A critical review of the newest biologically-inspired algorithms for flowshop scheduling problem

Jerzy Duda

AGH University of Science and Technology, Faculty of Management,
Dept. of Applied Computer Science, ul. Gramatyka 10, 30-067 Kraków, Poland
jduda@zarz.agh.edu.pl

Abstract. Three the most recent bioinspired heuristics proposed in OR literature for solving a flowshop scheduling problem are revised in the paper. All of these algorithms use some local search procedures in order to improve solutions achieved by the main procedure. The author tries to asses the gain coming from hybridising such heuristics with local search procedures. The results achieved provide the arguments for the thesis that simple local search algorithms can successfully conquer with much complex hybrids.

1 Introduction

The term biologically-inspired algorithms describes such methodologies like neural networks, evolutionary algorithms, particle swarm optimization, ant colony optimization and artificial immune systems. Most of these metaheuristics were successfully applied for various optimization problems in order to provide optimal or good suboptimal solutions.

The study presented in this paper focuses on the biologically-inspired heuristics applied for one of the most widely studied combinatorial optimization problems in OR literature – a flowshop scheduling problem. This problem is commonly used as a benchmark for testing new heuristic algorithms, although it can hardly be applied to shop floor sequencing problems found in real production systems. On the other hand, it is NP-complete in a strong sense [3], its definition is simple and the methods developed to solve classical flowshop problem can often be applied to more complex problems, which can be encountered not only in production management.

The task in flowshop scheduling is to find a sequence of $n$ independent jobs, which are to be processed on the $m$ machines. Only one job can be processed on a given machine at the same time and it cannot be interrupted. In its most popular version – permutation flowshop – all jobs has to be processed in the same order on all machines. Thus a valid solution is simply a permutation of all jobs. The most common optimisation criterion is to minimise the completion time of the last job on the last machine, which is called makespan ($C_{\text{max}}$).

A lot of heuristics have been proposed in order to provide a good approximation of the optimal solution for the flowshop problem in a reasonable time. Those include branch and bound techniques and many nature-inspired metaheuristics, including tabu...
search, simulated annealing and biologically-inspired algorithms, such as first of all genetic algorithms. An extensive review of heuristics applied to the flowshop problem can be found for example in [11].

In recent years some new biologically-inspired algorithms for the flowshop problem appeared in OR literature. Among them one can find genetic algorithms (GA) [12], ant colony optimisation (ACO) [9] and particle swarm optimisation (PSO) [19]. The common feature of all these algorithms is that they are in fact hybrid algorithms, as they use some local search procedures in order to improve the quality of solutions. Their authors prove that such hybrid metaheuristics perform better than metaheuristics without local search optimisation. In this study the author looks for an answer to another, even more important question: do the hybrid metaheuristics provide additional benefit over the local search procedures used in them?

2 Genetic algorithm

The genetic algorithm is the most popular biologically-inspired heuristic among the ones studied in this paper. Many authors tried to apply GA also for the flowshop scheduling problem. One of the first was GA proposed by Chen et al. [1]. It uses only one genetic operator – *partially mapped crossover* (PMX), defined by Goldberg and Lingle for solving Travelling Salesman Problem. The solutions are initialised with several constructive heuristics (CDS, Dannenberg), but less effective than NEH, which will be presented later in this section.

Much more effective genetic algorithm was proposed by Reeves [10]. It uses so called *C1 crossover*, which adopts simple one-point crossover to generate valid solutions for permutation representation of individuals. Contrary to the basic genetic algorithm, the offspring replace not their parents, but the individuals with fitness values below the average fitness of the population. Reeves’ algorithm makes use of the simple swap mutation, but with self adaptive probability value in order to maintain necessary variability of the population.

New interesting genetic algorithms for the flowshop sequencing have been recently proposed by Ruiz et al. [12]. According to the tests performed by their authors those algorithms outperform Chen et al. and Reeves algorithms by a large margin. The most effective variant of the proposed solutions was called HGA_RMA an is presented in the following section.

2.1 HGA_RMA

HGA_RMA uses well-known insertion mutation, but new crossover operator (SBOX). A *similar block order crossover* (SBOX) works as follows: first, the common jobs presented in both parents are copied into the offspring, but only those which build blocks, i.e. there are at least to consecutive common jobs. Next, a cut point is chosen and all the jobs up to that point are copied into offspring. Finally, the missing jobs are inserted according to their relative order in the other parent. One can notice that
SBOX is a combination of the one point crossover and the idea of common sequence, presented for example in a longest common sequence crossover (LCS-OX) [4].

The parents for the mating pool are taken on the basis of binary tournament selection – i.e. from two randomly chosen individuals the one with better fitness is taken. The author’s experience indicates that this simple selection scheme performs better for scheduling problems than for example proportional selection used in basic genetic algorithm.

The replacement scheme allows for replacing the worst solution in the population with a new offspring only, if the fitness function of the latter is better. This keeps the individual in the population at relatively high fitness level, but can lead to a quick convergence of the population. In order to prevent this, 80% of the population is replaced with new individuals after a given number of generations.

The scheme of HGA_RMA algorithm is shown in Figure 1.

Fig. 1. General scheme of hybrid GA (HGA_RMA) by Ruiz et al.

The HGA_RMA authors use Nawaz, Enscore and Ham (NEH) algorithm [7] for individuals initialisation, but also as a local search procedure. NEH algorithm is regarded as the best constructive heuristic defined for the flowshop scheduling problem [11]. It works in four steps. First, the sum of processing times on all machines is calculated for each job. Then, the jobs are sorted in a descending order of the calculated sums. Next, two first jobs are taken form the list and placed in such an order, that the makespan of the two-job sequence is the smallest. Finally, remaining $n-2$ jobs are placed, in turn in separate iterations, in such position that gives the smallest makespan of the partial sequences built so far. For example, the third job on the sorted list can be placed in 3 possible positions: before the first job, between the first and the second job and in the end of the partial sequence (third position). The position which gives the smallest makespan for the three jobs is chosen for the next iteration of this NEH algorithm step.
A local search procedure used in HGA_RMA corresponds to the last two steps of the NEH method and utilizes so called insertion neighbourhood. Starting from the job in the second position, jobs are placed in turn in all possible positions in partial sequences and the best position, which gives the smallest makespan is taken every time. The local search is applied with the probability of \( p_e \).

### 2.2 NEH-based Iterative Local Search algorithm

The simple iterated local search (ILS) algorithm has been constructed by the author in order to assess the impact of the local search on the solutions achieved by HGA_RMA. The ILS algorithm uses the same local search procedure, as in HGA_RMA. The NEH-based local search procedure allows only for extensive exploitation of the small fragment of the global search space. Thus it is necessary to introduce some mechanism allowing for escaping from the current search space area to another, maybe more prospective area. This mechanism is called modification step or disturbance step in the classical iterated local search scheme.

The general scheme of the ILS algorithm proposed in this paper is given in Figure 2 and corresponds to the general ILS scheme presented by Stuetzle in [15].

\[
\text{Initialise starting solution} \\
\text{Apply NEH local search procedure} \\
\text{Do} \\
\quad \text{Modify current solution} \\
\quad \text{If it is better than current best then accept it} \\
\quad \text{else accept it if acceptance criterion is met} \\
\text{Apply NEH local search procedure} \\
\text{Until termination criterion is met}
\]

**Fig. 2.** General scheme of iterated local search [15]

In the proposed algorithm modification step is done by swapping two jobs chosen randomly from the current solution (so called exchange neighbourhood). The number of swaps was experimentally set to \( 2^n \). A modified solution is accepted if it is better than the current best solution \( (b) \) or if an acceptance criterion is met. The acceptance criterion is identical with the one used in a basic simulated annealing algorithm. A worse solution \( (s) \) is accepted with a probability of:

\[
p_a = e^{(b-s)/T}
\]

Temperature \( (T) \) was set to 0.4 on the basis of experiments performed.

Note, that the main idea for building ILS algorithm was to keep it as simple as possible. It had to follow the local search scheme of HGA_RMA and not necessarily give the best possible results. The most complex step in the proposed ILS algorithm is the acceptance scheme of a newly generated solution based on simulated annealing.
This, however, can be replaced with any other acceptance mechanism preventing the algorithm from being stuck in one fragment of the search space.

2.3 HGA_RMA versus ILS_NEH

A set of Taillard standard problems [16] has been used in order to compare hybrid HGA_RMA algorithm and simple iterated local search algorithm built by the author. The problems with 20, 50 and 100 jobs were calculated in 10 reruns, while for the problems with 200 jobs only 5 independent runs were calculated due to noticeably longer computational time (up to 150–300 sec). In each single experiment 5000 iterations were computed.

The parameters for GA were set to the values provided by Ruiz et al. [12]:
- selection type: binary tournament
- crossover type: SBOX with probability \( p_c \): 0.4
- mutation type: insert with probability \( p_m \): 0.01
- population size: 20 individuals
- restart parameter \( G_r \): 25
- enhancement probability \( p_e \): 0.05
- percentage of individuals generated by modified NEH procedure \( B_i \): 25%.

The results showing relative increase over the best known solutions (as of April 2005) are reported in Table 1. The current list of best solutions can be found on-line at Taillard’s homepage [18]. The column marked as Avg presents the average increase from all 10 or 5 runs, while the column Min presents minimal deviation achieved in the best run out of 10 or 5 runs.

Table 1. Average percentage increase over the best known solutions for GA and ILS.

<table>
<thead>
<tr>
<th>Instance</th>
<th>HGA_RMA</th>
<th>ILS_NEH</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg</td>
<td>Min</td>
</tr>
<tr>
<td>20×5</td>
<td>0.12</td>
<td>0.04</td>
</tr>
<tr>
<td>20×10</td>
<td>0.21</td>
<td>0.07</td>
</tr>
<tr>
<td>20×20</td>
<td>0.22</td>
<td>0.12</td>
</tr>
<tr>
<td>50×5</td>
<td>0.03</td>
<td>0.01</td>
</tr>
<tr>
<td>50×10</td>
<td>1.28</td>
<td>1.09</td>
</tr>
<tr>
<td>50×20</td>
<td>2.23</td>
<td>1.76</td>
</tr>
<tr>
<td>100×5</td>
<td>0.06</td>
<td>0.00</td>
</tr>
<tr>
<td>100×10</td>
<td>0.45</td>
<td>0.23</td>
</tr>
<tr>
<td>100×20</td>
<td>2.30</td>
<td>2.00</td>
</tr>
<tr>
<td>200×10</td>
<td>0.28</td>
<td>0.24</td>
</tr>
<tr>
<td>200×20</td>
<td>2.20</td>
<td>2.13</td>
</tr>
<tr>
<td>Average</td>
<td>0.85</td>
<td>0.70</td>
</tr>
</tbody>
</table>

The average results obtained by HGA_RMA algorithm are little worse than achieved in experiments done by its authors [12]. This may be due to the following two reasons. Firstly, Ruiz et. al. did not provide iteration numbers, which were
counted, but only computational time, which is hard to compare on different machines, while using different programming languages. Secondly, they compared their results with the best results known in April 2004, i.e. a year before the date of the current list of best known solutions (some of them have been updated).

Nevertheless, the most important conclusion is that hybrid GA does not perform any better than the proposed iterated local search algorithm, excluding the problems with only 5 machines. Based mainly on the local search procedure used in the hybrid GA, proposed ILS algorithm is much more simple to implement and it requires only half of time to count the same number of iterations than HGA_RMA. This, however, does not mean that the genetic algorithm proposed by Ruiz et. al is a poor performer. The variant of genetic algorithm without local search called GA_RMA is the best pure genetic algorithm for the flowshop scheduling problem presented so far in literature. It is outperformed only by hybrid algorithms like HGA_RMA or PACO, which will be described in the next section.

It is also worth to notice that the idea of applying ILS to the flowshop scheduling problem has been already investigated by Stuetzle in [15]. He used the same local search procedure and acceptance criterion (though different temperature value), but different modification method. Contrary to the modification method proposed in this paper Stuetzle ILS algorithm used only small modifications (only 3 swaps, preferably between direct neighbours). The results shown in Table 1 for ILS are better than the ones obtained by Stuetzle [15]. However, also in this case it is hard to compare the two algorithms directly for the same reasons.

3 Ant Colony Optimization

Ant Colony Optimization has been proposed by Dorigo et al. in 1996 [2]. The main idea behind it is to treat the construction process of a solution as move of a single ant. A colony of ants can consist of many ants and each can leave its pheromone on the path it traversed while building a solution. The level of the pheromone evaporates in the next generations.

The general scheme of ant colony optimization is shown in Figure 3.

Initialise pheromone table \( \tau_{ij} \)
For each iteration do
  For each ant do
    Do
      Build the solution for the ant
      Until solution is built
      Evaluate the fitness value of the ant
      Apply local search algorithm (optionally)
    Next ant
    Update pheromone values \( \tau_{ij} \)
Next iteration

Fig. 3. General scheme of basic ACO algorithm
Only few algorithms has been proposed for solving the flowshop scheduling problem so far. The first of them was Max-Min Ant System (MMAS) developed by Stuetzle [14]. It builds a solution on the basis of the best solution found earlier. MMAS also makes use of a local search algorithm based on insertion neighbourhood. The pheromone for the first ant is initialised on the basis of the solution generated by NEH algorithm.

Rajedan and Ziegler in [9] proposed two ant colony algorithms for flowshop scheduling problems. The first, called M-MMAS, is an extended version of the Stuelze MMAS algorithm. It uses so called summation rule of ant pheromones and a local search procedure based on job indexes. These features are also present in the second algorithm called PACO (for the proposed ant colony optimization). PACO performs better then M-MMAS, thus it is presented in details in the next section.

3.1 PACO

Both PACO and M-MMAS algorithms builds only one ant in each iteration. It means that each ant lays its pheromone on the solution path and undergoes local optimisation process. First ant is initialised using NEH algorithm and then applying a local search algorithm for the three times. The local search algorithm is based on insertion neighbourhood, but this time jobs are inserted on the basis of their index in a sequence and not on jobs order, like in other cases presented in this paper (detailed scheme of this algorithm is shown in section 3.2).

Once the sequence for the first ant is created the pheromone values are initialised according to the following rule:

\[
\tau^{0}_{ij} = f^{-1}, \quad \text{if } |\text{pos}(i) - k| + 1 \leq n / 4
\]

\[
\tau^{0}_{ij} = (2 \times f)^{-1}, \quad \text{if } |\text{pos}(i) - k| + 1 \leq n / 2
\]

\[
\tau^{0}_{ij} = (4 \times f)^{-1}, \quad \text{otherwise}
\]

where:
- \( f \) – objective function for the ant sequence,
- \( \text{pos}(i) \) – returns the position of the job \( i \) in the ant sequence.

In next iterations ant sequences are constructed mainly on the basis of pheromone values. In MMAS algorithm each pheromone value \( \tau_{ij} \) indicates a ‘desire’ of placing job \( i \) in the position \( j \). Whereas PACO and M-MMAS use so called summation rule for pheromone calculation. The sum of pheromone is calculated in the following way:

\[
T_{ij} = \sum_{k=1}^{n} \tau_{ik}
\]

The value of pheromone represents now a ‘desire’ of placing job \( i \) not later than in position \( j \).
There are three possibilities for choosing yet unscheduled job $i$ for position $j$ in PACO:

- take the first unscheduled job from the best sequence obtained so far
- from the set of the first five unscheduled jobs in the best sequence choose the one with the highest $T_{ij}$
- draw one job from the set of the first five unscheduled jobs in the best sequence with the probability proportional to their $T_{ij}$

The first two possibilities has a probability of 0.4, while the last can occur with a probability of 0.2.

Right after building process of the ant sequence is finished the local search procedure is applied for the three times and the pheromone values are updated according to the schema:

$$
\tau'_{ij} = \rho \times \tau_{ij}^{t-1} + \left( \left| \text{pos}_b(i) - j \right| + 1 \right)^{0.5} \times f \quad \text{if } \left| \text{pos}(i) - j \right| \leq 1
$$

$$
\tau'_{ij} = \rho \times \tau_{ij}^{t-1}, \quad \text{otherwise}
$$

where:
- $\rho$ – evaporation rate of the pheromone value,
- $\text{pos}_b(i)$ – returns the position of the job $i$ in the best sequence found so far.

Remaining symbols are the same as in Formula (2).

When all iterations are finished yet another local search algorithm is applied to the final solution. This time it is based on exchange neighbourhood presented in section 4.2, but similarly to the first local search algorithm it works with job indexes instead of job order.

### 3.2 Iterated Local Search with job-based-insertion

The iterated local search built by the author of this paper uses the same job-index-based local search procedure as in PACO. The detailed scheme of this procedure is shown in Figure 4.

```plaintext
s=global_best
For i=1 to n
    For j=1 to n
        if index(i)<>i then
            s1=insert(s,i,j)
            if f(s1)<f(s) then
                f(s)=f(s1)
                n1=i; n2=j
            j=j+1
    Next j
Next i
if f(s)<f(global_best) then
    global_best=insert(s,n1,n2)

Fig. 4. Detailed scheme of join-index-based local search used in PACO. Based on [9]
```
The job-index-based local search is a variant of well-know job insertion neighbourhood used in NEH. According to Rajendran and Ziegler experiments it performs better than other simple local search strategies for job sequencing. The results presented in this paper do not indicate for any superiority of this approach comparing to classical, i.e. job-order-based local search. Moreover, the computational complexity of this algorithm step is \( O(n^2m) \), while Taillard shown in [17], that insertion neighbourhood can be evaluated in \( O(nm) \).

For the purpose of comparison the PACO algorithm against its local search engine a simple iterated local search has been created by the author of this paper. It generally follows the basic iterated local search scheme presented in section 2.2. This time, however, modification step is done only by swapping two randomly chosen jobs in the sequence and only better solution is accepted in each iteration (no acceptance criterion is used).

### 3.3 PACO versus ILS_JBI

In order to compare PACO with ILS based on job-index-insertion the same experiments as in the case of HGA_RMA and ILS were done. The number of iteration in PACO was set to 40 by its authors, however in order to maintain similar computational time as in previous experiment 200 iterations were calculated instead. It also is worth to notice that the local search algorithm used in PACO is more computational expensive than the other LS algorithms presented in this paper.

The remaining parameters of PACO were set according to Rajendran and Ziegler [9] as follows:

- ant colony size: 1 ant,
- evaporation rate: \( q = 0.75 \),

The relative increase over the best known solutions provided by both algorithms is shown in Table 2.

**Table 2.** Average percentage increase over the best known solutions for PACO and ILS.

<table>
<thead>
<tr>
<th>Instance</th>
<th>PACO</th>
<th>ILS_JBI</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n \times m )</td>
<td>Avg</td>
<td>Min</td>
</tr>
<tr>
<td>20\times5</td>
<td>0.78</td>
<td>0.33</td>
</tr>
<tr>
<td>20\times10</td>
<td>1.06</td>
<td>0.33</td>
</tr>
<tr>
<td>20\times20</td>
<td>0.78</td>
<td>0.31</td>
</tr>
<tr>
<td>50\times5</td>
<td>0.12</td>
<td>0.05</td>
</tr>
<tr>
<td>50\times10</td>
<td>1.21</td>
<td>0.85</td>
</tr>
<tr>
<td>50\times20</td>
<td>3.64</td>
<td>3.01</td>
</tr>
<tr>
<td>100\times5</td>
<td>0.11</td>
<td>0.04</td>
</tr>
<tr>
<td>100\times10</td>
<td>0.56</td>
<td>0.38</td>
</tr>
<tr>
<td>100\times20</td>
<td>2.96</td>
<td>2.62</td>
</tr>
<tr>
<td>200\times10</td>
<td>0.28</td>
<td>0.14</td>
</tr>
<tr>
<td>200\times20</td>
<td>2.07</td>
<td>1.88</td>
</tr>
<tr>
<td>Average</td>
<td>1.23</td>
<td>0.90</td>
</tr>
</tbody>
</table>
The quality of the results achieved depends on the size of the test problems. For the problems with 20 jobs, as well as for the problems with 50 and 100 jobs, but the ones with smaller number of machines, ILS algorithm performs better than PACO algorithm. This is true especially for 20 job problems, when ILS outperforms PACO by a large margin, no matter if the average result or the best solution is taken into consideration.

This indicates that the pheromone table, which is in fact a long term memory for the search procedure may, help ILS or other simple algorithm to achieve better results for large size problems. On the other hand, an ILS version with NEH-based local search presented in section 2.2 achieves better results than PACO with the exception for 200x10 problem instances. This algorithm does not use any long term memory mechanism.

4 Particle Swarm Optimisation

PSO like ACO is a relatively new biologically-inspired metaheuristic and it was developed by Kennedy and Eberhard in 1995 [5]. It is based on the observation of social behaviour of animals like birds or fishes. Its main idea is that the members of a swarm (particles) can cooperate with each other, adjusting their positions (by increasing or decreasing their speeds in particular dimension) in order to avoid a predator or to find some food.

The basic PSO algorithm is based on so called global neighbourhood model – it is when the particles in the swarm move towards the global best solution and their best positions (solutions) find so far. The scheme of such PSO algorithm is shown in Figure 5.

**Fig. 5. General scheme of basic PSO algorithm**

4.1 PSO_VNS

Perhaps the first PSO algorithm to solve the permutation flowshop scheduling problem has been proposed by Tasgetiren et al. [19]. The first problem with the application of basic PSO algorithm to the flowshop sequencing is that it works with real representation of solutions. Thus the authors
proposed a simple algorithm, which transforms a sequence of real numbers into a correct permutation. The Smallest Position Values (SPV) rule places the jobs according to their position on the sorted list. For example, the sequence of particle values (2.03, -1.82, 3.25, -0.54, 0.15) gives the sequence of jobs (2, 4, 5, 1, 3). The translation stage is done before the fitness value is evaluated. The SPV algorithm seems to be efficient one. The author of this paper developed some other translating strategies, including self optimising one. Unfortunately they did not improve the PSO algorithm, even that they were more time consuming.

In initial generation of PSO each particle is given a position in \( n \) dimensions (where \( n \) is the number of jobs), randomly from the range of \([0.0, 4.0]\). Also initial velocities are generated randomly from uniform distribution from the range of \([-4.0, 4.0]\). The velocity is adjusted in next generations using the following formula:

\[
v'_{ij} = w^{t-1}v_{ij}^{t-1} + c_1r_1(p_{ij}^{t-1} - x_{ij}^{t-1}) + c_2r_2(g_{j}^{t-1} - x_{ij}^{t-1})
\]

(5)

where:
- \( t \) – current generation,
- \( ij \) – \( j \)-th dimension of \( i \)-th particle,
- \( w \) – inertia weight, decreased in every generation by a \( \beta \) factor,
- \( c_1, c_2 \) – social and cognitive parameters,
- \( r_1, r_2 \) – random numbers from uniform distribution,
- \( p, g \) – particle personal best and the global best respectively,
- \( x \) – current position of particle regarding \( j \)-th dimension.

After the velocity is updated the positions of the particles are also updated according to the formula:

\[
x'_{ij} = x_{ij}^{t-1} - v'_{ij}
\]

(6)

The best performing PSO presented in [19] is the variant of the PSO hybridised with variable neighbourhood search (PSO_VNS). The details of this VNS algorithm are presented in the next section.

4.2 Variable Neighbourhood Search initialised with NEH

The algorithm used in PSO_VNS is a reduced version of variable neighbourhood method and it is based on the insert\+interchange variant of VNS, which was described in [6].

Variable Neighbourhood Search (VNS) is similar to Iterated Local Search algorithm, but its search procedure works in a different way. In the case of ILS the algorithm tries to find a local optimum for the solution generated in the modification step, i.e. in a single iteration exploitation of the search space is done after its exploration. In VNS the search procedure bases mainly on intensive exploration of the search space. In so called reduced VNS (compare [6]) no local optimization method is
utilized. VNS in its basic version also does not use any acceptance criterion for a solution worse than the current solution found.

The detailed scheme of the algorithm is shown in Figure 6.

\[
\begin{align*}
    & s = \text{global\_best} \\
    & n1 = \text{rand}(1,n) \\
    & n2 = \text{rand}(1,n) \\
    & s = \text{insert}(s,n1,n2) \\
    & \text{loop} = 0 \\
    & \text{Do} \\
    & \quad k = 0 \\
    & \quad \text{max\_method} = 2 \\
    & \quad \text{Do} \\
    & \quad \quad n1 = \text{rand}(1,n) \\
    & \quad \quad n2 = \text{rand}(1,n) \\
    & \quad \quad \text{if} \ k = 0 \ \text{then} \\
    & \quad \quad \quad s1 = \text{insert}(s,n1,n2) \\
    & \quad \quad \quad \text{if} \ k = 1 \ \text{then} \\
    & \quad \quad \quad \quad s1 = \text{interchange}(s,n1,n2) \\
    & \quad \quad \quad \quad \text{if} \ f(s1) < f(s) \ \text{then} \\
    & \quad \quad \quad \quad \quad k = 0; \ s = s1 \\
    & \quad \quad \quad \text{else} \\
    & \quad \quad \quad \quad k = k + 1 \\
    & \quad \quad \text{While} \ k < \text{max\_method} \\
    & \quad \text{loop} = \text{loop} + 1 \\
    & \text{While} \ \text{loop} < n(n-1) \\
    & \quad \text{if} \ f(s) < f(\text{global\_best}) \ \text{then} \\
    & \quad \quad \text{global\_best} = s \\
\end{align*}
\]

**Fig. 6.** Detailed scheme of VNS used in PSO [19]

The insert operation removes the job from position \(n_1\) and puts it in position \(n_2\). The interchange operation simply swaps two jobs at positions \(n_1\) and \(n_2\).

Contrary to the local search algorithms used in HGA_RMA and PACO, VNS can be applied not only for exploitation of the search space, but also directly for its exploration. Thus the only modification of the VNS algorithm used in PSO_VNS in order to be run as an independent optimisation algorithm was the initialisation of the staring solution with the solution obtained by the NEH method. This is not obligatory, as the algorithm may start with a random solution, but speeds up the convergence in a relatively good point.

### 4.3 PSO_VNS versus VNS

The same set of Taillard’s benchmarks was used in all other tests presented earlier. This time 1200 iterations were computed to maintain comparable computational time of all experiments. The parameters of PSO_VNS were set according to Tasgetiren et al [19] as follows:

- swarm size: \(2^n\).
− social and cognitive parameters: $c_1 = c_2 = 2.0$,
− initial inertia weight ($w_0$): 0.9; $w_t \geq 0.4$, decrement factor ($\beta$): 0.975.

The relative increase over the best known solutions provided by both algorithms is shown in Table 3.

### Table 3. Average percentage increase over the best known solutions for PSO and VNS.

<table>
<thead>
<tr>
<th>Instance</th>
<th>PSO</th>
<th>VNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n \times m$</td>
<td>Avg Min</td>
<td>Avg Min</td>
</tr>
<tr>
<td>20x5</td>
<td>0.10 0.04</td>
<td>0.10 0.04</td>
</tr>
<tr>
<td>20x10</td>
<td>0.22 0.09</td>
<td>0.15 0.04</td>
</tr>
<tr>
<td>20x20</td>
<td>0.25 0.05</td>
<td>0.18 0.02</td>
</tr>
<tr>
<td>50x5</td>
<td>0.05 0.01</td>
<td>0.05 0.02</td>
</tr>
<tr>
<td>50x10</td>
<td>0.97 0.67</td>
<td>0.81 0.54</td>
</tr>
<tr>
<td>50x20</td>
<td>1.84 1.22</td>
<td>1.57 1.14</td>
</tr>
<tr>
<td>100x5</td>
<td>0.06 0.02</td>
<td>0.04 0.00</td>
</tr>
<tr>
<td>100x10</td>
<td>0.35 0.23</td>
<td>0.22 0.13</td>
</tr>
<tr>
<td>100x20</td>
<td>1.92 1.53</td>
<td>1.80 1.48</td>
</tr>
<tr>
<td>200x10</td>
<td>0.22 0.17</td>
<td>0.25 0.18</td>
</tr>
<tr>
<td>200x20</td>
<td>1.64 1.52</td>
<td>1.59 1.48</td>
</tr>
<tr>
<td>Average</td>
<td>0.69 0.50</td>
<td>0.61 0.46</td>
</tr>
</tbody>
</table>

The results clearly show that variable neighbourhood search completely dominates over the main search process of PSO metaheuristic, making it virtually meaningless. There is no difference between the results achieved by hybrid PSO and the results generated only by its local search algorithm. In all cases, except for 200x10 instances, VNS achieved slightly better results than the one combined with PSO. VNS algorithm also performed twice faster than combined with PSO.

Recently Pan, Tasgetiren and Liang [8] proposed a new PSO algorithm called Discrete Particle Swarm Optimisation (DPSO). It is completely different version of the PSO proposed earlier, however it still follows the general PSO scheme. First of all the algorithm makes use of some operators typical rather for genetic algorithms like crossover (based on simple two cut crossover) and mutation (insert mutation is used). The best performing variant of DPSO uses also a local search procedure very similar to the iterated local search procedure presented in the section 2.2.

### 5 Summary

The most recent hybrid algorithms for the flowshop scheduling problem, based on three different biologically-inspired metaheuristics has been investigated in this paper. The goal of the experiments conducted was to assess the performance margin coming from the main search scheme of a metaheuristic over its local search procedure, which is used in theory only to make some improvements to the main algorithm.
The extensive results presented in the paper indicate, however, that there is no extra gain from using hybrid biologically-based metaheuristics comparing with pure neighbourhood search methods. Algorithms based on metaheuristics are usually more difficult in implementation and are computationally more expensive.

This conclusion can be confirmed also by carefully studying the newest papers concerning the application of metaheuristics for the flowshop scheduling problems in OR literature. A discrete version of Particle Swarm Optimization presented in [8] performs almost equally as the Iterated Greedy (IG) algorithm proposed recently by Ruiz and Stueltzle [13]. The latter is yet another version of local search-based metaheuristic and it is very similar to the iterated local search proposed in section 2.2 of this paper. Instead of performing several mutations to the sequence (modification phase of ILS) it removes some randomly chosen jobs from the sequence (so called destruction phase in IG) and reinserts them using NEH procedure (so called construction phase). It is worth to notice that IG also outperforms the HGA_RMA algorithm, which was also developed by Ruiz.

In the author’s opinion the above conclusion may not necessarily be true for more complex scheduling problems, for example, the ones with limited resources or time windows. The nature-inspired metaheuristics, including biologically-inspired ones, may prove their real value if the search algorithm has to deal with many different constraints or very large size of the problem (see the experience with PACO and ILS_JIB in section 3.3). This will be the subject of the author’s experiments in the near future.

Results achieved for the flowshop scheduling problem certainly cannot be directly generalized to other combinatorial problem. However, the author believes they should interest all researchers in the field of discrete optimization.

Acknowledgments

This study was supported by the State Committee for Scientific Research (KBN) under Grant No. H02D 086 29.

References