Stochastic Algorithms for Optimization and Application to Job-Shop-Scheduling

D. Duvivier[⋆] and Ph. Preux[⋆⋆] and E-G. Talbi[⋆⋆⋆]

Abstract. Combinatorial optimization problems are often used to test heuristics. Among heuristics, stochastic ones deserve particular consideration being generally meta-heuristics that aim at performing reasonnably well on a wide spectrum of problems. Among them, evolutionary algorithms have recently appeared. Emphasis have been put on them by researches that have shown that they are able to solve efficiently a wide range of problems, among them combinatorial optimization problems. It is however not easy to compare meta-heuristics between each others, and with other more classical heuristics. However, this kind of cross-paradigm comparisons are much worthy. In this paper, we perform a comparison of three kinds of algorithms (multi-start hill-climber, Tabu search, and evolutionary algorithm) on the job-shop-scheduling problem using a series of well-known standard benchmarks. Not surprisingly, hybrid algorithms perform better than pure algorithms. However, it is noteworthy that a multi-start hill-climber often outperforms the other algorithms.

1 Introduction

Various stochastic meta-heuristics have been proposed which have generally proven to be reasonnably efficient on various problems, among them, combinatorial optimization problems. However, we think that current studies encompass two drawbacks that are related : first, being meta-heuristics, these algorithms need some parameter tuning (and these parameters are not only numerical ones, but also encompass procedures to move in the search space) which is not easy to perform in a quite thorough or objective manner; second, comparisons between algorithms on a fair basis is not so often encountered and is really difficult to perform. Expertize in the algorithms that are to be compared is required for the comparison to be meaningful, a good benchmark is needed, and an expertise knowledge is also required on the problems that are used to be able to perform comparisons with state-of-the-art heuristics dedicated to this problem. Even if we restrict the comparison to one kind of problems, it is still not easy to achieve the goal.

In our research group (Perform), we aim at performing this kind of cross-paradigm comparison on a small set of problems (job-shop-scheduling, quadratic assignment, and traveling salesman), and a variety of stochastic algorithms, the

⋆ Laboratoire d'Informatique du Littoral, BP 689, 62228 Calais Cedex, France, duvivier@lil.univ-littoral.fr

⋆⋆ Laboratoire d'Informatique du Littoral, preux@lil.univ-littoral.fr

⋆⋆⋆ Laboratoire d'Informatique Fondamentale de Lille, URA CNRS 369, Cité scientifique, 59655 Villeneuve d'Ascq Cedex, France, talbi@lifl.fr

very core of them being evolutionary algorithms. Thus, we aim at comparing them, studying their hybridization and how they can bring something to each other on a cooperation basis. We also put a particular emphasis on simple algorithms that are able to achieve good performance using only small amounts of computational ressources. Following this line of work, we wish to assess the real capabilities of evolutionary algorithms which are not yet so clear.

In this paper, we will restrict ourselves to only one problem, namely the job-shop-scheduling on which we have obtained some significant results. In section 2, we will precise the definition of the JSP we use as well as various ways of representing solutions. In section 3, we detail the algorithms we have used so far. Section 4 gives our current results compared with the best results known so far. To be able to compare our results with others, we used the ORLIB benchmarks [Orl], that is a set of around 150 problems of size ranging from simple instances to very difficult ones. Finally, we will discuss these results and give our perspectives.

2 Definition of the JSP

There are various definitions of the JSP. Hence, before going any further, we define the JSP we are interested in this paper.

Definition 1 (JSP) *Given a set $\mathcal{J}$ of J jobs. Each job $\mathcal{J}_i$ may be realized using one plan of work among a set of possible plans $\{\mathcal{P}_{ij}\}$. Each plan $\mathcal{P}_{ij}$ is composed of a set of operations $\{\mathcal{O}_{ijk}\}$ among which there exist constraints of fabrication, resulting in a total order relation $<$ between the $\mathcal{O}_{ijk}$ of each plan $\mathcal{P}_{ij}$. Finally, each operation $\mathcal{O}_{ijk}$ may be performed on any machine of a subset of all M machines $\mathcal{M}_{ijk} = \{(\mathcal{M}_{ijkl,t_{ijkl}})\} \subset \mathcal{M}$ in time t_{ijkl}.*

Definition 2 (simple JSP) *A JSP is said to be simple if there exists one and only one plan to achieve any one job and if all the operations may only be performed on a sole machine.*

Definition 3 (extended JSP) *A JSP is said to be extended if there exists several plans to achieve one job, or if the operations may be performed on different machines.*

In this paper, we focus ourselves on the simple JSP. In this case, each job has one and only one operation that has to be performed on each machine. The following conditions should be fulfilled:

$\mathcal{C}_1$: all jobs have to be realized;
$\mathcal{C}_2$: any job is realized according its plan;
$\mathcal{C}_3$: a plan is accomplished by realizing the operations it is composed of;
$\mathcal{C}_4$: there is only one machine that is able to perform any operation;
$\mathcal{C}_5$: the operations should be performed in a predefined order. Hence, the operations of a job can not be executed concurrently on two different machines;
$\mathcal{C}_6$: a machine can only perform one operation at a time.

We assume that the time of execution of a job is solely the sum of the times of the execution of the operations of the job, and the waiting times. The goal is then to minimize this total time elapsed between the beginning of the execution of the first operation that is scheduled to the end of execution of the last scheduled operation. This total time is called the makespan.

3 Representation of data for the JSP

In this section, we discuss various representations of data that may be handled by the algorithms that we are focussing on. We restrict ourselves to the simple JSP as long as we have only experimented with this class of problems yet.

Basically, two classes of representations of data may be envisaged for the JSP, these two schemes of representation being qualified either as direct, or indirect. Clearly, the goal of the JSP is to obtain a schedule of occupation of the machines which indicates for each time slot of any machine if it is free or, if not, which operation of which job is currently performed. We review several representations of each kind and introduce a new direct representation that we use.

3.1 Indirect representation

When using an indirect representation, the data structure is not a schedule of occupation of the machines. A decoder is required to express the schedule given the data structure. According to the information that is present in the data structure, the decoder has more or less to work to be able to derive a schedule. Constraints are handled by the decoder and guarantees the validity of the schedule that is derived. The decoder may be non deterministic.

Representation 1

In this representation, the data structure is simply an array of J elements, each one being the number of one job. The entire structure is a permutation of the integers comprised between 1 and J [Bru93] [BUMK91].

0	...	$J-1$
Job i	...	Job j

The search space is limited to the set of the permutations of J integers, that is of size $J!$. The decoder has a very limited set of information and should derive much more to obtain valid schedules. Various decoders may be imagined, with a variable degree of stochasticity. A very simple one would consider the data structure as a priority list and would derive a schedule that always gives the priority to the operations belonging to the most prioritary jobs.

Representation 2

In this case, the data structure is an array of $J \times M$ entries. Each job is assigned a class of markers. All the markers associated to one job have the same tag (the job number in our example, below). The markers are then shuffled in the array. We give an example of such a representation for a 4×4 JSP:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	3	1	0	2	3	2	1	0	3	1	2	2	3	0

The decoder considers this array from the "left to the right". For each entry, it then schedules as soon as possible the next not-yet scheduled operation of the job associated to the marker that has been found.

Representation 3

In this case, the data structure is an array of J elements, each one being constituted of a list of allocations of machines on which the operations are to be executed.

	0	1	...	$M-1$
Job	Op7	Op3	...	...
i	m2	m3	...	...
...	...	...	...	...
...	...	...	...	...
Job	Op6	Op4	Op1	...
j	m2	m4	m1	...

In this case, the decoder has to allocate time slots to the operations [BUMK91] [Bru93]. Lines may be shuffled to change the priorities between jobs.

3.2 Direct representation

In the case of a direct representation, the data structure actually represents a schedule of occupation of the machines. Hence, no decoder is required.

Representation 4

For each job, the list of machines and the time slots that are used to perform the operations are given [Bru93].

	0	1	... $M-1$
Job	Op7	Op3	...
i	m2	m3	...
	1,3	13,17	...
...	...	...	
Job	Op6	Op4	Op1
j	m2	m4	m1
	20,22	34,36	51,55

For instance, the job i is constituted of the operations Op7 and Op3. Operation Op7 is executed on machine m2 from time 1 to time 3. The operation Op3 is performed on machine m3 from time 13 to 17, ...

With regards to representation 3, time slots have been added. The assignment of an operation consists in the association of a machine as well as a time slot taking in consideration precedence constraints. The order of execution of operations is defined at the level of operations.

Representation 5

In order to enhance the exploitation of the latest representation, we introduce a new one. It is a two-dimensional representation, one dimension ranging among jobs, the other one among machines. For each machine, we have the list of operations that are performed with their time slot. For each job, we have, just as in representation 4, the list of operations tagged with their time slot.

	Op8	...	Op1	...
$\mathcal{M}_k$	Job$i$	...	Job$j$	...
	1,10	...	51,55	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
	Op5	...	Op4	...
$\mathcal{M}_l$	Job$i$	...	Job$j$	...
	11,31	...	34,36	...

This representation makes it easier to detect available time slots. This characteristic will be exploited.

4 Algorithms

For the moment, we have mainly worked with three classes of primitive algorithms:

1. hill-climber
2. Tabu search
3. evolutionary algorithm

We have also investigated hybrid algorithms tinkered with these three algorithms as building-blocks.

The hill-climber is considered as a very simple heuristic that does not require heavy computational ressources, but which may easily be trapped in a local optimum. We also used a variant of the basic hill-climber, namely a multi-start hill-climber.

The Tabu search may be seen as a sophisticated hill-climber which may escape from a local optimum [Glo89a, Glo89b]. We should emphasize the fact that we use a very rudimentary version of Tabu algorithm. Hence, we have mainly encountered the (classical) problems of the sizing of the Tabu list, and that of the handling of a big neighborhood. Other mechanisms such as intensification and diversification have not been used for the moment.

Evolutionary algorithms (EAs for short) are meta-heuristics based on J. Holland's genetic algorithms [Hol75] with which we, and others, have obtained good results on various NP-hard problems such as graph partitioning [TB91], set partitioning [Lev94], graph coloring [FF95b]. We will suppose here that the basic scheme of evolutionary algorithms is known by the reader or propose him to read [FF95a] before going on any further which, apart a presentation of evolutionary algorithms, address the problems of their use for combinatorial optimization: the presentation of evolutionary algorithms and all subtulties that are required to know to perform efficient combinatorial optimization with them would require more than the volume of this whole paper. The issues are representation of solutions, operators, and hybridization schemes.

Let us now turn to a detailed presentation of these algorithms.

4.1 Multi-start Hill-climber

Starting at a random point, the hill-climber tries to move uphill. When, after a certain amount of attempts, it always fails to climb, the current best point is recorded and the climbing is restarted from another point in the space, chosen at random. To be more precise, it performs the algorithm displayed in figure 1.

In the work reported here, `number_of_periods` has been set to 20.

The delicate part in using a hill-climber lies in the choice of the move operator. Using the representation 2, the basic move turns out to be a swap of markers.

The decoder takes the elements of the array, from the left to the right (in that strict order) and schedules the corresponding operation as soon as possible. If possible, it uses a free time slot. If not, it is scheduled at the end of the current schedule.

4.2 Tabu search

The Tabu search has only been used with the direct representation 5. The move swaps two operations of two different jobs. The size of the neighborhood of each point is $JM(J-1)/2$ which is quite cumbersome. The move swaps two operations of two different jobs. The Tabu searches the whole neighborhood to find the best non-taboo move before actually moving, while the hill-climber takes the first favorable move, a strategy that shortens the search a lot. We also use an aspiration criterion that accepts a movement even though it is taboo if it leads to the best ever found schedule (in this case, though accepted, the movement remains taboo).

We have also begun to experiment a variant of the Tabu that only considers a restricted neighborhood, chosen stochastically. In this case, only $k\ JM$ neighbors are taken into consideration. The movements are selected in order to reach uniformally spared points in the neighborhood. We set k to 2 in our current experiments.

The size of the taboo list is a critical parameter (since we use a fixed size of this list) which has been set empirically, after numerous trials. The taboo list size depends on the size of the problem (see table 1 for more details).

```
current_schedule := random_schedule
best_ever_found_schedule := current_schedule
best_schedule := current_schedule
for i := 1 to number_of_periods do
  for j := 1 to threshold do
    candidate_schedule := perform move from current_schedule
    if candidate_schedule is better than current_schedule then
      current_schedule := candidate_schedule
    fi
  done
  if current_schedule is better than best_schedule then
    best_schedule := current_schedule
    if current_schedule is better than the best_ever_found_schedule then
      best_ever_found_schedule := current_schedule
    fi
  else /* Trigger a new experiment */
    best_schedule := current_schedule := random_schedule
  fi
done
output ("I have found " best_ever_found_schedule)
```

Fig. 1. The multi-start hill-climber algorithm used. `current_schedule` is the makespan of the schedule that is currently being worked on. `best_schedule` is the makepsan of the best schedule found in the current period. `best_ever_found_schedule` is the makespan of the best schedule ever found since the algorithm has begun its execution. The value of the upper bound `threshold` is given in the caption of table 1.

4.3 Evolutionary algorithm

Operators As for the TABU, the EA only works with a direct representation and we have used the representation 5.

Crossover We use a recombination operator largely inspired by the GA/GT crossover introduced in [NY92]. This operator relies on Giffler and Thompson's algorithm [GT69] to produce an offspring given two parents. The principle of the GT algorithm is as follows:

- $C \leftarrow$ first operation (which is not yet scheduled) of each job compute ECT of operations $\in C$
- repeat
  1. select $O^* \in C$ which has minimum ECT operations $\in C$ sharing the same machine with O^* and conflicting with O^* (processing overlaps)
  2. choose an operation $O^{**} \in G$ at random
  3. schedule O^{**} as soon as possible according to ECTs
  4. update C and ECTs
- until all operations are scheduled

```
current_schedule := random_schedule
best_ever_found_schedule := current_schedule
do
  apply usual TABU search during threshold iterations
  if current_schedule is better than best_ever_found_schedule then
    best_ever_found_schedule := current_schedule
  fi
while best schedule is enhancing enough
output ("I have found " best_ever_found_schedule)
```

Fig. 2. The TABU algorithm that is used in this paper. The variable `threshold` has been set according to the size of the problem and is given in table 1.

Problem size	Size of the taboo list	Threshold
less than 10×10	23	$1.8\,10^4$
10×10, 15×5, 20×5	67	$3\,10^4$
15×10	97	$4.5\,10^4$
15×15	147	$6.75\,10^4$
20×10	131	$6\,10^4$
20×15	197	$9\,10^4$
20×20	257	$1.2\,10^5$
30×10	197	$9\,10^4$
30×15	289	$1.35\,10^5$
30×20	387	$1.8\,10^5$
50×15	481	$2\,10^5$
50×20	641	$3\,10^5$
100×20	1279	$4\,10^5$

Table 1. The size of the taboo list and the threshold (with regards to the algorithm given in figure 2) that are used in our experiments for different sizes of JSP problem. For the multi-start hill-climber, the value of `threshold` is multiplied by 10.

where

C is a "cut"
G is a set of operations conflicting with O^*
ECT means Earliest Completion Time

Mutation The mutation performs a swap of two operations. To lead to a valid schedule, the swap should only consider operations of two different jobs. It works as follows:

1. choose a machine at random
2. choose two operations on this machine

3. among these two operations, compute the date of the earliest scheduled operation and copy the operations that are scheduled before this date in the result
4. swap the two operations chosen at step 2
5. schedule, as soon as possible, all the operations that are not yet scheduled

Basic scheme of the EA Grounded on experimental evidences, we use a different scheme of application of the operators than usually used. More precisely, the EA performs as follows:

- initialize population at random
- while completion criterion is not fulfilled do
  - reproduce
  - apply operators on offsprings
  - operated offspring form the next population

The operators are applied as follows:

1. pick up two individuals χ_1 and χ_2 in the population
2. perform cross-over on them and stochastically (with bias 0.75) keep the best of the two resulting individuals: χ_1'
3. perform mutation on one individual among χ_1 and χ_2 choosing one or the other at random. This results in χ_2'.
4. put χ_1' and χ_2' in the population of offsprings

This non-conventional scheme of application of operators has proven to perform better.

4.4 Hybrid algorithms

Various hybridization schemes are possible (see [PT95]). We have currently experimented one scheme on the JSP, namely the synchronous hybridization which uses a heuristic as a mutation operator. In this case, the hill-climber is used. The basic scheme of the hybrid algorithm is as seen before except for the application of operators which proceeds as follows:

```
if (current generation is odd) then
  apply GA/GT crossover and swap mutation
else
  apply heuristic
fi
```

Again, this choice is grounded on previous experimentations.

Only direct representations have been used in the case of hybrid algorithms (all algorithms then working on the same representation). The use of indirect representation has not proven to be as efficient as the direct representation.

The EA+hill-climber hybrid We have experimented a hill-climber, which is no longer a multi-start hill-climber. It iterates a certain amount of moves, starting from a schedule which is the individual it is applied on. The amount of moves that are allowed while there is no improvement is set to 50. When this number of iterations is reached, the algorithm continues to move as long as it climbs up.

Parameters The following table displays the value of the parameters that have been used for the EA and the hybrid EA:

Population size	300
Reproduction strategy	ranking (best individual has probability 1.25 to produce offspring(s), the worst has probability 0.75)
Selection	generational and elitism
GA/GT application rate	0.6
Swap mutation application rate	1.0
Hill-climber application rate	0.8

5 Results

We now turn to the results we have obtained with previously reviewed algorithms (*cf.* table 2). In our benchmark suite, we use the Muth and Thompson's problems, the 8 Carlier's problems, as well as 6 Lawrence's problems, and 5 problems of big size that have been generated using the problem generator of the ORLIB[4]. The size of the problems ranges from 6×6 to 100×20. In all cases, the table displays the best makespan that has been found.

Generally speaking, the hybrid EA+HC always performs better than the EA alone. The hybrid has found the optimal schedule for some problems (MT6×6, car1, car2, and car4) while the EA did only find the optimum for the mt6×6 which is found by all the algorithms.

For their parts, the hybrid EA, MSHC, TABU and RNST all achieve the same kinds of results, lying within a few percents from the optimal schedule. Taking a glance at table 3 reveals that, when considering the distribution of points found in a series of experiments with a given algorithm, it clearly appears that the standard deviation of optimal schedules is very different from one algorithm to another, and from one series of problem to another. The MSHC generally has a low to moderate standard deviation, the EA and the hybrid EA a moderate one, the TABU based heuristic (TABU itself and RSNT) quite a high one. Furthermore, for all heuristics, the standard deviation obtained on the Carlier's problem is much higher than the standard deviation obtained for Muth and Thompson's and Lawrence's problems. Hence, on the test-suite, the MSHC always finds the same quality of makespans, while the TABU finds various quality of schedules.

Concerning the mean time of execution of the various algorithms, the very short time of execution the MSHC is noteworthy. The EAs (both pure and hybrid) have

[4] these problems are all available via anonymous `ftp` on: `ftp@ftp.univ-littoral.fr:pub/users/duvivier/benchmarks/jsp`

Problem	Size	optimum	EA	EA+HC	MSHC	Tabu	RSNT
mt6x6	6x6	55	55	55	55 (0)	55 (0)	55 (0)
mt10x10	10x10	930	953 (2)	946 (2)	947 (2)	930 (0)	949(2)
mt20x5	20x5	1165	1180 (1)	1180 (1)	1175 (<1)	1165 (0)	1178(1)
car1	11x5	7038	7101 (<1)	7038 (0)	7038 (0)	7038 (0)	7038 (0)
car2	13x4	7166	7576 (6)	7166 (0)	7166 (0)	7166 (0)	7166 (0)
car3	12x5	7312	7594 (4)	7399 (1)	7312 (0)	7422 (1)	7543 (3)
car4	14x4	8003	8423 (5)	8003 (0)	8003 (0)	8003 (0)	8163 (2)
car5	10x6	7702	8047 (4)	7779 (1)	7738 (≈0)	7822 (2)	7854 (2)
car6	8x9	8313	8699 (5)	8562 (3)	8313 (0)	8438 (1.5)	9118 (10)
car7	7x7	6558	–	–	6558 (0)	6573 (≈0)	79 (3)
car8	8x8	8264	–	–	8294 (≈0)	8407 (2)	8407 (2)
la01	10x5	666	–	–	666 (0)	666 (0)	666 (0)
la02	10x5	655	–	–	655 (0)	655 (0)	655 (0)
la03	10x5	597	–	–	597 (0)	597 (0)	597 (0)
la04	10x5	590	–	–	590 (0)	590 (0)	590 (0)
la05	10x5	593	–	–	593 (0)	593 (0)	593 (0)
abz7	20x15	(654, 668)	757	690	682	–	–
la36	15x15	1268	1408 (11)	1297 (2)	1281 (1)	–	–
	20x20	(1217, 1663)	1968	1788	1725	1778	–
	30x15	(1764, 1770)	2098	1916	1807	–	–
	30x20	(1850, 2064)	2566	2262	2143	–	–
	50x15	2760	3225 (17)	2982 (8)	2760 (0)	3071 (11)	–
	100x20	5464	6065 (11)	5757 (5)	5755 (5)	–	–

Table 2. This table sums up our results. The problems are identified with their name when one is known. The unnamed problems have been generated with the JSP problem generator of the Orlib and are available on our `ftp` site. The second column gives the size of the problem. The third one gives the value of the optimum if it is known, a range of it when it is unknown. The fourth column gives the best result obtained using the evolutionary algorithm. The fifth column gives the result obtained with the hybrid EA+hill-climber (EA+HC), the sixth with the multi-start hill-climber (MSHC), the seventh with the Tabu, the eighth with the restricted stochastic neighborhood Tabu (RSNT). In (), we have indicated the percent from the optimum the best found schedule lies in. All algorithms have been run from 10 to 20 experiments.

moderate times of execution. The TABU based heuristics requires much more computation efforts. It is particular noteworthy that the EAs were run during 2000 generations. However, the best schedule that the algorithm is able to find in a run is usually synthesized during the first hundred generations. Hence, that means that the EA can be stopped after a hundred generations without losing, most of the times, the best solution that it would have been able to find in the run. It also means that we can perform 20 runs in the time of 1, and have much more chance to obtain good solutions while using the same amount of computational resources.

As a general remark, the EA generally quickly finds a good solution but it is unable to enhance it so quickly. The TABU is able to find a good solution, though slowly. Hence, hybridization of the EA with TABU would be able to yield good

Problem	EA	EA+HC	MSHC	TABU	RSNT
car1	17	3	0.0	76	68
car2	19	24	0.0	77	67
car3	6	10	5	50	23
car4	8	19	1	64	51
car5	22	8	14	31	43
car6	5	7	20	45	30
la01	–	–	0.0	2	0
la02	–	–	8	9	9
la03	–	–	3	20	22
la04	–	–	3	5	–
la05	–	–	0.0	0.0	–
mt6 $\times$ 6	0.0	0.0	0.0	0.0	0.0
mt10 $\times$ 10	2	13	17	29	–
mt20 $\times$ 5	1	11	10	–	–

Table 3. This table gives the rate between the standard deviation and the value of the optimum for the makespan of the schedules that were found in 10 to 20 experiments. The values have been multiplied by 1000.

Problem	EA	EA+HC	MSHC	TABU	RSNT
MT6 $\times$ 6	46"	4"	0",1	8",5	2"
MT10 $\times$ 10	25'	18'	1'20"	48'	28'
MT20 $\times$ 5	18'	22'	2'4"	3h43	–

Table 4. Order of magnitude of the time required to find the best schedule (timings have been done on SGI-Indy workstations; these are wall-clock times).

results.

In table 5, we have summarized the results obtained by several authors on the Muth and Thompson's problems, mainly using evolutionary algorithms. We have only taken into consideration works with pure EAs with random initial population.

The Branch-and-Bound (B&B) is an exact method. Carlier and Pinson [CP89] have obtained the optimal solutions using this method.

Using an AE with an indirect representation, Nakano et Yamada [NY91] have obtained the optimal solution to the MT6$\times$6. Nakano et Yamada [NY92] have obtained the optimal solution to MT10$\times$10. They had turned to a direct representation and the GA/GT as the recombination operator. The optimal schedule of the MT20$\times$5 has not been found using the same algorithm. It should be noted that the optimal schedule of the MT10$\times$10 has only been found in 4 experiments out of 600. [KOY95], using a new operator, have obtained the optimum of the mt10 $\times$ 10 51 times out of 100 experiments, that is much more often than Nakano and Yamada. It should also be emphasized that among the reported results, only Kobayashi *et al* [KOY95] work solely with active plannings.

Reference	Algo-rithme	MT 6 × 6	MT 10 × 10	MT 20 × 5
Carlier and Pinson [CP89]	B&B	55	930	1165
Nakano and Yamada [NY91]	EA	**55**	965	1215
Nakano and Yamada [NY92]	EA	55	**930**	1184
Dorndorf and Pesh [DP92]	–	55	938	1178
Fang, Ross and Corne [FRC93]	EA	–	949	1189
Juels and Wattenberg [JW94]	EA	–	937	1174
Soares [Soa94]	EA	58	997	–
Kobayashi *et al* [KOY95]	EA	–	930	–
Our results	EA	55	953	1180
Optimum		**55**	**930**	**1165**

Table 5. This table displays the results obtained by different authors on the Muth and Thompson problems. The first column gives the references to the work. The second column indicates the kind of algorithm that have been used. The three subsequent columns give the results that are reported (best found makespan) for the 3 problems. [NY91] uses an indirect representation. [NY92] uses a direct representation with the GA/GT operator. [JW94] uses the representation 2. They use an other recombination operator.

Using an AE, only Nakano and Yamada [NY92] and [KOY95] have obtained the optimum of the MT10×10. No one has obtained the optimum of the MT20×5.

It is noteworthy that for the big problems, the research space is huge and only a small fraction of it is visited by the algorithm.

It is also noteworthy that using EAs with a population 2, or 4 times bigger does not improve the quality of best found schedules. The rapid loss of diversity probably accounts for this fact.

We would like to put an emphasis on the performance achieved by the (multi-start) hill-climber. The best found schedules are either the optimal schedule or very close to it. This prompts us with two remarks. First, the design of the hill-climber is very simple while the design of the EA is not so easy (notably because of the recombination operator). Second, the basic movement of the MSHC is the mutation that was used in the EA. Hence, this questions the efficiency of the GA/GT operator.

6 Discussion and perspectives

In this paper, we have compared different stochastic meta-heuristics on a test-suite composed of job-shop-scheduling problems, ranging form small size to big size problems. The meta-heuristics are hill-climbers, TABU search and evolutionary algorithms. Best schedules that were found, standard deviation, times to obtain this schedule are reported for 5 algorithms. Hybrid algorithm are observed to achieve better results than pure algorithms. Furthermore, a simple multi-start hill-climber often obtains very good solutions. For this kind of algorithms, the trade-off between its design and its efficiency is then very good.

Given the results obtained with the pure EA and the hybrid EA+hill-climber, we are prompted to question the efficiency of the recombination operator as long as this is the real difference between these two algorithms. This observation is radically different from the one we obtained on the set partitioning problem where the EA worked much better (both with regards to the solution that was obtained and the time to obtain it) than several kinds of hill-climbers (see [Tal93]). Hence, further studies on the role of the recombination are urgently required.

As it was observed in our experiments, different classes of problem hardness seem to exist. Very pragmatically, we would like to say that a problem is easy if it can be solved optimally with a simple algorithm. This way, we optimize the trade-off between the time of design of the algorithm and the quality of solutions it is able to find. Furthermore, the multi-start hill-climber has also proved to be very fast to find optimal solutions on a series of problems.

In the forthcoming future, we will increase the number of experiments in order to obtain representative distributions of the solutions that are found by the studied algorithms. We will enhance our Tabu equipping it with an adaptive size of the taboo list as is usually done. We will also go further with the restricted neighborhood version RSNT. Test of the hill-climber with a direct representation is also needed to be able to fuel the direct versus the indirect representation debate. We also strongly support the view of using "smart decoders" for the indirect representation to give better results.

In the mid-term, we will further study other kinds of hybridizations, and pay some attention to other problems.

References

[BB91] Richard K. Belew and Lashon B. Booker, editors. *Proc. of the Fourth International Conference on Genetic Algorithms*, La Jolla, CA, USA, July 1991. Morgan Kaufmann, San Mateo, CA, USA.

[Bru93] Ralf Bruns. Direct chromosome representation and advanced genetic operators for production scheduling. In *[For93]*, pages 352–359, 1993.

[BUMK91] Sugato Bagchi, Serdar Uckun, Yutaka Miyabe, and Kazuhiko Kawamura. Exploring problem-specific recombination operators for job shop scheduling. In *[BB91]*, pages 10–17, 1991.

[CP89] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.

[DP92] U. Dorndorf and E. Pesh. Evolution-based learning in a job shop scheduling environment. Technical Report RM 92-019, Rijksuniversiteit Limburg, The Netherlands, 1992.

[Esh95] Larry J. Eshelman, editor. *Proc. of the Sixth International Conference on Genetic Algorithms*, Pittsburgh, PA, USA, July 1995. Morgan Kaufmann, San Mateo, CA, USA.

[FF95a] Charles C. Fleurent and Jacques A. Ferland. Algorithmes génétiques hybrides pour l'optimisation combinatoire. *RAIRO*, 1995.

[FF95b] Charles C. Fleurent and Jacques A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operation Research*, 1995.

[For93] Stephanie Forrest, editor. *Proc. of the Fifth International Conference on Genetic Algorithms*, Urbana-Champaign, IL, USA, July 1993. Morgan Kaufmann, San Mateo, CA, USA.

[FRC93] Hsiao-Lan Fang, Peter Ross, and Dave Corne. A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In *[For93]*, pages 488–493, 1993.

[Glo89a] F. Glover. Tabu search — part I. *ORSA Journal of Computing*, 1(3):190–206, 1989.

[Glo89b] F. Glover. Tabu search — part II. *ORSA Journal of Computing*, 2(1):4–31, 1989.

[GT69] B. Giffler and G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1969.

[Hol75] John H. Holland. *Adaptation in Natural and Artificial Systems.* Michigan Press University, Ann Arbor, MI, 1975.

[JW94] Ari Juels and Martin Wattenberg. Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. Technical Report csd-94-834, University of California, 1994.

[KOY95] Shigenobu Kobayashi, Isao Ono, and Masayuki Yamamura. An efficient genetic algorithm for job shop scheduling problems. In *[Esh95]*, pages 506–511, 1995.

[Lev94] David Levine. *Parallel Genetic Algorithms for the Set Partioning Problem.* PhD thesis, Argonne National Laboratory, Maths and Computer Science Divivsion, May 1994. report ANL 94/23.

[MM92] R. Männer and B. Manderick, editors. *Proc. of the Second Conf. on Parallel Problem Solving in Nature.* Elsevier Science Publishers, Amsterdam, 1992.

[NY91] Ryohei Nakano and Takeshi Yamada. Conventional genetic algorithm for job shop problems. In *[BB91]*, pages 474–479, 1991.

[NY92] Ryohei Nakano and Takeshi Yamada. A genetic algorithm applicable to large-scale job-shop problems. In *[MM92]*, pages 281–290, 1992.

[Orl] Orlib. available via anonymous ftp on `mscmga.ms.ic.ac.uk:/pub`.

[PT95] Ph. Preux and E-G. Talbi. Assessing the evolutionary algorithm paradigm to solve hard problems. In *Constraint Processing, workshop on really hard problem solving*, September 1995.

[Soa94] C. Soares. Evolutionary computation for the job-shop scheduling problem. Technical Report UU-CS-1994-52, Utrecht University, The Netherlands, December 1994.

[Tal93] E. G. Talbi. *Allocation de processus sur les architectures parallèles à mémoire distribuée.* PhD thesis, Institut National Polytechnique de Grenoble, Mai 1993.

[TB91] E-G. Talbi and P. Bessière. A parallel genetic algorithm for the graph partitioning problem. *ACM Int. Conf. on Supercomputing, Cologne, Germany*, June 1991.

This article was processed using the LaTeX macro package with LLNCS style