A Simple Genetic Algorithm for a Minimal Overlapping Scheduling Problem

Alfredo Olivera Sergio Nesmachnow
Instituto de Computación, Facultad de Ingeniería,
Universidad de la República, Montevideo, Uruguay.
{aolivera, sergion}@fing.edu.uy

July 30, 2004

Abstract

This article introduces a new version of the Multiple Machine Scheduling Problem: the Scheduling Problem with Time Windows and Minimal Overlap (SPTWMO). Given a set of non-preemptive jobs with time windows and a number of identical machines, the problem consists on finding a starting time for each job which satisfies time window constraints while minimizing a measure of *resource infeasibility* (the Total Overlap). The problem is NP-Complete even in the case when only one machine is considered. We present a simple genetic algorithm applied to the SPTWMO, reporting efficient numerical results according to lower bounds obtained solving the preemptive version of the problem.
Keywords: Genetic Algorithms, Optimization, Scheduling.

Destinated to Agents and Intelligent Systems Workshop

1 Introduction

Scheduling problems consist in determining execution times for a set of jobs in order to satisfy *temporal constraints* and *resource constraints*, while optimizing a *quality measure*. Traditionally, time windows (a time interval in which a job must be executed) and precedence constraints (linear inequalities relating starting times of pairs of jobs) appear as temporal constraints. Resource constraints model the availability of machines over time (usually one machine is able to process only one job at a time) and may also impose compatibility restrictions between jobs and machines. Many alternatives have been proposed as quality measures, for example, the total execution time (makespan) and the number of late jobs, among others.

In some cases it not possible to satisfy both temporal and resource constraints, and some restrictions are relaxed, including them in the quality measure. The most common approach consists in relaxing temporal constraints, that means, to find schedules that satisfy every resource constraint and a *good* subset of temporal constraints. The idea behind this approach is that when resource constraints are violated, more machines will be needed to put the schedule into practice. When temporal constraints are time windows and there is only one machine available, the problem is known as One Machine Problem [1] or $1|r_j|\sum U_j$ in the classification proposed by Graham et al. [2]. In the One Machine Problem, the objective is to find a schedule that minimizes the number of *late* jobs (those jobs violating temporal constraints). This is a well known problem, and various methods have been proposed to solve it [3, 4, 5].

An alternative approach consists on relaxing resource constraints. In this case, the problem is to find a schedule that satisfies every temporal constraint while minimizing some measure of the violated resource constraints. In a survey performed, we have not found references to this approach in the literature. In our previous work [6] we presented the problem and developed a simple genetic approach to solve the single machine version. In this article, we extend this idea for a problem with multiple identical machines, non-preemptive jobs and time windows as temporal constraints.

The rest of the paper is organized as follows. Section 2 presents the problem, its mathematical formulation as well as some applications and the computational complexity analysis. In Section 3, a genetic algorithm to solve the problem is specified. Section 4 summarizes and comments the results of the algorithm's execution over a set of test cases. Finally, in Section 5, we state the conclusions and propose lines for future work.

2 Problem Description

In the Scheduling Problem with Time Windows and Minimal Overlap, a set of non-preemptive jobs $J = \{1, \ldots, n\}$ is to be scheduled over m identical machines. Each job $j \in J$ has an associated time window $[r_j, d_j]$ and a processing time p_j ($r_j, d_j, p_j \in \mathbb{Z}$). Time window constraints state that the starting time t of each job j must verify that $t \geq r_j$ (the job cannot start before r_j) and $t + p_j \leq d_j$ (it cannot end after d_j).

It can be assumed, without loss of generality, that $\min_{j \in J}\{r_j\} = 0$. Also, as every task j must be completed by d_j, there will be no task executing later than $T = \max_{j \in J}\{d_j\} - 1$. So, the relevant time interval for the scheduling problem is $[0, T]$.

Given a feasible schedule (i.e., a schedule that satisfies the time window constraints), the expression in Equation 1 defines the *Overlap* at time $t \in [0, T]$ where q_t stands for the number of jobs running at time t.

$$O_t = \begin{cases} 0 & \text{if } q_t \leq m \\ q_t - m & \text{otherwise} \end{cases} . \tag{1}$$

The non-overlapping situation (first case in Equation 1) occurs when all running jobs can be executed using the m available machines, while on the other case only m of the jobs can be processed and the remaining $q_t - m$ jobs overlap. A more compact expression for the Overlap at time t is

Table 1: Jobs for an example instance of the SPTWMO.

j	1	2	3	4	5
r_j	0	5	1	2	3
d_j	5	11	8	9	10
p_j	4	5	5	6	5

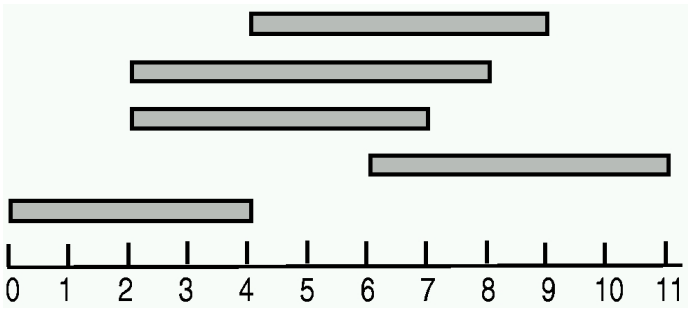

Figure 1: A solution to the example instance with Total Overlap 7.

$O_t = \max(0, q_t - m)$. The SPTWMO proposes to minimize the *Total Overlap* $\sum_{t=0}^{T} O_t$, which evaluates the resource constraints violation in the relevant time interval.

Considering the simple SPTWMO instance with 2 machines and 5 jobs specified in Table 1 and the Gantt diagram for a feasible schedule shown in Figure 1. At times 0 and 1, only one job is being executed, so $O_0 = O_1 = 0$. On each of the next four time instants 3 jobs are running, so $O_2 = O_3 = O_4 = O_5 = 1$. There are 4 jobs running at time 6, so $O_6 = 2$. Finally, $O_7 = 1$ and $O_8 = O_9 = O_{10} = 0$. So, the presented solution has a Total Overlap value of 7.

2.1 Mathematical Formulation

The SPTWMO may be formulated as a Mixed Integer Linear Program. Given a job $j \in J$, a binary decision variable $s_{j,t}$ is defined to indicate whether job j starts at time t or not. The index t of $s_{j,t}$ takes values only in $[r_j, d_j - p_j]$, which is the time interval where j can start without violating the time window constraint. In addition, a positive real variable O_t is defined for every $t \in [0, T]$, evaluating the Overlap at time t. The problem is formulated in Equations 2-6.

$$\min \sum_{t=0}^{T} O_t \tag{2}$$

$$\text{s.t.} \sum_{t=r_j}^{d_j - p_j} s_{j,t} = 1 \qquad \forall j \in J \tag{3}$$

$$\sum_{j=1}^{n} \sum_{t'=\max(t-p_j+1, r_j)}^{\min(t, d_j - p_j)} s_{j,t'} \leq O_t + m \qquad \forall t \in [0, T] \tag{4}$$

$$s_{j,t} \in \{0, 1\} \qquad \forall j \in J, \forall t \in [r_j, d_j - p_j] \tag{5}$$

$$O_t \in \mathbb{R}^+ \qquad \forall t \in [0, T] \tag{6}$$

In this model, Equation 2 gives the objective function, which corresponds to minimizing the Total Overlap. Restriction 3 states that job j must start exactly once within $[r_j, d_j - p_j]$. Equation 4 determines the Overlap values at each time unit in the relevant time interval. Finally, Equations 5 and 6 set the domain for the decision variables.

2.2 Applications

An application of this particular scheduling problem arises when post-processing the output of the Critical Path Method (CPM) [7]. The CPM allows to find a minimal duration schedule for a project, composed by a set of jobs subject to precedence constraints, without considering resource constraints. After solving the problem, for each job j the earliest starting time t_j and a slack s_j are obtained, such that the job can start anytime in the interval $[t_j, t_j + s_j]$ without retarding the whole project duration. If resource constraints are considered for some jobs, the problem of finding starting times satisfying resource constraints (as much as possible) without affecting the project duration can be modeled as a SPTWMO.

The SPTWMO can also be applied in the Vehicle Routing with time windows domain [8]. Classic Vehicle Routing algorithms do not consider fleets having a finite number of vehicles and also do not allow vehicles to perform more than one route in a planning period. When such cases must be considered, assigning routes to vehicles is not a trivial task [9, 10]. For finding solutions, the problem can be decomposed in a *routing phase* that generates a candidate set of routes, and a *scheduling phase* which determines if those routes can be assigned to the fleet of vehicles. Given that routes are subject to time windows, the scheduling phase can be solved as a SPTWMO. If a schedule with Total Overlap 0 is obtained, then the routes can be assigned to vehicles. If not the SPTWMO solution provides information about congested time intervals (those in which the Overlap is not 0), which can be used as feedback to perform a routing phase iteration. In fact, the problem appeared while solving a Vehicle Routing Problem.

2.3 Computational Complexity

The SPTWMO is NP-Complete, even for the simplified case when only one machine is considered ($m = 1$), as we prove as follows.

Consider the decision problem (**SP**): "given a finite set of tasks with time windows and known processing times, does there exist a schedule over one machine satisfying time windows and resource constraints?". **SP** is known to be NP-Complete [11]. The decision problem associated to SPTWMO with $m = 1$ is (**DP**): "given a finite set of tasks with time windows and known processing times and $K \in \mathbb{Z}$, does there exist a schedule over one machine satisfying time windows constraints having a Total Overlap value less than or equal to K?".

Any algorithm that solves **DP** can be used to solve **SP**. Given an instance of **SP**, an instance of **DP** is solved with the same input data and $K = 0$. If the answer to **DP** is "yes", then a schedule exists such that time windows are respected and the Total Overlap value is less than or equal to 0. As Total Overlap cannot be negative, it should be equal to 0, which implies that the resource constraint is also satisfied. So the answer to **SP** is "yes". On the other hand, if the answer to **DP** is "no", then every schedule that satisfies time windows violates the resource constraint, so the answer to **SP** is "no".

The previous argumentation shows that **SP** can be reduced to **DP**; since input data of both problem instances are the same, the reduction can be performed in polynomial time. Given that **SP** is NP-Complete, **DP** also is. Moreover, as **DP** is the decision problem associated to SPTWMO with $m = 1$, the latter is NP-Complete. Finally, if m has not a fixed value the problem is a generalization, and so it remains NP-Complete.

The fact that the SPTWMO is NP-Complete, suggests the convenience of using heuristic algorithms to find solutions of acceptable quality in reasonable amounts of time, since the low efficiency of traditional exact methods makes them impractical for solving large size problem instances.

3 The Genetic Algorithm

Evolutionary techniques have been successfully applied to solve a variety of combinatorial optimization problems, including scheduling problems [12, 13]. The proposed Genetic Algorithm (GA)

follows the generational model [14], but it incorporates an elitist mechanism, perpetuating the best individual over generations. A pseudocode of the GA is shown in Algorithm 1.

Algorithm 1 Simple Genetic Algorithm with Elitism.

```
Initialize population P_0 with PS individuals
N ← 1
repeat
  Create an empty population P_N ← {}
  while |P_N| < PS do
    Select individuals p_1 and p_2 from P_{N-1}
    Apply crossover to p_1 and p_2 to obtain s_1 and s_2
    Mutate s_1 and s_2
    Add s_1 and s_2 to P_N: P_N ← P_N ∪ {s_1, s_2}
  end while
  b ← the best individual of P_{N-1}
  if b ∉ P_N then
    w ← the worst individual of P_N
    Replace w with b in P_N: P_N ← (P_N \ {w}) ∪ {b}
  end if
  N ← N + 1
until N = nGen
Return the best individual of P_N
```

The GA uses a fixed effort stopping criteria, evolving during $nGen$ generations and returning the best solution found. Considering that this particular problem has not been previously tackled, we decided to set a high number of generations as stopping criteria ($nGen = 2000$), in the quest for accurate solutions. The GA was implemented using the `GAlib v2.4` [15] library.

3.1 Problem Encoding

Choosing a good encoding scheme for the individuals has high impact in GA's performance [16]. In traditional scheduling problems is usual to use permutation based codifications indicating the order in which jobs start [17]. This codification is adequate for problems where, given the starting order of the jobs, the starting times can be easily inferred. Since this is not the case in the SPTWMO, we discarded using a permutation based codification scheme. We also decided not to use binary codifications, since it would imply the need of using a decodification heuristic procedure as the one used in Crawels et al. [12].

To encode SPTWMO solutions we used vectors in $\mathbb{Z}^n$, where the integer in position j indicates the starting time of job j. For instance, the solution presented on the example of Section 2 (Figure 1) is represented by the vector $(0, 6, 2, 2, 4)$. Using this integer-based codification most of the traditional evolutive operators are easy to implement, as we present in the next subsection.

3.2 Genetic Operators

The following list describes the operators used by the GA implemented.

Initialization: it randomly builds individuals, choosing the starting time of each job j uniformly over the interval of its possible starting times: $[r_j, d_j - p_j]$. This procedure ensures working only with feasible individuals in the initial population.

Selection: proportional selection is used.

Crossover: three classic [16] operators were evaluated: Single and Double Point crossover (SPX and DPX) and uniform crossover (UX). This three operators maintain the feasibility of individuals, and they are applied with a certain probability p_c.

Mutation: three operators were considered, which are applied to each individual with probability p_m. The classic swap mutation (SM) [16], that randomly picks two jobs and interchanges their starting times, and two problem specific operators which modify the starting time of one job in the schedule: Flip Mutation (FM) and Push Mutation (PM). FM operator randomly picks a new starting time for job j from its possible starting time interval $[r_j, d_j - p_j]$, similarly to the initialization operator. PM operator increases or decreases in one unit (with probability 0.5) the starting time of the job. SM and PM operators may generate infeasible solutions that violate time window constraints. Infeasible solutions will be penalized by the fitness function.

3.3 Fitness Function

The fitness function is the sum of two terms: one term evaluates the Total Overlap of the schedule (which is the objective of the problem) and the other penalizes time windows constraints violations, pressing the evolutive search towards the feasible region of the problem. Equation 7 shows the fitness function expression, where ω stands for the number of time instants that tasks are running out of their time windows and ϕ is a generic penalization function.

$$fitness = \sum_{t=0}^{T} O_t + \phi(\omega) \ . \tag{7}$$

Two models were proposed for ϕ: a linear penalization model where $\phi(\omega) = k\omega$ and a quadratic penalization model in which $\phi(\omega) = k\omega^2$. The empirical evidence showed that, in practice, using the linear model with $k = 5$ allowed to obtain a satisfactory evolution behavior, not allowing the perpetuation of non feasible solutions for the test problems considered.

4 Computational Results

4.1 Test Problems

As it was previously mentioned, as far as we know this work is the first reference about this problem, and therefore it was not possible to use standard test suites to evaluate our results. In order to do that, we designed a test suite of 1040 instances, generated following the scheme proposed by Baptiste et al. [3] for the One Machine Problem. Although SPTMWO is not exactly the One Machine Problem, the input data for both problems are very similar (SPTWMO has m as an additional parameter) and we assume that considerations taken into account for designing test instances in Baptiste et al. are also valid for our problem.

Two independent sets of test problems were generated in order to avoid biases in the experiments. One set of problems was used in the parameter configuration phase and the other was used for the GA evaluation. The parameter configuration test problem set consists of 50 problems with n varying between 20 and 100 and $m = 1$.

Given a feasible schedule for a problem instance with $m > 1$, if the number of machines is increased by 1, then the overlap of the given schedule in the new instance decreases in at least $|\{t : O_t > 0\}|$ units (where O_t is defined as in Equation 1). As a consequence, problem instances with low values of m are harder to schedule, so we assume are more difficult to solve using the genetic algorithm. The motivation for using $m = 1$ in the parameter configuration test problems is to tune the genetic algorithm parameters with a subset of the harder problem instances.

The evaluation problem set includes 165 sets of jobs with n varying between 20 and 100; each set of jobs was tested with values of m varying from 1 to 6, so there is a total of 990 test problems.

4.2 Lower Bounds

Given that optimal solutions to the test problems are not known, we used lower bounds (obtained solving the problems considering preemptive jobs) to estimate the quality of the GA's solutions. The preemptive version of SPTWMO can be modelled as a minimum cost network flow problem. Consider the net $N = (X, U, W)$ defined in Equations 8-12.

$$X = J \cup [0, T] \cup \{u, z\} \tag{8}$$

$$U = \{(j, t) \,|\, j \in J,\, t \in [r_j, d_j - 1]\} \cup \{(t, u), (t, z) \,|\, t \in [0, T]\} \tag{9}$$

$$W(j, t) = 1 \quad \forall j \in J,\, \forall t \in [r_j, d_j - 1] \tag{10}$$

$$W(t, z) = m \quad \forall t \in [0, T] \tag{11}$$

$$W(t, u) = \infty \quad \forall t \in [0, T] \tag{12}$$

In the previous flow network, X is the set of nodes, U is the set of arcs and W is the capacity of each arc. There is one node for each job, one node for each possible time instant and 2 auxiliary sink nodes (u and z) which help to express the objective function. Equation 10 states that each job j can send at most one flow unit to each time instant in its time window and must send a total of p_j units. Equation 11 indicates that each time instant can send at most m flow units to node z while Equation 12 states that each time instant can send flow to node u with no limit. Feasible solutions to the SPTWMO can be represented as flows over N, while the objective is to minimize the total flow reaching the sink node u. In an optimal solution, the flow that each time instant t sends to u will be O_t, the Overlap value at time t.

The linear relaxation of a network flow problem with integer parameters is known to have integer basic solutions [18]. So, the optimal solution to the flow problem presented (i.e., the desired lower bound) can be obtained using Linear Programming techniques. Equations 13-18 present the formulation of the preemptive version of the SPTWMO as a min-cost network flow problem.

$$\min \sum_{t=0}^{T} x_{t,u} \tag{13}$$

$$\text{s.t.} \sum_{t=r_j}^{d_j-1} x_{j,t} = p_j \qquad \forall j \in J \tag{14}$$

$$\sum_{\{j \in J |\, t \in [r_j, d_j - 1]\}} x_{j,t} = x_{t,u} + x_{t,z} \qquad \forall t \in [0, T] \tag{15}$$

$$0 \leq x_{j,t} \leq 1 \qquad \forall j \in J,\, \forall t \in [r_j, d_j - 1] \tag{16}$$

$$0 \leq x_{t,z} \leq m \qquad \forall t \in [0, T] \tag{17}$$

$$x_{t,u} \geq 0 \qquad \forall t \in [0, T] \tag{18}$$

Being LB the optimal value of this problem and s_{GA} the value of the solution obtained by the Genetic Algorithm, we propose using $GAP = 100(s_{GA} - LB)/LB$ as an estimation value for the GA's solutions quality. GAP measures the difference between the preemptive lower bound and the value of the solution found by the GA, in terms of the lower bound. If the preemptive lower bound is close to the unknown optimal value for the non-preemptive problem then GAP is a good quality measure. So, it must be considered as an upper bound for the distance to the optimal solution, whose tightness is unknown.

4.3 Operator and Parameter Configuration

The GA operators and parameters were determined in three phases: the analysis of crossover and mutation operators, the determination of the population size (PS) and the configuration of crossover and mutation probabilities (p_c and p_m).

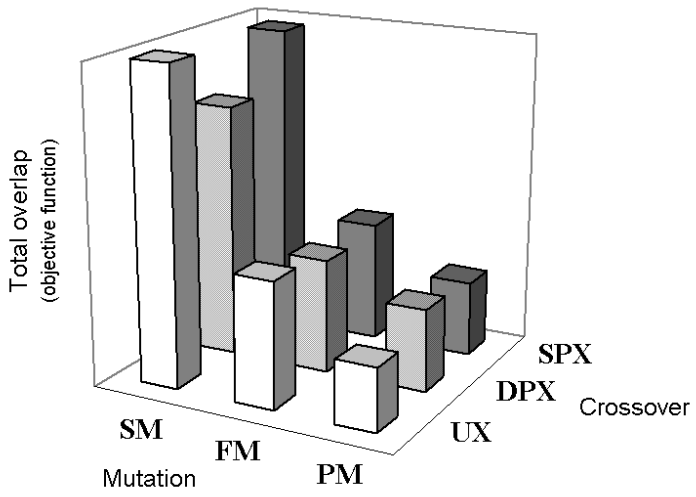

Figure 2: Representative case obtained when varying crossover and mutation operators.

Table 2: Results obtained varying the population size.

n	T_{250}	T_{500}	$\frac{T_{500}}{T_{250}}$	$S_{500} = S_{250}$	$S_{500} < S_{250}$	$S_{500} > S_{250}$
20− 40	53	108	2.04	93%	–	7%
50− 70	108	220	2.04	55%	45%	–
80−100	142	289	2.04	53%	27%	20%

In order to select the most appropriate operators, we tested every combination of the crossover and mutation operators proposed in Section 3.2. Each combination was run over the parameter configuration test problems, using $PS = 500$, $p_c = 0.7$ and $p_m = 0.05$. For the majority of the problems, the objective function followed the same pattern, which is shown in Figure 2. The empirical analysis showed that the choice of the crossover operator does not have a significant influence on the quality of the results, and so we chose to use UX in the GA evaluation experiments. Related to mutation operators, solutions obtained when using SM shows low-quality values when compared with those obtained by FM and PM, which presented close results. We opted to use PM in the GA evaluation experiments, considering that PM introduces slight perturbations, performing a local search in the neighborhood of each solution, while FM simply selects new starting times, possibly disrupting accurate schedules.

To determine an adequate population size, we evaluated the solutions obtained using populations of 250 and 500 individuals for 5 independent runs of the GA over the parameter configuration test problems. We used UX and PM operators, with $p_c = 0.7$ and $p_m = 0.05$. Table 2 summarizes the results, where T_{PS} and S_{PS} stand for the average execution time (in seconds) and the objective value, respectively, when using PS individuals. Even though the execution times duplicated when using $PS = 500$, solution quality increased. Given the low execution times obtained even for problems with $n = 100$, we decided to use 500 individuals.

To choose the best crossover and mutation probabilities, three values were considered for each: 0.5, 0.7 and 0.9 for p_c and 0.01, 0.05 and 0.1 for p_m. The configuration test problems were solved with each of the 9 combinations. The average GAP values obtained in each case are presented in table 3. We opted to use the combination $p_c = 0.9$ and $p_m = 0.05$, because it produced the best results.

Table 3: Average GAP obtained varying p_c and p_m.

p_c	p_m	GAP	p_c	p_m	GAP	p_c	p_m	GAP
0.5	0.01	12%	0.7	0.01	12%	0.9	0.01	12%
0.5	0.05	10%	0.7	0.05	9%	0.9	0.05	4%
0.5	0.1	5%	0.7	0.1	9%	0.9	0.1	8%

Table 4: Results for the evaluation test problems.

m	n	GAP	T	G	Optimal	m	n	GAP	T	G	Optimal
1	20−40	2%	64	370	67%	4	20−40	1%	62	151	91%
1	50−70	2%	123	409	66%	4	50−70	5%	125	477	61%
1	80−100	2%	189	577	52%	4	80−100	8%	189	559	65%
2	20−40	4%	62	405	73%	5	20−40	0%	60	69	94%
2	50−70	2%	125	525	57%	5	50−70	2%	124	394	76%
2	80−100	3%	184	659	44%	5	80−100	3%	186	486	75%
3	20−40	1%	62	210	90%	6	20−40	0%	60	38	98%
3	50−70	4%	125	538	57%	6	50−70	4%	125	361	76%
3	80−100	1%	188	644	54%	6	80−100	2%	189	400	77%

4.4 Evaluation Results

To evaluate the results, the GA was used to solve the 990 test problems with the parameters determined in the configuration phase. Table 4 summarizes the results, reporting the average values of GAP, the execution time in seconds (T) and the average generation in which the best individual was found (G), and the percentage of the cases in which a provably optimal solution was found (i.e. $GAP = 0$). We used a Pentium III machine at 300 MHz having 192 MB RAM with Windows ME operating system as execution platform.

Related to solution quality, provably optimal solutions were found on 71% of the cases; in 20% of the cases GAP was not greater than 5%. Only for 9% of the test problems was GAP greater than 5%. Given that GAP is an upper bound on the distance to the optimal value, we consider this results as acceptable.

Figure 3 shows that as m increases, the percentage of cases in which a provably optimal solution is found becomes bigger. This results support the conjecture stating that instances with a high number of machines become easier to solve for the GA.

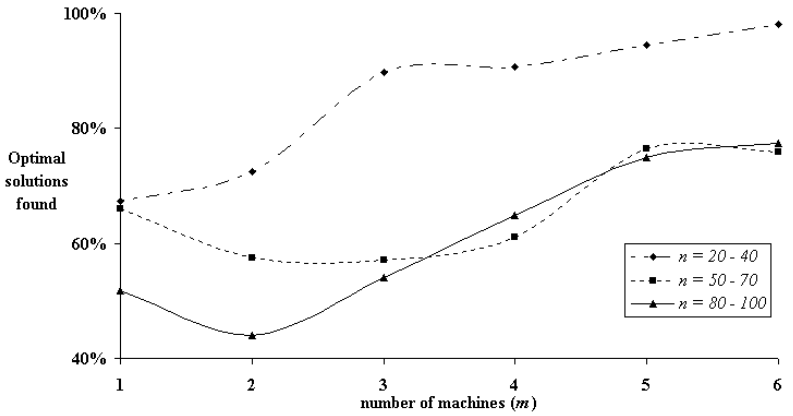

Figure 3: Percentage of the cases in which a provably optimal solution was found.

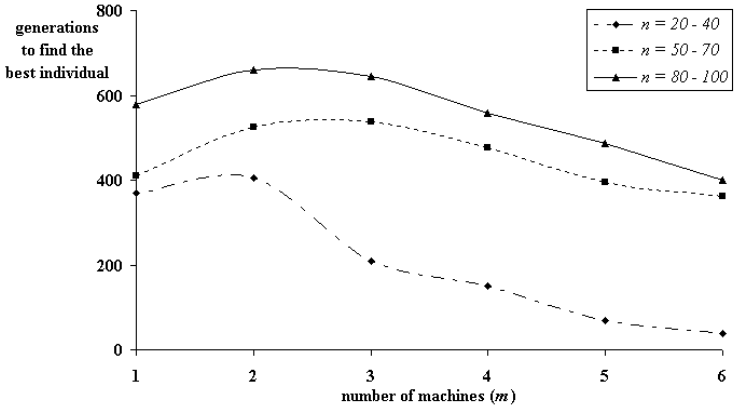

Figure 4: Average number of generations needed to find the best individual.

On average, the best individual is found earlier when more machines are available, as is depicted in Figure 4. It also can be seen that, for a given number of machines, problems with less jobs take less generations to reach a convergence state.

Execution times were moderate in all cases, and can be reduced (modifying the fixed effort criterion) observing that on average the GA finds the best individual in much less than the 2000 generations it evolves.

Table 4 does not reveal a relation between the average GAP and the number of jobs and machines. GAP is a good quality measure only when the preemptive lower bound is near to the non-preemptive optimal value. We have seen that this proximity depends on each particular case.

5 Conclusions and Future Work

In this paper we have presented a genetic algorithm that solves a scheduling problem over multiple machines minimizing the Total Overlap. The problem was mathematically formulated, as well as its preemptive version, which was used to obtain lower bounds for estimating the quality of the solutions obtained by the GA.

Several aspects of the GA configuration were analyzed, including empirical analysis of operators and parameters setting, searching for high quality results on a set of SPTWMO configuration instances.

The algorithm was evaluated over a wide set of test problems with 20 to 100 jobs and 1 to 6 machines. Results show that the GA was effective to solve the problem, reaching high-quality solutions (including several optimal solutions), using low execution times.

Further work should be made to find tighter lower bounds, which would allow a more precise estimation of solution quality. Other difficulty measures (rather than m and n) should be proposed, in order to distinguish hard problem instances from simpler ones. Finally, it would be interesting to calculate exact results for the problem; as well as to design other heuristics to solve this problem and compare the results with the GA solutions presented.

References

[1] Carlier, J.: The one machine sequencing problem. European Journal of Operations Research **11** (1982) 42–47

[2] Graham, R., Lawler, E., Lenstra, J., Rinnooy Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics **5** (1979) 287–326

[3] Baptiste, P., Le Pape, C., Peridy, L.: Global constraints for partial csps: A case-study of resource and due date constraints. Lecture Notes in Computer Science **1520** (1998) 87–101

[4] Dauzère-Pérès, S.: Minimizing late jobs in the general one machine scheduling problem. European Journal of Operational Research **81** (1995) 134–142

[5] Moore, J.: An n job, one machine sequencing algorithm for minimizing the number of late jobs. Management Science **15** (1968) 102–109

[6] Olivera, A., Nesmachnow, S.: Algoritmo genético aplicado a un problema de scheduling con mínimo solapamiento. In: 3er Congreso Español de Metaheursticas, Algoritmos Evolutivos y Bioinspirados (MAEB '04). (2004) 486–493 Text in Spanish.

[7] Kaufmann, A., Desbazeille, G.: The Critical Path Method. Gordon and Breach (1969)

[8] Toth, P., Vigo, D.: The Vehicle Routing Problem. SIAM (2002)

[9] Brandão, J., Mercer, A.: The multi-trip vehicle routing problem. Journal of the Operational Research Society **49** (1998) 799–805

[10] Taillard, D., Laporte, G., Gendreau, M.: Vehicle routing with multiple use of vehicles. Journal of the Operations Research Society **47** (1996) 1065–1070

[11] Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company (1979)

[12] Crauwels, H., Potts, C., Van Wassenhove, L.: Local search heuristics for the single machine total weighted tardiness scheduling problem. Informs Journal on Computing **10** (1998) 341–350

[13] Avci, S., Akturk, M., Storer, R.: A problem space algorithm for single machine weighted tardiness problems. IIE Transactions **35** (2003) 479–486

[14] Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold (1991)

[15] Wall, M.: Galib: A C++ library of genetic algorithm components. http://lancet.mit.edu/ga (1996)

[16] Golberg, D.: Genetic algorithms in search, optimization, and machine learning. Addison Wesley Longman (1989)

[17] Bierwirth, C., Mattfeld, D., Kopfer, H.: On permutation representations for scheduling problems. Lecture Notes in Computer Science **1141** (1996) 310–318

[18] Nemhauser, G., Wolsey, L.: Integer and Combinatorial Optimization. John Wiley and Sons (1998)