A Constructive Genetic Algorithm for Permutation Flowshop Scheduling

Version 2

Marcelo Seido Nagano*

Department of Industrial Engineering, School of Engineering of São Carlos, University of São Paulo

Av. Trabalhador Sãocarlense, 400, 13566-590, São Carlos, São Paulo, Brazil

Rubén Ruiz

Department of Applied Statistics and Operations Research, Polytechnic University of Valencia

Camino de Vera S/N, 46021, Valencia, Spain

Luiz Antonio Nogueira Lorena

Computer and Applied Mathematics Laboratory, Brazilian Space Research Institute

Av. dos Astronautas, 1758, 12227-010, São José dos Campos, São Paulo, Brazil

---

* Corresponding author. Tel.: +55-16-3373-9428; Fax: +55-16-3373-9425.

E-mail address: drnagano@usp.br (M.S. Nagano).

Av. Trabalhador Sãocarlense, 400 - Centro

Zip Code: 13566-590

São Carlos - SP/Brazil

A Constructive Genetic Algorithm for Permutation Flowshop Scheduling

Abstract

The general flowshop scheduling problem is a production problem where a set of *n* jobs have to be processed with identical flow pattern on *m* machines. In permutation flowshops the sequence of jobs is the same on all machines. A significant research effort has been devoted for sequencing jobs in a flowshop minimizing the makespan. This paper describes the application of a Constructive Genetic Algorithm (CGA) to makespan minimization on flowshop scheduling. The CGA was proposed recently as an alternative to traditional GA approaches, particularly, for evaluating schemata directly. The population initially formed only by schemata, evolves controlled by recombination to a population of well-adapted structures (schemata instantiation). The CGA implemented is based on the NEH classic heuristic and a local search heuristic used to define the fitness functions. The parameters of the CGA are calibrated using a Design of Experiments (DOE) approach. The computational results are compared against some other successful algorithms from the literature on Taillard's well known standard benchmark. The computational experience shows that this innovative CGA approach provides competitive results for flowshop scheduling problems.

Keywords: Flowshop, Constructive Genetic Algorithm, Makespan

1. Introduction

The general flowshop scheduling problem is a production problem where a set of *n* jobs have to be processed with identical flow pattern on *m* machines. When the sequence of job processing on all machines is the same we have the permutation flowshop sequencing production environment. Since there is no job passing, the number of possible schedules for *n* jobs is *n!*. Usually, the schedule performance measure is related to an efficient resource utilization looking for a job sequence that minimizes the makespan; that is the total time to complete the schedule and the problem is then denoted as $n/m/P/F_{max}$ or as $F/prmu/C_{max}$ (Pinedo, 2002).

This scheduling problem is generally modelled on the following assumptions:

i) The operation processing times on the machines are known, fixed and some of them may be zero if some job is not processed on a machine;

ii) Set-up times are included in the processing times and they are independent of the job position in the sequence of jobs;

iii) At a time, every job is processed on only one machine, and every machine processes only one job;

iv) The job operations on the machines may not be preempted.

A significant research effort has been devoted for sequencing jobs in a flowshop with the objective of finding a sequence that minimizes the makespan. For problems with 2 machines, or 3 machines under specific constraints on job processing times, the efficient Johnson's algorithm (Johnson, 1954) obtains an optimal solution for the problem. However, since this scheduling problem is NP-hard (Garey et al., 1976) the search for an optimal solution is of more theoretical than practical importance. Therefore, since the 1960s a

number of heuristic methods that provide near optimal or good solutions with limited computation effort have been proposed for flowshop sequencing. The performance of some earlier heuristics was evaluated by Dannenbring (1977) and King & Spachis (1980).

Heuristic methods can be classified according to two major categories: constructive or improvement methods. The constructive algorithms obtain directly a solution for the scheduling problem, i.e. a *n*-job sequence, by using some procedure which assigns to each job a priority or an *index* in order to construct the solution sequence (see for example, Palmer, 1965; Campbell et al., 1970; Gupta, 1971; Dannenbring, 1977; Nawaz et al., 1983; Koulamas, 1998; Davoud Pour, 2001; Nagano & Moccellin, 2002; Kalczynski & Kamburowski, 2007). An improvement method starts from a given initial solution, and looks for a better one normally by using some neighbourhood search procedure. Metaheuristics can also be considered as improvement heuristics. Within this type of techniques we find genetic algorithms (GAs), simulated annealing (SA), tabu search (TS) and other procedures or hybrid methods.

The first proposed metaheuristics for the permutation flowshop scheduling problem (PFSP) are the simulated annealing algorithms by Osman & Potts (1989) and Ogbu & Smith (1990). Widmer & Hertz (1989), Taillard (1990), Reeves (1993) and Nowicki & Smutnicki (1996) demonstrated different tabu search approaches. Other algorithms are the path-based method of Werner (1993) or the iterated local search (ILS) of Stützle (1998). Recently, Rajendran & Ziegler (2004) have proposed two very effective ant-colony optimization (ACO) algorithms and Grabowski & Wodecki (2004) a very fast TS approach. Ruiz & Maroto (2005) give an updated and comprehensive review of flowshop heuristics and metaheuristics. Another recent review is given by Hejazi & Saghafian (2005).

We focus now on work dealing with GAs and the PFSP. A genetic algorithm is a computerized iterative search optimization technique. It is based on the mechanics of natural selection and natural genetics. This deals with the population of solution rather than with a single solution. This type of algorithms provide near optimal schedules. The optimum value depends on the operators like crossover, mutation, number of iterations (‘generations’), etc. In every generation, a new set of artificial individuals (strings) is created. This algorithm combines survival of the fittest among string structures.

This paper describes the application of a Constructive Genetic Algorithm (CGA) to the PFSP. The CGA has a number of new features compared to a traditional genetic algorithm. These include a population of dynamic size composed of schemata and structures, and the possibility of using heuristics in structure representation and in the fitness function definitions.

The CGA is compared against some other successful algorithms from the literature on Taillard (1993) well known standard benchmark, composed of 120 different problem instances ranging from 20 jobs and 5 machines to 500 jobs and 20 machines. This benchmark contains some instances that have proved to be very difficult to solve in the past 10 years.

The rest of paper is organized as follows: Section 2 presents an overview of the existing genetic algorithms for the PFSP that are available from the literature. Section 3 describes in detail the new proposed genetic algorithm (CGA). An extensive comparison of CGA against some other successful algorithms from the literature is given in Section 4. Finally, in Section 5 we provide some conclusions about the study, along with some future research directions.

2. Genetic Algorithm for the PFSP

Flowshop scheduling is one of the most well-known problems in the area of scheduling. Various approaches to this problem have been proposed since the pioneering work of Johnson (1954). GAs have been applied to combinatorial optimization problems such as the traveling salesman problem and scheduling problems (see for example, Fox & McMahon, 1991; Ishibushi et al., 1994).

One of the earliest GAs for the PFSP was proposed by Chen et al. (1995). The initial population in this algorithm is generated by using several heuristic rules. The first $m-1$ population members are generated by the $m-1$ sequences obtained after applying the CDS heuristic of Campbell et al., (1970), the *m*th member is obtained from the *rapid access* (RA) heuristic of Dannenbring. The remaining members are generated from simple job exchanges of the already generated sequences. Only crossover is applied, there is no mutation. The crossover operator used is the *partially mapped crossover* (PMX) by Goldberg & Ling (1985). Reeves (1995) also proposed a GA. This algorithm uses a different generational scheme, called "termination with prejudice" in which the offspring generated after each mating do not replace the parents but members of the population with a fitness value below average. The algorithm uses a C1 crossover, essentially similar to the *one-point order crossover*. Another remarkable feature of the algorithm is the adaptive mutation used. Reeves' algorithm also initializes the population by using heuristics. In this case, one of the population members is generated by applying the NEH heuristic. Murata et al. (1996) proposed a hybrid GA with a *two-point order crossover*, a shift mutation and elitism strategy. The algorithm is hybridized with local search, which resulted in a clear performance gain over a non-hybrid version. Another hybrid GA is that of Reeves & Yamada (1998). In this case a special crossover, called *multi-step crossover fusion* or

MSXF is used, coalescing a typical crossover operator with local search. Ponnambalam et al. (2001) valuate a GA with a *generalized position crossover* or GPX crossover, a shift mutation and random population initialization. Aldowaisan & Allahverdi (2003) have proposed a simple yet effective GA for the permutation flowshop with no-wait constraints. And more recently, Ruiz et al. (2006) have proposed two new advanced genetic algorithms for the problem considered here that provide good results at the expense of some added complexity in the proposed methods.

All the previously cited work on GAs lacks a methodological approach to obtain the correct choice of operators and parameters. Usually, authors make use of short computer simulations or preliminary experiments to set parameters on a "one factor at a time" basis, i.e. changing one parameter while maintaining the remaining factors unaltered. While this approach might be useful when setting some of the algorithms' parameters, it has several shortcomings when setting some of the important operators (like crossover operator or population size). In this paper we use a more comprehensive approach for calibrating the operators and parameters.

3. Constructive Genetic Algorithm for the PFSP

Lorena & Furtado (2001) proposed recently the Constructive Genetic Algorithm (CGA) as an alternative to a traditional GA approaches, particularly, for evaluating schemata directly. The CGA evolves a population initially formed only by schemata, controlled by recombination, to a population of well-adapted structures (schemata instantiation) and schemata. It was applied to clustering problems (Lorena & Furtado, 2001), location problems and timetabling problems (Ribeiro Filho & Lorena, 2001), gate matrix layout problems (Oliveira & Lorena, 2002) and to point feature cartographic label placement

problems (Yamamoto & Lorena, 2005). In this paper, the concept of CGA will be first described and then applied to the solution of the PFSP.

3.1. CGA Modeling

In this section the modeling phase of the CGA is described. The problem in consideration (in this case the PFSP) is initially formulated as a *bi-objective optimization problem* (BOP) to attain the objective of evaluating schemata and structures in a common way. Two fitness functions are defined on the space of all schemata and structures that can be obtained using a specific representation. The BOP is then indirectly solved by an evolution process (described in the next section) that considers the two objectives on an adaptive rejection threshold, which gives ranks to individuals in population and yields a dynamic population.

Very simple structure and schema representations are implemented to the PFSP. A direct alphabet of symbols (natural numbers) represents the jobs permutation. The symbol # is used to express indetermination (# - do not care) on schemata. For example s_i = (1 5 3 2 4) is a structure representation of a PFSP with 5 jobs, while s_k = (# 5 # # 4) is a possible schema.

Let X be the space of all schemata and structures that can be created by this representation. The PFSP is modeled as the following *Bi-objective Optimization Problem* (BOP):

$$
\begin{aligned}
& & \textit{Min} \quad & \{g(s_k) - f(s_k)\} & \\
\textit{(BOP)} & & \textit{Max} \quad & g(s_k) & \\
& & \textit{Subject to} \quad & g(s_k) \geq f(s_k) \qquad & \forall\, s_k \in X
\end{aligned}
$$

Function g is the fitness function that reflects the makespan of a given permutation of jobs in s_k after application of NEH heuristic of Nawaz et al. (1983). Therefore, it is defined by the following procedure:

- The jobs in s_k are extracted in their relative order;
- These jobs form the initial ordering for the application of the NEH;
- After the application of the NEH heuristic, the makespan of the resulting sequence (which might not be a complete sequence) is the value of $g(s_k)$.

The second fitness function f is defined to drive the evolutionary process to a population trained by a heuristic. Thus, function f is defined by the following insertion heuristic (IH):

- One job of the schedule obtained after calculating the $g(s_k)$ value is extracted at random;
- This job is inserted in all possible positions of the sequence and the best makespan obtained is retained.

This procedure is repeated until a local optima is achieved of $f(s_k)$ is the makespan obtained as a result.

Considering the definition of function g, the maximization objective on BOP appears to be a contradiction. However, it is needed to give distinct treatment to structures and schemata. By definition, f and g are applied to structures and schemata, just differing in the amount of information and consequently, in the values associated to them. More information means larger values. Therefore, the g maximization objective in BOP drives the search for feasible solutions (structures) to PFSP.

The interval minimization (g-f) shows that better individuals (schemata or structures) are those having little improvement by IH, which indirectly reproduces the makespan minimization in PFSP.

3.2. Evolution Process

The BOP defined above is not directly considered as the set X is not completely known. Alternatively an evolutionary process is considered to attain the objectives (interval minimization and g maximization) of the BOP. At the beginning of the process, two expected values are given to these objectives:

- g maximization: a non-negative real number g_{max} that is expected to be an upper bound to the PFSP makespan;
- Interval minimization: an interval length $d \cdot g_{max}$, obtained from g_{max} considering a real number $0 < d \leq 1$, that will be the expected improvement of IH.

The evolution process is then conducted considering an adaptive rejection threshold, which considers both objectives in BOP. Given a parameter $\alpha \geq 0$, the expression

$$g(s_k) - f(s_k) \geq d \cdot g_{max} - \alpha \cdot d \cdot [g_{max} - g(s_k)] \quad (1)$$

presents a condition for rejection from the current population of a schema or structure s_k. The right hand side of (1) is the threshold, composed of the expected value to the interval minimization $d \cdot g_{max}$, and the deviation $g_{max} - g(s_k)$, that shows the difference of $g(s_k)$ and g_{max} evaluations.

Expression (1) can be examined varying the value of α. For $\alpha = 0$, both schemata and structures are evaluated by the difference g-f (first objective of BOP). When α increases,

schemata are most penalized than structures by the difference $g_{max} - g$ (second objective of BOP).

Parameter α is related to time in the evolution process. Considering that the good schemata need to be preserved for recombination, the evolution parameter α starts from 0, and then increases slowly, in small time intervals, from generation to generation. The population at the evolution time α, denoted by P_α, is dynamic in size accordingly the value of the adaptive parameter α, and can be emptied during the process. The parameter α is now isolated in expression (1), thus yielding the following expression and corresponding rank to s_k:

$$\alpha \geq \frac{d \cdot g_{max} - [g(s_k) - f(s_k)]}{d[g_{max} - g(s_k)]} = \delta(s_k) \qquad (2)$$

At the time they are created, structures and/or schemata receive their corresponding rank value $\delta(s_k)$. These ranks are compared with the current evolution parameter α. The higher the value of $\delta(s_k)$, and better is the structure or schema to the BOP, and they also have more surviving and recombination time.

For the PFSP, the overall bound g_{max} is obtained at the beginning of the CGA application, by generating a structure and making g_{max} receive the g evaluation for that structure. In order to ensure that g_{max} is always an upper bound, after recombination, each new structure generated s_{new} is rejected if $g_{max} \leq g(s_{new})$.

In the case of the PFSP we have chosen to initialize the value of g_{max} with the NEH heuristic rule of Nawaz et al. (1983). In this case g_{max} will be the makespan of this constructed schedule.

3.2.1. Initial Population

The initial population is composed exclusively of schemata, where a proportion of jobs are replaced by labels # (indetermination). For the PFSP, the random process of creating the initial population of schemata is guided so that every job is present in schemata at least once and that for every position in the permutation there is at least one schema with a label different from #. Along the generations, the population increases by addition of new offspring generated out of the combination of two schemata aiming the structure creation.

3.2.2. Selection

There are two purposes on the evolution process: to obtain structures (good solutions to the g maximization objective on the BOP), and that these structures be good ones (best solutions to the interval minimization objective on the BOP). Thus, the population is maintained classified by rank (expression (2)), and the individuals with more genetic information (structures or semi-complete schemata) and presenting small improvements by heuristic IH, appear in first order places on the population.

Two structures and/or schemata are selected for recombination. The first is called the base (s_{base}) and is randomly selected out of the first positions in P_{α}, and in general it is a good structure or a good schema. The second structure or schema is called the guide (s_{guide}) and is randomly selected out of the total population. The objective of the s_{guide} selection is the conduction of a guided modification on s_{base}.

3.2.3. Recombination

In the recombination operation, the current labels in corresponding positions are compared. Let s_{new} be the new structure or schema (offspring) after recombination. Structure or schema s_{new} is obtained by applying the following operations:

{Recombination}

For i from 1 to individual length

I - if s_{base} (i) = # and s_{guide} (i) = # then

set s_{new} (i) = #

II - if s_{base} (i) <> # and s_{guide} (i) = # then

if s_{base} (i) is not in s_{new} then

set s_{new} (i) = s_{base} (i)

else set s_{new} (i) = #

III - if s_{base} (i) = # and s_{guide} (i) <> # then

if s_{guide} (i) is not in s_{new} then

set s_{new} (i) = s_{guide} (i)

else set s_{new} (i) = #

IV - if s_{base} (i) <> # and s_{guide} (i) <> # then

if s_{base} (i) is not in s_{new} then

set s_{new} (i) = s_{base} (i)

else

if s_{guide} (i) is not in s_{new} then

set s_{new} (i) = s_{guide}

else set s_{new} (i) = #

Observe that s_{base} is a privileged individual to compose s_{new}, but it is not totally predominant. There is a small probability of the s_{guide} gene information to be used instead of s_{base} one.

3.2.4. The Algorithm

The Constructive Genetic Algorithm can be summed up by the following pseudo-code. The ε increment is a linear step that increases the adaptive rejection threshold. Each distinct value of α corresponds to a generation. The stop conditions occur with an emptied population (assured by a sufficiently higher α) or when a predetermined stopping criterion is met. The population increases, after the initial generations, reaching an upper limit (in general controlled by storage conditions), and decreases for higher values of the evolution parameter α.

CGA {Constructive Genetic Algorithm}

Given g_{max} and d; $\alpha := 0$; ε;

Initialize P_α ;

for all $s_k \in P_\alpha$ do compute $g(s_k)$, $f(s_k)$, $\delta(s_k)$;

while (not stop condition) do

    while (number of recombinations) do

        Select Base and Guide from P_α ;

        Recombine Base and Guide;

        Evaluate Offspring;

        Update Offspring in P_α ;

end_while

$\alpha := \alpha + \varepsilon$;

for all $s_k \in P_\alpha$ satisfying $\alpha > \delta(s_k)$ do

Eliminate s_k from P_α;

end_for

end_while

The CGA algorithm begins with the recombination procedures (over schemata only) and the constructive process builds structures (full individuals) progressively at each generation. The constructive process repeatedly uses genetic information contained in two individuals to generate another one. However, the constructive process can be complemented using especially designed mutation and filling heuristics, searching for a better overall performance.

The mutation is always applied to structures, no matter how they are created (after recombination or after the filling process). It performs an additional local search step to the NEH method used to define the function g, applying heuristic IH more intensively than in definition of function f. The insertion neighbourhood is fully examined to find a local optima (i.e. if after performing an insertion move, a better schedule is obtained, all insertion moves are performed again).

The filling heuristic is performed whenever the offspring generated is a schema. The procedure is simple and can be summarized as follows:

- Construct a vector with the missing jobs in the schema;

- Shuffle this vector of missing jobs and apply the NEH procedure to fill the schema (transform it into structure). The order in which the jobs are filled in the NEH heuristic is taken from the shuffled vector;
- Apply the intensive form of the local search IH to the completed structure.

It is important to mention that the completed structures are not kept in the population. However, the completed structure is kept if the value of the makespan is the best found so far.

3.2.5. Calibration of parameters by means of Design of Experiments

As it has been mentioned, the proposed Constructive Genetic Algorithm has some parameters that need to be determined to attain best performance. These parameters are the increment ε, used for determining how fast "bad" schemata die off in the population, the value "d" used in the interval minimization of the BOP problem inside the CGA and two main parameters of the CGA, the size of the initial schemata population or P_α and the number of jobs that are filled in each schemata in this initial population, parameter that will be refered to as "fill_in". A priori, it is expected that these four parameters would have little impact on the overall performance of the CGA since both d and ε affect only the evaluation of schemata and P_α and fill_in control the characteristics of the initial population which is dynamic in size anyways.

Still, a correct and complete calibration is in order using a complete factorial design of experiments (DOE) (see Montgomery, 2005) for analyzing the four aforementioned factors at the following levels:

- P_α : 6 levels, 50, 60, 70, 80, 90 and 100;

- fill_in: 6 levels, 5, 6, 7, 8, 9 and 10;
- ε: 3 levels, 0.005, 0.01 and 0.015;
- d: 3 levels, 0.05, 0.1 and 0.15.

The total number of combinations and thus, different Constructive Genetic Algorithms is 6*6*3*3=324.

In order to test all these combinations we could use Taillard's benchmark, but this would probably result in an algorithm calibrated for an instance set that is afterwards used for comparative evaluations. A better approach is to "train" or calibrate the CGA with a set of instances different from those used for testing. Thus, a new set of 340 difficult PFSP problems that come after considering different values for *n* and *m* where *n*={20,50,80,110,…,500} and *m*={5,10,15,20} is generated, yielding 68 combinations of *n* and *m* with 5 repetitions per combination where the processing times have been sampled from the distribution U[1,99] following the same methodology showed in Taillard (1993). This set of 340 problems produced 324 combinations of parameters for the CGA that were run with the following stopping criteria: *n*(m/2)*10* elapsed milliseconds on an Athlon XP 1600 computer (running at 1400 MHz) with 512 Mbytes of RAM. This stopping criterion assures that the same amount of time is allowed for every one of the 324 combinations running on every instance. Also, more time is given as both *n* and *m* increase in the instance.

With this stopping criteria the following measure is calculated:

$$\%\ \text{Increase Over Lower Bound} = \frac{\text{Heu}_{\text{sol}} - \text{LB}}{\text{LB}} \cdot 100 \qquad (3)$$

Where LB is the lower bound obtained as in Taillard (1993) for any of the 340 problems. Therefore, the response variable in the experiment is the average percentage increase obtained in the Cmax of the CGA over the LB for the set of 340 problems.

The following table shows the ANOVA results from the experimental analysis.

[Insert table 1 about here]

As can be seen, the number of jobs (*n*) and the number of machines (*m*) are by very far, the two most important factors that explain the response variable, i.e. the average percentage increase depends on the difficulty (size) of a given instance rather than to the four controlled parameters in the experiment. With this result we carried out 68 different analyses by analyzing the behavior of the four controlled parameters when *n* and *m* are fixed to a given value from the 68 possible. In this case the best combinations of the four controlled parameters depending on the difficulty or size of the instance are analyzed. For example, for *n*=50 and *m*=20 the ANOVA table is:

[Insert table 2 about here]

Where the controlled factors, fill_in and P_α do have a clear effect on the response variable according to the P-Value. If the response variable is plotted against the different levels of the factor P_α the following means plot arises:

[Insert figure 1 about here]

Figure 1 shows that an initial population of 50 schemata is statistically worse than a population of 60 or more schemata with no clear trend after a population of 60. In this case, any value of P_α but 50 results in a statistically equivalent performance.

By following the same procedure for all 68 combinations and all parameters results the following “best-case” calibration of the proposed CGA algorithm:

- P_{α} : 100;
- fill_in: 8;
- ε: 0.005;
- d: 0.05.

4. Computational Results

In this section the performances of the proposed CGA algorithm and the CGA algorithm with local search (referred to as CGALS) are compared against a representative set of genetic algorithms as well as other methods proposed in the literature.

The implemented heuristics are: the NEH heuristic of Nawaz et al (1983), the Simulated Annealing of Osman & Potts (1989) (SAOP), the Tabu Search of Widmer & Hertz (1989) (SPIRIT), the genetic algorithms of Chen et al. (1995) (GAChen), Murata et al. (1996) (GAMIT), Reeves (1995) (GAReev), Ponnambalam et al. (2001) (GAPAC), Aldowaisan & Allahverdi (2003) (GAAA). We also implement recent heuristics such as the genetic algorithm of and Ruiz et al. (2006) (GARMA), the differential evolution method of Onwubolu & Davendra (2006) (DE) and the adaptive-learning approach algorithm of Agarwal et al. (2006) that uses NEH as initialization (NEH_ALA).

All these algorithms are implemented in Delphi 2006 and share datasets and functions and are run against Taillard's instances which comprise a set of 120 problems ranging from 20 jobs and 5 machines to 500 jobs and 20 machines which have proven to be especially difficult in the past 10 years. For this set the heuristics are compared with the following performance measure:

$$\%\ \text{Increase Over the Best Solution} = \frac{\text{Heu}_{\text{sol}} - \text{Best}_{\text{sol}}}{\text{Best}_{\text{sol}}} \cdot 100 \qquad (4)$$

Where Heu_{sol} is the best makespan obtained by a given algorithm and $Best_{sol}$ is the given optimum makespan for each instance in the OR Library (http://mscmga.ms.ic.ac.uk/jeb/orlib/flowshopinfo.html) or the lowest known upper bound if the optimum makespan for that instance is yet unknown.

The response variable is, therefore, the average percentage increase over the optimum or lowest known upper bound for 120 instances. All the algorithms are run on a Pentium IV computer running at 3.2 GHz with 2 Gbytes of RAM memory and the stopping criteria for the metaheuristic methods is *n*(m/2)*60* elapsed milliseconds. This way all the algorithms are run with the same computer for the same amount of time which results in comparable results for the experiment. All tested algorithms, with the exception of the constructive heuristic NEH are stochastic. Therefore, we run 5 independent replicates of each instance in order to have a better picture of the results. The results, averaged by instance size are shown in Table 3.

[Insert table 3 about here]

Table 3 shows that the CGA algorithm with local search (CGALS) is considerably better than the version without local search. Considering that for both versions are allowed the same CPU time we can safely observe that the hybridization with the IH heuristic is very effective.

Table 3 also shows that CGALS algorithm is better all other genetic algorithms, including the recent GARMA algorithm although the differences with this last algorithm are small. In any case, CGALS shows better results for most instance sizes. A simple statistical mean test is run to check whether there are statistically significant differences between the average percentage increases obtained by the different algorithms. This test can be carried out by means of a one-way ANOVA test. Notice that we have 120 instances and 5

replicates for each instance and 13 different algorithms. Therefore, the number of data points (7,800) is large enough to ascertain a very high power in the statistical test. The ANOVA Table is:

[Insert table 4 about here]

We see that there are statistically significant differences between the different algorithms used in the evaluation. To further clarify these differences we show the following means plot:

[Insert figure 2 about here]

This experiment confirms the idea that the CGALS algorithm has a performance which is statistically equivalent to that of the GARMA algorithm. However, this result is due to the similarities in performance for the larger instances. In the small instances, and as Table 3 shows, CGALS gives better results and this is confirmed by limiting the previous experiment to those instances. We also observe that all other algorithms are statistically worse to CGALS and GARMA, including the very recent CGA and NEH_ALA algorithms. This extensive experimentation shows that the CGA algorithm is a new and innovative class of genetic algorithms for PFSP that has competitive performance as far as genetic algorithms and other simple approaches are concerned.

5. Conclusions and future research

In this paper we have proposed the application of a new class of genetic algorithms, called Constructive Genetic Algorithms (CGA), to the permutation flowshop scheduling problem. The CGA maintain in population parts of schedules called schemata, where not all jobs are given to makespan evaluation. That characteristic appears to be beneficial to the search for good feasible complete schedules.

The simple CGA and the local search CGA have their parameters calibrated by an extensive design of experiments and tested against other genetic, tabu search and simulated annealing algorithms. The results are very promising and show that the CGA is competitive with other successful methods.

Furthermore, the CGA studied could be easily modified to application in different problems for flowshop, for example: no-wait flowshop, hybrid flowshop and sequence dependent setup times (SDST flowshop).

References

Agarwal, A., Colak, S. & Eryarsoy, E. (2006). Improvement heuristic for the flow-shop scheduling problem: An adaptive-learning approach. *European Journal of Operational Research*, 169, 801-815.

Aldowaisan, T. & Allahverdi, A. (2003). New heuristics for no-wait flowshops to minimize makespan. *Computers and Operations Research*, 30, 1219-1231.

Campbell, H.G., Dudek, R.A. & Smith M.L. (1970). A heuristic algorithm for n job, m machine sequencing problem. *Management Science*, 16(10), 630-637.

Chen, C.-L., Vempati, V.S. & Aljaber, N. (1995). An application of genetic algorithms for flow shop problems. *European Journal of Operational Research*, 80, 389-396.

Dannenbring, D.G. (1977). An evaluation of flow-shop sequencing heuristics. *Management Science*, 23, 1174-1182.

Davoud Pour, H. (2001). A new heuristic for the n-job, m-machine flow-shop problem. *Production Planning and Control*, 12(7), 648-653.

Fox, B.R. & McMahon, M.B. (1991). Genetic operations for sequencing problems, foundations of genetic algorithms. *Rawlins GJE, editor. San Mateo: Morgan Kaufmann Publishers*, 284-300.

Garey, M.R., Johnson, D.S. & Sethi, R. (1976). Complexity of flow-shop and job-shop scheduling. *Mathematics of Operations Research*, 1(2), 117-129.

Goldberg, D. & Lingle Jr., R. (1985). Alleles, loci, and the travelling salesman problem. In: Grefenstelle J.J. (Ed.), Proceedings of the first international conference on genetic algorithms and their applications, Hillsdale, N.J., *Lawrence Erlbaum associates*, 154-159.

Grabowski, J. & Wodecki, M. (2004). A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. *Computer & Operations Research*, 31, 1891-1909.

Gupta, J.N.D. (1971). A functional heuristic algorithm for the flowshop scheduling problem. *Operational Research Quarterly*, 22(1), 39-47.

Hejazi, S.R. & Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, 43(14), 2895-2929.

Ishibuchi, H., Yamamoto, N., Murata, T. & Tanaka, H. (1994). Genetic algorithms and neighborhood search algorithms for fuzzy flowshop scheduling problems. *Fuzzy Sets and Systems*, 67, 81-100.

Johnson, S.M. (1954). Optimal two-and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1, 61-68.

Kalczynski, P.J. & Kamburowski, J. (2007). On the NEH heuristic for minimizing the makespan in permutation flow shops. *OMEGA, The International Journal of Management Science*, 35(1), 53-60.

King, J.R. & Spachis, A.S. (1980). Heuristics for flowshop scheduling. *International Journal of Production Research*, 18, 345-357.

Koulamas, C. (1998). A new constructive heuristic for the flowshop scheduling problem. *European Journal of Operational Research*, 105, 66-71.

Lorena, L.A.N. & Furtado, J.C. (2001). Constructive genetic algorithm for clustering problems. *Evolutionary Computation*, 9(3), 309-327.

Montgomery, D.C. (2005). Design and analysis of experiments. Sixth Edition, New York: *John Wiley & Sons.*

Murata, T., Ishibuchi, H. & Tanaka, H. (1996). Genetic algorithms for flow shop scheduling problems. *Computers and Industrial Engineering*, 30, 1061-1071.

Nagano, M.S. & Moccellin, J.V. (2002). A high quality solution constructive heuristic for flow shop sequencing. *Journal of the Operational Research Society*, 53, 1374-1379.

Nawaz, M., Enscore, Jr., E.E. & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The International Journal of Management Science*, 11(1), 91-95.

Nowicki, E. & Smutnicki, C. (1996). A fast tabu algorithm for the permutation flow-shop problem. *Journal of Operational Research*, 91, 160-175.

Ogbu F.A. & Smith D.K. (1990). The application of the simulated annealing algorithms to the solution of the $n/m/C_{max}$ flowshop problem. *Computers & Operations Research*, 17(3), 243-253.

Oliveira, A.C.M. & Lorena, L.A.N. (2002). A constructive genetic algorithm for gate matrix layout problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*. vol. 21 (8), 969-974.

Onwubolu, G. & Davendra, D. (2006). Scheduling flow shops using differential evolution algorithm. *European Journal of Operational Research*, 171, 674-692.

Osman, I.H. & Potts, C.N. (1989). Simulated annealing for permutation flow-shop scheduling. *OMEGA, The International Journal of Management Science*. 17(6), 551-557.

Palmer, D.S. (1965). Sequencing jobs through a multi-stage process in the minimum total time – a quick method of obtaining a near optimum. *Operational Research Quarterly*, 16(1), 101-107.

Ponnambalam, S.G., Aravindan, P. & Chandrasekaran, S. (2001). Constructive and improvement flow shop scheduling heuristics: an extensive evaluation. *Production Planning and Control*, 12(4), 335-344.

Pinedo, M. (2002). Scheduling: theory, algorithms and systems. 2nd ed., *Englewood Cliffs*, NJ: Prentice Hall.

Rajendran, C. & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, 155, 426-436.

Reeves, C.R. (1993). Improving the efficiency of tabu search for machine scheduling problems. *Journal of the Operational Research Society*, 44(4), 375-382.

Reeves, C.R. (1995). A Genetic algorithm for flowshop sequencing. *Computers and Operations Research*, 22 (1), 5-13.

Reeves, C.R. & Yamada, T. (1998). Genetic algorithms, path relinking, and the flowshop sequencing problem. *Evolutionary Computation*, 6(1), 45-60.

Ribeiro Filho, G. & Lorena, L.A.N. (2001). A constructive evolutionary approach to school timetabling. In Applications of Evolutionary Computing, Boers, E.J.W., Gottlieb, J.,

Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H., *Springer Lecture Notes in Computer Science*. vol. 2037 (pp.130-139). Germany: Springer-Verlag.

Ruiz, R. & Maroto, C. (2005). A Comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165, 479-494.

Ruiz, R., Maroto, C. & Alcaraz, J. (2006). Two new robust genetic algorithms for the flowshop scheduling problem. *OMEGA, The International Journal of Management Science*, 34, 461-476.

Stützle, T. (1998). Applying iterated local search to the permutation flow shop problem. *Technical report*, AIDA-98-04, TU Darmstadt, FG Intellektik.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64, 278-285.

Werner, F. (1993). On the heuristic solution of the permutation flowshop problem by path algorithms. *Computers & Operations Research*, 20(7), 707-722.

Widmer, M. & Hertz, A. (1989). A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*. 41, 186-193.

Yamamoto, M. & Lorena, L. A. N. (2005). A constructive genetic approach to point-feature cartographic label placement. In *Metaheuristics: Progress as real problem solvers*, Ibaraki, T., Nonobe, K., Yagiura, M. Kluwer Academic Press, 285-300.

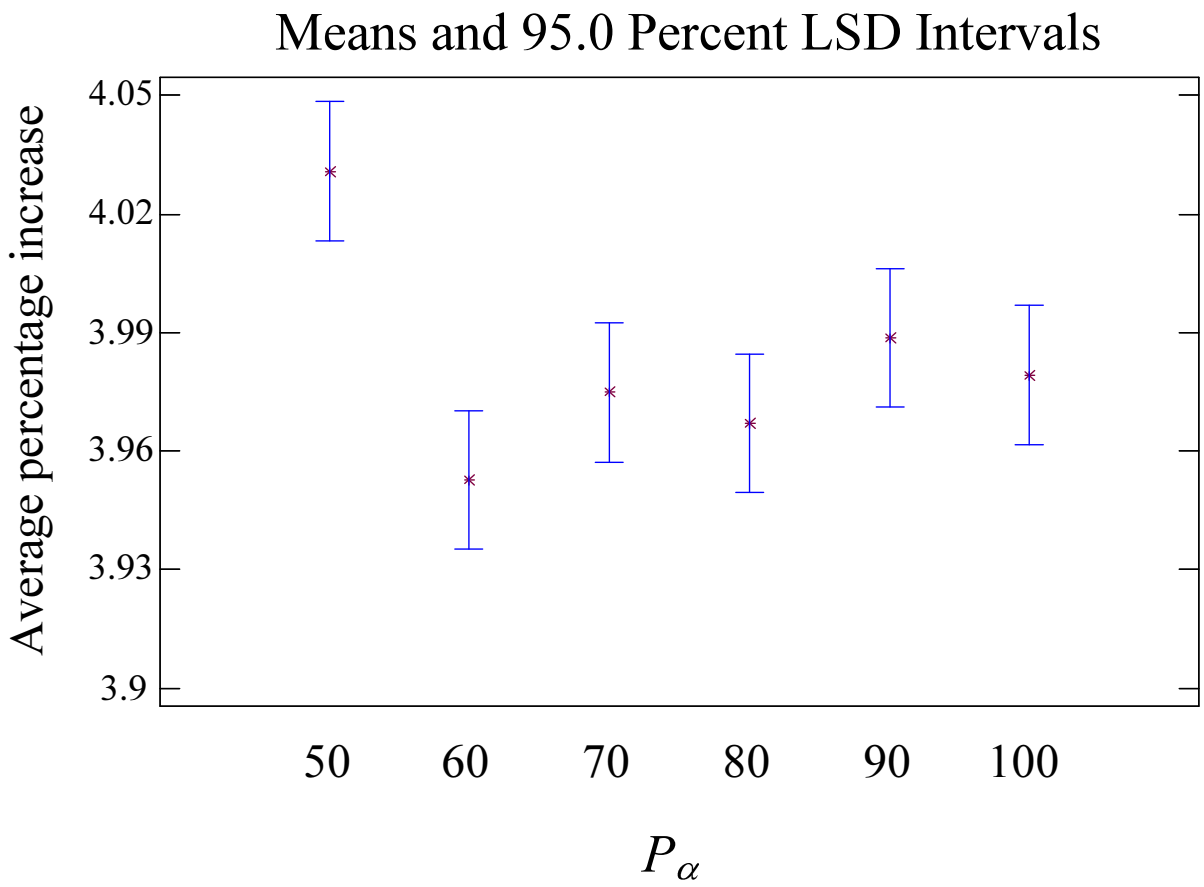

Figure 1: Means plot for population at the evolution time α (P_α) factor.

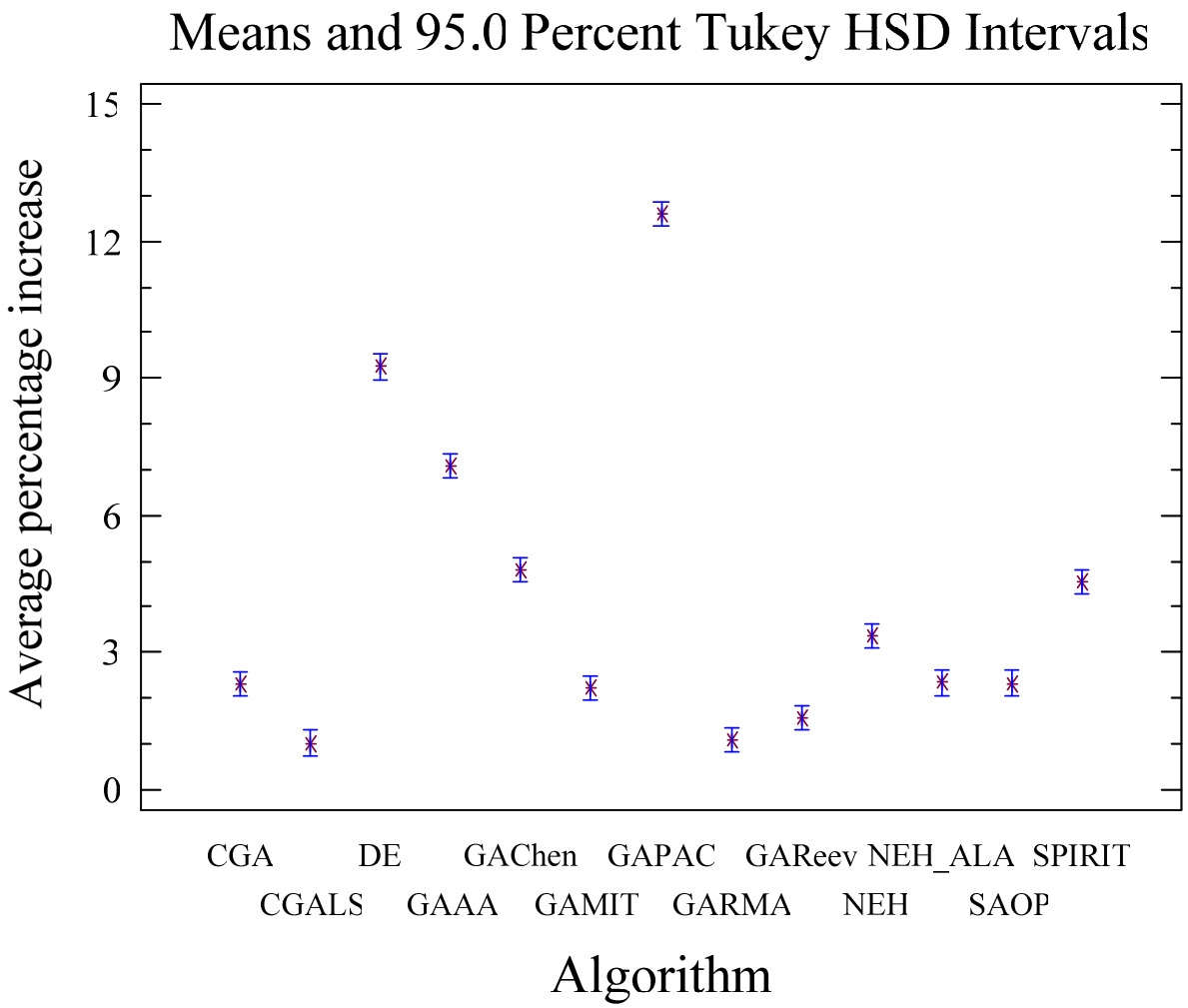

Figure 2: Means plot for the different algorithms evaluated.

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:d	0.0580	2	0.0290	0.0400	0.9585
B:fill_in	3.6931	5	0.7386	1.0800	0.3697
C:M	545629	3	181876	265669.05	0.0000
D:N	2239280	16	139955	204434.43	0.0000
E:Pα	21.5362	5	4.3073	6.2900	0.0000
F:ε	1.8928	2	0.9464	1.3800	0.2510
INTERACTIONS					
AB	0.2141	10	0.0214	0.0300	1.0000
AC	0.0071	6	0.0012	0.0000	1.0000
AD	1.2082	32	0.0378	0.0600	1.0000
AE	0.1481	10	0.0148	0.0200	1.0000
AF	0.0675	4	0.0169	0.0200	0.9988
BC	2.5369	15	0.1691	0.2500	0.9986
BD	37.4064	80	0.4676	0.6800	0.9865
BE	4.5496	25	0.1820	0.2700	0.9999
BF	0.2331	10	0.0233	0.0300	1.0000
CD	419583	48	8741.3	12768.53	0.0000
CE	11.9448	15	0.7963	1.1600	0.2928
CF	0.4299	6	0.0717	0.1000	0.9959
DE	5.8213	80	0.0728	0.1100	1.0000
DF	8.9975	32	0.2812	0.4100	0.9987
EF	0.4703	10	0.0470	0.0700	1.0000
RESIDUAL	75129.8	109743	0.6846		
TOTAL (CORRECTED)	3.2797E+06	110159			

Table 1: ANOVA table for the constructive genetic algorithm experiment.

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
MAIN EFFECTS					
A:d	0.2227	2	0.1113	2.57	0.0769
B:fill_in	0.5100	5	0.1020	2.35	0.0386
C:Pα	0.9626	5	0.1925	4.44	0.0005
D:ε	0.0796	2	0.0398	0.92	0.3992
INTERACTIONS					
AB	0.2542	10	0.0254	0.59	0.8261
AC	0.3076	10	0.0308	0.71	0.7159
AD	0.0341	4	0.0085	0.2	0.9402
BC	2.2084	25	0.0883	2.04	0.0018
BD	0.1899	10	0.0190	0.44	0.9282
CD	0.7655	10	0.0766	1.77	0.0619
RESIDUAL	66.5678	1536	0.0433		
TOTAL (CORRECTED)	72.1024	1619			

Table 2: ANOVA table for the constructive genetic algorithm experiment (n=50 and m=20)

Instances	NEH	GARMA	SAOP	SPIRIT	GAChen	GAReev	GAMIT	GAPAC	GAAA	CGALS	CGA	DE	NEH_ALA
20x5	3.35	0.29	0.93	4.01	3.54	0.53	0.57	*9.07*	1.69	**0.05**	1.33	3.98	1.38
20x10	5.02	0.63	2.59	5.65	5.17	1.79	1.75	*13.10*	1.60	**0.19**	2.42	5.86	2.22
20x20	3.73	0.41	2.33	4.84	4.29	1.40	1.48	*9.80*	1.61	**0.08**	2.08	4.53	1.78
50x5	0.84	0.06	0.48	1.90	2.14	0.19	0.24	*7.00*	2.40	**0.02**	0.32	4.28	0.46
50x10	5.12	1.76	3.34	5.84	6.47	2.11	3.38	*16.86*	9.88	**1.65**	3.72	11.48	3.44
50x20	6.31	**2.62**	4.47	7.46	7.86	3.60	4.92	*18.85*	12.35	2.67	4.98	14.73	4.66
100x5	0.46	0.07	0.28	0.93	1.32	0.16	0.24	*5.71*	2.15	**0.02**	0.21	4.27	0.46
100x10	2.13	**0.60**	1.53	2.96	3.99	0.80	1.53	*12.39*	7.88	**0.60**	1.46	10.42	1.54
100x20	5.23	**2.52**	4.68	6.26	7.99	3.32	4.87	*18.65*	14.27	2.84	4.52	16.08	4.49
200x10	1.43	0.43	0.99	2.06	2.72	0.48	1.00	*10.17*	6.95	**0.35**	0.99	8.34	1.30
200x20	4.52	**2.24**	4.14	5.17	7.37	2.87	4.18	*16.95*	13.75	2.56	3.79	15.44	4.11
500x20	2.24	1.28	2.21	7.59	4.81	1.47	2.54	*12.47*	10.61	**1.22**	1.93	11.58	2.24
Average	3.37	1.08	2.33	4.56	4.81	1.56	2.22	*12.59*	7.10	**1.02**	2.31	9.25	2.34

Table 3: Average percentage increase over the best solution known for the metaheuristic algorithms. Maximum elapsed time stopping criterion. Worst values in italics and best values in bold.

Source	Sum of Squares	Df	Mean Square	F-Ratio	P-Value
Between groups	87992.2	12	7332.69	931.34	0.0000
Within groups	61309.0	7787	7.87324		
TOTAL (CORRECTED)	149301.0	7799			

Table 4: ANOVA table for the evaluation of the differences between algorithms