TASK QUARTERLY 11 No 1–2, 7–19

A CRITICAL REVIEW OF THE NEWEST BIOLOGICALLY-INSPIRED ALGORITHMS FOR THE FLOWSHOP SCHEDULING PROBLEM

JERZY DUDA

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

(Received 30 December 2006; revised manuscript received 01 February 2007)

Abstract: The three most recent bio-inspired heuristics proposed in the OR literature for solving the flowshop scheduling problem are revised in the paper. All of these algorithms use local search procedures to improve solutions achieved by the main procedure. The author tries to asses the gains from hybridizing such heuristics with local search procedures. The achieved results confirm that simple local search algorithms can compete successfully with much complex hybrids.

Keywords: flowshop scheduling, metaheuristics, local search, hybrid algorithms

1. Introduction

The term *biologically-inspired algorithms* comprises methodologies such as neural networks, evolutionary algorithms, particle swarm optimization, ant colony optimization and artificial immune systems. Most of these metaheuristics have been successfully applied in 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 the OR literature – the 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. At the same time, it is NP-complete in a strong sense [1], its definition is simple and the methods developed to solve the 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 to be processed on 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 permutation flowshop

version, all jobs have to be processed in the same order on all machines. Thus, a valid solution is simply a permutation of all jobs. The most common optimization criterion is to minimize the completion time of the last job on the last machine, which is called *makespan* ($C_{\max}$).

A lot of heuristics have been proposed 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, *e.g.* first-of-all genetic algorithms. An extensive review of heuristics applied to the flowshop problem can be found in [2].

New biologically-inspired algorithms for the flowshop problem have appeared in the OR literature in recent years, including genetic algorithms (GA) [3], ant colony optimization (ACO) [4] and particle swarm optimization (PSO) [5]. A common feature of all these algorithms is that they are in fact hybrid algorithms, as they use local search procedures to improve the quality of solutions. Their authors have proven that such hybrid metaheuristics perform better than metaheuristics without local search optimization. In this study, the author attempts to determine whether hybrid metaheuristics provide additional benefits over the local search procedures used therein.

2. The genetic algorithm

The genetic algorithm is the most popular biologically-inspired heuristic of those studied in this paper. Many authors attempted to apply it also to the flowshop scheduling problem, one of the first being the GA proposed by Chen *et al.* [6]. It uses a single genetic operator, *partially mapped crossover* (PMX), defined by Goldberg and Linge for solving the Travelling Salesman Problem. The solutions are initialized with several constructive heuristics (CDS, Dannenberg), none of them more effective than NEH, which will be presented later in this section.

A much more effective genetic algorithm was proposed by Reeves [7]. It uses the so-called *C1 crossover*, adapting simple one-point crossover to generate valid solutions for permutation representation of individuals. Unlike in the basic genetic algorithm, offspring replace not their parents but individuals with fitness values below the population's average fitness. Reeves' algorithm makes use of the simple swap mutation, but includes self-adaptive probability value in order to maintain the necessary variability of the population.

New interesting genetic algorithms for flowshop sequencing have been proposed recently by Ruiz *et al.* [3]. According to the tests performed by those authors, the algorithms outperform the Chen *et al.* and Reeves algorithms by a large margin. The most effective variant of the proposed solutions, called HGA_RMA, is presented in the following section.

2.1. HGA_RMA

HGA_RMA uses the well-known insertion mutation and a new crossover operator, *similar block order crossover* (SBOX), which works as follows. First, the common jobs present in both parents are copied into the offspring, but only those which build blocks, *i.e.* when there are at least two consecutive common jobs. Next, a cut point

is chosen and all the jobs up to that point are copied into the offspring. Finally, the missing jobs are inserted according to their relative order in the other parent. Noticeably, SBOX is a combination of one-point crossover and the idea of common sequence, present for example in *longest common sequence crossover* (LCS-OX) [8].

The parents chosen for the mating pool are subject to binary tournament selection, *i.e.* the individual with better fitness is taken from two randomly chosen individuals. The author's experience indicates that this simple selection scheme performs better in scheduling problems than proportional selection used in the basic genetic algorithm.

The replacement scheme allows one to replace the worst solution from a population with new offspring only if the fitness function of the latter is better. This keeps individuals in the population at relatively high fitness levels, but may lead to 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 the HGA_RMA algorithm is shown in Figure 1.

```
Initialize individuals in population
Evaluate fitness values of the population
Do
  Select mating pool using binary tournament
  Perform SBOX crossover with p_c probability
  Perform insert mutation with p_m probability
  Apply LS algorithm to offspring with p_e probability
  Evaluate fitness values of the offspring
  Find the worst individuals in the population
  If the offspring are better and unique replace
    the worst solutions with them
  Find the best solution in the population
  If it is new then apply local search algorithm to
    it with 2*p_e probability
  If no improvement has been made for G_r generations
    restart the population, leaving 20% best individ.
Until Termination criterion is met
```

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

The HGA_RMA authors used the Nawaz, Enscore and Ham (NEH) algorithm [9] to initialize individuals, but also as a local search procedure. The NEH algorithm is regarded as the best constructive heuristic defined for the flowshop scheduling problem [2] and 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 from the list and placed in such an order that the makespan of the two-job sequence is the smallest. Finally, the remaining $n-2$ jobs are placed, in turns of separate iterations, in positions of the smallest makespan of 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 at the end of the partial sequence (*i.e.* third). The position of 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 the so-called *insertion neighbourhood*. Starting from

the job in the second position, jobs are placed in turns in all possible positions in partial sequences and the best position, *i.e.* that of the smallest makespan, is taken every time. The local search is applied with probability p_e.

2.2. The NEH-based iterative local search algorithm

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

A 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 [10].

```
Initialize starting solution
Apply NEH local search procedure
Do
  Modify current solution
  If it is better than current best then accept it
  else accept it if acceptance criterion is met
  Apply NEH local search procedure
Until termination criterion is met
```

Figure 2. General scheme of iterated local search [10]

In the proposed algorithm, the modification step is performed by swapping two jobs chosen randomly from the current solution (the so-called *exchange neighbourhood*). The number of swaps has been set experimentally at $2n$. 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 that used in a basic simulated annealing algorithm. A worse solution, s, is accepted with a probability of:

$$p_a = e^{(b-s)/T}. \tag{1}$$

Temperature, T, has been set at 0.4 on the basis of experiments performed.

The main idea of building the 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. However, it can be replaced with any other acceptance mechanism preventing the algorithm from being stuck in a fragment of the search space.

2.3. HGA_RMA versus ILS_NEH

A set of Taillard standard problems [11] has been used to compare the hybrid HGA_RMA algorithm and the simple iterated local search algorithm deisigned by the author. Problems with 20, 50 and 100 jobs were calculated in 10 reruns, while only 5 independent runs were calculated for problems with 200 jobs due to the noticeably longer computational time (up to 150 – 300 sec). In each experiment 5000 iterations were computed.

The parameters for GA were set at the values provided by Ruiz *et al.* [3]:

- 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 a relative advantage over the best solutions known as of April 2005 are reported in Table 1. The current list of best solutions can be found on-line at Taillard's homepage [12]. The column marked as **avg** presents the average increase from all 10 or 5 runs, while the **min** column presents minimal deviation achieved in the best of 10 or 5 runs.

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

instance	HGA_RMA		ILS_NEH	
$n \times m$	avg	min	avg	min
20×5	0.12	0.04	0.10	0.04
20×10	0.21	0.07	0.20	0.07
20×20	0.22	0.12	0.21	0.09
50×5	0.03	0.01	0.07	0.00
50×10	1.28	1.09	1.02	0.86
50×20	2.23	1.76	2.20	1.92
100×5	0.06	0.00	0.09	0.05
100×10	0.45	0.23	0.48	0.28
100×20	2.30	2.00	1.99	1.65
200×10	0.28	0.24	0.33	0.22
200×20	2.20	2.13	1.87	1.79
average	0.85	0.70	0.78	0.63

The average results obtained by the HGA_RMA algorithm are slightly worse than those achieved in experiments performed by its authors [3]. This may be due to two reasons. Firstly, Ruiz *et al.* did not provide iteration numbers but computational time only, which is difficult to compare on different machines, 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, and some of them have been updated during this time.

Nevertheless, the most important conclusion is that hybrid GA does not perform any better than the proposed iterated local search algorithm, save for problems limited to 5 machines. Based mainly on the local search procedure used in the hybrid GA, the proposed ILS algorithm is much easier to implement and requires only half the HGA_RMA time to count the same number of iterations. However, this 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 in literature so far. It is outperformed only by hybrid algorithms, such as HGA_RMA or PACO, described in the next section.

The idea of applying ILS to the flowshop scheduling problem was investigated earlier by Stuetzle in [10]. He used the same local search procedure and acceptance criterion (though a different temperature value), but a different modification method. Contrary to the modification method proposed in this paper, Stuetzle's ILS algorithm used only small modifications: just 3 swaps, preferably between direct neighbours. The results for ILS shown in Table 1 are better than those obtained by Stuetzle [10]. However, also in this case it is difficult to compare the two algorithms directly for reasons as above.

3. Ant colony optimization

Ant colony optimization was proposed by Dorigo *et al.* in 1996 [13]. Its main idea is to treat the solution construction process as movements of a single ant. A colony of ants can consist of many ants and each ant leaves its pheromone on the path it traverses while building a solution. The pheromone evaporates in next generations.

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

```
Initialize pheromone table τij
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 τij
Next iteration
```

Figure 3. General scheme of the basic ACO algorithm

Only few algorithms have so far been proposed for solving the flowshop scheduling problem. The first of these was the 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 initialized on the basis of the solution generated by the NEH algorithm.

Rajedan and Ziegler proposed two ant colony algorithms for flowshop scheduling problems in [4]. One of them, called M-MMAS, is an extended version of the Stuelze MMAS algorithm, using the so-called summation rule of ant pheromones and a local search procedure based on job indices. These features are also found in the other algorithm, called PACO (Proposed Ant Colony Optimization). PACO performs better than M-MMAS and is presented in detail in the following section.

3.1. PACO

Both PACO and M-MMAS algorithms build only one ant in each iteration, which means that each ant lays its pheromone on the solution path and undergoes a local optimization process. The first ant is initialized using the NEH algorithm followed by a local search algorithm being applied thrice. 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 jobs' order. A detailed scheme of this algorithm is given in Section 3.2.

Once the sequence for the first ant has been created, the pheromone values are initialized according to the following rule:

$$\begin{aligned} \tau_{ij}^0 &= f^{-1}, \quad \text{if } |\text{pos}(i)-k|+1 \le n/4, \\ \tau_{ij}^0 &= (2\times f)^{-1}, \quad \text{if } |\text{pos}(i)-k|+1 \le n/2, \\ \tau_{ij}^0 &= (4\times f)^{-1}, \quad \text{otherwise}, \end{aligned} \tag{2}$$

where f is the objective function for the ant sequence and $\text{pos}(i)$ returns the position of job i in the ant sequence.

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

$$T_{ij} = \sum_{k=1}^{j} \tau_{ik}. \tag{3}$$

The value of pheromone now represents a 'desire' of placing job i not farther than in position j.

There are three possibilities for choosing a yet unscheduled job i for position j in PACO:

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

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

Right after the ant sequence building process has been completed, the local search procedure is applied thrice and the pheromone values are updated according to the following schema:

$$\begin{aligned} \tau_{ij}^t &= \rho\times\tau_{ij}^{t-1} + ((|\text{pos}_b(i)-j|+1)^{0.5}\times f), \quad \text{if } |\text{pos}(i)-j| \le 1, \\ \tau_{ij}^t &= \rho\times\tau_{ij}^{t-1}, \quad \text{otherwise}, \end{aligned} \tag{4}$$

where ρ is the evaporation rate of the pheromone value and $\text{pos}_b(i)$ returns the position of job i in the best sequence found so far. The 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 the exchange neighbourhood presented in Section 4.2 but, like the first local search algorithm, it works with job indices instead of a 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 PACO. A detailed scheme of this procedure is shown in Figure 4.

```
s=global_best
For i=1 to n
  For j=1 to n
    if index(j)<>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)
```

Figure 4. Detailed scheme of job-index-based local search used in PACO based on [4]

The job-index-based local search is a variant of the well-know job insertion neighbourhood used in NEH. According to the Rajendran and Ziegler experiments, it performs better than other simple local search strategies for job sequencing. The results presented in this paper do not suggest any superiority of this approach comparing to the classical, *i.e.* job-order-based, local search. Moreover, the computational complexity of this algorithm step is $O(n^2m)$, while Taillard had shown in [15] that an insertion neighbourhood can be evaluated in $O(nm)$.

For the purpose of comparing the PACO algorithm with its local search engine, a simple iterated local search has been created by the present author, generally following the basic iterated local search scheme presented in Section 2.2. However, in this case the modification step is performed by simply swapping two randomly chosen jobs in the sequence, only the better solution being 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 were performed as in the case of HGA_RMA and ILS. Although the number of iterations in PACO had originally been set at 40, in order to maintain computational time similar to that of the previous experiment, 200 iterations were calculated instead. Notably, the local search algorithm used in PACO was more computational expensive than the other LS algorithms presented in this paper.

The remaining parameters of PACO were set according to Rajendran and Ziegler [4] 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.

The quality of the obtained results depends on the size of the test problems. For problems with 20 jobs, as well as for problems with 50 and 100 jobs with smaller numbers of machines, the ILS algorithm has performed better than the PACO algorithm, especially for 20 job problems, where ILS outperforms PACO by a large margin, no matter whether the average result or the best solution are considered.

This indicates that the pheromone table, which is in fact long term memory for the search procedure, may help ILS or another simple algorithm to achieve better

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

instance	PACO		ILS_JBI	
$n \times m$	avg	min	avg	min
20×5	0.78	0.33	0.16	0.04
20×10	1.06	0.33	0.31	0.07
20×20	0.78	0.31	0.28	0.03
50×5	0.12	0.05	0.09	0.03
50×10	1.21	0.85	1.18	0.82
50×20	3.64	3.01	3.69	3.11
100×5	0.11	0.04	0.08	0.02
100×10	0.56	0.38	0.57	0.33
100×20	2.96	2.62	3.03	2.64
200×10	0.28	0.14	0.34	0.17
200×20	2.07	1.88	2.11	1.91
average	1.23	0.90	1.08	0.83

results for large-size problems. At the same time, the version of ILS with NEH-based local search presented in Section 2.2 has achieved better results than PACO except for 200×10 problem instances. This algorithm does not use any long-term memory mechanism.

4. Particle swarm optimization

Like ACO, particle swarm optimization (PSO) is a relatively new biologically-inspired metaheuristic, developed by Kennedy and Eberhard in 1995 [16]. It is based on the observation of social behaviour of animals like birds and fish. Its main idea is that members of a swarm (particles) can cooperate with each other, adjusting their positions (by increasing or decreasing their speeds in a particular dimension) in order to avoid a predator or find food.

The basic PSO algorithm is based on the so-called global neighbourhood model of swarm particles moving towards the global best solution and their own best positions (solutions) found so far. A scheme of such a PSO algorithm is shown in Figure 5.

```
Initialize particles
Evaluate the fitness values of the swarm
Do
  Find the personal bests
  Find the global best
  Update velocity of the particles
  Update positions of the particles
  Evaluate their fitness values
  Apply local search algorithm (optionally)
Until Termination criteria are met
```

Figure 5. General scheme of the basic PSO algorithm

4.1. PSO_VNS

The first PSO algorithm to solve the permutation flowshop scheduling problem has been proposed by Tasgetiren *et al.* [5].

One problem with applying the basic PSO algorithm to flowshop sequencing was that it worked with real representation of solutions. Thus, the authors proposed a simple algorithm, which transformed 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, a sequence of particle values of (2.03, −1.82, 3.25, −0.54, 0.15) yields a sequence of jobs of (2, 4, 5, 1, 3). The translation stage is performed before the fitness value is evaluated. Although the SPV algorithm appears to be efficient, the author of this paper has developed other translating strategies, including a self-optimizing one. Unfortunately, they did not improve the PSO algorithm, even when they were more time consuming.

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

$$v_{ij}^{t} = w^{t-1} v_{ij}^{t-1} + c_1 r_1 (p_{ij}^{t-1} - x_{ij}^{t-1}) + c_2 r_2 (g_j^{t-1} - x_{ij}^{t-1}), \tag{5}$$

where:

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

After velocity has been updated, the particles' positions are updated as well, according to the following formula:

$$x_{ij}^{t} = x_{ij}^{t-1} - v_{ij}^{t}. \tag{6}$$

The best performing PSO presented in [5] is a variant of the PSO hybridized with variable neighbourhood search (PSO_VNS). The details of this VNS algorithm are presented in the following section.

4.2. Variable neighbourhood search initialized with NEH

The algorithm used in PSO_VNS is a reduced version of the variable neighbourhood method based on the *insert+interchange* variant of VNS described in [17].

Variable neighbourhood search (VNS) is similar to the iterated local search algorithm, but its search procedure works differently. In the case of ILS, the algorithm seeks a local optimum for the solution generated in the modification step, *i.e.* in a single iteration the exploitation of the search space is performed after its exploration. In VNS, the search procedure is mainly based on intensive exploration of the search space. In the so-called *reduced VNS* (*cf.* [17]), no local optimization method is utilized. Neither does VNS in its basic version use any acceptance criterion for a solution worse than the current found solution.

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

```
s=global_best
n1=rand(1,n)
n2=rand(1,n)
s=insert(s,n1,n2)
loop=0
Do
  k=0
  max_method=2
  Do
    n1=rand(1,n)
    n2=rand(1,n)
    if k=0 then
      s1=insert(s,n1,n2)
    if k=1 then
      s1=interchange(s,n1,n2)
    if f(s1)<f(s) then
      k=0; s=s1
    else
      k=k+1
  While k<max_method
  loop=loop+1
While loop<n*(n-1)
if f(s)<f(global_best) then
  global_best=s
```

Figure 6. Detailed scheme of VNS used in PSO [5]

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

Contrary to the local search algorithms used in HGA_RMA and PACO, VNS can be applied not only to exploit the search space, but also directly to explore it. Thus, the only modification of the VNS algorithm used in PSO_VNS in order to be run as an independent optimization algorithm was the initialization of the starting solution with the solution obtained by the NEH method. This is not obligatory, as the algorithm may start with a random solution, but it speeds up 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.* [5] as follows:

- swarm size: $2n$,
- social and cognitive parameters: $c_1 = c_2 = 2.0$,
- initial inertia weight (w_0): 0.9; $w_t \geq 0.4$, decrement factor (β): 0975.

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

The results demonstrate that variable neighbourhood search completely dominates the main search process of the PSO metaheuristic, rendering it virtually meaningless. There is no difference between the results achieved with a hybrid PSO and those generated only by its local search algorithm. In all cases except for the 200×10 instances, VNS achieved slightly better results on its own than combined with PSO. The VNS algorithm also performed twice faster on its own than combined with PSO.

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

instance	PSO_VNS		VNS	
$n \times m$	avg	min	avg	min
20×5	0.10	0.04	0.10	0.04
20×10	0.22	0.09	0.15	0.04
20×20	0.25	0.05	0.18	0.02
50×5	0.05	0.01	0.05	0.02
50×10	0.97	0.67	0.81	0.54
50×20	1.84	1.22	1.57	1.14
100×5	0.06	0.02	0.04	0.00
100×10	0.35	0.23	0.22	0.13
100×20	1.92	1.53	1.80	1.48
200×10	0.22	0.17	0.25	0.18
200×20	1.64	1.52	1.59	1.48
average	0.69	0.50	0.61	0.46

Pan, Tasgetiren and Liang [18] have recently proposed a new PSO algorithm called discrete particle swarm optimization (DPSO). It is completely a different version of the PSO proposed earlier, though it still follows the general PSO scheme. First of all, the algorithm makes use of some operators more typical for genetic algorithms like crossover (based on simple two-cut crossover) and mutation (insert mutation is used). The best performing variant of DPSO also uses a local search procedure very similar to the iterated local search procedure presented in Section 2.2 above.

5. Summary

The most recently proposed hybrid algorithms for the flowshop scheduling problem, based on three different biologically-inspired metaheuristics, have been investigated. The aim of the conducted experiments was to asses the performance margin of the main search scheme of a metaheuristic over its local search procedure, theoretically used merely to improve to the main algorithm.

The presented results indicate that there is no gain from using hybrid biologically-based metaheuristics compared with pure neighbourhood search methods. Algorithms based on metaheursitics are usually more difficult to implement and computationally more expensive.

The above conclusion is further confirmed by a careful study of the latest papers concerning the application of metaheursitics in flowshop scheduling problems. A discrete version of particle swarm optimization presented in [18] performs almost equally well as the iterated greedy (IG) algorithm proposed recently by Ruiz and Stueltzle [19]. The latter is yet another local search-based metaheuristic, very similar to the iterated local search presented in Section 2.2 above. Instead of performing several mutations of the sequence (the *modification phase* of ILS), it removes some randomly chosen jobs from the sequence (the so-called *destruction phase* of IG) and reinserts them using the NEH procedure (the so-called *construction phase*). Likewise, IG outperforms the HGA_RMA algorithm, also developed by Ruiz.

Nevertheless, it is the author's opinion that the above conclusion may not be necessarily true for more complex scheduling problems, *e.g.* those with limited

resources or time windows. Nature-inspired metaheuristics, including biologically-inspired ones, may prove their real value if the search algorithm has to deal with many different constraints or problems of very large size (see the case of PACO and ILS_JIB in Section 3.3 above). This will be the subject of the author's forthcoming experiments.

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

Acknowledgements

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

References

[1] Garey M R, Johnson D S and Sethi R 1976 *Math. Oper. Res.* **1** 117
[2] Ruiz R and Maroto C 2005 *Eur. J. Oper. Res.* **165** 479
[3] Ruiz R, Maroto C and Alcaraz J 2006 *OMEGA* **34** 461
[4] Rajendran C and Ziegler H 2004 *Eur. J. Oper. Res.* **115** 426
[5] Tasgetiren M F, Liang Y-C, Sevkli M and Gencyilmaz G 2007 *Eur. J. Oper. Res.* **177** 1930
[6] Chen C-L, Vempati V S and Aljaber N 1995 *Eur. J. Oper. Res.* **80** 389
[7] Reeves C 1995 *Comput. Oper. Res.* **22** 5
[8] Iyer S K and Saxena B 2004 *Comput. Oper. Res.* **31** 593
[9] Nawaz M, Enscore E and Ham I 1983 *OMEGA Int. J. Manage. Sci.* **11** 91
[10] Stuetzle T 1998 *Technical Report AIDA-98-04 FG*, Intellektik, TU Darmstadt
[11] Taillard E 1993 *Eur. J. Oper. Res.* **64** 278
[12] Taillard E 2005 *Summary of Best Known Lower and Upper Bounds for Taillard's Instances*, http://ina2.eivd.ch/collaborateurs/etd
[13] Dorigo M, Maniezzo V and Colorni A 1996 *IEEE Trans. Syst. Man, and Cybernetics* **26** 29
[14] Stuetzle T 1998 *Proc. th Eur. Congress on Intelligent Techniques and Soft Computing*, Verlag Mainz, pp. 1560–1564
[15] Taillard E 1990 *Eur. J. Oper. Res.* **47** 65
[16] Kennedy J and Eberhart R C 1995 *Proc. IEEE Int. Conf. on Neural Networks, Piscataway*, pp. 1942–1948
[17] Mladenovic N and Hansen P 1997 *Comput. Oper. Res.* **24** 1097
[18] Pan Q-K, Tasgetiren M F and Liang Y-C 2007 *Comput. Oper. Res.* (to be published)
[19] Ruiz R and Stuetzle T 2007 *Eur. J. Oper. Res.* **177** 2033