# A steady-state evolutionary algorithm for the job shop problem

**Patrick Van Bael, Dirk Devogelaere, Marcel Rijckaert**

K.U. Leuven - Chemical Engineering Department

De Croylaan 46, 3001 Heverlee, Belgium

{Patrick.vanbael}@cit.kuleuven.ac.be

## Abstract

This paper describes an evolutionary search scheduling algorithm (ESSA) for the job shop scheduling problem (JSSP). If no specific knowledge of the problem is included in the EA then it is less likely that an EA will find the global optimum of a JSSP. However, the proposed steady state ESSA can find global optima if a particular population and offspring size is utilized. Adding specific knowledge through a Lamarckian Learning process improves the performance significantly. In this case, a good balance of the amount of Lamarckian Learning can realize a good performance in both efficiency and effectiveness.

## 1 INTRODUCTION

Many of the practical scheduling problems in industry are simplified and modeled as a *nxm* job shop scheduling problem. Such a problem is described as a set of *n* jobs that are processed on a set of *m* machines. Each job consists of activities to be processed on a machine. Every job has a predefined route through the shop of resources. Each activity must be processed without preemption and each machine can process only one activity at a time. The completion time of a job is the finishing time of its last activity. The maximal completion time of all jobs is the makespan of the schedule and is the criterion to evaluate schedules. The general job shop scheduling problem is NP-hard and can be extremely difficult to solve.

The basic evolutionary algorithm many researchers use to solve the JSSP problem is shown in figure 1. In this algorithm the genotype of the JSSP is used. The genotype is a symbolic representation or encoding of the JSSP problem. A decoding system transfers the genotype in the phenotype. The phenotype is a real JSSP-solution or schedule, which can be executed if needed. This phenotype is used to evaluate the solution. Note that the genotype is in fact the way the JSSP is modeled and used to explore the search space of the real problem.

**Algorithm** ESSA

Initialize population P of size μ

Evaluate μ individuals in P

**While** not termination **do**

Select λ individuals from P

Mate individuals to produce λ offspring

Evaluate λ offspring

Update λ offspring

Insert λ offspring in P

**End while**

**End algorithm**

Figure 1. Evolutionary search template

The evolutionary search begins by initializing a population of individuals and evaluates their fitness. Individual solutions are selected from the population and mated to produce new offspring. Next the offspring are evaluated. After evaluation the offspring's genotype can be updated if necessary. New offspring will be inserted in the population via total or partial replacement of the old population. This cycle is called a generation and is repeated until a predefined termination criterion is met.

The general template (figure 1) can be used as ESSA to solve the JSSP. The most recent ESSAs are hybrid. This means they use a local search algorithm within an ESSA, denoted as Lamarckian Learning. Usually the update function contains the local search algorithm, which tries to locally optimize the created offspring as much as possible before inserting them in the population. The idea is to optimize a population that consists of local optima. Hence the local search procedure focuses on optimizing offspring, while the ESSA can emphasize merging good individuals of the population. The local search algorithm contains specific knowledge of the problem concentrated in special neighborhood structures, which are compatible with the evaluation criterion. A general update function is shown in figure 2.

Genotype g

**Function** Update

p := Decode g

p := Local Optimizer(p)

g := Force (p)

**Return** Evaluate g

**End function**

Figure 2: General update function (g: genotype; p: phenotype)

The update function decodes an offspring into a schedule. If a local optimizer is used, the offspring is optimized accordingly. The best schedule is forced into a new genotype and replaces the initial original genotype. Figure 3 shows the process of the update function.

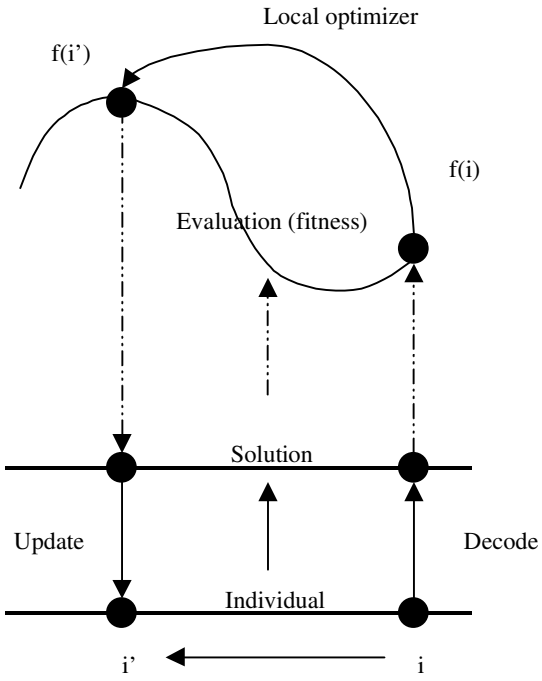


Figure 3: The update function illustrated.

In spite of the improvements of hybrid ESSA's versus classic ESSA's, we first focussed on classic ESSA to see how far this ESSA can reach. In a second experiment we investigate the amount of local search to apply, aiming at optimizing hybrid ESSAs in both efficiency and effectiveness.

The major differences in the applied (Hybrid) ESSAs are (i) the representation of a schedule (genotype), (ii) the general or specific operators used to produce offspring (e.g. crossover and/or mutation), (iii) the decoding algorithm to create a schedule from a genotype and (iv) the update function. Minor implementation differences can be found in the EA management. It consists of the population size, the parent's selection mechanism, the insertion mechanism and the operators' rates.

In the next experiments, we first examine the influence of the population and the offspring size in a steady state ESSA omitting Lamarckian Learning. Both determine the balance between exploration and exploitation. Results on common benchmarks [Vaessens 96] have to illustrate the usefulness of good tuned population and offspring sizes. In a second experiment, we investigate how many offspring should be improved through Lamarckian Learning to have an efficient and effective hybrid ESSA. Results on similar benchmarks should indicate the robustness of this parameter setting.

## 2 OPTIMAL POPULATION AND OFFSPRING SIZE

The experiments apply a steady state EA where overlapping populations occur instead of a generational EA, which replaces the old population by all offspring generated. At each generation, a portion of the old population is replaced by the offspring. At one extreme, only one individual may be replaced each generation. At the other extreme, the steady state EA can become a generational EA when offspring entirely replace the population. The steady-state EA selects a first time individuals for the mating mechanism. It selects a second time individuals to be replaced by the offspring. The replacement strategy allows inserting an offspring into the population only if an acceptance criterion is met. It is ignored otherwise. Two advantages arise for a steady state EA including a replacement strategy. A first evident advantage is that increasing the mating mechanism rates can increase the EA performance. A high rate will not introduce too much random perturbation since offspring does not always pass the replacement strategy. A second advantage is that replacement strategies correct the tight selection pressures independent of the selection mechanism. A high selection pressure focuses exploitation of the better individuals. This correction avoids quick convergence to the better solutions, especially if large offspring sizes are chosen.

The population size is sometimes used to diversify between two common approaches. The evolutionary strategies apply a small population size while genetic algorithms (broadly evolutionary algorithms) use a large population size. The former strategy is aiming at exploitation. The few good solutions should be improved as much as possible. The exploring phase is usually added via other complimentary functions. The latter strategy aims at exploration using a large sample size of the wide search space, hence the large population. Exploitation is realized using a specific mating method. This means the EA can be used in a stand-alone method using no specific knowledge at all.

The chosen approach should avoid specific knowledge, the ESSA should do the work. Consequently, a large population size was chosen. However, this parameter is regarded crucial for the EA performance. If the population size is too small, the knowledge-processing feature (exploitation) is virtually disabled. In addition, the initial population should consist of as much different knowledge (individuals) as possible, which afterwards will be merged effectively to obtain a sub-optimal solution. To obtain a good performance two options are left. First, a large population size can be chosen. Second, a multi-population run with rather small population sizes can be chosen.

In practice, these two approaches will be combined. A large population size will be considered but precautions will be taken that there is convergence towards several local optima instead of one (i.e. niching). However, the population sizes must be limited. There seems to be an asymptotic towards the performance. Therefore, an experiment should indicate appropriate sizes. However, it can only be done if an adequate offspring size is given, which is also missing. A selection mechanism selects a set of parents equal to the offspring size. The size of this subset also regulates the exploration or exploitation. A small subset exploits the knowledge of few individuals. If these are fed back into the population, it will not disturb the mean fitness that much. However, it can focus only on the better individuals to be reproduced, guiding the search towards local optima. This approach is therefore more exploitation directed. On the contrary, large offspring sizes are more focussed on exploration. The offspring will be a combination of more distinct individuals making it difficult to converge to near optimal solutions. However, they initially disturb the mean fitness of the population very much. Hence, the offspring size must be empirically determined.

The ESSA uses a representation of individuals described in [Van Bael $99_a$]. It is an individual using activities as genes and integer numbers as alleles. These integer numbers represent the order in which activities are assigned an earliest start time (i.e. the decoder). Hence an active schedule is build. The operators to be used depend on the representation applied. Therefore, the $n_{max}$-point crossover will be used where $n_{max}$ is equal to the number of jobs in the problem. The mutation operator mutates an allele within an interval of [*0,60*]. Crossover is used in *90%* of the cases otherwise mutation is activated. As selection, a binary tournament scheme is preferred, while an insertion strategy replaces the worst individuals whenever offspring are better than the average of the population. This balances the tight selection pressure keeping a diversified selection possible. Once the offspring are created, new fitness evaluations take place. An update changes the individuals via reverse decoding of the schedule. The EA used a niching technique where the number of same fitness values in the population was at most *100*. The EA incorporated the bootstrapping function when the best individual did not change for *400* generations and lasts for *400* generation. Bootstrapping means a reverse inserting mechanism replacing the best solutions temporally. The total amount of generations was set to *20000* generations. Details of the strategy can be found in [Van Bael $99_a$]. The type of EA which will be most effective (the generational or steady state version) is an open question. An investigation of the population size versus offspring size is necessary.

To determine the offspring and population size, the EA is first tested on the mt10 problem [Muth 63]. Average results of 20 runs were obtained with different offspring and population sizes. Figure 4 and 5 shows the results graphically.

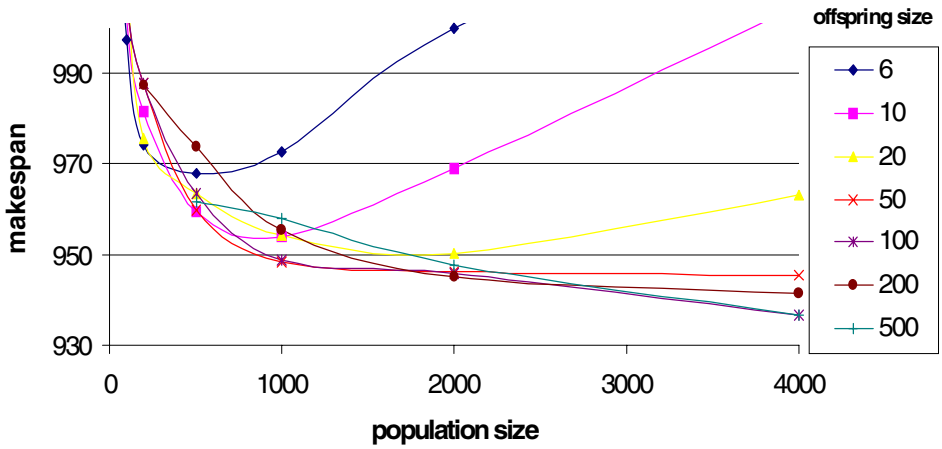


Figure 4: Different population sizes versus average makespan achieved for different absolute offspring sizes for a steady state evolutionary algorithm solving the mt10 problem.

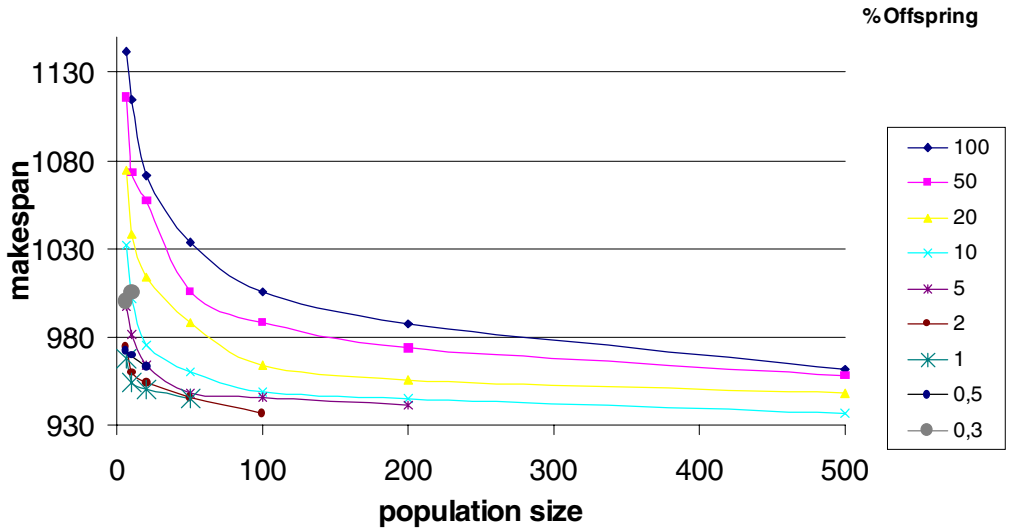


Figure 5: Different population sizes versus average makespan achieved for different relative offspring sizes for a steady state evolutionary algorithm solving the mt10 problem.

Figures 4 and 5 show an interesting result. Small relative offspring sizes and high absolute population sizes are preferred. The large population sizes make it possible to contain as much as possible diversified individuals that can be recombined into near optimal solutions. Figure 5 indicates that one or two percent offspring gives the best results. However, the absolute offspring size is limited by the computation time. Although not illustrated here, the computation time correlates linear with the offspring size. In practice, this means that absolute offspring sizes are limited to 50. Large benchmark problems tend to consume even more computation time, justifying this choice of sizes. Consequently, the population size taken is 2000. Next, the ESSA is applied to a set of common JSSP benchmarks [Vaessens 96]. Table 1 illustrates the results.

The general EA performs well. Moreover, even without specific knowledge it can find optimal schedules within the subset of benchmarks. The ESSA performs well with a mean MRE of 2.24 % within a total CI-CPU time 41707 (MRE and CI-CPU are independent comparison measures introduced in [Vaessens 96]). If one compares the general ESSA algorithms [Bierwirth 95][Norman 97] with the proposed EA then it shows improved performance. It finds more global optima, however the MRE of the best solutions is similar to the one from Norman [Norman 97]. In the class of EA algorithms (hybrid as well) only the hybrid genetic local search algorithms of Mattfeld [Mattfeld 96] and Yamada [Yamada 96] can do better. Nevertheless, the EA does not use specific criterion based operators and will be more generally applicable.

Computation times seemed to be reasonable since runs take only moderate computation times. The total amount of computation time is also the result of the bootstrapping technique that is used. It takes a long time to (i) decide weather to change the search direction or to stay a little longer in the same search region and (ii) to process the bootstrapping phase itself. The EA takes almost 66% for the bootstrapping method and only 34% to really converge to a best solution.

Table 1: Results of the ESSA method on a subset of benchmarks and the better known pure ESSAs.

| JSSP | A** | B** | Best | Average | Worst | Time(sec)* |
|---|---|---|---|---|---|---|
| **FT 06** | **55** | **-** | ***55*** | **55** | **55** | 0.15 |
| **FT 10** | 936 | 937 | ***930*** | 935 | 945 | 292 |
| **FT 20** | 1181 | **1165** | ***1165*** | 1179 | 1184 | 452 |
| **LA 2** | - | 666 | ***655*** | **655** | **655** | 168 |
| **LA 19** | - | 851 | ***842*** | 843 | 854 | 264 |
| **LA 21** | - | 1055 | 1059 | 1074 | 1085 | 472 |
| **LA 24** | - | 966 | *955* | 964 | 970 | 460 |
| **LA 25** | - | 987 | *984* | 986 | 998 | 464 |
| **LA 27** | 1269 | 1257 | 1268 | 1282 | 1294 | 760 |
| **LA 29** | 1233 | 1179 | 1207 | 1219 | 1234 | 764 |
| **LA 36** | 1297 | 1287 | *1281* | 1302 | 1315 | 720 |
| **LA 37** | 1447 | 1418 | 1435 | 1457 | 1467 | 680 |
| **LA 38** | 1251 | 1217 | 1233 | 1248 | 1268 | 752 |
| **LA 39** | 1251 | 1258 | *1251* | 1255 | 1263 | 736 |
| **LA 40** | 1252 | 1234 | 1246 | 1251 | 1252 | 376 |
| **CI-CPU** | 9494 | - | | - | | 41.707 |
| **MRE(%)** | 2.61 | 1.47 | 1.45 | 2.24 | 3.05 | - |

(-) not applicable or not given
(*) Time for one run using a Pentium II 450 Mhz., $TF_{\text{Pentium II 450 Mhz}} = 85$
A**[Bierwirth 95] - B**[Norman 97]

## 3 OPTIMAL AMOUNT OF LAMARCKIAN LEARNING

ESSAs are capable of rapidly finding promising regions in the search space and exploit them intensively. However, they take a very long time to converge to local optima. On the contrary, local improvement procedures find the local optimum fast if a good search region is at hand but typically lack global search features. The combination of EA and local improvement techniques was attempted to improve the algorithms' performance through a Lamarckian Learning concept. The concept

allows individuals' fitness to be determined based on learning, i.e. the application of local improvements. Lamarckian learning changes the individuals' genotype to reflect the result of learning. Figure 3 illustrated the approach.

The learning schemes used are depicted in figure 6 (the update function is illustrated in figure 3). The figure shows a specialized decoder (left) or a local optimizer (right) is used, which is similar as using active scheduling (left) or local search based scheduling (right). If both are used together, a powerful learning scheme arises.

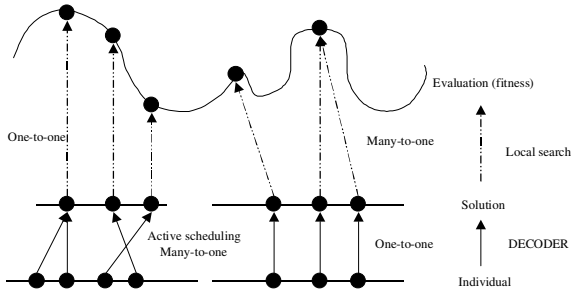


Figure 6: Learning schemes for the JSSP.

The ESSA as previously applied will be extended with a local optimizer. A simulated annealing algorithm is incorporated as learning procedure. One of the basic disadvantages of the SA approach is its time consuming element. Therefor the EA will not be implemented as a memetic algorithm but one, which now and then refocuses the search direction using the local optimizer sparingly. Moreover, the simulated annealing is intended to create local improvements. Therefor, it is used with an initial control parameter of *5*, a linear decrease function using a factor of *0.01* whenever 20 improvements were found and iterates for only *2000* iterations. It uses a neighborhood structure called critical end insert with repair [Van Bael 98]. The ESSA will apply the local optimizer with a probability of *x%*. This means, whenever an offspring is not selected for crossover it will be selected in *x%* of the cases for local search. If both operators are not selected mutation takes place. Since no arguments clearly indicate how much Lamarckian learning is needed, an experiment with different rates is performed. Rates of 1, 2, 5, 10, 20, 50, 75 and 95 % are examined for one instance, namely mt10. Figure 7 shows average results of the number of generations and the total number of evaluations processed to find the global optimum.

The choice of 1% Lamarckian Learning was made to limit the computation time. Note that this means that the local search procedure is applied on 0.1% (=(1-0.9)*1%) of the offspring. As can be seen in figure 7, it is about the best choice one could make. A high rate of Lamarckian Learning is able to generate the global optimum within only a few generations. On the contrary, the time needed to reach it is as the total evaluations processed and is significantly more (almost the double).

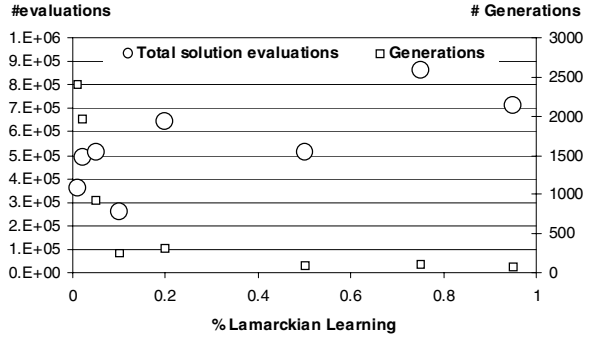


Figure 7: Total number of generations and evaluations versus rate of Lamarckian Learning to find the global optimum of mt10 (average of *10* runs).

Table 2: Hybrid evolutionary algorithm using *1%* Lamarckian learning

| JSSP | Optimum | Best | Average | Worst | Time(sec*) |
|---|---|---|---|---|---|
| **FT 06** | **55** | **55** | **55** | **55** | 281 |
| **FT 10** | **930** | **930** | **930** | **930** | 1818 |
| **FT 20** | **1165** | **1165** | 1174 | 1178 | 5217 |
| **LA 2** | **655** | **655** | **655** | **655** | 750 |
| **LA 19** | **842** | **842** | **842** | **842** | 952 |
| **LA 21** | **1046** | **1046** | 1047.5 | 1053 | 2962 |
| **LA 24** | **935** | **935** | 938.3 | 941 | 3200 |
| **LA 25** | **977** | **977** | 978.4 | 981 | 3529 |
| **LA 27** | **1235** | 1242 | 1249.6 | 1255 | 5106 |
| **LA 29** | **1152** | 1154 | 1169 | 1180 | 6233 |
| **LA 36** | **1268** | 1281 | 1281 | 1281 | 9056 |
| **LA 37** | **1397** | **1397** | 1409.4 | 1418 | 5274 |
| **LA 38** | **1196** | **1196** | 1205.5 | 1216 | 6486 |
| **LA 39** | **1233** | **1233** | 1237.8 | 1243 | 5714 |
| **LA 40** | **1222** | 1226 | 1229.5 | 1233 | 4615 |
| | | **MRE (%)** | | | **CI-CPU** |
| | | 0.209 | 0.51864 | 0.853 | 346798 |

(*) Time for one run using a Pentium II 450 Mhz., $TF_{\text{Pentium II 450 Mhz}} = 85$

The hybrid ESSA using 1% Lamarckian Learning was applied to the same instances as the previous ESSA. Table 2 shows the results. The hybrid evolutionary algorithm is very effective. It can compete with the best hybrid ESSA. It reaches an MRE of *0.209* for the *15* instances, which is only improved by [Yamada 96] solving only *10* of the set. If compared to the same 10 instances our hybrid method was superior with an MRE of *0.11*. The low MRE could be reached due to the fact that *11* of the *15* instances could be solved to global optimality. On average, the MRE is just over *0.5%* and in the worst case it is still below *1%,* which indicates its robust character. Improvements could be made tailoring the parameter instantiations to the problem as illustrated elsewhere for the mt20 problem [Van Bael $99_b$]. Its efficiency is not that bad as mentioned (*CI-CPU = 346798*). As already explained, the bootstrapping function takes about *66%* of the computation time. The EA converges about *6* times during every run. Thus, the computation time needed can

if necessary drop below *20000 CI-CPU* if only one convergence is allowed each run.

## 4 CONCLUSION

We presented an evolutionary algorithm solving the JSSP, once without and once including specific knowledge. Both are a steady state ESSA. The former performs well, finding optimal solutions for smaller JSSP benchmarks and attaining a worst case MRE of 3%. However, the population size and offspring size had to be tuned. It indicated that large population sizes were preferred generating 1% offspring each generation. The large population size was limited by the time consuming schedule generation. An optimal tuning of 2000 individuals generating 50 offspring was selected as best performing. The ESSA including Lamarckian Learning performs very well. It was able to find the majority of global optima of the benchmarks and achieve a good overall effectiveness (worst case MRE below 1%). A Lamarckian Learning concept was sparingly utilized (0.1%) within the ESSA making the ESSA efficient as well.

**References**


Bierwirth, C., A Generalized Permutation Approach to Job shop Scheduling with Genetic Algorithms, OR Spektrum, vol. 17, 87-92. 1995

Mattfeld, D.C. Evolutionary sear and the job shop: Investigations on Genetic Algorithms for Production Scheduling. Physica-Verlag. Heidelberg. 1996

Muth J.F., Thompson G.L. (eds) 1963. Industrial Scheduling, Prentice-Hall, Englewood Cliffs, N.J. 1963

Norman B.A., Bean J.C., Random Keys Genetic Algorithm for Job-Shop Scheduling. Engineering Design & Automation 3(@), p. 145-156. 1997

Vaessens, R.J.M., Aarts, E.H.L., Lenstra, J.K. Job Shop Scheduling by Local Search. INFORMS Journal on Computing 8, pp. 302-317. 1996

Van Bael P., Rijckaert M., Solving Job Shop Problems with Critical Block Neighborhood Search, Proceedings of the Tenth Netherlands/Belgium Conference on Artificial Intelligence (NAIC 98), Amsterdam, Holland, p.203-209, 1998

Van Bael P., Devogelaere D. Rijckaert M., The Job Shop Scheduling Problem Solved with Simple, Basic Evolutionary Search Elements. Proceedings of GECCO99, Florida, p. 665-669, $1999_a$

Van Bael P., Devogelaere D. Rijckaert M., A hybrid Evolutionary Search Scheduling Algorithm to Solve the Job Shop Scheduling Problem. Proceedings of CEC99, Washington, USA, p. 1104-1109, $1999_b$

Yamada T., Nakano R., Scheduling by Genetic Local Search with Multi-Step Crossover, The Fourth International Conference on Parallel Problem Solving from Nature (PPSN IV) Berlin, pp.960-969. 1996