 2984)	2984	2984
DMU11	30×15	(3395, 3453)	3457	3455
DMU12	30×15	(3481, 3518)	3519	3522
DMU13	30×15	(3681, 3697)	[3683]	3687
DMU14	30×15	3394	3394	3394
DMU15	30×15	(3332, 3343)	3343	3343
DMU16	30×20	(3726, 3781)	3781	[3772]
DMU17	30×20	(3697, 3848)	3841	[3836]
DMU18	30×20	(3844, 3849)	3850	3852
DMU19	30×20	(3650, 3807)	[3775]	[3775]
DMU20	30×20	(3604, 3739)	3731	[3712]
DMU21	40×15	4380	4380	4380
DMU22	40×15	4725	4725	4725
DMU23	40×15	4668	4668	4668
DMU24	40×15	4648	4648	4648
DMU25	40×15	4164	4164	4164
DMU26	40×20	(4647, 4667)	4688	4688
DMU27	40×20	4848	4848	4848
DMU28	40×20	4692	4692	4692
DMU29	40×20	4691	4691	4691
DMU30	40×20	4732	4741	4749
DMU31	50×15	5640	5640	5640
DMU32	50×15	5927	5927	5927
DMU33	50×15	5728	5728	5728
DMU34	50×15	5385	5385	5385
DMU35	50×15	5635	5635	5635
DMU36	50×20	5621	5621	5621
DMU37	50×20	5851	5851	5851
DMU38	50×20	5713	5713	5713
DMU39	50×20	5747	5747	5747
DMU40	50×20	5577	5577	5577

Table 2 (*Continued*)

Prob	$\|J\| \times \|M\|$	(LB, UB)	Alg5	AlgFix
DMU41	20×15	(2839, 3267)	[3264]	3278
DMU42	20×15	(3066, 3401)	3404	3412
DMU43	20×15	(3121, 3443)	3450	3450
DMU44	20×15	(3112, 3489)	3489	3489
DMU45	20×15	(2930, 3273)	3273	3273
DMU46	20×20	(3424, 4099)	[4043]	4071
DMU47	20×20	(3353, 3972)	3953	[3950]
DMU48	20×20	(3317, 3810)	[3808]	3813
DMU49	20×20	(3369, 3754)	[3724]	3725
DMU50	20×20	(3379, 3768)	[3737]	3742
DMU51	30×15	(3839, 4202)	4216	[4202]
DMU52	30×15	(4012, 4353)	4357	[4353]
DMU53	30×15	(4108, 4419)	4424	[4419]
DMU54	30×15	(4165, 4424)	[4413]	[4413]
DMU55	30×15	(4099, 4331)	[4303]	4321
DMU56	30×20	(4366, 5049)	5017	[4985]
DMU57	30×20	(4182, 4779)	4727	[4709]
DMU58	30×20	(4214, 4829)	[4787]	[4787]
DMU59	30×20	(4199, 4694)	[4638]	[4638]
DMU60	30×20	(4259, 4888)	[4827]	[4827]
DMU61	40×15	(4886, 5293)	5310	5310
DMU62	40×15	(5004, 5342)	5358	[5330]
DMU63	40×15	(5049, 5431)	5467	5431
DMU64	40×15	(5130, 5367)	5406	5385
DMU65	40×15	(5072, 5269)	[5242]	5322
DMU66	40×20	(5357, 5902)	[5864]	5886
DMU67	40×20	(5484, 6012)	5967	[5938]
DMU68	40×20	(5423, 5934)	5861	[5840]
DMU69	40×20	(5419, 5891)	5882	[5868]
DMU70	40×20	(5492, 6072)	[6023]	6028
DMU71	50×15	(6050, 6302)	6452	6437
DMU72	50×15	(6223, 6571)	6624	6604
DMU73	50×15	(5935, 6283)	6361	6343
DMU74	50×15	(6015, 6376)	6432	6467
DMU75	50×15	(6010, 6380)	[6379]	6397
DMU76	50×20	(6329, 6974)	7021	6975
DMU77	50×20	(6399, 6930)	6940	6949
DMU78	50×20	(6508, 6962)	6911	[6928]
DMU79	50×20	(6593, 7158)	7091	[7083]
DMU80	50×20	(6435, 6824)	6880	6861

6 Concluding remarks

The goal of our research was to show that techniques similar to simulated annealing can be efficiently used for obtaining approximate solutions for the JSP. The proposed algorithm was able to obtain new upper bounds for Taillard's and Demirkol's instances, which are considered one of the most difficult test sets for the JSP.

The Job Shop Scheduling problem (JSP) is discussed in the present paper. The publicly available instances of the JSP exhibit some specific features of the distribution of the high-quality solutions in the search space (the backbone property and the "big valley" property). The goal of this paper was to show that these properties can be effectively used to design the approximate algorithms for the JSP. The dynamics of the proposed approach are based on the notion of Hamming distance between some subset of locally optimal solutions. The computational results revealed the high effectiveness of the proposed technique both in terms of quality and computational time, when comparing with the best techniques known up-to-date. Several new upper bounds were obtained for the benchmark instances discussed in the literature.

References

1. Aiex, R., Binato, S., Resende, M.: Parallel grasp with path-relinking for job shop scheduling. Parallel Comput. **29**, 393–430 (2003)
2. Balas, E., Vazacopoulos, A.: Guided local search with shifting bottleneck for job shop scheduling. Manag. Sci. **44**, 262–275 (1998)
3. Binato, S., Hery, W., Loewenstern, D., Resende, M.: A GRASP for job shop scheduling. In: Essays and Surveys on Metaheuristics, pp. 59–79. Kluwer Academic, Dordrecht (2001)
4. Blazewicz, J., Domschke, W., Pesch, E.: The job-shop scheduling problem: Conventional and new solution techniques. Eur. J. Oper. Res. **93**(1), 1–33 (1996)
5. Darwen, P.J.: Looking for the big valley in the fitness landscape of single machine scheduling with batching, precedence constraints, and sequence-dependent setup times. In: 5th Australasia-Japan Joint Workshop University of Otago, Dunedin, New Zealand, 19–21 November 2001
6. Demirkol, E., Mehta, S., Uzsloy, R.: Benchmarks for job shop scheduling problems. Eur. J. Oper. Res. **109**, 137–141 (1997)
7. Grabowski, J., Nowicki, E., Smutnicki, C.: Block algorithm for scheduling of operations in job-shop system. Prz. Stat. **35**, 67–80 (1988) (in Polish)
8. Internet: Home page Eric Taillard http://www.eivd.ch/ina/collaborateurs/etd/default.htm (2008). Accessed 22 January 2008
9. Jain, A., Meeran, S.: Deterministic job shop scheduling: Past, present and future. Eur. J. Oper. Res. **113**, 390–434 (1999)
10. Internet: Job Shop Scheduling webpage http://plaza.ufl.edu/shylo/jobshopinfo.html (2008). Accessed 22 January 2008
11. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
12. Lawler, E.L., Lenstra, J.-K., Rinnooy Kan, A.: Recent Developments in Deterministic Sequencing and Scheduling, pp. 35–73. Reidel, Dordrecht (1982)
13. Lee, C.-Y., Pinedo, M.: Optimization and heuristic scheduling. In: Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Applied Optimization, pp. 569–584. Oxford University Press, London (2002)
14. Lourenco, H., Martin, O., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics. Kluwer Academic, Dordrecht (2003)
15. Nowicki, E., Smutnicki, C.: A fast tabu search algorithm for the job-shop problem. Manag. Sci. **42**(6), 797–813 (1996)
16. Nowicki, E., Smutnicki, C.: An advanced tabu search algorithm for the job shop problem. J. Sched. **8**(2), 145–159 (2005)

17. Nowicki, E., Smutnicki, C.: Some new ideas in TS for job shop scheduling. In: Operations Research/Computer Science Interfaces Series, vol. 30, pp. 165–190. Springer, Berlin (2005). Part II
18. Nowicki, E., Smutnicki, C.: Some aspects of scatter search in the flow-shop problem. EJOR **169**(2), 654–666 (2006)
19. Pardalos, P., Shylo, O.: An algorithm for the job shop scheduling problem based on global equilibrium search techniques. Comput. Manag. Sci. **3**(4), 331–348 (2006)
20. Roy, B., Sussman, B.: Les problèm d'ordonnancement avec contraintes disjonctives. Note DS9 bis, SEMA, Paris (1964) (in French)
21. Shylo, V.: A global equilibrium search method. Kybern. Syst. Anal. **1**, 74–80 (1999) (in Russian)
22. Streeter, J., Smith, S.: How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. J. Artif. Intell. Res. **26**, 247–287 (2006)
23. Taillard, E.: Benchmarks for basic scheduling problems. Eur. J. Oper. Res. **64**, 278–285 (1993)
24. Van Laarhoven, P., Aarts, E., Lenstra, J.: Job shop scheduling by simulated annealing. Oper. Res. **40**, 113–125 (1992)
25. Watson, J., Howe, A., Whitley, L.: An analysis of iterated local search for job-shop scheduling. In: Fifth Metaheuristics International Conference (MIC 2003), September 2003
26. Watson, J.-P., Beck, J., Howe, A., Whitley, L.: Problem difficulty for tabu search in job-shop scheduling. Artif. Intell. **143**(2), 189–217 (2003)