10

Hybrid Genetic Algorithms for the Single Machine Scheduling Problem with Sequence-Dependent Setup Times

Aymen Sioud[1], Marc Gravel[1] and Caroline Gagné[2]
[1]*Département D'informatique et de Mathématique, Université du Québec à Chicoutimi*
[2]*Département Des Sciences Économiques et Gestion, Université du Québec à Chicoutimi*
Canada

1. Introduction

Several researches on scheduling problems have been done under the assumption that setup times are independent of job sequence. However, in certain contexts, such as the pharmaceutical industry, metallurgical production, electronics and automotive manufacturing, there are frequently setup times on equipment between two different activities. In a survey of industrial schedulers, Dudek et al. (1974) reported that 70% of industrial activities include sequence-dependent setup times. More recently, Conner (2009) has pointed out, in 250 industrial projects, that 50% of these projects contain sequence-dependent setup times, and when these setup times are well applied, 92% of the order deadline could be met. Production of good schedules often relies on management of these setup times (Allahverdi et al., 2008). This present chapter considers the single machine scheduling problem with sequence dependent setup times with the objective to minimize total tardiness of the jobs (SMSDST). This problem, noted as $1|s_{ij}|\Sigma T_j$ in accordance with the notation of Graham et al. (1979), is an NP-hard problem (Du & Leung, 1990).

The $1|s_{ij}|\Sigma T_j$ may be defined as a set of n jobs available for processing at time zero on a continuously available machine. Each job j has a processing time p_j, a due date d_j, and a setup time s_{ij} which is incurred when job j immediately follows job i. It is assumed that all the processing times, due dates and setup times are non-negative integers. A sequence of the jobs $S = [q_0, q_1,..., q_{n-1}, q_n]$ is considered where q_j is the subscript of the j^{th} job in the sequence. The due date and the processing time of the j^{th} job in sequence are denoted as d_{q_j} and p_{q_j}, respectively. Thus, the completion time of the j^{th} job in sequence will be expressed as $C_{q_j} = \sum_{k=1}^{j}(s_{q_{k-1}q_k} + p_{q_k})$ while the tardiness of the j^{th} job in sequence will be expressed as $T_{q_j} = max(0, C_{q_j} - d_{q_j})$. The objective of the scheduling problem studied is to minimize the total tardiness of all the jobs which will be expressed as $\sum_{j=1}^{n} T_{q_j}$.

Different approaches have been proposed by a number of researchers to solve the $1|s_{ij}|\Sigma T_j$ problem. Rubin & Ragatz (1995) proposed a Branch and Bound approach, which quickly showed its limitations. It could optimally solve only small instances of benchmark files of

15, 25, 35 and 45 jobs proposed by these authors. Bigras et al. (2008) have optimally solved all instances proposed by Rubin & Ragatz (1995) using a Branch and Bound approach with linear programming relaxation bounds. They also demonstrated and used the problem's similarity with the time-dependent traveling salesman problem (TSP). This Branch and Bound approach solved some of these instances in more than 7 days. Because this problem is NP-hard, many researchers used a wide variety of metaheuristics to solve this problem, such as a genetic algorithm (Franca et al., 2001; Sioud et al., 2009), a memetic algorithm (Armentano & Mazzini, 2000; Franca et al., 2001; Rubin & Ragatz, 1995), a simulated annealing (Tan & Narasimhan, 1997), a GRASP (Gupta & Smith, 2006), an ant colonies optimization (ACO) (Gagné et al., 2002; Liao & Juan, 2007) and a Tabu/VNS (Gagné et al., 2005). Heuristics such as Random Start Pairwise Interchange (RSPI) (Rubin & Ragatz, 1995) and Apparent Tardiness Cost with Setups (ATCS) (Lee et al., 1997) have also been proposed for solving this problem. For their part, Sioud et al. (2010) introduce a constraint based programming approach proposing an ILOG API C++ model.

Concerning the genetics algorithms (GA), only Sioud et al. (2009) succeeded in proposing an efficient GA, suggesting that this metaheuristic is not well suited to deal with the specificities of this problem. Indeed, the authors have proposed a GA integrating the RMPX crossover operator which takes greater account of the relative and absolute position of a job. Indeed, Armentano & Mazzini (2000); Rubin & Ragatz (1995); Tan & Narasimhan (1997) have shown the importance of relative and absolute order positions for solving the $1|s_{ij}|\Sigma T_j$ problem. The proposed GA outdoes the performance of all the GAs found in the literature but is still less efficient than the Tabu/VNS of Gagné et al. (2005) which represents the best approach found in the literature.

The main purpose of this chapter is to show that GAs can be efficient approaches for solving the $1|s_{ij}|\Sigma T_j$ problem when the different mechanisms of the algorithm are specially design to deal with the specificities of the problem. Indeed, in their respective works, Rubin & Ragatz (1995) and Sioud et al. (2009) have shown the importance of relative and absolute order positions for the $1|s_{ij}|\Sigma T_j$ problem. Thereby, all the used crossover operators into the genetic algorithms from literature maintain the absolute position, or the relative position or both. So, to reach good results, the presented genetic algorithms must ensure the preservation of both the relative and the absolute order positions while maintaining diversification during their evolving. In this context, the presented algorithms will take this into consideration. Indeed, we present, in this chapter, two hybrid GAs for solving the $1|s_{ij}|\Sigma T_j$ where the different mechanisms of the algorithms are specially design to deal with the specificities of the problem. The first hybridization incorporates Constraint Based Scheduling (CBS) in a GA. The hybridization of the CBS approach with the GA is done at two levels. Even, the CBS is used in the reproduction and intensification processes of GA separately. The second hybridization introduces a hybrid crossover in a GA. The proposed crossover uses concepts from the multi-objective evolutionary algorithms and ant colony optimization. Both hybridizations use the specificities of the problem to reach good results.

This chapter is organized as follows: Section 2 presents the used pure GA of Sioud et al. (2009). Section 3 introduce the two hybrid algorithms. The computational testing and discussion are presented in Section 4: we present several versions of hybridizations and compare our results to the Tabu/VNS of Gagné et al. (2005). Finally, we conclude with some remarks and future research directions.

2. Genetic algorithm

Based on the GA proposed by Sioud et al. (2009), we define a simple genetic algorithm. A solution is coded as a permutation of the considered jobs. The population size is set to n to fit with the considered instance size. Sixty percent of the initial population is generated randomly, 20% using a pseudo-random heuristic which minimizes setup times, and the last 20% using a pseudo-random heuristic which minimizes the due dates. A binary tournament selects the chromosomes for the crossover. The proposed GA uses the OX crossover (Michalewicz, 1996) to generate 30% of offspring and the RMPX crossover (Sioud et al., 2009) to generate the rest of the child population. The RMPX crossover can be described in the following steps : (i) two parents *P1* and *P2* are considered and two distinct crossover points *C1* and *C2* are selected randomly, as shown in Figure 1; (ii) an insertion point p_i is then randomly chosen in the offspring *O* as p_i = *random (n - (C2 - C1))*; (iii) the part [*C1*, *C2*] of *P1*, shaded in Figure 1, is inserted in the offspring O from p_i, from the position 2 shown in Figure 1; and (iv) the rest of the offspring *O* is completed from *P2* in the order of appearance from its first position.

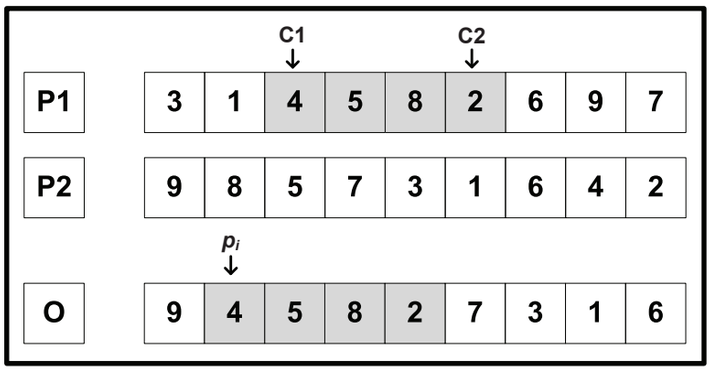


Fig. 1. Illustration of RMPX

The crossover probability p_c is set to 0.8, therefore *n*0.8* offspring are generated at each generation. A mutation is also applied with a probability p_m equal to 0.3. The mutation consists of exchanging the position of two distinct jobs which are randomly chosen. The replacement is elitist and the duplicate individuals in the population are replaced by chromosomes generated by one of the pseudo-random heuristics used in the initialization phase.

3. Hybrid genetic algorithms

Several researchers have attempted to relieve the metaheuristic shortcomings and limitations by modifying the traditional executing for some problems. Indeed, to improve the effectiveness of these methods, some researchers have used metaheuristics variations and hybridizations (Puchinger & Raidl, 2005; Talbi, 2009). In general, hybridization combines two or more methods in a single algorithm to solve combinatorial optimization problems. Hybrid approaches in general and hybrid metaheuristics in particular are gaining popularity because these approaches obtained the best results for several combinatorial optimization problems (Jourdan et al., 2009; Talbi, 2009). Also, according to Blum et al. (2005), the hybridization of metaheuristics is the most promising avenue for improving the quality of solutions in

many real applications. Puchinger & Raidl (2005) divide hybrid methods into two categories : collaborative and integrative hybridization. The algorithms that exchange information in a sequential, parallel or interlaced way fall into the category of collaborative hybridization. We talk about an integrative hybridization when a technique is an embedded component of another technique. In this chapter, we introduce first a collaborative hybridization which incorporates CBS approach with the GA at two levels. Indeed, the CBS is used in the reproduction and intensification processes of GA separately. In fact, the CBS approach is integrated in a crossover operator and in the intensification search space process using additional constraints for both of them. Second, we introduce an integrative hybridization at a new hybrid crossover, integrating concepts from two different techniques: archives as in the multi-objective evolutionary algorithms and a transition rule as in ant colony optimization.

3.1 The collaborative hybrid genetic algorithm

Constraint solving methods such as domain reduction and constraint propagation have proved to be well suited for a wide range of industrial applications (Fromherz, 1999). These methods are increasingly combined with classical solving techniques from operations research, such as linear, integer, and mixed integer programming (Talbi, 2002), to yield powerful tools for constraint-based scheduling by adopting them. The most significant advantage of using such CBS is to separate the model from the algorithms which solve the scheduling problem. This makes it possible to change the model without changing the algorithm used and vice versa.

In the recent years, the CBS has become a widely used form for modeling and solving scheduling problems using the constraint programming approach (Allahverdi et al., 2008; Baptiste et al., 2001). A scheduling problem is the process of allocating tasks to resources over time with the goal of optimizing one or more objectives (Pinedo, 2002). A scheduling problem can be efficiently encoded like a constraint satisfaction problem (CSP).

The activities, the resources and the constraints, which can be temporal or resource related, are the basis for modeling a scheduling problem in a CBS problem. Based on representations and techniques of constraint programming, various types of variables and constraints have been developed specifically for scheduling problems. Indeed, the domain variables may include intervals domains where each value represents an interval (processing or early start time for example) and variable resources for many classes of resources. Similarly, various research techniques and constraints propagation have been adapted for this kind of problem.

In Constraint Based Scheduling, the single machine problem with setup dependent times can be efficiently encoded in terms of variables and constraints in the following way. Let M be the single resource. We associate an activity A_j for each job j. For each activity A_j four variables are introduced, *start*(A_j), *end*(A_j), *proc*(A_j) and *dep*(A_j). They represent the start time, the end time, the processing time and the departure time of the activity A_j, respectively. The departure time represents the needed setup time of an activity when the latter starts the schedule.

Figure 2 presents the pseudo-code for the $1|s_{ij}|\Sigma T_j$ problem modeling with the C++ API of ILOG Scheduler 6.0. The main procedure ModelSMSDST calls the two procedures CreateMachine and CreateJob. CreateMachine procedure (lines 3 to 6) uses the class *IloUnaryResource*. This allows handling unary resources, that is to say, a resource whose

```
L1   Modeling SMSDST :
L2
L3      procedure CreateMachine (SetupMatrix)
L4         Create the setup matrix parameter
L5         Create the single machine and associate the setup matrix parameter
L6      end CreateMachine()
L7
L8      procedure CreateJob (ProcessingTimes, StartingTimes, type)
L9         Create a job with a type and processing time
L10        Set a starting time for the created job
L11     end CreateJob()
L12
L13     procedure ModelSMSDST(ProcessingTimes, StartingTimes, DueDates SetupMatrix)
L14        CreateMachine(SetupMatrix)
L15        Define an array for the jobs completion time C
L16        Define a variable for the total tardiness Tard
L17        for each i in NB_JOBS do
L18           job ← CreateJob (ProcessingTimes, StartingTimes, i)
L19           C[i] ← max(0, job.end - DueDates[i])
L20        end for
L21        Tard ← Sum(C)
L22        Minimize (Tard)
L23     end ModelSMSDST
```

Fig. 2. C++ API model for the $1|s_{ij}|\Sigma T_j$ problem

capacity is equal to one. This resource cannot therefore handle more than one job at a time. The use of the setup times in CBS and also with ILOG Scheduler (ILOG, 2003a) indicates that they are resource-related and not activity-related such as is the case in our problem. It is possible to overcome this problem by associating a type for each activity and creating setup times associated with these types. For this purpose, we use the class *IloTransitionParam* which is managing and setting setup times. The setup matrix is then associated to this class which will be related to the unary machine (line 5). Thus, when we calculate the objective function, it is possible to associate the setup times between two distinct types of activities. To model the total tardiness, we must first define a variable *Tard* (line 16). Then we define an array C containing the completion times C_i of the different activities times A_i during the research phase (line 15). When we create the activities in the model, we add a constraint that combines the activities A_i to the corresponding times C_i (line 19). After that, we add a constraint which combines the variable *Tard* with the sum of the C_i in the table C (line 21). Finally, we add a constraint that minimizes the variable *Tard* (line 22). Thus, we obtain the objective function which will be added to the model.

ILOG Solver (ILOG, 2003a) provides several predefined search algorithms named as *goals* and activity selectors. We used the *IloSetTimesForward* algorithm with the *IloSelFirstActMinEndMin* activity selector. The *IloSetTimesForward* algorithm schedules activities on a single machine forward initializing the start time of the unscheduled activities. The activity selector defines the heuristic scheduling variables representing start times, which chooses the next activity

to schedule. The *IloSelFirstActMinEndMin* tries first the activity with the smallest start time and in case of equality the activity with the smallest end time. For his part, ILOG Scheduler (ILOG, 2003b) provides four strategies to explore the search tree : the default *Depth-First Search* (DFS), the *Slice-Based Search* (SBS) (Beck & Perron, 2000), Interleaved Depth-First Search (IDFS) (Meseguer, 1997) and the *Depth-Bounded Discrepancy Search* (DDS) (Walsh, 1997) which is used in this work.

The hybridization of an exact method such as the CBS and a metaheuristic such as the GA can be carried out in several ways. Talbi (2002) presents a taxonomy dealing with the hybrid metaheuristics in general. Puchinger & Raidl (2005) and Jourdan et al. (2009) present a taxonomy for the exact methods and metaheuristics hybridizing. In this chapter we present two different approaches of hybridization. The first approach is to integrate the CBS in the GA reproduction phase and more precisely in a crossover operator, while the second approach is to use CBS as an intensification process in the GA.

When we handle a basic single machine model, there is no precedence constraint between activities as is the case in a flow-shop or job-shop where adding constraints improves the CBS approach. The main idea of integrating the CBS in a crossover is to provide to this latter precedence constraints between activities when generating offspring. In this work, we consider only the direct constraints during the crossover. Therefore, the conceived crossover promotes the relative order positions such as the PPX crossover (Bierwirth et al., 1996). The proposed crossover operator is designated *Indirect Precedence Constraint Crossover (IPCX)* and can be described in the two following steps : *(i)* all individuals in the current population are considered and the indirect precedence constraints between jobs concurrently in all the individuals are kept, as shown in Figure 3; and *(ii)* the CBS approach tries to solve the problem while adding the indirect precedence constraints built in the previous step and an upper bound consisting of the objective function value of the best parent. The upper bound is added to discard faster bad solutions when branching during the solver process. As a reminder, the ILOG Solver uses a Branch and Bound approach to solve a problem (ILOG, 2003b).

In the case of Figure 3,we consider a population with 4 individuals. Only the three indirect precedence constraints *(1 before 6)*, *(8 before 2)* and *(9 before 7)* are in the four individuals. So these three indirect constraints are added to the model and will be propagated. Thus, they preserve the relative positions of the pairs of activities *(1,6)*, *(8,2)* and *(9,7)*. After that, in a potential offspring we will find this indirect precedence constraints. Finally, if no solution is found by the *IPCX* crossover, the offspring is generated by one of the pseudo-random heuristics used in the initialization phase. The *IPCX* crossover will be done under probability p_{IPCX}.

Integrating an intensification process in a genetic algorithm has been applied successfully in several fields. The incorporation of heuristics and/or other methods, i.e. an exact method such as the CBS approach, into a genetic algorithm can be done in the initialization process to generate well-adapted initial population and/or in the reproduction process to improve the offspring quality fitness. Following this latter reasoning, the strategy proposed in this section is based on the intensification in specific space search areas. However, we can find in literature only few papers dealing with such hybridization Puchinger & Raidl (2005); Talbi (2009).

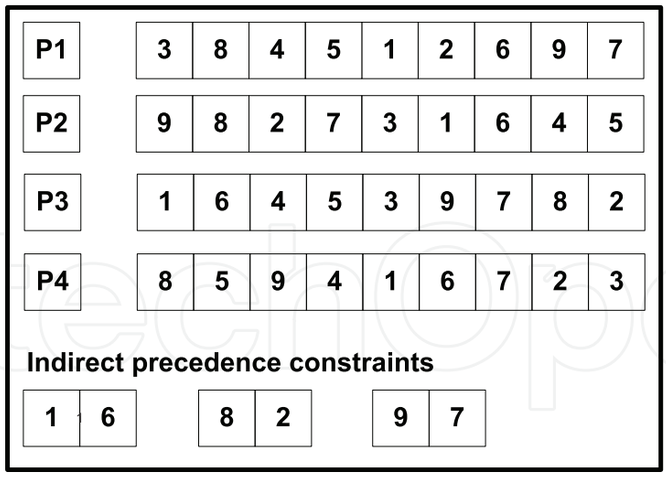


Fig. 3. Illustration of IPCX

In the same vein of the *IPCX* conservation precedence constraints, an intensification process is applied by giving a generated offspring to the CBS approach and fixing a block of α positions. Thus, the absolute order position will be preserved for these fixed positions while the relative order position will be preserved for the other activities. Indeed, the activities on the left of the fixed block will be scheduled before this late block, while the activities on the right will be scheduled after this block. The fixed block size should be neither too large nor too small : if its size is too large, the CBS approach will have no effect and if its size is too small the CBS approach will consume more time to find a better solution. Thereby, at each time this intensification is done, α continuous positions are fixed with $0.2*n \prec \alpha \prec 0.4*n$. We use to this end two different procedures based on the CBS approach. The first one, noted as IP_{TARD}, selects a generated offspring and tries to solve the problem using the CBS approach which minimizes the total tardiness described above while adding an upper bound consisting of the objective function value of this offspring. So as a result, the CBS approach may return a better solution when scheduling separately the activities on the left and the right of the fixed block activities.

Using the similarity of the studied problem with the time-dependent traveling salesman problem (Bigras et al., 2008), the second intensification procedure, noted as IP_{TSP}, works like IP_{TARD} but in this case the CBS approach minimizes the makespan. The makespan optimization aims to minimize the setup times and then, in some specific configurations, will give promising solutions under total tardiness optimization otherwise explore a different areas search space. The makespan criterion is represented by an additional variable Makespan. Its value is determined by $Makespan = \sum_{A_j=1}^{n} max(end(A_j))$. The model minimizing the makespan is similar to that in Figure 2. Indeed, we just delete the declaration of the array C at line 15 and define an activity Makespan with time processing equal to 0 at line 16. Then, a constraint stating that all jobs must be completed before the Makespan start time is added in the loop. Finally, lines 19 and 21 are removed and line 22 minimizes in this case the Makespan end time.

Thereby, an offspring is selected with a tournament under probability p_{IP} and then, one of the two intensification procedures IP_{TARD} and IP_{TSP} is chosen under probability p_{cip} to be applied on this offspring. Figure 4 illustrates the intensification process based on the CBS

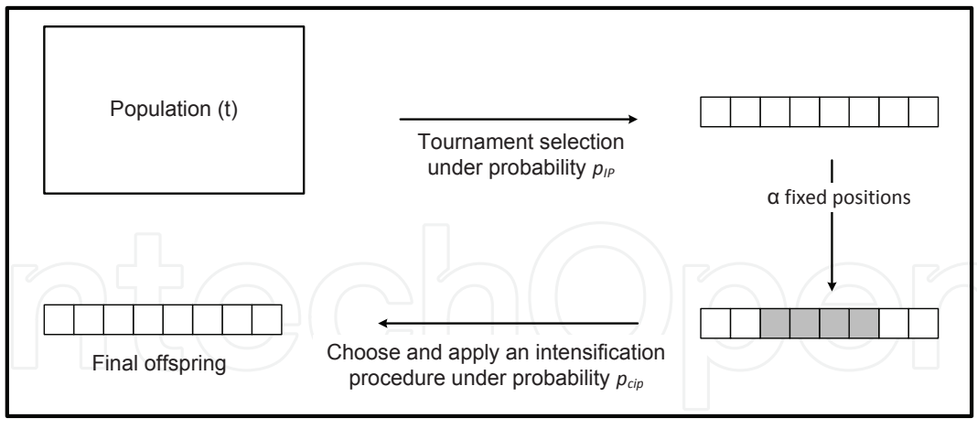


Fig. 4. The intensification process

approach. At each generation, an offspring is selected under probability p_{IP} with tournament selection. After fixing α positions and choosing an intensification procedure, IP_{TARD} or IP_{TSP} under probability p_{cip} , the solver tries to find a solution. If no solution is found the offspring is unchanged.

3.2 The integrative hybrid genetic algorithm

In this section, we introduce the integrative hybrid genetic algorithm which integrates concept from two different techniques : archives as in the multi-objective evolutionary algorithms and a transition rule as in ant colony. This hybridization is done in a crossover noted as *ICX* which evolves in two step : (i) from the first parent, we place the cross section, which represents the section between the two crossover points as the RMPX crossover processing; (ii) use a transition rule to fill the remaining jobs using two lists formed from the second parent.

The transition rule is used in the *ICX* crossover operator as a mechanism taking into account the problem's properties and memory information. After defining the cross section (jobs set from the first parent), the filling section (jobs set from the second parent) is completed using a transition rule adapted to the $1|s_{ij}|\Sigma T_j$ problem, similar to that used by the ant colony optimization (ACO) (Dorigo & Gambardella, 1997) and inspired by the work of Gagné et al. (2002).

From an identified cross section (section from the first parent), it is possible to insert the jobs to the right of this section from the latest job as a classical ant or inversely to the left. First, the number of jobs to be inserted on the right and left of the cross section are determined. Then, two lists from the second parent are built: a job list which will be inserted on the left of the cross section and a jobs list which will be inserted on the right. From the beginning of the second parent, the left list is formed by the jobs not yet placed according to the jobs number to be placed on the left, and the rest of the jobs not yet placed form the right list. In Figure 5 we consider the two parents *P1*, *P2* and the offspring *O*. Both three positions remain unfilled on the left and on the right of the cross section. Looking through the parent *P2*, the left list is then formed by jobs 9, 5 and 7 while the right one is formed by jobs 3, 1 and 4.

Firstly, we consider the job insertions on the right of the cross section. The second case, very similar to the first, requires only few changes and is subsequently treated further. From the

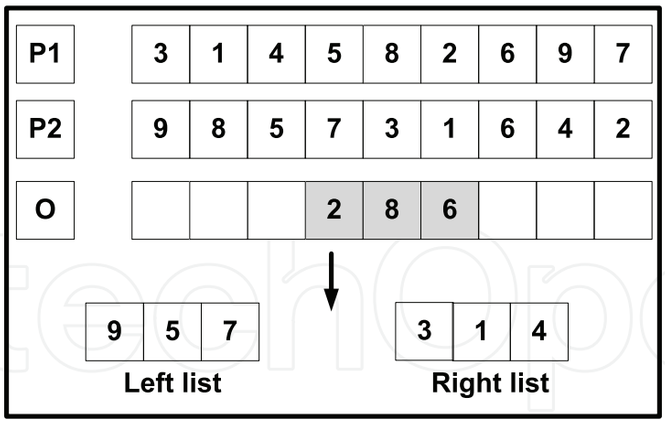


Fig. 5. List construction for the transition rule step

$$j = \begin{cases} arg \ \ max \left\{ \left[SUCC_{ij}(A_t) \right]^{\alpha} * \left[\frac{1}{\overline{s_{ij}}} \right]^{\beta} * \left[\frac{1}{\overline{U_{ij}}} \right]^{\phi} \right\} & if \ q \leq q_0 \\ J & if \ q > q_0 \end{cases} \quad (1)$$

where J is chosen according to the probability p_{ij}

$$p_{ij} = \frac{\left[SUCC_{ij}(A_t) \right]^{\alpha} * \left[\frac{1}{\overline{s_{ij}}} \right]^{\beta} * \left[\frac{1}{\overline{U_{ij}}} \right]^{\phi}}{\sum \left[SUCC_{ij}(A_t) \right]^{\alpha} * \left[\frac{1}{\overline{s_{ij}}} \right]^{\beta} * \left[\frac{1}{\overline{U_{ij}}} \right]^{\phi}} \quad (2)$$

last inserted job, the choice of the next job is made using the pseudo-random-proportional transition rule expressed in Equations (1) and (2). As in an ACO, in Equation (1), q is a random number and q_0 is a parameter; both are between 0 and 1. The parameter q_0 determines the relative importance of the existing information exploitation and the new solutions search space exploration. Indeed, Equation (2) states that the next job will be chosen by a greedy rule when $q \leq q_0$ or by the probabilistic rule of Equation (2) when $q > q_0$. Equation (2) describes the biased exploration rule p_{ij} adapted to the $1|s_{ij}|\Sigma T_j$ problem when inserting job j after job i.

In Equations (1) and (2), the element $\overline{s}_{ij}$=s_{ij}/MAX s_{ij} are the relative setup times and represents the visibility as in the ACO of Gagné et al. (2002).

We describe in the following, the other two elements of Equations (1) and (2) where two new concepts are introduced : $SUCC_{ij}(A_t)$ which represents the pheromone trail as in an ACO and $\overline{U}_{ij}$ which represents an heuristic for look-ahead information.

In an classical ACO, the pheromone trail contains information based on solution quality. Indeed, in the pheromone matrix, if the intensity of the pheromone value between two jobs i and j increases, then the probability to insert j after i increases. In our case, we construct a matrix $SUCC$ from an archive that stores the best solutions throughout the evolution process

as in some cases in multi-objective evolutionary algorithms using the Pareto-optimal concept (Zitzler & Thiele, 1999). This archive is updated at every new offspring creation. The archive size, denoted as N, is equal to the problem size and contains the N best individuals found during the genetic algorithm search.

If n is the number of jobs processed and the archive size, the matrix $SUCC$ is calculated as follows:

$$\begin{array}{c} For\ every\ jobs\ pair\ (i,j)\ where\ i \in [1,n]\ and\ j \in [1,n] \\ SUCC[i][j] = \dfrac{number\ \ of\ \ times\ \ that\ \ j\ \ immediatly\ \ succeeds\ \ i}{n} \end{array} \quad (3)$$

Thereby built from the archive individuals, the $SUCC$ matrix will contain the trail information. Thus, the more job j succeeds job i in the archive individuals, the more important the trail is. This information, then favors the succession of job j after job i in the transition rule. This matrix is calculated as needed from the archive and is updated at each archive update. Consequently, in Equations (1), $SUCC_{ij}(A_t)$ represents the trail quantity that job j immediately succeeds job i in the archive at time t.

In Gagné et al. (2002), the authors have considered a lower bound determined by the tardiness sum of the sequenced jobs and an estimate for the not yet sequenced jobs, as proposed by Ragatz (1993). This lower bound is used as a look ahead function to anticipate the choice of an ant and it is incorporated in the transition rule of their ACO. In this present integrative hybridization, we propose to use an heuristic that also anticipates the choices in the transition rule. However, this heuristic is based on an upper bound of the total tardiness. Indeed, considering a defined cross section in an empty job sequence, we use this heuristic noted as U_{ij} for successively placing the jobs on the right of the cross section first until the end of the sequence. Placing the jobs on the right is very similar and needs only few changes.

Starting from a partial sequence where only the cross section is defined, the heuristic uses the maximum values of processing time $p_{(max)}$ and setup times $s_{ij\ (max)}$ and the minimum due dates $d_{(min)}$ in its calculation to complete the empty positions. Thus, we consider a job sequence Q where a cross section is defined and $Q = [q_0, q_1, ..., q_{n-1}, q_n]$ where q_j is the subscript of the j^{th} job in the sequence. Thereby, the completion time of the j^{th} job in sequence will be expressed as $C_{q_j\ (max)} = \sum_{k=1}^{j}(s_{q_{k-1}q_k\ (max)} + p_{q_k\ (max)})$ while the tardiness of the j^{th} job in sequence will be expressed as $T_{q_j\ (max)} = max(0, C_{q_j\ (max)} - d_{q_j\ (min)})$. In these last two equations, the exact values of processing time, due dates and setup times are used instead the maximum or minimum values for the cross section jobs which are already placed. So, if we want to place a job j immediately after the cross section last job i, then U_{ij} will be defined as $\sum_{j=1}^{n} T_{q_j\ (max)}$.

To better understand how this heuristic works, consider the 9-job example in Figure 6. The first table in this figure shows the respective processing times p_j and due dates d_j.

The second table in Figure 6 presents a partial sequence with only the cross section composed of jobs 2, 5 and 8. The processing time, the related setup times (s_{2-5} *and* s_{5-8}) and due dates of jobs 2, 5 and 8 are used to calculate $\sum_{j=1}^{n} T_{q_j\ (max)}$. We suppose that the right list contains jobs 1, 3, 6 and 7. So, the left list contains the two remaining jobs 9 and 4. Considering the right section filling, we use the maximum values of processing times $p_{\ (max)}$ (the maximum

Job j	1	2	3	4	5	6	7	8	9
p_j	102	100	97	99	96	102	97	107	100
d_j	640	596	602	585	625	635	616	645	608

Q	-	-	*2*	*5*	*8*	-	-	-	-	$\sum_{j=1}^{n} T_{q\,j\,(\max)}$
C_j	120	240	360	456	568	690	812	934	1056	
d_j	585	585	596	625	645	602	602	602	602	
$\overline{T}_{q_j}$	0	0	0	0	0	88	210	332	*454*	**1084**

Q	-	-	2	5	8	1	-	-	-	$\sum_{j=1}^{n} T_{q\,j\,(\max)} = U_{ij}$
C_j	120	240	360	456	568	680	802	924	1033	
d_j	585	585	596	625	645	640	602	602	602	
$\overline{T}_{q_j}$	0	0	0	0	0	40	200	322	431	**993**

Q			2	5	8	6				$\sum_{j=1}^{n} T_{q\,j\,(\max)} = U_{ij}$
C_j	120	240	360	456	568	686	803	920	1037	
d_j	585	585	596	625	645	635	602	602	602	
$\overline{T}_{q_j}$	0	0	0	0	0	51	201	318	435	**1005**

Q			2	5	8	7				$\sum_{j=1}^{n} T_{q\,j\,(\max)} = U_{ij}$
C_j	120	240	360	456	568	666	788	910	1032	
d_j	585	585	596	625	645	616	602	602	602	
$\overline{T}_{q_j}$	0	0	0	0	0	50	186	308	430	**974**

Q			2	5	8	3				$\sum_{j=1}^{n} T_{q\,j\,(\max)} = U_{ij}$
C_j	120	240	360	456	568	670	792	914	1036	
d_j	585	585	596	625	645	602	616	616	616	
$\overline{T}_{q_j}$	0	0	0	0	0	68	176	298	420	**962**

Fig. 6. H_{ij} example

processing time of the right list jobs, here 102) and setup times $s_{ij\,(max)}$ and the minimum due dates $d_{(min)}$ (the minimum due date of the right list jobs, here 602) of the jobs in the right list jobs. We suppose also that the maximum setup times is equal to 20. For the left section filling the maximum values of processing times $p_{\,(max)}$ equal 100 (the maximum processing time of the left list jobs) the maximum setup times $s_{ij\,(max)}$ also equal 20 and the minimum due dates $d_{(min)}$ equal 585 (the minimum due date of the left list jobs). We obtain an upper bound $\sum_{j=1}^{n} T_{q_j\,(max)}$ which equal 1084.

Then for each remaining job in the right list, the heuristic U_{ij} is calculated. So, the respective processing time, due date and setup times following the job 8 (s_{8-*}) are updated and the total tardiness is calculated. For example, if we suppose that job 1 is directly inserted after the cross section and that s_{8-1} is equal to 10, then we obtain the third table in Figure 6. In this partial sequence, we update the setup times s_{8-1} (10 instead of 20) and the due date (640 instead 602). By inserting job 1, the heuristic U_{ij} value equal 993. If we suppose now that job 3 is directly inserted after the cross section and that s_{8-3} is equal to 5, then we obtain the last table in Figure 6. In this partial sequence we update the setup times s_{8-3} (5 instead of 20) and the minimum due date $d_{(min)}$ for the remaining jobs (616 instead 602). For this case, the heuristic value U_{ij} equal 962. The fourth and fifth tables in Figure 6 represent the insertion of the jobs 6

and 7 directly after the cross section, respectively. The heuristic value U_{ij} equal 1005 and 974, respectively.

The normalized values $\overline{U}_{ij}$=U_{ij}/MAX U_{ij} are then used in Equations (1) and (2) to determine which job will be placed. Thus, in the previous example, we obtain $\overline{U}_{8-1}$ = 0.98, $\overline{U}_{8-3}$ = 0.95, $\overline{U}_{8-6}$ = 1 and $\overline{U}_{8-7}$ = 0.97. It is obvious that the higher the normalized value, the lower the probability of placing job increases.

Since the cross section is already placed, placing the remaining jobs on the left of this section can be done either from the first offspring position from left to right as a classical ant or inversely from the first cross section position. In both cases, we will have a resulting setup time either at the junction with the cross section if we proceed from left to right, or with the latest job of the previous period (the initial setup time) if we proceed from right to left. During the application of the hybrid crossover *ICX*, we use equiprobable one of the two methods of left insertion.

In the case of placing jobs from right to left, we make some changes in Equations (1) and (2). Indeed, $\overline{s}_{ij}$ and $\overline{U}_{ij}$ are replaced by $\overline{s}_{ji}$ and $\overline{U}_{ji}$, respectively. Also, the matrix $SUCC_{ij}(A_t)$ is replaced by $PRED_{ij}(A_t) = {}^{T}SUCC_{ij}(A_t)$, the transposed matrix of $SUCC_{ij}(A_t)$ from the archive A. In fact, the trace must be built from relevant information related to the predecessors in this case.

So, with these elements, this transition rule uses past, present and future information from the archive, the visibility and the look ahead, respectively. The transition rule is used in this way for all job insertions until the end of the sequence. Finally, the parameters α, β *and* ϕ associated with each transition rule matrix in Equations (1) and (2), can privilege certain elements depending on the characteristics of the problem.

4. Computational results and discussion

The benchmark problem set consists of eight instances, each with a number of jobs of 15, 25, 35 and 45 jobs, and it is taken from the work Ragatz (1993). These instances are available on the Internet at https://www.msu.edu/~rubin/files/c&ordata.zip. The job processing times are normally distributed with a mean of 100 time units and the setup times are also uniformly distributed with a mean of 9.5 time units. Each instance has three factors which have both high and low levels. These factors are due date range, processing time variance and tardiness factor. The tardiness factor determines the expected proportion of jobs that will be tardy in a random sequence. The second instance subset, taken from the work of Gagné et al. (2002), consists of eight instances each with 55, 65, 75 and 85 jobs. These instances which are called "large instances set" are available at http://wwwdim.uqac.ca/∼c3gagne/DocumentRech/ProblemDat
aSet55to85.zip. These instances are also generated similarly as in the smaller instances. All the experiments were run on an Itanium with a 1.4 GHz processor and 4 GB RAM. Each instance was executed 10 times and all the algorithms are coded in C++ language and under the ILOG IBM CP constraint environment using ILOG Solver and Scheduler via the C++ API (ILOG, 2003a;b) for the CBS approach. In order to obtain a reliable comparison, the stop criterion for all the proposed algorithms is 50 000 evaluations. This criterion is used by the Tabu/VNS of Gagné et al. (2005) which represents the best approach found in the literature. First, we

PRB	OPT	GA	CBS	Collaborative hybridization			Ingrative hybridization			TVNS
				CGA^{IPCX}	CGA^{IP}	CGA^{COL}	IGA^{L-R}	IGA^{R-L}	IGA^{OrOpt}	
401	**90**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
402	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
403	**3418**	0.0	0.0	0.0	0.4	0.0	0.0	0.0	0.0	0.0
404	**1067**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
405	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
406	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
407	**1861**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
408	**5660**	0.0	0.9	0.0	0.1	0.0	0.0	0.0	0.0	0.0
501	**261**	0.0	0.4	0.0	0.5	0.0	0.0	0.0	0.0	0.0
502	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
503	**3497**	0.0	2.5	0.0	0.3	0.0	0.0	0.0	0.0	0.0
504	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
505	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
506	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
507	**7225**	0.0	1.8	0.0	0.7	0.0	0.0	0.0	0.0	0.0
508	**1915**	0.0	35.8	0.0	1.8	0.0	0.0	0.2	0.0	0.0
601	**12**	16.9	41.7	5.7	7.5	2.4	1.0	1.7	0.0	0.0
602	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
603	**17587**	0.2	6.5	0.8	1.1	0.2	0.0	0.0	0.0	0.0
604	**19092**	0.2	21.1	0.9	1.3	0.5	0.0	0.0	0.0	0.0
605	**228**	1.3	122.4	2.6	3.5	0.3	1.0	0.4	0.0	0.0
606	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
607	**12969**	0.2	17.7	0.6	1.9	0.2	0.0	0.0	0.0	0.0
608	**4732**	0.2	156.6	0.5	1.2	0.0	0.0	0.0	0.0	0.0
701	**97**	3.0	20.6	5.3	8.3	2.1	1.2	1.0	0.6	0.3
702	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
703	**26506**	0.2	2.8	1.2	1.8	0.7	0.0	0.0	0.0	0.0
704	**15206**	0.3	94.8	1.3	2.1	0.5	0.2	0.2	0.0	0.0
705	**200**	3.4	72.5	3.2	6.5	1.1	2.3	1.0	0.4	0.2
706	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
707	**23789**	0.2	20.4	1.0	1.9	0.3	0.0	0.0	0.0	0.0
708	**22807**	0.3	50.0	1.4	2.1	1.0	0.0	0.0	0.0	0.1

Table 1. Comparison of different approaches for the small problem set

discuss the collaborative and integrative hybridization on the small instances, then only the integrative genetic algorithm on the large instances.

Table 1 compares the results of different approaches and the best results are shaded. In this table, PRB denotes the instance names and OPT the optimal solution found by the B&B of Bigras et al. (2008). These authors have not given information about the execution time of their approach. They only said that some instances have been resolved after more than seven days. The GA column shows the results average deviation to the optimal solution of the genetic algorithm described in the section 3.1 which gives the best results among all genetic algorithms in the literature without an intensification process (Sioud et al., 2009). The GA average CPU time is equal to 13.4 seconds for the 32 instances. The GA generally obtained fairly good results only for the instances 601, 605, 701 and 705. These instances are low due date range and large tardiness factor. Thus, for this kind of instances, "good" solutions may not generate "good" offspring. Furthermore, considering that the tardy jobs are scheduled at the end of the sequence, it may be sufficient to schedule the other jobs by minimizing

the setup times. It is the aim of introducing the IP_{TSP} intensification procedures. The CBS column shows the deviations of the CBS approach minimizing the total tardiness defined in Section 3.1. For this approach, the execution time is limited to 60 minutes. It can be noticed that the CBS approach results deteriorate with increasing the instances size and especially for the **4, **5 and **8 instances. The CGA_{IPCX} column shows the average deviation of the genetic algorithm in which the crossover operator *IPCX* is integrated. The probability p_{IPCX} is equal to 0.2 and the CBS approach execution time is limited to 15 seconds. The CGA_{IPCX} average time execution is equal to 12.8 minutes for the 32 instances. The first observation is that the CGA_{IPCX} algorithm is always optimal for 15 and 25 jobs instances. It should be noted that the integration of the IPCX crossover improves all of the GA results and especially for the instances **1 and **5 where the deviation became less than 6%. For example, the deviation was reduced from 16.9% to 6.7% for the 601 instance. Using the direct precedence constraints allows the PCX crossover to enhance both the GA exploration and the CBS search; and consequently reaching better schedules.

The CGA_{IP} column shows the average deviation of the genetic algorithm in which we include the IP_{Tard} and IP_{TSP} intensification procedures under probability p_{IP} equal to 0.1. The CBS approach execution time is limited to 20 seconds for the IP_{Tard} and IP_{TSP}. The CGA_{IP} average time execution is equal to 13.5 minutes for the 32 instances. The CGA_{IP} improves most GA results and specially the **1 and **5 instances but gives worse results than the CGA_{IPCX} and this was expected because in 50% of the cases the intensification procedure minimizes the makespan and not the total tardiness. The CGA_{COL} column shows the average deviation of the GA_{PCX} algorithm where we include the IP_{Tard} and IP_{TSP} intensification procedures. The probabilities p_{IP} and p_{cip} are equal to 0.1 and 0.5 respectively like the CGA_{IP}. The CBS approach execution time is also limited to 20 seconds for the IP_{Tard} and IP_{TSP} in the CGA_{COL}. The CGA_{COL} average time execution is equal to 20.5 minutes for the 32 instances. This hybrid algorithm improves all the results found by the CGA_{IPCX}. These improvements are more pronounced with the integration of local search procedures. The introduction of the two intensification procedures improves essentially the **1 and the **5 instances. Also, the optimal schedule is always reached by CGA_{COL} for the 608 instance. The CGA_{COL} found the optimal solution for all the instances at least one time and this was not the case either for CGA_{IPCX} or CGA_{IP}.

The convergence of both GA and the CGA_{IPCX} algorithms are similar. Indeed, the average convergence generation is equal to 1837 and 1845 generations for GA and CGA_{IPCX}, respectively. Concerning the CGA_{IP} algorithm, the average convergence generation is equal to 1325 generations. So, we can conclude that the two intensification procedures based on the CBS approach are permitting a faster genetic algorithm convergence than the *IPCX* crossover but achieving worse results. The CGA_{COL} average convergence generation is equal to 825 and compared to the CGA_{IPCX}, the introduction of the intensification procedures speeds up the convergence of the solution with reaching better results.

Exact methods are well known to be time expensive. The same applies to their hybridization of them with metaheuristics. Indeed, times execution increases significantly with such hybridization policies due to some technicality during the exchange of information between the two methods (Jourdan et al., 2009; Puchinger & Raidl, 2005; Talbi, 2002; 2009) and this is what has been observed here. However, in this chapter, the solution quality is our main concern. So, we concentrated our efforts on it. Then, because the high consuming time and

memory, the collaborative algorithm will not be applied on the large problem set. Finally, we are also aware of the fact that we can't compare the collaborative hybridization with the other approaches because the CBS approach executes more than the 50 000 stop criterion evaluations.

The two row noted as IGA^{L-R} and IGA^{R-L} present the results of the genetic algorithm where the hybrid crossover *ICX* is integrated and the filling section placement is executed by the transition rule, respectively, on the left then on the right (IGA^{L-R}) and on the right then on the left (IGA^{R-L}). The purpose of this comparison is to show the impact of the look ahead element $\overline{U}_{ij}$ in the transition rule.

Indeed, if the results of the two algorithms outperform those of IGA^{1-2}, those of IGA^{R-L} are better than those of IGA^{L-R}, and specially for instances of type **1 and **5. This can be explained by two aspects: (i) in both cases, the look-ahead element $\overline{U}_{ij}$ improves the search for jobs to be placed, by calculating the impact of placing a job in a sequence where some jobs are already placed; and (ii) starting to place jobs on the right allows the transition rule to be more directive concerning the jobs in the beginning of the sequence, specially for instances of **1 and **5 where tardy jobs are usually at the end of the sequence. Similarly, the trace elements, $SUCC_{ij}(A_t)$ and $PRED_{ij}(A_t)$ built from the archive, play an important role to guide the transition rule in order to maintain and preserve the relative order according to an already placed job. Finally, IGA^{R-L} finds optimal solutions at least once except for the instance 704, which is not the case for IGA^{L-R}.

The row noted as IGA^{OrOpt} presents the results of the genetic algorithm IGA^{R-L} where a local search is applied at each offspring creation under probability equal to 0.1. The used local search heuristic in this case is the or-opt (Or, 1976) adapted to the total tardiness. This heuristic is also used by the Tabu/VNS of Gagné et al. (2005) whose results are summarized in the last row in Table 1 and noted as TVNS. In this hybrid algorithm, at each call to the heuristic, we generate a single neighborhood of size 40. The integration of the local search allows the hybrid genetic algorithm to have similar results to those of the Tabu/VNS and improve some average results for instances 604, 607, 701, 703, 704, 705 and 708. The Tabu/VNS achieved better performance only for instances 601 (0.0 against 0.0) and 605 (0.0 against 1.0). Concerning the integrative hybridization execution times, IGA^{L-R}, IGA^{R-L} and IGA^{OrOpt} have an average of 1.6, 1.6 and 2.1 minutes respectively on the small instances group.

Table 2 summarizes the comparison of different algorithms for the large instance set of Gagné et al. (2005). The subrow noted as (B) and (M) present the best and the median deviation of the presented algorithms, respectively. The best results of the B row are shaded in dark gray and the best results of the M row are shaded in gray. Overall, we observe a similar algorithm behavior as in the first group of instances. Indeed, IGA^{R-L}, which gives better results than HGA^{L-R}. Indeed, placing jobs at the end of the sequence before those at the beginning allows the hybrid crossover *ICX* better guiding for job placement using the look ahead element $\overline{U}_{ij}$ and the normalized setup times $\overline{s}_{ij}$ in the transition rule.

It should be noted that IGA^{R-L} lowers the minimum known bound for instances 557 and 858. This can be explained by the nature of these instances and by the fact that the transition rule uses the characteristics of the problems, including due dates and setup times, when calculating the look ahead element $\overline{U}_{ij}$.

PRB	OPT	GA		$IHGA^{L-R}$		$IHGA^{R-L}$		$IHGA^{OrOpt}$		TVNS	
		B	M	B	M	B	M	B	M	B	M
551	**183**	3.6	5.7	0.3	1.2	0.0	0.7	0.0	0.6	0.1	0.6
552	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
553	**40540**	0.1	0.2	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1
554	**14653**	0.3	0.5	0.1	0.3	0.1	0.1	0.0*	0.0	0.0	0.2
555	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
556	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
557	**35813**	0.2	0.3	0.0	0.1	0.0*	0.0	0.0*	0.0	0.0	0.0
558	**19871**	0.3	0.4	0.0	0.2	0.0	0.0	0.0	0.0	0.0	0.1
651	**268**	1.6	4.3	0.0	1.0	0.0	0.9	0.0	0.3	0.0	0.2
652	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
653	**57569**	0.2	0.3	0.0	0.2	0.0	0.1	0.0	0.1	0.0	0.1
654	**34301**	0.4	0.6	0.1	0.3	0.1	0.1	0.1	0.1	0.1	0.1
655	**2**	120.0	185.3	45.0	77.8	17.0	52.0	15.0	25.0	0.0	12.5
656	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
657	**54895**	0.2	0.3	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.1
658	**27114**	0.4	0.5	0.0	0.3	0.1	0.1	0.0*	0.0	0.1	0.1
751	**241**	3.2	4.8	0.5	2.0	0.8	1.7	0.2	0.8	0.0	0.3
752	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
753	**77663**	0.3	0.4	0.1	0.2	0.1	0.1	0.1	0.1	0.1	0.1
754	**35200**	0.3	0.7	0.2	0.4	0.1	0.3	0.0*	0.0	0.1	0.3
755	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
756	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
757	**59735**	0.2	0.3	0.1	0.2	0.0	0.1	0.0	0.0	0.0	0.0
758	**38339**	0.3	0.5	0.1	0.3	0.1	0.2	0.0*	0.0	0.1	0.2
851	**384**	2.8	5.4	1.3	1.7	0.9	1.6	0.2	0.4	0.0	0.2
852	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
853	**97642**	0.3	0.4	0.1	0.2	0.3	0.1	0.0*	0.0	0.0	0.0
854	**79278**	0.4	0.5	0.2	0.3	0.1	0.2	0.0*	0.1	0.2	0.1
855	**283**	6.0	7.5	0.5	2.3	1.1	2.0	0.3	1.5	0.0	1.3
856	**0**	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
857	**87244**	0.3	0.4	0.1	0.2	0.0	0.1	0.0	0.0	0.0	0.1
858	**74785**	0.3	0.5	0.1	0.2	0.0*	0.1	0.0*	0.0	0.1	0.2

* New lower bound

Table 2. Comparison of different approaches for the large problem set

Nevertheless, for all the introduced algorithms, there are still significant differences for instances **1 and **5, and especially for instance 655 where it exceeds 75%. In these cases, this is due to the low value of the objective function.

The local search integration in IGA^{OrOpt} allows this algorithm to find six other new minimum values for instances 554, 557, 658, 754, 758, 853 and 854. This intensification process improves the genetic algorithm exploitation phase. Also, except for some deviations in instances **1 and **5, IGA^{OrOpt} improves several averages of TABU/ VNS and specially for instances 654, 657, 658, 754, 758, 854 , 857 and 858. Except for the 655 instance, where the deviation is 25% for the average result, TABU/ VNS surpasses IGA^{OrOpt} only in 7 instances (551, 651, 751, 753, 757, 851 and 855) and this with minor deviations. Of these 7 instances, 5 of them

are **1 and **5 instances. Finally, in addition to the 8 new minimum values found, IGA^{R-L} and IGA^{OrOpt} also found the best known value for instance 551 while TABU/VNS did not find it. Concerning instances 653, 654 and 753, the best solutions are found by the GRASP of Gupta & Smith (2006).

Concerning the execution times, IGA^{L-R}, IGA^{R-L} and IGA^{OrOpt} have an average of 3.1, 3.1 and 3.9 minutes respectively. Furthermore, these execution times are increased by the transition rule integration in IGA^{L-R} and IGA^{R-L}, and the archive management. Finally, the or-opt local search heuristic increases the execution time by 20% for both the small and the large instance group.

5. Conclusion

In this chapter, we have introduced two hybrid GA to solve the sequence-dependent setup times single machine problem with the objective of minimizing the total tardiness. Indeed, using classical operator, most found GA in literature are not well suited to deal with the specificities of this problem. The proposed approaches in this chapter are essentially based on adapting highly specialized genetic operators to the specificities of the studied problem. The numerical experiments allowed us to demonstrate the efficiency of the proposed approaches for this problem. A natural conclusion of these experimental results is that GA may be robust and efficient alternative to solve this problem.

We describe first a collaborative hybridization where both a crossover operator and intensification process based on Constraint Based Scheduling are integrated into a GA. Indeed, the *IPCX* crossover operator uses the indirect precedence constraints to improve the CBS search and consequently the schedules quality. The precedence constraints are built from all the individual population in the reproduction process. The intensification procedures are based on two different CBS approaches after fixing a jobs block : the first minimizes the total tardiness which represents the considered problem objective function while the second minimizes the makespan which also enhances the exploration process and is well adapted to some instances.

Then, we introduce a hybrid crossover in an integrative hybridization which uses concepts from multi-objective algorithms and ant colony optimization to enhance the relative and absolute job position conservation during the evolving phase. The integrative hybridization introduce the *ICX* crossover which evolves in two steps. Indeed, from the first parent we place firstly the cross section. Then, from two lists formed with the remaining jobs, we use a pseudo-random transition rule to place these jobs. This transition rule uses past, present and future information from the archive, the visibility and the look ahead, respectively. The different proposed adaptations have contributed to the performance of this approach. The use of the archive and the look-ahead information have been shown to improve solution quality also enhancing the relative and the absolute order.

The proposed hybrid GA in this chapter represent very interesting alternatives to find good solutions. In fact, The found results highlight the importance of incorporating specific problem knowledge and specificities into genetic operators, even if classical genetic operators could be used. The two hybridizations have proved effectiveness on sets of benchmark

problems taken from literature. Specially, the integrative one which even outdoes the performance of the best approach found in the literature.

For future work, we will work on improving the precedence constraints under the collaborative hybridization. Indeed, it is possible to consider constraints related to a jobs set or to intervals time. Also, it would be possible to employ a chromosome representation based on the start times of activities. Hence, it will be possible to get more accurate combination of start times. Concerning the integrative hybridization, we use it for other scheduling problems in particular and other optimization problems in general, specially real-world problems.

6. References

Allahverdi, A., Ng, C., Cheng, T. & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs, *European Journal of Operational Research* 187(3): 985 – 1032.

Armentano, V. & Mazzini, R. (2000). A genetic algorithm for scheduling on a single machine with setup times and due dates, *Production Planning and Controly* 11(7): 713 – 720.

Baptiste, P., LePape, C. & Nuijten, W. (2001). *Constraint-Based Scheduling : Applying Constraint Programming to Scheduling Problems*, Kluwer Academic Publishers.

Beck, J. C. & Perron, L. (2000). Discrepancy bounded depth first search, *CP-AI-OR'2000: Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp. 7–17.

Bierwirth, C., Mattfeld, D. C. & Kopfer, H. (1996). On permutation representations for scheduling problems, *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, Springer-Verlag, London, UK, pp. 310–318.

Bigras, L., Gamache, M. & Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times, *Discrete Optimization* 5(4): 663–762.

Blum, C., Roli, A. & Alba, E. (2005). *An Introduction to Metaheuristic Techniques*, Wiley Series on Parallel and Distributed Computing, Wiley.

Conner, G. (2009). 10 questions, *Manufacturing Engineering Magazine* pp. 93–99.

Dorigo, M. & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem, *IEEE Transactions on Evolutionary Computation* .

Du, J. & Leung, J. Y. T. (1990). Minimizing total tardiness on one machine is np-hard, *Mathematics and Operations Researchs* 15: 438–495.

Dudek, R., Smith, M. & Panwalkar, S. (1974). Use of a case study in sequencing/scheduling research, *Omega* 2(2): 253–261.

Franca, P. M., Mendes, A. & Moscato, P. (2001). A memetic algorithm for the total tardiness single machine scheduling problem, *European Journal of Operational Research* 132: 224–242.

Fromherz, M. P. (1999). Model-based configuration of machine control software, *Technical report*, In Configuration Papers from the AAAI Workshop.

Gagné, C., Gravel, M. & Price, W. L. (2005). Using metaheuristic compromise programming for the solution of multiple objective scheduling problems, *The Journal of the Operational Research Society* 56: 687–698.

Gagné, C., Price, W. & Gravel, M. (2002). Comparing an aco algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times, *Journal of the Operational Research Society* 53: 895–906.

Graham, R. L., Lawler, E. L., Lenstra, J. K. & Kan, A. G. H. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics* 5: 287–326.

Gupta, S. R. & Smith, J. S. (2006). Algorithms for single machine total tardiness scheduling with sequence dependent setups, *European Journal of Operational Research* 175(2): 722–739.

ILOG (2003a). *ILOG Scheduler 6.0. User Manual*, ILOG.

ILOG (2003b). *ILOG Solver 6.0. User Manual*, ILOG.

Jourdan, L., Basseur, M. & Talbi, E.-G. (2009). Hybridizing exact methods and metaheuristics: A taxonomy, *European Journal of Operational Research* 199(3): 620–629.
URL: *http://ideas.repec.org/a/eee/ejores/v199y2009i3p620-629.html*

Lee, Y., Bhaskaram, K. & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups, *IIE Transactions* 29: 45–52.

Liao, C. & Juan, H. (2007). An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups, *Computers and Operations Research* 34: 1899–1909.

Meseguer, P. (1997). Interleaved depth-first search, *IJCAI'97: Proceedings of the Fifteenth international joint conference on Artifical intelligence*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1382–1387.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs (3rd ed.)*, Springer-Verlag, London, UK.

Or, I. (1976). *Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking*, PhD thesis, Northwestern University, Illinois.

Pinedo, M. (2002). *Scheduling Theory, Algorithm and Systems*, Prentice-Hall.

Puchinger, J. & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification, *Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Las Palmas, Spain, LNCS*.

Ragatz, G. L. (1993). A branch-and-bound method for minimumtardiness sequencing on a single processor with sequence dependent setup times, *Proceedings twenty-fourth annual meeting of the Decision Sciences Institute*, pp. 1375–1377.

Rubin, P. & Ragatz, G. (1995). Scheduling in a sequence-dependent setup environment with genetic search, *Computers and Operations Research* 22: 85–99.

Sioud, A., Gravel, M. & Gagné, C. (2010). A modeling for the total tardiness smsdst problem using constraint programming., *Proceedings of the 2010 International Conference on Artificial Intelligence, ICAI 2010, July 12-15, 2010, Las Vegas Nevada, USA, 2 Volumes*, CSREA Press, pp. 588–594.

Sioud, A., Gravel, M. & Gagné, C. (2009). New crossover operator for the single machine scheduling problem with sequence-dependent setup times, *GEM'09: The 2009 International Conference on Genetic and Evolutionary Methods*.

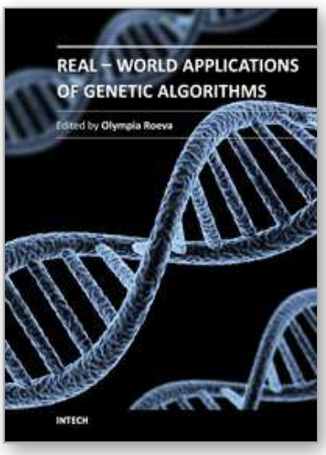
Talbi, E. (2002). A taxonomy of hybrid metaheuristics, *Journal of Heuristics* 8: 541–564.

Talbi, E.-G. (2009). *Metaheuristics : from design to implementation*, John Wiley & Sons.

Tan, K. & Narasimhan, R. (1997). Minimizing tardiness on a single processor with setup-dependent setup times: a simulated annealing approach, *Omega* 25: 619 – 634.

Walsh, T. (1997). Depth-bounded discrepancy search, *IJCAI'97: Proceedings of the Fifteenth international joint conference on Artifical intelligence*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 1388–1393.

Zitzler, E. & Thiele, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach, *IEEE Transactions on Evolutionary Computation* 3(4): 257–271.



Real-World Applications of Genetic Algorithms
Edited by Dr. Olympia Roeva

ISBN 978-953-51-0146-8
Hard cover, 376 pages
Publisher InTech
Published online 07, March, 2012
Published in print edition March, 2012

The book addresses some of the most recent issues, with the theoretical and methodological aspects, of evolutionary multi-objective optimization problems and the various design challenges using different hybrid intelligent approaches. Multi-objective optimization has been available for about two decades, and its application in real-world problems is continuously increasing. Furthermore, many applications function more effectively using a hybrid systems approach. The book presents hybrid techniques based on Artificial Neural Network, Fuzzy Sets, Automata Theory, other metaheuristic or classical algorithms, etc. The book examines various examples of algorithms in different real-world application domains as graph growing problem, speech synthesis, traveling salesman problem, scheduling problems, antenna design, genes design, modeling of chemical and biochemical processes etc.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Aymen Sioud, MarcGravel and Caroline Gagné (2012). Hybrid Genetic Algorithms for the Single Machine Scheduling Problem with Sequence-Dependent Setup Times, Real-World Applications of Genetic Algorithms, Dr. Olympia Roeva (Ed.), ISBN: 978-953-51-0146-8, InTech, Available from: http://www.intechopen.com/books/real-world-applications-of-genetic-algorithms/hybrid-genetic-algorithms-for-the-single-machine-scheduling-problem-with-sequence-dependent-setup-ti

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen