Minimizing the weighted number of tardy jobs in a hybrid flow shop with Genetic Algorithm

Caroline Desprez, Feng Chu, Chengbin Chu

Minimizing the weighted number of tardy jobs in a hybrid flow shop with Genetic Algorithm

Journal:	*International Journal of Computer Integrated Manufacturing*
Manuscript ID:	TCIM-2007-IJCIM-0152.R2
Manuscript Type:	Original Manuscript
Date Submitted by the Author:	10-Sep-2008
Complete List of Authors:	Desprez, Caroline; Université de technologie de Troyes, Institut Charles Delaunay Chu, Feng; Université de technologie de Troyes, Institut Charles Delaunay Chu, Chengbin; Université de technologie de Troyes, Institut Charles Delaunay
Keywords:	FLOW SHOP SCHEDULING, GENETIC ALGORITHMS, RESOURCE ALLOCATION
Keywords (user):	multi-resource operations, re-entrant shop

Minimizing the weighted number of tardy jobs in a hybrid flow shop with Genetic Algorithm

C. Desprez[(1)(2)], F. Chu[(2)], C. Chu[(3)]
[(1)] Commissariat à l'Énergie Atomique, Centre de Valduc – 21120 Is-sur-Tille, France
[(2)]Université de technologie de Troyes, Institut Charles Delaunay (FRE CNRS 2848) - 10010 Troyes, France
(3) Laboratoire Génie Industriel, Ecole Centrale Paris, Châtenay-Malabry, France

Abstract. This paper considers a real-world industrial problem in order to minimize the (weighted) number of tardy jobs. This problem occurs in a company where due dates are associated with parts, and penalties incur when the parts are completed after the due dates, whatever the magnitude of the tardiness. Therefore, the objective function can be modelled as minimization of the (weighted) number of tardy jobs. The system studied is a hybrid flow shop with re-entrance (or recirculation). In order to deal with large size problems arising in real life, a Genetic Algorithm (GA) is implemented. A coding system, adapted to the considered problem, is designed, and existing crossover and mutation operators are adapted to this coding. To evaluate the effectiveness of the proposed method, it is tested against a commercial software package. The results show that the proposed GA performs well on the scheduling part for a given resource allocation, but it still requires an effective resource allocation procedure.

Keywords : *hybrid flow shop, re-entrance, number of tardy jobs, Genetic algorithm*

1 Introduction

The problem addressed in this paper is a real-world industrial problem. This study is carried out for a manufacturer in order to improve its customers' satisfaction.

This firm manufactures complex objects. All the objects manufactured can be considered as identical, and thus one can talk about series production. The number of objects to be produced being relatively small, due to their complexity, the problem cannot be considered as a cyclic production. But the number of operations involved in an object is very large and the total number of operations to schedule may be as large as 5000 during the planning horizon.

The shop considered can be seen as a flow shop, which is a very classical production system, but the originality of the problem lies in the number of constraints to take into account, and the large number of operations to schedule.

The main features to be taken into account in the model are presented here.

Firstly, every object must follow a fixed sequence of operations. This means that the order of the operations in any object is known and these operations cannot be swapped: there are precedence constraints between them. Moreover, the first operation of each object is not necessarily ready to start at time 0: There is a release date for each object.

Secondly, each operation may need more than one resource to be processed. These resources can be machines as well as operators: A control operation, for instance, requires a control machine and at least one controller. In the considered problem, the number of resources required for an operation is either 2 or 3. It is thus a multi-resource problem.

Thirdly, for each operation, and for each type of resource needed, there exists more than one resource capable of processing the operation. For instance, if one considers a painting operation that requires one painter, several painters are qualified to perform this operation. For each operation, a resource has to be chosen among a given set of qualified resources. Furthermore, it is not possible to aggregate some resources to obtain a mono-resource problem: A given operator does not always work with the same machine or the same other operator.

Fourthly, the operators are polyvalent. This is not a constraint itself, but makes the problem more complex because, since the operators can process more than one kind of operation, they are involved in different stages of a job. This explains the re-entrance of the system. As an example, consider two different control operations, control A and control B. The object will go through control A, then through a number of other operations, and finally back to the same operator for control B: The operator who makes control A and control B is a re-entrant resource.

Finally, the due dates are fixed and known in advance. Each delivery date is associated with a number of objects to deliver. If there is one or more object missing, then the firm gets a penalty, and, since the transportation costs are significant, the objects have to be delivered at the next due date: The object is either on time or late, and if it is late there is no need to know the magnitude of the lateness. To improve the customers' satisfaction, the firm wants to minimize the number of late objects. This number may be weighted by the penalty cost, different for each customer and thus for each delivery. As all objects are identical, a due date is not associated with an object in particular. These due dates are said to be generalized or positional.

To summarize, the problem dealt with here is the minimization of the weighted number of tardy jobs in a re-entrant flow shop with precedence constraints, release dates, multi-resources operations, and generalized due dates.

The firm is currently using a commercial software package for production scheduling. This software is named CADPlan®. It is based on simple priority rules, such as SPT or EDD. This software does not perform too

badly, but the rules are not well adapted to the complexity of the problem, so this paper aims at developing an effective method that would outperform this software with a comparable computation time.

Due to the complexity and the size of the problem, this paper only deals with the scheduling part while assuming that for each operation, the required resource is already assigned. Therefore, it is assumed that the resource assignment is given, either randomly or by CADPlan®.

This problem is very difficult and its sub-problems are NP-hard. It's obvious that it cannot be optimally solved for large size industrial problems. Since Genetic Algorithms have been well used to solve many scheduling problems, this work develops a Genetic Algorithm. For this purpose, this paper proposes a new coding system avoiding twin solutions. The classical crossover and mutation operators are adapted to this new coding system and guarantee the feasibility of the yielded solutions, despite the complexity of the problem. Because of the complexity related to the re-entrance, the multi-resource operations and the flexibility of resources, a specific decoding procedure is needed to calculate the fitness of individuals. To evaluate its efficiency, the obtained results are compared with those given by the commercial software called CADPlan® currently used in the company. The improvement is very significant since it can be nearly 50% on the average, despite the complex features of the problem (precedence constraints, release dates, multi-resource operations). This shows that the developed method can contribute very much to the performance improvement of the company. Due to the fact that the problem has never been addressed in the literature, the method also makes a contribution to the state of the art of the research in production scheduling.

Preliminary results for this problem were presented at ICSSSM 06 conference in Troyes (France), see [11]. This paper is an extended version that completes and improves the algorithm and results reported in the conference paper.

The remainder of this paper is organized as follows. Section 2 presents a brief review of the literature, both on the scheduling and resource allocation problems. Section 3 describes the proposed method. Section 4 reports the numerical results of the experiments. Finally, section 5 concludes the paper and provides some perspectives for further works.

2 Literature review

The minimization of (weighted) number of tardy jobs has been studied in single machine case and in classical flow shop. This section reviews the related literature for this problem.

2.1 Single machine problems

2.1.1 Minimizing the number of tardy jobs

For the unweighted problem, if there are no particular constraints (i.e. all release dates are equal, no precedence relations, etc.), the problem can be solved to optimality in polynomial time due to the algorithm of Moore [17] which runs in $O(n^2)$ time, where n is the number of jobs to be scheduled. If release dates are different, the problem becomes NP-hard. [8] presented a lower bound for this problem, along with a heuristic that shows good results on small size instances (up to 50 jobs). [3] proposed some dominance rules and a Branch-and-Bound algorithm based on dominance rules, which solves to optimality all small size instances and more than 50% of larger size instances (up to 200 jobs). [10] also used a Branch-and-Bound approach, based on the notion of master sequence; i.e., a sequence containing at least one optimal solution. They managed to solve 95% of 140-job instances optimally in less than one hour. [6] presented a slightly different approach. The authors started by identifying a critical path from which they computed the maximum tardiness, and then they used a Branch-and-Bound algorithm to find a sequence that minimizes the number of tardy jobs while respecting the maximum

tardiness. In [1] the case with preemptive jobs was studied, and an $O(n^4)$ algorithm based on dynamic programming was proposed.

When generalized due dates are involved, [7] gave some dominance rules, and showed that the SPT rule is optimal, if the release dates are identical. For problems with precedence relations, [23] established the strong NP-hardness.

2.1.2 *Minimizing weighted number of tardy jobs*

In the weighted case, with no additional constraints, the problem is polynomially solvable if all processing times are equal (see [24]). In the general case (i.e. when processing times are different), [14] developed a Genetic Algorithm to minimize the weighted number of both early and tardy jobs, and reduced this number by a third when compared to an existing heuristic, on problems with up to 80 jobs. [22] also provided a Genetic Algorithm and showed that the quality of the initial population is of high importance for the efficiency of the final result. This GA was used on problems with up to 100 jobs.

[2] dealt with the problem with release dates, and with equal processing times. The author proved that this problem is polynomial, and proposed an $O(n^7)$ algorithm, based on dynamic programming. He also proposed an $O(n^{10})$ algorithm for the preemptive case.

With precedence relations between jobs, the weighted problem is NP-hard, according to [24]. If the due dates are generalized, [21] showed that, when the weights are associated to the due dates, the SPT rule is optimal. When the weights are on the jobs, then the problem is still polynomially solvable and an algorithm was provided. However, the problem with release dates, weights being on due dates or on jobs, is intractable.

2.2 *m machines problems*

For the problem with more than one machine, only two articles were found. The first one ([5]) dealt with a 2-machine flow shop, and with the objective of minimizing the weighted number of tardy jobs. The authors proposed an exact method in $O(n^3\log(n))$. The second one considered m machines, but the objective function is unweighted: [19] proposed a GA that solves small size instances (30 jobs on 15 machines) in less than 30 seconds, and to near-optimality.

2.3 *Hybrid flow shops*

In the papers cited above, whether with one or two machines, the basic assumption was that there is no choice in the machine to perform each operation. In the case considered here, for each operation, a machine can be chosen among several qualified ones. In the literature, this model is called hybrid flow shop (HFS). In an HFS (also called multiprocessor flow shop), a job visits all stages, and at one or more stage, more than one machine is available to process the operation. A good review on HFS can be found in [16]. According to this review, most of the papers on this subject focused on 2-stage HFS, which are NP-complete. The only study it reported with more than three stages was on minimizing the sum of machine changeovers. This review is not related to any study on minimizing tardiness or even makespan. [18] was another review on HFS, but is rather concentrated on adapting local search methods from the flow shop scheduling to the hybrid flow shop scheduling. In this field, taboo search and variable-depth search of a neighbourhood seem to be the best-performing methods.

There is a very interesting paper on the hybrid flow shop with 3 stages and recirculation ([4]). Recirculation means that a job may go through a given stage more than once. According to the authors, there seem to be no other paper dealing with an HFS with recirculation to minimize the total weighted number of tardy jobs. In this paper, they implemented several dispatching rules, and included the obtained sequences into the initial population of a Genetic Algorithm. As it can be seen in their experimental results, this association performs better than GA

alone or dispatching rules alone. The problems tested vary from 20 to 400 jobs.

To be closer to the industrial case studied in the present paper, a further particularity should be added to the model: in addition to an HFS, a multi-resource Hybrid Flow Shop has to be dealt with. This means that at each stage, the processing of a job requires more than one resource (machine or agent), and each of these resources can be chosen among a set of available ones (flexibility). To the best of the knowledge of the authors, there is no such study in the literature, except [9] that considered a multi-resource Flexible Job Shop with nonlinear precedence relations (i.e., an operation may have more than one predecessor and/or more than one successor). In this latter paper, the authors introduced a modified disjunctive graph representation to take into account the flexibility and the multiplicity of resources, as well as nonlinear routing. Then they defined a neighbourhood and explored it with a taboo search approach. This method gave an improvement of 24% of the makespan, from the initial solution to the best solution found.

As far as the authors know, [11] is the first to consider the problem of the minimization of the weighted number of tardy jobs in a re-entrant flow shop with precedence constraints, release dates, multi-resources operations, and generalized due dates in the literature. [11] uses a Genetic Algorithm and developers a new code that is suitable to any series production problem, but does not describe in detail the algorithm developed.

This paper is an extended version of the conference paper [11]. In this paper, the problem is described in detail and the literature review is completed. The present paper improves the efficacy of the method proposed in [11], by introducing heuristics in the initial population. The comparisons for the crossover operators and the effect of the mutation phase are detailed. The evaluation section is strengthened by clarifying the decoding. More numerical experiments were done in this paper. For example, both fixed resources and flexible resource allocation cases have been considered; tests for the unweighted and the weighted problems are compared. This paper also indicates further research directions to further improve the performance of the proposed Genetic Algorithm, especially in the flexible resource allocation setting.

3 Scheduling with the Genetic Algorithm

As can be seen from section 2, most of the existing models in scheduling problems deal with a small number of jobs and with few constraints. In the considered case, in order to comply with industrial requirements, a method is needed that would combine both quickness and efficiency even on large size problems such as the problem at hand. Indeed, the number of feasible schedules exponentially increases with the number of operations to be scheduled.

In this context, metaheuristics are known to be very powerful methods. Metaheuristics enclose a set of general methods adaptable to different problems. For this purpose, Genetic Algorithm is chosen, because it is able to explore a good part of the solution space in a reasonable amount of time.

The Genetic Algorithm was first described in [12]. It was originally used for signal processing. [20] reviewed its application to scheduling problems.

The Genetic Algorithm (GA) is based on the principles of biological sexual reproduction: two individuals, called parents and constituted of genes, reproduce and give birth to two new individuals, called children, each child receiving a part of each parent's genes or characteristics. Sometimes, random mutations appear during the reproduction process and the children get genes coming from none of the parents. Then, eldest or weakest individuals die and leave room for younger and stronger individuals. In this way the population gradually improves itself, following the idea of Darwin's Natural Selection (see fig. 1).

Insert Figure 1 here

In the GA, the individuals are feasible solutions and the equivalent of the sexual reproduction is called crossover. It starts with an initial population, constituted of a number of individuals generated most often either randomly or using heuristics. Some parents are selected among this population, and are subjected to a crossover operator. This operator will create two new feasible solutions (the children). A mutation operator will then be applied to the children with a generally low probability. At some point of the algorithm, the renewal of the population consists in selecting some individuals from the current population and to replace them with newly-created children. A new generation is obtained, and the algorithm goes on until a stopping criterion is reached. The stopping criterion is often related to the improvement of the best solution between successive generations.

The crossover part is called intensification phase, because it is assumed that, by taking two parents with good features and by crossing them, the children will combine the good features from both parents to get even better features (and thus, lead to a solution of better quality). The mutation is called the diversification part. Its aim is to maintain a good level of diversity in the population, and to avoid redundancy between the solutions explored: It prevents the search from staying stuck in a local optimum.

A good description of GA scheme, and of several crossover and mutation operators, is given in [13].

As said above, a metaheuristic is just a general skeleton of a method. Then, each component of the GA must be adapted to the considered problem, starting with the coding of the individuals.

3.1 Coding

The coding makes it possible to represent feasible solutions, namely feasible schedules, into individuals that can be handled by the algorithm. In scheduling problems, the most classical coding is a coding by operation number. Each operation is attributed a unique number, and the schedule is represented as a list of operation numbers. This coding is called indirect, because it does not represent directly the schedule. To obtain a schedule with starting dates the individual must be decoded.

Example 1:

Consider 5 objects to be manufactured. Each object must go through 4 operations A, B, C and D in this order. The data concerning these operations (considering that the operations A, B, C and D are the same for all 5 objects) is in the table 1.

Insert Table 1 here

To use the coding by operation number, a number must be assigned to each operation as in Table 1. Then, an individual for the GA, representing a feasible solution for the considered problem, can be coded as I1 (see fig. 2).

Insert Figure 2 here

This coding is widely used for representing solutions in scheduling problems. However, it is not well adapted to the case considered in this paper. In fact, it could be used, but it does not take into account the fact that the objects are identical. Consider another feasible individual I2 with respect to the data given in example 1 (see fig. 2).

Since all objects are identical, operations 0 – 4 – 8 – 12 – 16 are the same operation (type A), as well as 1 – 5 – 9 – 13 – 17 (type B), 2 – 6 – 10 – 13 – 18 (type C), and 3 – 7 – 11 – 15 – 19 (type D). Then, individuals I1 and I2 represent the same list of operations and thus the same schedule. When the number of objects to manufacture is large, this situation very frequently occurs, and therefore the number of "twin individuals"; i.e., the number of different individuals representing the same schedule, will be very large.

In order to take advantage of the fact that all objects are identical, and therefore that any operation on an object is strictly equivalent to the same operation on any other object,

coding based on the type of the operations, rather than on their number, is developed. Using this new coding, both individuals I1 and I2 above will be represented as I3 in fig. 2.

In this way, the number of twin individuals can be significantly reduced, and then the population diversity is improved. This coding is based on the fact that all object manufactured are identical; therefore it can be used in any serial production problem.

In this considered problem, the type of operation is represented by a number rather than by a letter as I3' in fig. 2, because it will be simpler to implement in the algorithm.

3.2 Feasibility constraints

After the definition of the coding system, the conditions for an individual to represent a feasible schedule must be expressed.

Two constraints must be respected. The first one is that each type of operation must occur in the individual as many times as the number of objects. Define:

$N_w^I(k)$ as the number of occurrences of operations of type w in individual I up to position k (included). I can be dropped when there is no ambiguity about the individual considered;

m as the number of objects to manufacture;

n as the number of operations in each object (and therefore the number of types).

With these notations, the constraint expressed above can be written as:

$$N_w(n*m-1) = m \quad \forall w \in \{0,1,...,n-1\}$$

This constraint also ensures that the number of genes in the individual will be equal to the total number of operations.

As an example, individual I4 in figure 3 does not respect this first constraint:

$N_0(11) = 4 > m$

$N_3(11) = 2 < m$

The second constraint is precedence relations. This constraint can be written as follows:

$$N_w^I(k) \le N_{w-1}^I(k) \quad \forall w \in \{1,...,n\}$$

This inequality requires that, if at position k of an individual, only x operations of type w-1 have been scheduled, then at the same position there cannot be more than x operations of type w scheduled. Otherwise, it would mean that at least one operation of type w has been scheduled before its predecessor.

As an example, individual I5 in figure 3 does not respect this second constraint:

$N_1(2) = 2 > N_0(2) = 1$

Insert Figure 3 here

3.3 Evaluation

The evaluation step consists in determining the fitness of an individual. The fitness is the performance of the considered individual related to the optimization criterion. Therefore, in the proposed GA, the evaluation step will be the calculation of the wNT (weighted number of tardy jobs) induced by an individual.

Beforehand, the individual must be decoded; i.e., the sequence of genes must be translated into a feasible schedule. In biology, this decoding corresponds to the translation of the genotype into the phenotype.

Consider an individual I, coded following the proposed coding. L(i) is the value of the gene at position i where i=0, …, n×m-1; L'(i) will be the value of the gene at the same position in the decoded individual I' (I' will then contain a list of operations rather than a list of type of operations). The relation between L(i) and L'(i) is given by:

$$L'(i) = L(i) + n \times N_{L(i)}^I(i-1)$$

Figure 4 presents an example of decoding.

Insert Figure 4 here

Then, by considering successively each gene from the decoded individual L', the starting dates of the operations can be calculated.

Let $\Pi(i)$ denote the set of operations that have at least one resource in common with operation i, and that appear before operation i

in the individual. Let i^- be the immediate predecessor of operation i in an object. If i is the first operation of the object, then i^- is set to -1, without loss of generality.

Then, the starting date $S(i)$ of the corresponding operation i is:

$$S(i) = \max_{j \in \Pi(i)} \left\{ S(i^-) + p_{i^-} ; S(j) + p_j \right\}$$

where p_j is the processing time of operation j.

The last step of this evaluation phase is to calculate the weighted number of tardy jobs associated with the solution.

Insert Figure 5 here

Figure 5 gives an example. In this figure, objects 0 and 1 are completed on time for deliveries (due dates) #0 and #1. Object 2 is completed too late for delivery #2: It will be delivered in delivery #3, for which it is on time. Object 3 is delivered in delivery #4. Object 4 is delivered in delivery #6 since it is completed too late for delivery #5. Objects 5 and 6 will be used to replace deliveries #2 and #5, which are late. The weighted number of tardy jobs is the sum of weights of the deliveries which are late (#2 and #5), i.e. 2 + 2 = 4.

3.4 Generation of the initial population

There are several ways to generate an initial population of feasible individuals. Generally, these individuals are generated randomly or using simple heuristics. In the proposed method, they are first generated randomly to avoid increasing the computing time. Then a few simple heuristics are implemented to improve the algorithm's efficiency.

The heuristics used are the following:

- EDD (Earliest Due Date), which schedules first the operations belonging to the object with the earliest due date. It means that, in the sequence, all operations from an object will be placed together. For example, if there are 10 objects constituted of 5 operations each, the sequence given by EDD will be: 0-1-2-3-4 repeated 10 times;
- SPT (Shortest Processing Time) which places the operations in nondecreasing order of their processing times.
- FIFO (First In First Out) which places the operations in nondecreasing order of their availability date (i.e. the release date of the object plus the sum of processing times of the preceding operations in this object)
- Slack (Remaining Time) which places the operations in nondecreasing order of the slack.
- BATCH which places all operations of a same type together. For example, with 10 objects of 5 operations, the sequence will be: 0-0-0-0-0-0-0-0-0-0-1-1-1-1-1-…

In case of ties, the operation with the smallest type is scheduled.

As a consequence, the initial population contains 5 individuals generated by heuristics. The others are generated randomly. To ensure that the individuals are feasible, before generating a gene, all possible operation types are sorted according to one of the above rules. The gene is either chosen randomly from the list or the head of the list is chosen.

This way of generating individuals ensures that they will be feasible.

3.5 Crossover

Before the crossover operation, a number of parents must be selected. Some authors use the whole population as parents. In the method presented, only a part of it is selected. For the selection, each individual is given a probability of being chosen: a number S is randomly drawn such that:

$S \in [0.2 \times \min; 1.2 \times \max]$ where min and max are the minimum and maximum values of the weighted number of tardy jobs in this generation, respectively. Any individual I such that the weighted number of tardy jobs is less than S is selected with a probability $p_S(I)$ to be selected:

$p_S(I) = \frac{1.2 \times \max - wNT(I)}{1.2 \times \max - 0.2 \times \min}$ where $wNT(I)$ is the criterion value corresponding to individual I.

In this way, the better the individual, the higher chance it gets to be selected as a parent.

The individuals are examined in nondecreasing order of their wNT (i.e. best individuals first). At the end of the selection procedure, if the number of parents selected is odd, the last parent selected is discarded. The crossover takes place between two parents randomly drawn in the list.

For the crossover, a crossover operator is needed. There exist a large variety of operators. Some of them do not create feasible individuals: They require a repairing afterwards. Other operators create feasible individuals: They can be more complex but avoid repairing the created children. In the proposed GA, an operator of this latter kind is chosen.

Three different crossover operators have been tested. All three of them already existed in the literature, but they had to be adapted to the proposed coding system and to the feasibility constraints.

3.5.1 1X operator

1X means 1-point crossover. The principle of this operator is simple: Each parent is divided into two parts, and the children are both made with one part from each parent (see fig. 6).

Insert Figure 6 here

A cross point (CP) is randomly chosen between position A = 0.25**n***m* and position B = 0.75**n***m*. These positions ensure that the crossover will produce children different from their parents: If the cross point is too close from an end of the individual, most of the genes will be copied directly from one parent to the child. Then, both children will be too similar to their parents.

Then there are three steps:

Step 1: The first part of parent 1 is copied into child 1.
Step 2: All genes appearing in the first part of parent 1 are deleted from parent 2.
Step 3: The remaining genes are copied from parent 2 to child 1.

These steps are then repeated for the second child by inversing the role of parents 1 and 2.

Since the genes coming from parent 1 are identical to this parent, and genes coming from parent 2 are in the same relative order as in parent 2, one can conclude that, if both parents respect the feasibility conditions, then both children will also do.

3.5.2 2X operator

2X means 2-point crossover. This operator works similarly to 1X. Here two cross points are randomly chosen, the first one cpA $\in$ [0; CP-0.25**n***m*] and the second one cpB $\in$ [CP+0.25**n***m*; *n***m*-1]. Then, the steps are virtually the same as in 1X crossover (see fig. 7)

Insert Figure 7 here

Step 1: All genes between cpA and cpB are copied from parent 1 to child 1.
Step 2: All genes appearing in the middle part of parent 1 are deleted from parent 2.
Step 3: The remaining genes of parent 2 are copied to child 1.

These steps are then repeated for child 2 by inversing the role of the parents.

Both children respect the feasible conditions if both parents do, for the same reason as in 1X crossover.

3.5.3 POX operator (Precedence preserving Order based Crossover)

This operator has been designed by [15].

Here the positions of all the operations from a randomly chosen task (object) are simply swapped between the parents (see fig. 8).

Insert Figure 8 here

Step 1: A task number is chosen randomly.

Step 2: The operations from this task are copied identically from parent 1 to child 1.

Step 3: The rest of child 1 is filled with the operations of the other tasks from parent 2.

Here, all the operations of each task keep the same relative order as in parents 1 and 2. Then, the feasibility conditions are met for the children if they are also met for the parents.

Some experiments were performed on these operators, to know whether one of them dominates the others. They were applied to the same instances, and compared both in terms of computation time and in terms of solution quality. The results show that POX operator performs clearly better than 1X and 2X (see figure 9). This conclusion is not surprising: given that POX operator was initially designed for problems with precedence constraints, and thus closer to the considered problem, while 1X and 2X were not specifically designed to take into account this kind of constraints.

Insert Figure 9 here

In further experiments, the operator used will always be POX.

3.6 Mutation

After crossover is complete, a mutation is applied on some children. The mutation aims to explore new solutions that may not be reached by crossing existing solutions. It consists in a random modification of an individual. For each child, the probability of being mutated is 5%. As a mutation operator, the PPS (Precedence Preserving Shift mutation) operator is adapted. As for the POX operator, it has been designed by [15] for problems with precedence constraints.

In the PPS operator, a position is randomly chosen. The corresponding gene is then swapped with another gene, randomly chosen between the immediate predecessor and successor of the operation corresponding to the drawn position (see fig. 10). The relative positions of operations in the chosen task remain the same, thus the feasibility constraints are not violated.

Insert Figure 10 here

Some additional experiments were performed to check the interest of mutation in the GA. The GA was run with and without mutation on the same instances, and the computation time, the solution quality and the convergence of the algorithm were compared. These experiments showed that without mutation, the solution converges much more quickly, in a few generations (see fig.11). On the contrary, with the mutation step, the computation time only slightly increased, but the solutions are much better. Therefore this mutation step is kept in the algorithm.

Insert Figure 11 here

3.7 Renewal

Once the crossover and mutation are complete, a number of children can be inserted into the population in order to form a new generation. In the proposed GA the population size is kept constant, so as many individuals as the number of children created must be discarded. As for the selection step, each individual is examined. For this purpose, the population is sorted twice: first in nondecreasing order of the weighted number of tardy jobs, and then in nondecreasing order of age (expressed in number of generations). Let rV(I) and rA(I) denote the ranks of an

individual I in these two lists, respectively. Then, a random number is drawn between 0 and 2*pop, where pop is the number of individuals in a population. If this number is less than the sum of rV(I) and rA(I), then the individual I is deleted and replaced by a child. The probability for an individual I to be deleted is then:

$$p_{del}(I) = \frac{rA(I) + rV(I)}{2 \times pop}$$

In this way, individuals with a better criterion value and/or younger in terms of number of generations have less chances to be deleted.

This procedure is repeated until all children are inserted into the new population.

3.8 Stopping criterion

There are several ways to stop a Genetic Algorithm. For example, if a lower bound exists, the stopping criterion can be the gap between the best solution found and the lower bound. The GA then stops when it reaches a solution that is within this gap. In the considered case, there is no lower bound to refer to. The number of generations without improvement is used as a stopping criterion: if the best solution found is not improved since the last g generations with g being a control parameter, then this solution is considered as unlikely to be improved any more and so the algorithm is stopped.

4 Numerical results

To check its effectiveness, the GA presented in the previous section was tested against the software currently used: CADPlan®. Since there is no known method to tackle the problem studied here, this software provides an interesting comparison tool that maintains this study in an industrial context.

CADPlan® is able to assign resources itself. To ensure that the comparison between the software and the proposed method (which considers a fixed resource allocation), the problems used in the following tests will either consider fixed resource allocation (the same for both CADplan® and the proposed algorithm), or a flexible resource allocation, in which case the resources are assigned to operations by CADPlan®, and this assignment is used as the fixed resource allocation for the algorithm. In any case, the resource allocation is the same for CADPlan® and for the proposed algorithm.

Two series of tests were conducted. In the first series, the resources used are fixed and known in advance. In the partner company, this is the case when operations are pre-assigned to machines and operators (fixed resources). It means that it is known before scheduling who processes each given operation, and with which machine. Moreover, this resource allocation is considered to be the same for all objects. In the second series of tests, the case when an operation may be done by several machines and/or several operators equally qualified is considered (flexible resources). It means that, for each operation, the set of qualified machines and operators is known, and any machine/operators combination can be chosen among a set. This can be the case, for example, when operators belong to laboratories, and within a laboratory each agent is qualified on the operations processed in the laboratory.

For each series of tests, 5 methods will be compared. The first one is CADPlan®, noted CP in the following tables; the second is the proposed GA. The remaining three are derived from the proposed GA: GA* is the Genetic Algorithm to which the solution computed by CADPlan® is fed as part of the initial population; in GAh the presented heuristics are used to generate a part of the initial population, and GAh* is the GAh with the solution from CADPlan® in the initial population.

For each instance, the size is given by the number of objects to manufacture (nb obj.) and the number of operations to realize for each object (nb ope.).

The mean improvement ratio has been calculated by comparing the sums of wNT for all methods. The mean improvement ratio between two methods A and B is then the difference between the sum of wNT given by A minus the sum of wNT given by B, divided

by the sum of wNT by A (and multiplied by 100 to get a percentage).

All 5 methods are tested on both weighted and unweighted problems.

The GA was coded in C, using Visual C++. The computer used to run the tests is a 3GHz processor with 1Go RAM.

4.1 Fixed resources

Table 2 shows the comparison results when the resource allocation is fixed and known, and on problems where the due dates are unweighted (all weights=1).

Insert Table 2 here

For each instance, GA outperforms CP. The mean improvement ratio is 21.3% for the GA when compared to CP. When the solution from CP is added to the initial population of the GA (GA*), the improvement ratio is 33.9%; using heuristics to generate a part of the initial population (GAh and GAh*) seems to be profitable. GAh gives better results than GA*. However, the combination of the two methods (GAh*) improves both results, with an improvement of 41.2%. This means that, not only a good initial solution leads to good final results, but also the number of good solutions in the initial population helps the Algorithm to generate better solutions by combining them through the crossover phase, and then to reach even better results.

Insert Table 3 here

Table 3 gives the results of the same comparison but now the due dates are weighted. One can see here that the improvements are not affected by the addition of weights.

4.2 Flexible resource allocation

This is the case where machines and operators can be chosen among sets of competent resources. Since the proposed GA does not provide a resource allocation module, CADPlan® must be run first and then the resources chosen by CADPlan® are used as fixed resources for the Genetic Algorithm. Table 4 shows the comparison results for the unweighted case.

Insert Table 4 here

First notice that the proposed GA performs badly, whether the solutions generated with heuristics (GA and GAh) are included in the initial population. This is due to the fact that the resources have been fixed by CADPlan® simultaneously with the scheduling: this allocation is linked with the schedule.

Operations are scheduled progressively, and resources are chosen for each operation among the available resources. Thus, the allocation suitable for a given schedule is not efficient when it is used with other schedules, and then a schedule that fits well to the resource allocation cannot be found.

However, when the solution obtained with CADPlan® is included in the initial population, the final solution is much better. The improvement is nearly 50% on the average for GA* and GAh*. This is because both the resource allocation and the initial schedule were obtained from CADplan®. The GA can then generate new solutions from CADPlan®'s, which have common points with the initial schedule. The GA needs a good starting point to find its way towards a good solution. In table 5, the results in the weighted case show that the addition of weights does not change the validity of these observations.

Insert Table 5 here

4.3 Statistical analysis

These results are analyzed to check whether all methods were significantly different or not. The Wilcoxon matched-pairs signed-ranks test was used. This test aims at

determining whether two populations are significantly different.

The results from this analysis show that all five methods are different one from another, except in two cases.

- With fixed allocation, in the unweighted case, GAh and GAh* are not significantly different (GA with heuristics, with or without the solution from CADPlan® in the initial population);
- With flexible allocation, in both weighted and unweighted cases, GA* and GAh* are not different (GA with the solution from CADPlan®, with or without solutions yielded by the heuristics in the initial population).

These results confirm initial observations.

The Wilcoxon test was also used to see whether the flexibility of the allocation was influent on the solution found by a given method. This test was applied to each method, both in the unweighted case and weighted case, to compare between the results with fixed allocation and the results with flexible allocation.

The results of this analysis show that, as far as CADPlan® is concerned; i.e. in GA, GA* and GAh* methods, there is a significant difference between the results according to the type of allocation (fixed or flexible). This is due to the fact that, in the flexible case, the resource assignment is done by CADPlan®. When CADPlan is not involved (GA and GAh methods), the results are not significantly different: The resource allocation is just an input data, whether it comes from CADPlan® or is generated randomly.

4.4 Results on computation time

Previous sections show that the proposed GA performs better than CADPlan® in terms of solution quality. The experiments made also show its superiority on the computation time level, as can be seen in figure 12.

Insert Figure 12 here.

5 Conclusions

This paper proposed a Genetic Algorithm to minimize the weighted number of tardy jobs in a real-world multiresource hybrid flow shop involving re-entrance. This problem corresponds to an industrial case, and this paper aims at developing a method that gives efficient results in a reasonable amount of computation time. For this purpose, a Genetic Algorithm is used. A new coding system was proposed and existing crossover and mutation operators were adapted to take into account the specificities of the problem, and several heuristics were added to generate a part of the initial population.

To check the efficiency of the proposed method, it is compared with a commercial software package named CADPlan®. The results show that, if resources are fixed in advance, the proposed GA performs much better than CADPlan®, and even better if some initial solutions come from CADPlan® and/or heuristics, with an improvement of up to 42%. When resources are flexible, and assigned using CADPlan®, the proposed GA needs the schedule from CADPlan® as one of the initial solutions to give efficient results and the improvement is as much as nearly 50%.

One can conclude that, to be efficient even with flexible resources, resources allocation and operations scheduling must be processed simultaneously. It would be interesting to develop further research by simultaneously scheduling the operations and assigning the required resources.

6 References

[1] P. Baptiste, *An $O(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs.* Operations Research Letters 24, pp. 175-180, 1999

[2] P. Baptiste, *Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times.* Journal of Scheduling 2, pp. 245-252, 1999

[3] P. Baptiste, L. Peridy, E. Pinson, *A branch and bound to minimize the number of late jobs on a single machine with release time constraints,* European Journal of Operational Research 144, pp. 1-11, 2003

[4] S. Bertel, J. C. Billaut, *A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation.* European Journal of Operational Research 159, pp. 651-662, 2004

[5] R. L. Bulfin, R. M'Hallah, *Minimizing the weighted number of tardy jobs on a two-machine flow shop.* Computers and Operations Research 30, pp. 1887-1900, 2003

[6] P. C. Chang, L. H. Su, *Scheduling n jobs on one machine to minimize the maximum lateness with a minimum number of tardy jobs.* Computers and Industrial Engineering 40, pp. 349-360, 2001

[7] C. Chu, *A new class of scheduling criteria and their optimization.* RAIRO Operations Research vol. 30 n°2, pp. 489-509, 1996

[8] S. Dauzère-Pérès, *Minimizing late jobs in the general one machine scheduling problem.* European Journal of Operationnal Research 81, pp. 134-142, 1995

[9] S. Dauzère-Pérès, W. Roux, J. B. Lasserre, *Multi-resource shop scheduling with resource flexibility*. European Journal of Operational Research 107, pp. 289-305, 1998

[10] S. Dauzère-Pérès, M. Sevaux, *An exact method to minimize the number of tardy jobs in single machine scheduling*. Journal of Scheduling 7, pp. 405-420, 2004

[11] C. Desprez, C. Chu, F. Chu, *A Genetic Algorithm for minimizing the weighted number of tardy jobs*, International Conference on Service Systems and Service Management, 25-27 october 2006, Troyes, pp. 1271-1276, 2006

[12] J. H. Holland, *Adaptation in natural and artificial systems*. The University of Michigan Press, Ann Arbor, MI, 1975

[13] I. Kacem, *Ordonnancement multi-critère des job shops flexibles: formulation, bornes inférieures et approche évolutionniste cooperative*. Ph. D. thesis, Ecole Centrale de Lille, University of Lille 1, pp. 43-63, 2003

[14] C. Y. Lee, J. Y. Choi, *A genetic algorithm for job sequencing problems with disctinct due dates and general early-tardy penalty weights.* Computers Operations Research vol. 22 n° 8, pp. 857-869, 1995

[15] K. M. Lee, T. Yamakawa, K. M. Lee, *A genetic algorithm for general machine scheduling problems*. 2nd International Conference on Knowledge-Based Intelligent Electronic Systems, 1998

[16] R. Linn, W. Zhang, *Hybrid flow shop scheduling: a survey.* Computers and Industrial Engineering 37, pp. 57-61, 1999

[17] J. M. Moore, *An n jobs, one machine sequencing algorithm for minimizing the number of late jobs.* Management Science vol. 15 n°1, pp. 102-106, 1968

[18] E. G. Negenman, *Local search algorithms for the multiprocessor flow shop scheduling problem.* European Journal of Operational Research 128, pp. 147-158, 2001

[19] G. C. Onwubolu, M. Mutingi, *Genetic Algorithm for minimizing tardiness in flow shop scheduling.* Production Planning and Control vol. 10 n° 5, pp. 462-471, 1999

[20] M. C. Portmann, *Genetic algorithms and scheduling: A state of the art and some propositions*. Proceedings of the

Workshop on Production Planning and Control, Mons, Belgium, pp. i-xxiv, 1996

[21] X. Qi, G. Yu, J. F. Bard, *Single machine scheduling with assignable due dates.* Discrete Applied Mathematics 122, pp. 211-233, 2002

[22] M. Sevaux, S. Dauzère-Pérès, *Genetic algorithms to minimize the weighted number of late jobs on a single machine.* European Journal of Operational Research 151, pp. 296-306, 2003

[23] C. Sriskandarajah, *A note on generalized due dates scheduling problems.* Naval Research Logistics vol. 37, pp. 587-597, 1990

[24] C. S. Wong, M. Yan, G. H. Young, *Complexity of scheduling problems with generalized due dates.* Journal of Combinatorial Mathematics and Combinatorial Computing 22, pp. 51-63, 1996.

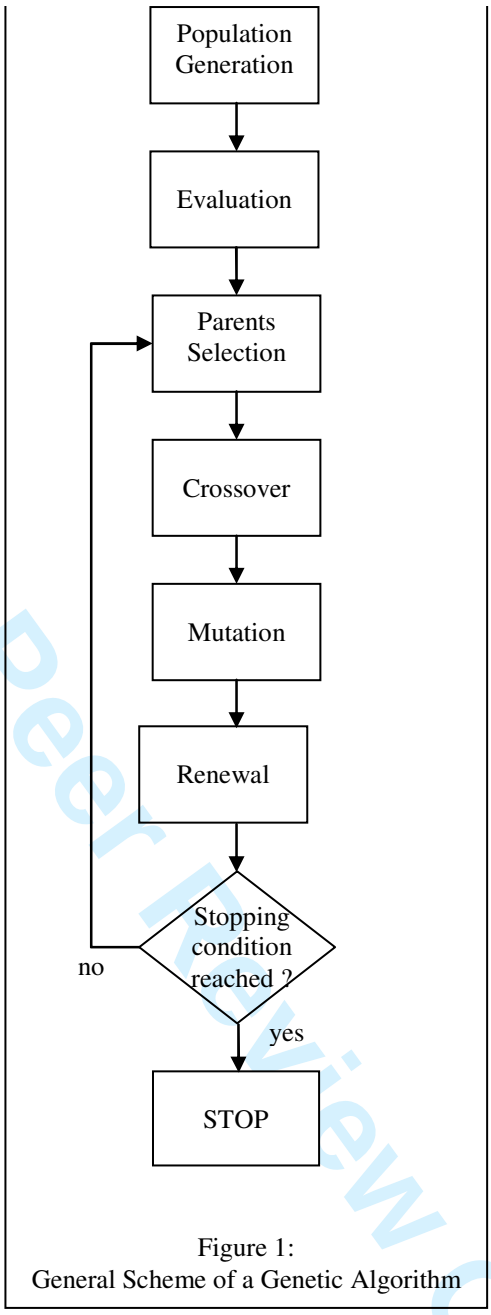

Figure 1:
General Scheme of a Genetic Algorithm

Operation	A	B	C	D
Processing time	5	3	2	4
Resources	1 – 2	3 – 4	1 – 3	2 – 4

k	1	2	3	4	5
Release date of object k	0	0	10	20	20
k^{th} due date	20	22	24	26	28

Object	1				2				3				4				5			
Operation	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Table 1
Data for example 1

I1:	0	1	4	2	5	6	7	8	3	12	9	10	16	11	13	17	14	15	18	19
I2:	0	1	4	2	5	6	3	12	7	8	13	14	16	15	9	17	10	11	18	19
I3:	A	B	A	C	B	C	D	A	D	A	B	C	A	D	B	B	C	D	C	D
I3':	0	1	0	2	1	2	3	0	3	0	1	2	0	3	1	1	2	3	2	3

Figure 2
Individuals I1 and I2 represented with our coding as I3 (with letters) or I3' (with numbers)

I4 : 0 1 0 2 1 0 0 2 1 2 3 3

I5: 0 1 1 0 2 3 0 2 3 1 2 3

Figure 3 :
Example of individuals not respecting
feasibility constraints for n = 4 and m = 3

Coded individual :

I:	0	1	0	2	1	0	1	2	3	3	2	3

Decoded individual :

I':	0	1	4	2	5	8	9	6	3	7	10	11

Figure 4 :
Example of decoding an individual

Object	Completion date	Delivery #	Date	Weight
0	15	0	20	5
1	20	1	20	3
2	25	**2**	**20**	**2**
3	30	3	30	1
4	35	4	30	3
5	40	**5**	**30**	**2**
6	45	6	50	1

Weighted number of tardiness : 4

Figure 5 :
Example of weighted number of tardiness computation

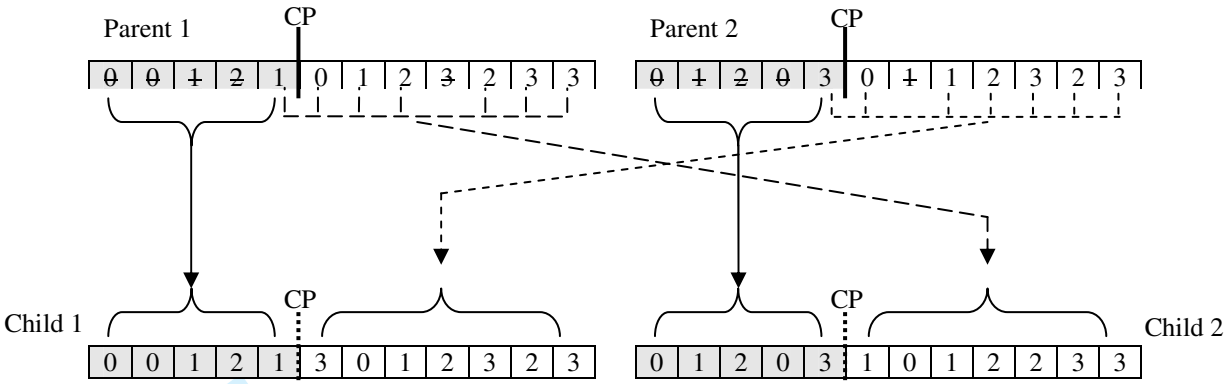

Figure 6 :
1X Crossover Operator

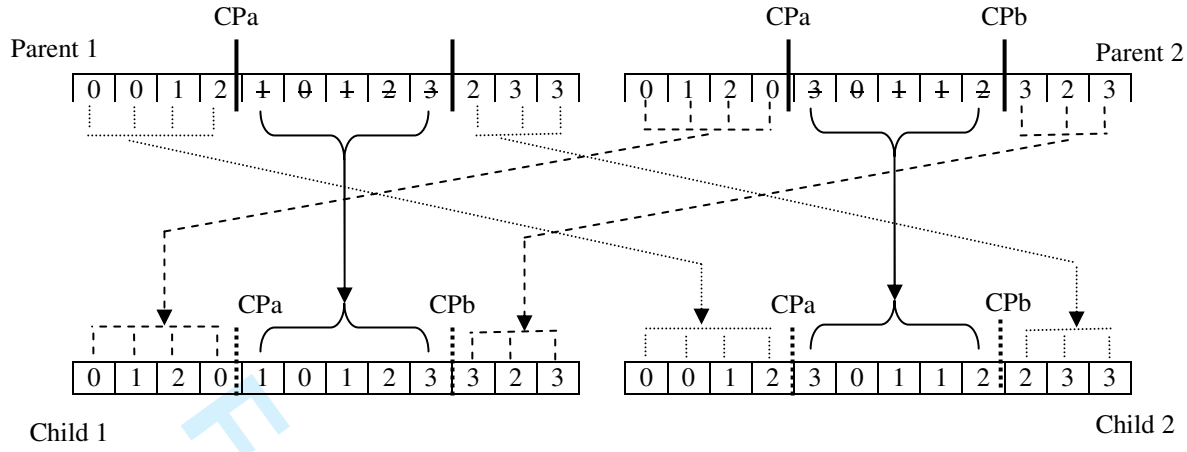

Figure 7 :
2X Crossover Operator

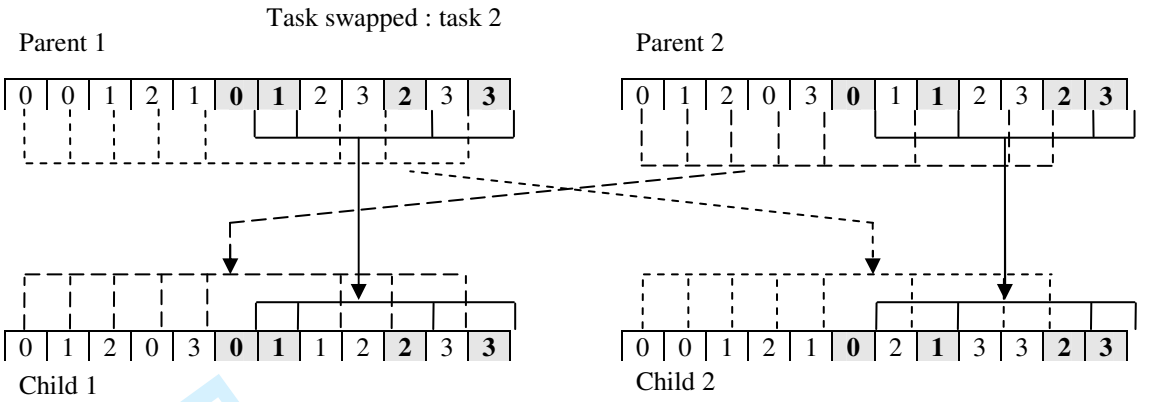

Figure 8 :
POX Crossover Operator

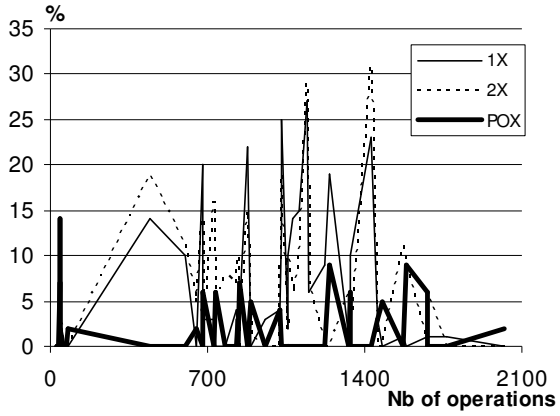

Figure 9 :
Deviation between the wNT found by each operator and the minimum value found

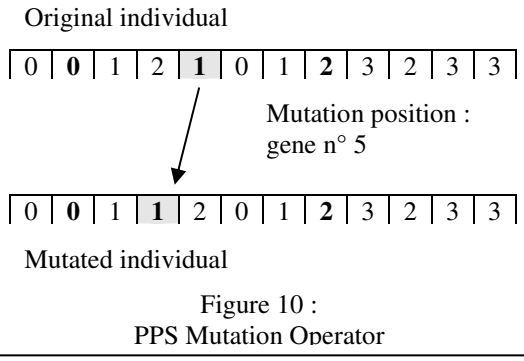

Figure 10 :
PPS Mutation Operator

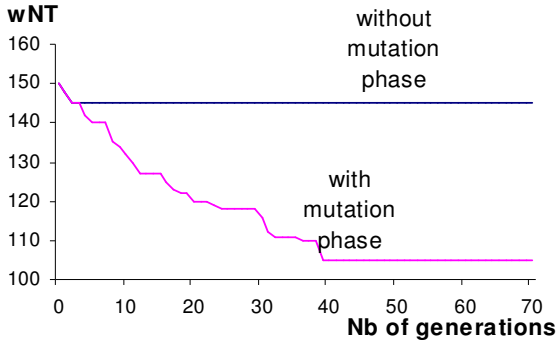

Figure 11 :
Interest of the mutation phase

Pb	nb obj.	nb ope.	CP	GA	GAh	GA*	GAh*
1	50	35	46	37	22	29	22
2	55	37	39	35	26	33	26
3	60	38	38	41	27	27	27
4	65	41	60	49	33	42	33
5	70	37	61	41	33	31	29
6	75	27	62	36	33	35	32
7	80	40	67	52	35	37	35
8	85	33	68	44	37	42	37
9	90	27	30	28	26	26	26
10	95	38	78	57	40	47	41
11	100	37	68	62	47	46	46
12	50	34	50	27	23	24	23
13	55	34	55	40	29	30	29
14	60	32	60	47	32	32	32
15	65	30	46	46	31	33	31
16	70	47	65	50	42	46	42
17	75	44	69	53	42	52	42
18	80	43	56	50	38	44	38
19	90	47	77	67	53	68	53
Sum			*1095*	*862*	*649*	*724*	*644*

Table 2 :
Comparison results
Fixed allocation, unweighted problems

Pb	nb obj.	nb ope.	CP	GA	GAh	GA*	GAh*
1	50	35	144	108	67	83	66
2	55	37	123	113	83	109	83
3	60	38	107	117	80	97	80
4	65	41	181	141	106	127	106
5	70	37	194	124	112	96	91
6	75	27	199	105	103	105	100
7	80	40	211	149	95	116	113
8	85	33	188	122	101	110	101
9	90	27	95	101	88	90	90
10	95	38	234	169	118	150	120
11	100	37	199	172	149	131	131
12	50	34	142	80	71	57	61
13	55	34	156	112	79	89	82
14	60	32	166	123	89	93	89
15	65	30	142	139	104	99	99
16	70	47	159	111	101	120	94
17	75	44	182	128	115	129	100
18	80	43	187	149	116	132	123
19	90	47	240	196	145	203	148
Sum			*3249*	*2459*	*1922*	*2136*	*1877*

Table 3 :
Comparison results
Fixed allocation, weighted problems

Pb	nb obj.	nb ope.	CP	GA	GAh	GA*	GAh*
1	50	35	26	36	28	13	13
2	55	37	16	37	27	7	7
3	60	38	6	41	28	2	2
4	65	41	37	50	39	19	19
5	70	37	30	45	34	13	13
6	75	27	0	37	26	0	0
7	80	40	36	54	39	18	18
8	85	33	0	45	37	0	0
9	90	27	0	45	33	0	0
10	95	38	46	63	48	23	23
11	100	37	29	70	53	12	12
12	50	34	20	33	25	11	11
13	55	34	18	39	26	8	8
14	60	32	0	35	17	0	0
15	65	30	0	35	27	0	0
16	70	47	46	54	41	24	24
17	75	44	52	57	46	26	26
18	80	43	30	53	40	12	12
19	90	47	67	70	57	33	33
Sum			*459*	*899*	*671*	*221*	*221*

Table 4 :
Comparison results
Flexible allocation, unweighted problems

Pb	nb obj.	nb ope.	CP	GA	GAh	GA*	GAh*
1	50	35	83	111	90	47	47
2	55	37	54	120	91	24	24
3	60	38	13	116	75	5	5
4	65	41	117	142	121	54	54
5	70	37	98	136	102	41	41
6	75	27	0	113	84	0	0
7	80	40	107	156	124	60	60
8	85	33	0	118	106	0	0
9	90	27	0	139	109	0	0
10	95	38	136	184	132	75	75
11	100	37	84	196	149	29	29
12	50	34	64	89	73	39	39
13	55	34	47	110	82	27	27
14	60	32	0	89	45	0	0
15	65	30	0	110	74	0	0
16	70	47	122	126	101	53	53
17	75	44	137	130	112	78	78
18	80	43	97	164	134	37	37
19	90	47	195	199	168	98	98
	Sum		*1354*	*2548*	*1972*	*667*	*667*

Table 5 :
Comparison results
Flexible allocation, weighted problems

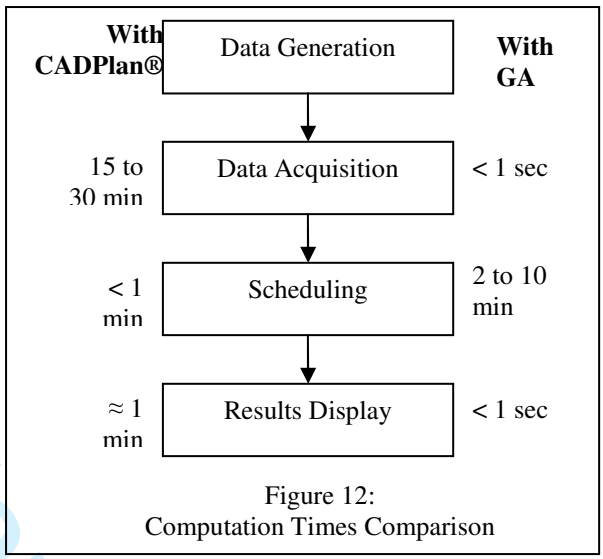

Figure 12:
Computation Times Comparison