International Conference on Computational Science, ICCS 2010

The new golf neighborhood for the flexible job shop problem

Wojciech Bożejko, Mariusz Uchoński, Mieczysław Wodecki

Abstract

In this paper we propose a new neighborhood for the flexible job shop problem. A main idea of the proposed neighborhood is to execute a 'long shot' of an operation from the current operation’s machine to another machine of the same type, and then to make a small move by using a local optimization algorithm without changing operations-to-machines assignment. We call this method 'the golf neighborhood'. Computational experiments executed on the benchmark instances from the literature show the efficiency of this solution.

Keywords: scheduling, flexible job shop, metaheuristics

1. Introduction

We consider here the flexible job shop problem (FJSP). The FJSP is a generalization of the classic job shop problem as well as belongs to the strongly NP-hard class. Exact algorithms based on a disjunctive graph representation of the solution have been developed (see Pinedo [22]) but they are not effective for instances with more than 20 jobs and 10 machines. However, many approximate algorithms, mainly metaheuristic, have been proposed. Hurink [15] developed the tabu search method for this problem. Also Dauzère-Pérès and Pauli [9] used the tabu search approach extending the disjunctive graph representation for the classic job shop problem to take into consideration assigning operations to machines. Mastrolilli and Gambardella [18] proposed a tabu search procedure with effective neighborhood functions for the FJSP. Many authors have proposed a method of assigning operations to machines and then determined sequence of operations on each machines. Such an approach is followed by Brandimarte [7] and Pauli [20]. These authors solved the assignment problem (i.e. using dispatching rules) and next used metaheuristics to solve the job shop problem. Also genetic approaches have been adopted to solve the FJSP. Recent works are these of Jia et al. [16], Ho and Tay [14], Kacem et al. [17] and Pezzella et al. [21]. Gao et al. [10] proposed the hybrid genetic and variable neighborhood descent algorithm for this problem.
2. Flexible job shop problem

The flexible job shop problem (FJSP), also called the general job shop problem with parallel machines, can be formulated as follows. Let \( \mathcal{J} = \{1, 2, \ldots, n\} \) be a set of jobs which have to be executed on machines from the set \( \mathcal{M} = \{1, 2, \ldots, m\} \). There exists a partition of the set of machines into types, i.e. subsets of machines with the same functional properties. A job constitutes a sequence of some operations. Each operation has to be executed on an adequate type of machine (nest) within a fixed time. The problem consists in the jobs allocation to machines from the adequate type and the schedule of jobs execution determination on each machine to minimize the total jobs finishing time. The following constrains have to be fulfilled:

(i) each job has to be executed on only one machine of a determined type in each moment of time,
(ii) machines must not execute more than one job in each moment of time,
(iii) there are no idle times (i.e. the job execution must not be broken),
(iv) the technological order has to be obeyed.

Let \( \mathcal{O} = \{1, 2, \ldots, o\} \) be the set of all operations. This set can be partitioned into sequences which correspond to jobs where the job \( j \in \mathcal{J} \) is a sequence of \( o_j \) operations which have to be executed in an order on dedicated machines (i.e. in so-called technological order). Operations are indexed by numbers \( (l_{j-1} + 1, \ldots, l_{j-1} + o_j) \) where \( l_j = \sum_{i=1}^{j-1} o_i \) is the number of operations of the first \( j \) jobs, \( j = 1, 2, \ldots, n \), where \( l_0 = 0 \) and \( o = \sum_{i=1}^{n} o_i \).

The set of machines \( \mathcal{M} = \{1, 2, \ldots, m\} \) can be partitioned into \( q \) subsets of the same type (nests) where \( i \)-th (\( i = 1, 2, \ldots, q \)) type \( \mathcal{M}^i \) includes \( m_i \) machines which are indexed by numbers \( (t_{i-1} + 1, \ldots, t_{i-1} + m_i) \), where \( t_i = \sum_{j=1}^{i} m_j \) is the number of machines in the first \( i \) types, \( i = 1, 2, \ldots, q \), where \( t_0 = 0 \) and \( m = \sum_{j=1}^{q} m_j \).

An operation \( v \in \mathcal{O} \) has to be executed on the machines type \( \mu(v) \), i.e. on one of the machines from the set (nest) \( \mathcal{M}^{\mu(v)} \) in the time \( p_{v,j} \) where \( j \in \mathcal{M}^{\mu(v)} \).

Let \( \mathcal{O}^k = \{v \in \mathcal{O} : \mu(v) = k\} \) be a set of operations executed in the \( k \)-th nest (\( k = 1, 2, \ldots, q \)). A sequence of operations sets

\[
\mathcal{Q} = [\mathcal{Q}^1, \mathcal{Q}^2, \ldots, \mathcal{Q}^m],
\]

such that for each \( k = 1, 2, \ldots, q \)

\[
\mathcal{O}^k = \bigcup_{i=0, i+1}^{t_{i-1}} \mathcal{Q}^i \text{ and } \mathcal{Q}^i \cap \mathcal{Q}^j = \emptyset, \text{ } i \neq j, \text{ } i, j = 1, 2, \ldots, m,
\]

we call an assignment of operations from the set \( \mathcal{O} \) to machines from the set \( \mathcal{M} \) (or shortly, operations to machines assignment).

A sequence \([\mathcal{Q}^{k_1+1}, \mathcal{Q}^{k_2+2}, \ldots, \mathcal{Q}^{k_{m+1}}]\) is an assignment of operations to machines in the \( i \)-th nest (shortly, assignment in the \( i \)-th nest). In a special case a machine can execute no operations and then a set of operations assigned to be executed by this machine is an empty set.

If the assignment of operations to machines has been carried out, then the optimal schedule of operations execution determination (including a sequence of operations execution on machines) simplifies to solve the classic scheduling problem (Grabowski, Wodecki [12]).

Let \( \mathcal{K} = [K_1, K_2, \ldots, K_m] \) be a sequence of sets where \( K_i \in 2^{\mathcal{O}}, \text{ } i = 1, 2, \ldots, m \) (in particular case elements of this sequence can constitute empty sets). By \( \mathcal{K} \) we denote the set of all such sequences. The number of elements of the set \( \mathcal{K} \) is \( 2^{\mathcal{O}^1}, 2^{\mathcal{O}^1}, \ldots, 2^{\mathcal{O}^m} \).

If \( \mathcal{Q} \) is an assignment of operations to machines then \( \mathcal{Q} \in \mathcal{K} \) (of course, the set \( \mathcal{K} \) includes also sequences which are not feasible; that is such sequences do not constitute assignments of operations to machines).

For any sequence of sets \( K = [K_1, K_2, \ldots, K_m] (K \in \mathcal{K}) \) by \( \Pi_i(K) \) we denote the set of all permutations of elements from \( K_i \). Thereafter, let

\[
\pi(K) = (\pi_1(K), \pi_2(K), \ldots, \pi_n(K))
\]
be a concatenation \( m \) sequences (permutations), where \( \pi_i(K) \in \Pi_i(K) \). Therefore

\[
\pi(K) \in \Pi(K) = \Pi_1(K) \times \Pi_2(K) \times \ldots \times \Pi_m(K).
\]

It is easy to observe that if \( K = [K_1, K_2, \ldots, K_m] \) is an assignment of operations to machines then the set \( \pi_i(K) \ (i = 1, 2, \ldots, m) \) includes all permutations (possible sequences of execution) of operations from the set \( K_i \) on the machine \( i \). Further, let

\[
\Phi = \{(K, \pi(K)) : K \in \mathcal{K} \land \pi(K) \in \Pi(K)\}
\]

be a set of pairs, where the first element is a sequence set and the second – a concatenation of permutations of these sets elements. Any feasible solution of the FJSP is a pair \((Q, \pi(Q)) \in \Phi\) where \( Q \) is an assignment of operations to machines and \( \pi(Q) \) is a permutations concatenation determining the operations execution sequence which are assigned the each machine fulfilling constrains (i-iv).

3. Solution method

There is an exponential number of possible jobs to machines assignments, due to the number of operations. Each feasible assignment generates a NP-hard problem (job shop) whose solution consists in determining an optimal jobs execution order on machines. One has to solve exponential number of NP-hard problems to solve the FJSP.

Therefore, we will apply an approximate algorithm based of the tabu search method. The main element of this approach is a neighborhood – subsets of all the feasible solution set, generated from the current solution by transformations called moves. Searching a neighborhood we choose an element with the lowest cost function value, which we take as a new current solution in the next iteration of the algorithm. From the fixed solution it is possible to generate another solution by executing a move which consists in:

- an operation moving (transferring) from a machine into another machine in the same nest (of the same type), or
- changing operations execution order on a machines.

In the further part of the paper we describe in details both these moves.

Let \( \Theta = (Q, \pi(Q)) \) be a feasible solution for the FJSP, where \( Q = [Q^1, Q^2, \ldots, Q^m] \) is an operations to machines assignment, \( q_i \) is a number of operations executed on a machine \( M_i \) (i.e. \( q_i = |Q^i| \)) and

\[
\pi(Q) = (\pi_1(Q), \pi_2(Q), \ldots, \pi_m(Q))
\]

is a concatenation of \( m \) permutations. A permutation \( \pi_i(Q) \) determines an order of operations from the set \( Q^i \) which have to be executed on the machine \( M_i \) \( i = 1, 2, \ldots, m \). By \( C_{\text{max}}(\Theta) \) we define the cost function value for a solution \( \Theta \), i.e. the time of all the operations execution finishing time.

In the further part of this section we will omit operations assignment \( Q \) as permutations parameter in these places where it does not follow to a contradiction. Therefore \( \pi(Q) = (\pi_1(Q), \pi_2(Q), \ldots, \pi_m(Q)) \) will be written in the shorter form of \( \pi = (\pi_1, \pi_2, \ldots, \pi_m) \).

3.1. Transfer type moves

By \( t_j^i(k, l) \) we define a transfer type moves (shortly t-move) which consists in moving an operation from the position \( k \) in a permutation \( \pi_i \) (i.e. operation \( \pi_i(k) \)) into position \( l \) in a permutation \( \pi_j(k) \) (together with moving operations from position \( k, k+1,... \) with a single position to the right – so called insert type move). Execution of the move \( t_j^i(k, l) \) generates from \( \Theta = (Q, \pi) \) the new solution \( \Theta' = (Q', \pi') \) such that

\[
\pi'_v = \pi_v, \quad v \neq i, j, \quad v = 1, 2, \ldots, m
\]

and

\[
\pi'_i = (\pi_i(1), \pi_i(2), \ldots, \pi_i(k-1), \pi_i(k+1), \ldots, \pi_i(q_i - 1), \pi_i(k), \pi_i(l), \pi_i(l+1), \ldots, \pi_i(q_j + 1)) \tag{2}
\]

3.2. Insert type moves

By \( i_j^i(k, l) \) we define an insert type moves (shortly i-move) which consists in inserting an operation into a place between two operations in a permutation \( \pi_i \) (i.e. operation \( \pi_i(k) \)) into position \( l \) in a permutation \( \pi_j(k) \) (together with moving operations from position \( k, k+1,... \) with a single position to the left – so called insert type move). Execution of the move \( i_j^i(k, l) \) generates from \( \Theta = (Q, \pi) \) the new solution \( \Theta' = (Q', \pi') \) such that

\[
\pi'_v = \pi_v, \quad v \neq i, j, \quad v = 1, 2, \ldots, m
\]

and

\[
\pi'_j = (\pi_j(1), \pi_j(2), \ldots, \pi_j(l-1), \pi_j(k), \pi_j(l), \pi_j(l+1), \ldots, \pi_j(q_j + 1)) \tag{3}
\]
Execution of this move causes moving the operation \( \pi_i(k) \) from the set \( Q_i \) (i.e. from the machine \( M_i \)) to the set \( Q_j \) (i.e. to the machines \( M_j \)). Therefore,

\[
Q'' = Q', \quad v \neq i, j, \quad v = 1, 2, ..., m
\]

and

\[
Q^l_j = Q^j \setminus \{\pi_i(k)\}, \quad Q^j = Q^j \cup \{\pi_i(k)\}.
\]

A move \( t'_j(k, l) \) such that \( i = j \) and \( k = l \) we call a neutral \( t \)-move.

Computational complexity of the \( t \)-move execution is \( O(n) \). An execution of a \( t \)-move causes moving an operation from a machine into another one, i.e. a new operations to machines assignment in a nest. Therefore, from any solution (operations to machines assignment) by executing \( t \)-moves it is possible to obtain any other operations to machines assignment, i.e. partition of operations sets in particular nests.

If \( \tau \) is a \( t \)-move, then we define a solution generated from \( \Theta \) by executing the \( \tau \) move by \( \pi(\Theta) \). It is possible that the solution \( \pi(\Theta) \) is not feasible.

Let \( \Theta \) be a feasible solution. The set

\[
T_j^i(\Theta) = \{ t'_j(k, l) : \quad k \in Q^j \quad \mathrm{and} \quad l \in Q^i \}
\]

includes all \( t \)-moves which transfer operations from a machine \( M_i \) into a machine \( M_j \) and

\[
T(\Theta) = \sum_{i,j} T_j^i(\Theta)
\]

includes all \( t \)-moves for the solution \( \Theta \). The number of elements of this set has an upper bound \( O(qm^2o^2) \).

3.2. Insert type moves

To simplify let us assume that a permutation \( \pi = (\pi(1), \pi(2), ..., \pi(t)) \) determines an operations execution order on a machine.

Insert type move \( t^i_j \) (INS) moves an element \( \pi(k) \) (from the position \( k \) in \( \pi \)) into the position \( l \), generating a permutation \( \pi^l_j(\pi) = \pi^l_j(i) \). If \( l \geq k \) therefore:

\[
\pi^l_j(i) = \begin{cases} 
\pi(i), & \text{if } i < k \lor i > l, \\
\pi(i + 1), & \text{if } k \leq i < l, \\
\pi(k), & \text{if } i = l.
\end{cases}
\]

A similar case appears when \( k > l \).

This type of move we will call shortly an \( i \)-move. Its computational complexity is \( O(1) \). The number of all such moves (for a determined machines) is \( t(t - 1) \).

For a fixed feasible solution \( \Theta \), let \( I_k(\Theta) \) be a set of all \( i \)-moves for the machine \( M_k \in M \) and let

\[
I(\Theta) = \sum_{k=1}^{m} I_k(\Theta)
\]

be a set of all \( i \)-moves on all machines. It is easy to notice that not all moves from this set generate a feasible solution.

3.3. Golf neighborhood

A transfer type move is like a long shot in golf: it moves an operation into another machines. In turn, an insert type move makes only a little modification of operation sequences on machines. The inspiration of this research was a paper of Bożejko and Wodecki [4] which considered multimoves.

Let \( \Theta \) be a feasible solution and let \( T(\Theta) \) be a set of all \( t \)-moves. We consider the move \( t'_j(k, l) \in T(\Theta) \). It transfers an operation from the position \( k \)-th on the machine \( M_i \) to a position \( l \)-th on the machine \( M_j \). This move generates the solution \( \Theta'' = r'_j(k, l)(\Theta) \). The set \( \Theta(\Theta') \) includes all \( t \)-moves connected with a solution \( \Theta' \) and \( I_j(\Theta') \) – \( i \)-moves defines on operations executed on the machine \( M_j \). Let \( i^j_l \in I_j(\Theta') \) be an \( i \)-move. Its execution generates a new solution \( \Theta'' = i^j_l(\Theta') \) from the \( \Theta' \). The transformation which generates a solution \( \Theta'' \) from the \( \Theta \) we call an
it-multimove. It constitutes a product of the t-move \( t'_j(k,l) \) and i-move \( t_i \). We will denote this move shortly as \( t'_i \circ t_j(k,l) \). Therefore, \( \Theta'' = t'_i \circ t_j(k,l)(\Theta) \).

By \( I \circ T(\Theta) \) we denote a set of all it-multimoves determined for a solution \( \Theta \). The golf neighborhood \( \Theta \) is the set \( \mathcal{N}(\Theta) = \{ \lambda(\Theta) : \lambda \in I \circ T(\Theta) \} \).

In the paper Bożejko et al. [6] so-called elimination criterion was proved. It allows us to eliminate all i-moves and t-moves which generate unfeasible solutions. Moreover multimoves generating solutions, for which the cost function value is not less than \( C_{\text{max}}(\Theta) \), are also eliminated. Such determined golf neighborhood will be applied to the tabu search algorithm.

If we consider neutral t-moves only then the golf neighborhood ((4) simplifies to moves applied in the best approximate algorithm of the job shop problem solving (Nowicki and Smutnicki [19], Grabowski and Wodecki [12]). The survey of tabu search method approach for the classic job shop problem can be found in [19]. A TSAB [19] algorithm is used as a local optimization method.

4. Computational results

Proposed methods were tested on the Hewlett-Packard server with two Dual-Core AMD 1.0 GHz Opteron processors, 1 MB cache memory and 8 GB RAM working under 64-bit Linux Debian 5.0 operating system. Algorithms were tested on the set of benchmark problem instances taken from Brandimarte [7].

<table>
<thead>
<tr>
<th>problem</th>
<th>( n \times m )</th>
<th>TS1</th>
<th>TS2</th>
<th>TS3</th>
<th>TS4</th>
<th>TS(_{\text{TSAB}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mk01</td>
<td>10 \times 6</td>
<td>17.50%</td>
<td>30.00%</td>
<td>2.50%</td>
<td>5.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Mk02</td>
<td>10 \times 6</td>
<td>42.31%</td>
<td>46.15%</td>
<td>15.38%</td>
<td>23.08%</td>
<td>7.69%</td>
</tr>
<tr>
<td>Mk03</td>
<td>15 \times 8</td>
<td>7.35%</td>
<td>21.57%</td>
<td>0.00%</td>
<td>19.12%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Mk04</td>
<td>15 \times 8</td>
<td>35.00%</td>
<td>33.33%</td>
<td>8.33%</td>
<td>8.33%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Mk05</td>
<td>15 \times 4</td>
<td>11.05%</td>
<td>8.14%</td>
<td>1.16%</td>
<td>1.74%</td>
<td>0.58%</td>
</tr>
<tr>
<td>Mk06</td>
<td>10 \times 15</td>
<td>53.45%</td>
<td>55.17%</td>
<td>22.41%</td>
<td>20.69%</td>
<td>17.24%</td>
</tr>
<tr>
<td>Mk07</td>
<td>20 \times 5</td>
<td>42.45%</td>
<td>26.62%</td>
<td>6.47%</td>
<td>11.51%</td>
<td>10.07%</td>
</tr>
<tr>
<td>Mk08</td>
<td>20 \times 10</td>
<td>15.11%</td>
<td>12.62%</td>
<td>5.35%</td>
<td>3.63%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Mk09</td>
<td>20 \times 10</td>
<td>39.09%</td>
<td>30.29%</td>
<td>33.55%</td>
<td>8.79%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Mk10</td>
<td>20 \times 15</td>
<td>62.44%</td>
<td>55.33%</td>
<td>34.23%</td>
<td>9.64%</td>
<td>6.60%</td>
</tr>
</tbody>
</table>

Table 1: The percentage relative deviation to the best known solution for Brandimarte [7] benchmark instances for a fixed number of iterations.

<table>
<thead>
<tr>
<th>problem</th>
<th>( n \times m )</th>
<th>TS1</th>
<th>TS2</th>
<th>TS3</th>
<th>TS4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mk01</td>
<td>10 \times 6</td>
<td>17.50%</td>
<td>30.00%</td>
<td>2.50%</td>
<td>5.00%</td>
</tr>
<tr>
<td>Mk02</td>
<td>10 \times 6</td>
<td>42.31%</td>
<td>46.15%</td>
<td>15.38%</td>
<td>23.08%</td>
</tr>
<tr>
<td>Mk03</td>
<td>15 \times 8</td>
<td>7.35%</td>
<td>21.57%</td>
<td>0.00%</td>
<td>19.12%</td>
</tr>
<tr>
<td>Mk04</td>
<td>15 \times 8</td>
<td>35.00%</td>
<td>33.33%</td>
<td>8.33%</td>
<td>8.33%</td>
</tr>
<tr>
<td>Mk05</td>
<td>15 \times 4</td>
<td>11.05%</td>
<td>8.14%</td>
<td>1.16%</td>
<td>1.74%</td>
</tr>
<tr>
<td>Mk06</td>
<td>10 \times 15</td>
<td>53.45%</td>
<td>55.17%</td>
<td>22.41%</td>
<td>20.69%</td>
</tr>
<tr>
<td>Mk07</td>
<td>20 \times 5</td>
<td>42.45%</td>
<td>26.62%</td>
<td>6.47%</td>
<td>11.51%</td>
</tr>
<tr>
<td>Mk08</td>
<td>20 \times 10</td>
<td>15.11%</td>
<td>12.62%</td>
<td>5.35%</td>
<td>3.63%</td>
</tr>
<tr>
<td>Mk09</td>
<td>20 \times 10</td>
<td>39.09%</td>
<td>30.29%</td>
<td>33.55%</td>
<td>8.79%</td>
</tr>
<tr>
<td>Mk10</td>
<td>20 \times 15</td>
<td>62.44%</td>
<td>55.33%</td>
<td>34.23%</td>
<td>9.64%</td>
</tr>
</tbody>
</table>

Average 32.57% 31.92% 13.07% 11.15% 4.22%

Table 2: The percentage relative deviation to the best known solution for Brandimarte [7] benchmark instances for fixed computational time.

In the Table 1 and 2 particular columns mean:
• TS1 - tabu search algorithm in which the full neighborhood with using t-moves is generated; the best solution from this neighborhood is chosen and becomes the base solution in the next iteration,

• TS2 - in this algorithm the neighborhood is generated by the t-moves performed for operations from the critical path; the best solution from this neighborhood is chosen and becomes the base solution in the next iteration,

• TS3 - the tabu search algorithm in which the full neighborhood using t-moves is generated; the best solution from this neighborhood is chosen; for this solution the TSAB \[19\] algorithm is performed; the solution provided by the TSAB algorithm becomes the base solution in the next iteration,

• TS4 - in this algorithm the neighborhood is generated by t-moves performed for operations from the critical path; the best solution from this neighborhood is chosen; for this solution the TSAB algorithm is performed; the solution provided by the TSAB algorithm becomes the base solution in the next iteration,

• TS_{TSAB} - two level tabu search algorithm for flexible job shop problem.

Proposed algorithms were tested on 10 test instances taken from Brandimarte \[7\]. We compare our results with results from Gao et al. \[10\]. The Table 1 presents results for a fixed number of iteration (equals 1000) of the proposed algorithm. The Table 2 gives results for a fixed computational time (240 seconds) of the proposed algorithm. Obtained results show that for performing TSAB algorithm with the best solution in neighborhood as a starting solution an average value of the cost function is more than two times smaller than without using TSAB algorithm. Making a comparison of results for algorithms which use operations from critical path (TS2 and TS4) with algorithms which don’t use only these operations (TS1 and TS3) we can notice that for the smallest sizes of problem instances (Mk01, Mk02 or Mk03) the algorithm TS2 (TS4) provides with the worst results comparing to the algorithm TS1 (TS3). On the other hand, for a bigger size of the problem instances (Mk07, Mk09, Mk09 or Mk10) the algorithm TS2 (TS4) gives better results than the TS1 (TS3) algorithm.

We have also tested an algorithm based on the hybrid neighborhood (TShn). Proposed neighborhood is generated by performing t-moves and moves used in TSAB algorithm (moves for classical job shop problem). Results for TShn algorithm are given in the Table 3. We have compared our results with the hGA algorithm \[10\] and the TS3 algorithm.

<table>
<thead>
<tr>
<th>problem</th>
<th>hGA</th>
<th>TS3</th>
<th>TShn</th>
<th>TS3</th>
<th>TShn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mk01</td>
<td>40</td>
<td>40</td>
<td>41</td>
<td>0.00%</td>
<td>2.50%</td>
</tr>
<tr>
<td>Mk02</td>
<td>29</td>
<td>29</td>
<td>31</td>
<td>11.54%</td>
<td>19.23%</td>
</tr>
<tr>
<td>Mk03</td>
<td>204</td>
<td>204</td>
<td>204</td>
<td>0.00%</td>
<td>0.00%</td>
</tr>
<tr>
<td>Mk04</td>
<td>60</td>
<td>65</td>
<td>69</td>
<td>8.33 %</td>
<td>15.00 %</td>
</tr>
<tr>
<td>Mk05</td>
<td>172</td>
<td>173</td>
<td>173</td>
<td>0.58 %</td>
<td>0.58 %</td>
</tr>
<tr>
<td>Mk06</td>
<td>58</td>
<td>68</td>
<td>77</td>
<td>17.24%</td>
<td>32.76%</td>
</tr>
<tr>
<td>Mk07</td>
<td>139</td>
<td>144</td>
<td>144</td>
<td>2.88 %</td>
<td>3.60 %</td>
</tr>
<tr>
<td>Mk08</td>
<td>523</td>
<td>523</td>
<td>587</td>
<td>0.00 %</td>
<td>12.24 %</td>
</tr>
<tr>
<td>Mk09</td>
<td>307</td>
<td>326</td>
<td>347</td>
<td>6.19%</td>
<td>13.03 %</td>
</tr>
<tr>
<td>Mk10</td>
<td>197</td>
<td>227</td>
<td>265</td>
<td>15.23%</td>
<td>34.52%</td>
</tr>
<tr>
<td>average</td>
<td></td>
<td></td>
<td></td>
<td>6.20%</td>
<td>13.35%</td>
</tr>
</tbody>
</table>

Table 3: The percentage relative deviation to the best known solution for Brandimarte \[7\] benchmark instances for fixed computational time (600s) of proposed algorithm.

For the fixed execution time the TS3 algorithm provides better solutions than the TShn algorithm. The relative error (i.e. the percentage relative deviation to the best known solutions taken from the literature) for the TS3 algorithm is two times smaller than for the TShn algorithm. For a fixed number of iterations (see Table 4) the TShn algorithm is faster than TS3 algorithm but the quality of obtained solutions is worse.
Table 4: Computational time for 1000 iterations.

<table>
<thead>
<tr>
<th>problem</th>
<th>Cmax</th>
<th>time [sec.]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>hGA</td>
<td>TS3</td>
</tr>
<tr>
<td>Mk01</td>
<td>40</td>
<td>41</td>
</tr>
<tr>
<td>Mk02</td>
<td>29</td>
<td>30</td>
</tr>
<tr>
<td>Mk03</td>
<td>204</td>
<td>204</td>
</tr>
<tr>
<td>Mk04</td>
<td>60</td>
<td>65</td>
</tr>
<tr>
<td>Mk05</td>
<td>172</td>
<td>174</td>
</tr>
<tr>
<td>Mk06</td>
<td>58</td>
<td>71</td>
</tr>
<tr>
<td>Mk07</td>
<td>139</td>
<td>148</td>
</tr>
<tr>
<td>Mk08</td>
<td>523</td>
<td>551</td>
</tr>
<tr>
<td>Mk09</td>
<td>307</td>
<td>410</td>
</tr>
<tr>
<td>Mk10</td>
<td>197</td>
<td>267</td>
</tr>
</tbody>
</table>

5. Conclusions

We have proposed a new so-called golf neighborhood implemented inside a tabu search based algorithms for the flexible job shop problem. Our results show that using only ‘long’ moves (we call them t-moves – operations moves between different machines) there is no possibility to obtain results with high quality (in meaning of the value of the cost function). From the other hand, using ‘long’ moves together with the local improvement method (‘small’ moves) allows us to improve the quality of obtained solutions. As a future work it is possible to adapt the proposed approach for another flexible scheduling problems.

References


