Weighted Tardiness Minimization in Job Shops with Setup Times by Hybrid Genetic Algorithm

Miguel A. González, Camino R. Vela, and Ramiro Varela

Department of Computer Science,
University of Oviedo, (Spain) Campus de Viesques s/n, Gijón, 33271, Spain
http://www.aic.uniovi.es/tc

Abstract. In this paper we confront the weighted tardiness minimization in the job shop scheduling problem with sequence-dependent setup times. We start by extending an existing disjunctive graph model used for makespan minimization to represent the weighted tardiness problem. Using this representation, we adapt a local search neighborhood originally defined for makespan minimization. The proposed neighborhood structure is used in a genetic algorithm hybridized with a simple tabu search method. This algorithm is quite competitive with state-of-the-art methods in solving problem instances from several datasets of both classical JSP and JSP with setup times.

1 Introduction

In this paper we confront the Job Shop Scheduling Problem with Sequence-Dependent Setup Times (SDST-JSP) with weighted tardiness minimization. JSP has interested to researchers over the last decades, but in most cases the objective function is the makespan, even though other objective functions such as weighted tardiness or total flow time are sometimes more important in many real-life problems. Also, setup considerations are a relevant characteristic of many real scheduling problems that add to the difficulty of solving these problems with respect to their non-setup counterparts. Incorporating sequence-dependent setup times changes the nature of scheduling problems, so well-known results and techniques for the JSP are not directly applicable to the SDST-JSP. Some extensions have been done for makespan minimization in [1,6,13].

As far as we know, the best approach to weighted tardiness minimization in the SDST-JSP is the presented by Sun and Noble in [10]. They propose a shifting bottleneck algorithm and in their experimental study this algorithm is compared with very simple heuristics (some priority rules) across randomly generated instances that are not available for further comparison. However, weighted tardiness has been largely considered for the classic JSP, maybe the algorithms proposed in [3] and [7] being the most relevant approaches currently. In [3], Essafi et al. propose a hybrid genetic algorithm that uses a local search based in reversing critical arcs and an algorithm that iterates between hill climbing and random generation of neighbors to escape from local optima. In [7] Mati et al. propose a local search method that uses estimators to evaluate the neighbors and

J.A. Lozano, J.A. Gámez, and J.A. Moreno (Eds.): CAEPIA 2011, LNAI 7023, pp. 363–372, 2011.
© Springer-Verlag Berlin Heidelberg 2011

that is capable of minimizing any regular objective function, i.e. a non-decreasing function on the completion time of the operations.

We propose a hybrid algorithm that combines a genetic algorithm (GA) with tabu search (TS). The core of this algorithm is a variation of the neighborhood structure N_1^S introduced in [13] for the SDST-JSP with makespan minimization, which in its turn extends the structures proposed in [12] for the classical JSP. We define a disjunctive graph model for the SDST-JSP with weighted tardiness minimization to formalize this neigborhood. The proposed algorithm is termed $GA+TS$ in the following. We also define a method for estimating the weighted tardiness of the neighbors, and we will see that this estimation is less accurate and more time consuming than similar estimations for the makespan, due to the difference in the problem difficulty. We have conducted an experimental study across conventional benchmarks to compare $GA+TS$ with state-of-the-art algorithms in both classical JSP and SDST-JSP. The results show that the proposed algorithm outperforms the other methods.

The rest of the paper is organized as follows. In Section 2 we formulate the JSP and introduce the notation used across the paper. In section 3 we describe the main components of the genetic algorithm. The proposed neighborhood structure, the weighted tardiness estimation algorithm and the main components of the TS algorithm are described in Section 4. Section 5 reports results from the experimental study. Finally, in Section 6 we summarize the main conclusions and propose some ideas for future work.

2 Description of the Problem

In the job shop scheduling problem, a set of N jobs, $J = \{J_1, \ldots, J_N\}$, are to be processed on a set of M machines (resources), $R = \{R_1, \ldots, R_M\}$ while minimizing some function of completion times of the jobs, subject to constraints: (i) the sequence of machines for each job is prescribed, and (ii) each machine can process at most one job at a time. The processing of a job on a machine is called an operation, and its duration is a given constant. We denote by p_u the processing time of operation u. A time may be needed to adjust a machine between two consecutive operations, which is called a setup time, and which may or may not be sequence-dependent. We adopt the following notation for the setup times: S_{uv} is the setup time between consecutive operations $u, v \in R_j$, and S_{0u} is the setup time required before u if this operation is the first one scheduled on his machine. A job J_i may also have a due date d_i, that is a time before jobs should be completed, and a weight w_i, that is the relevance of the job. The objective here is to minimize the weighted cost of the jobs exceeding its due-dates, also known as the weighted tardiness. In the following, we denote by t_u the starting time of operation u, that needs to be determined.

The SDST-JSP has two binary constraints: precedence and capacity. Precedence constraints, defined by the sequential routings of the tasks within a job, translate into linear inequalities of the type: $t_u + p_u \leq t_v$, if v is the next operation to u in the job sequence. Capacity constraints that restrict the use of each

resource to only one task at a time translate into disjunctive constraints of the form: $t_u + p_u + S_{uv} \leq t_v \vee t_v + p_v + S_{vu} \leq t_u$, where u and v are operations requiring the same machine.

The objective is to obtain a feasible schedule that minimizes the weighted tardiness, defined as:

$$\sum_{i=1,\ldots,N} w_i T_i$$

where T_i is the tardiness of the job i, defined as $T_i = \max\{C_i - d_i, 0\}$, where C_i is the completion time of job i. This problem is denoted by $J|s_{ij}|\sum w_i T_i$ in the literature.

2.1 The Disjunctive Graph Model Representation

The disjunctive graph is a common representation in scheduling, its exact definition depending on the particular problem. For the $J|s_{ij}|\sum w_i T_i$ problem, we propose that it be represented by a directed graph $G = (V, A \cup E \cup I_1 \cup I_2)$. Each node in set V represents a task of the problem, with the exception of the dummy nodes *start* and end_i $1 \leq i \leq N$, which represent fictitious operations that do not require any machine. Arcs in A are called *conjunctive arcs* and represent precedence constraints while arcs in E are called *disjunctive arcs* and represent capacity constraints. Set E is partitioned into subsets E_i, with $E = \cup_{j=1,\ldots,M} E_j$, where E_j corresponds to resource R_j and includes two directed arcs (v, w) and (w, v) for each pair v, w of operations requiring that resource. Each arc (v, w) in A is weighted with the processing time of the operation at the source node, p_v, and each arc (v, w) of E is weighted with $p_v + S_{vw}$. Set I_1 includes arcs of the form $(start, v)$ for each operation v of the problem, weighted with S_{0v}. Set I_2 includes arcs $(\omega(i), end_i)$, $1 \leq i \leq N$, weighted with $p_{\omega(i)}$, where $\omega(i)$ is the last operation of job J_i

A feasible schedule is represented by an acyclic subgraph G_s of G, $G_s = (V, A \cup H \cup J_1 \cup I_2)$, where $H = \cup_{j=1\ldots M} H_j$, H_j being a minimal subset of arcs of E_j defining a processing order for all operations requiring R_j and where J_1 consists of arcs $(start, v_j)$, $j = 1 \ldots M$, v_j being the first operation of H_j. Finding a solution can thus be reduced to discovering compatible orderings H_j, or partial schedules, that translate into a solution graph G_s without cycles. Figure 1 shows a solution to a problem with 3 jobs and 3 machines; dotted arcs belong to H and J_1, while continuous arcs belong to A.

To calculate the weighted tardiness of the schedule we have to compute the cost of a critical path in G_s to each node end_i $1 \leq i \leq N$, i.e., a directed path in G_s from node *start* to node end_i having maximum cost. Nodes and arcs in a critical path are also termed critical. A critical path may be represented as a sequence of the form $start, B_1, \ldots, B_r, end_i$, $1 \leq i \leq N$, where each B_k, $1 \leq k \leq r$, is a critical block, a maximal subsequence of consecutive operations in the critical path requiring the same machine.

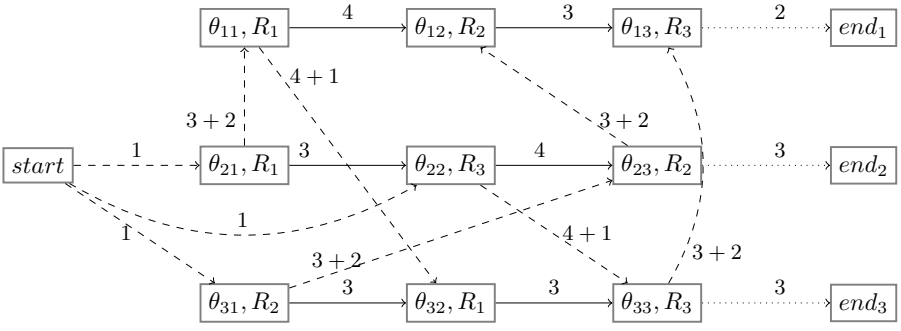

Fig. 1. A feasible schedule to a problem with 3 jobs and 3 machines

In order to simplify expressions, we define the following notation for a feasible schedule. Given a solution graph G_s for the SDST-JSP, the head of an operation v, denoted r_v, is the cost of the longest path from node *start* to node v, i.e., the starting time of v in the schedule represented by G_s. A tail q_v^i, $1 \leq i \leq N$ is the cost of the longest path from node v to node end_i, minus the duration of task in node v. For practical reasons we will take $q_v^i = -\infty$ when no path exist from v to end_i. Here, it is important to remark that we have had to define N tails for each operation, while for makespan minimization it is required just one. Let PJ_v and SJ_v denote respectively the predecessor and successor of v in the job sequence, and PM_v and SM_v the predecessor and successor of v in its machine sequence. We take node *start* to be PJ_v for the first task of every job and PM_v for the first task to be processed in each machine; note that $p_{start} = 0$. Then, the head of every operation v and every dummy node may be computed as follows:

$$r_{start} = 0$$
$$r_v = \max(r_{PJ_v} + p_{PJ_v}, r_{PM_v} + p_{PM_v} + S_{PM_v v})$$
$$r_{end_i} = r_v + p_v, \ (v, end_i) \in I_2, 1 \leq i \leq N$$

Similarly, for $1 \leq i \leq N$, we take node end_i as SJ_v for the last task of job i, and $p_{end_i} = 0$. So, the tail of all operations are computed as follows:

$$q_{end_i}^i = 0$$
$$q_{end_i}^j = -\infty, j \neq i$$
$$q_v^j = \begin{cases} \max(q_{SJ_v}^j + p_{SJ_v}, q_{SM_v}^j + p_{SM_v} + S_{vSM_v}) & \text{if } SM_v \text{ exists} \\ q_{SJ_v}^j + p_{SJ_v} & \text{otherwise} \end{cases}$$
$$q_{start}^j = \max_{v \in SM_{start}} \{q_v^j + p_v + S_{0v}\}$$

3 Genetic Algorithm for the SDST-JSP

In this paper we will use a conventional GA, with permutations with repetition as our encoding schema and *Job Order Crossover* (JOX) for chromosome mating, uniform selection and generational replacement using tournament between two parents and two offspring. To build schedules we have used the Serial Schedule Generation Schema (SSGS) proposed in [1] for the SDST-JSP. SSGS iterates over the operations in the chromosome sequence and assigns each one the earliest starting time that satisfies all constraints with respect to previously scheduled operations. SSGS produces active schedules, provided that the triangular inequality for the setup times holds for all operations requiring the same machine [1], and this is the case in the instances used in our experimental study.

When combined with GA, TS is applied to every schedule produced by SSGS. Then, the chromosome is rebuilt from the improved schedule obtained by TS, so as its characteristics can be transferred to subsequent offspring. This effect of the evaluation function is known as Lamarckian evolution.

4 Tabu Search for the Weighted Tardiness Minimization in the SDST-JSP

We use here a conventional TS algorithm [4], the particular implementation is borrowed from [5]. In the next subsections we describe in detail the components of the algorithm that has been adapted to the SDST-JSP with weighted tardiness minimization, namely the neighborhood structure and the procedure for weighted tardiness estimation after a move.

4.1 The Neighborhood Structure

As usual, this structure is based on changing processing orders in a critical block. However, the number of critical paths in a problem instance is usually large and so not all candidate moves can be considered in order to obtain a reasonable number of neighbors. To do that, we consider two options: all critical paths corresponding to tardy jobs or just the critical path that most contributes to the objective function. We have observed that neither option is clearly better than the other in some preliminary experiments. Moreover, it happens that there are substantial differences that depend on the instances, so we have finally opted to consider them both and choose randomly each time the TS algorithm is issued.

Also, we have opted to consider a neighborhood structure that generates a small number of neighbors from each critical block. For this reason, we adapted the structure N_1^S proposed in [13] for SDST-JSP with makespan minimization, which is based on previous structures given in [8] and [12] for the standard JSP. This structure can be formalized for the SDST-JSP with weighted tardiness minimization from the disjunctive model defined in 2.1. N_1^S is based on the following results.

Proposition 1. *Let H be a schedule and (v, w) a disjunctive arc that is not in a critical block. Then, reversing the arc (v, w) does not produce any improvement, provided that the triangular inequality for the setup times holds for all operations requiring the same machine.*

So we have to reverse a critical arc to obtain an improving schedule. In [13] the authors define non-improving conditions for some reversals of critical arcs in makespan optimization that in principle can not be translated to the weighted tardiness case. Regarding feasibility, the next result gives a sufficient condition for an alternative path not existing after the reversal of a critical arc. If such an alternative path exists then the resulting neighbor would be unfeasible because it would contain a cycle.

Proposition 2. *Let H be a schedule and (v, w) an arc in a critical block. A sufficient condition for an alternative path between v and w not existing is that*

$$r_{PJ_w} < r_{SJ_v} + p_{SJ_v} + min\{S_{kl} | (k, l) \in E, J_k = J_v\}$$

where J_k is the job of operation k.

So, the neighborhood structure N_1^S is defined as follows.

Definition 1. *(N_1^S) Given a schedule H, the neighborhood $N_1^S(H)$ consists of all schedules derived from H by reversing one arc (v, w) of a critical block, provided that feasibility condition given in proposition 2 holds.*

4.2 Weighted Tardiness Estimation

Even though computing the weighted tardiness of a neighbor only requires to recompute heads (tails) of operations that are after (before) the first (last) operation moved, for the sake of efficiency the selection rule is based on weighted tardiness estimations instead of computing the actual weighted tardiness of all neighbors. For this purpose, we have extended the procedure *lpath* given for the JSP in [11] to cope with both setup times and weighted tardiness. This procedure is termed *lpathSWT* and it is shown in Algorithm 1.

Remember that each task t has N tails denoted by $q_t^1 \dots q_t^N$. For each $i = 1 \dots N$, *lpathSWT* estimates the cost of the longest path from node *start* to each node end_i through the node v or the node w, and then estimates the weighted tardiness of the neighboring schedule from the estimations of these paths. It's easy to prove that *lpathSWT* produces a lower bound for the weighted tardiness when using N_1^S.

The makespan estimation algorithm *lpath* is very accurate and very efficient. However, *lpathSWT* is much more time consuming as it calculates N tails for each operation. Moreover, experiments conclude that weighted tardiness estimation is much less accurate than makespan estimation. We have conducted a series of experiments in several instances, generating 3 millions of neighbors for each instance, and for 81.56% of neighbors the makespan estimation coincided

Require: A sequence of operations (w, v) as they appear after a move
Ensure: A estimation of the weighted tardiness of the resulting schedule
$TotalEst = 0$;
$r'_w = \max\{r_{PJ_w} + p_{PJ_w}, r_{PM_w} + p_{PM_w} + S_{PM_w w}\}$;
$r'_v = \max\{r_{PJ_v} + p_{PJ_v}, r'_w + p_w + S_{wv}\}$;
for $i = 1$ to N **do**
$q'^i_v = \max\{q^i_{SJ_v} + p_{SJ_v}, q^i_{SM_v} + p_{SM_v} + S_{vSM_v}\}$;
$q'^i_w = \max\{q^i_{SJ_w} + p_{SJ_w}, q'^i_v + p_v + S_{wv}\}$;
$PartialEst = \max\{r'_w + p_w + q'^i_w\}, \{r'_v + p_v + q'^i_v\}$;
$TotalEst = TotalEst + (\max((PartialEst - d_i), 0) * w_i$;
return $TotalEst$;

Alg. 1. Procedure *lpathSWT*

with the actual value, but for weighted tardiness this value drops to 51.37% of the cases.

Other authors, for example Mati et al. in [7] or Essafi et al. in [3] opted to use more accurate estimations (they report results with exact estimations from 57% to 76%, depending on the particular instance). However, their estimation procedure is more time consuming as the complexity goes up from $O(1)$ to $O(N)$ for each path, where N is the number of jobs.

For these reasons, we have opted to evaluate the actual weighted tardiness when the neighbor's estimation is lower than the actual weighted tardiness of the original schedule. So, the use of *lpathSWT* allows the algorithm to discard a lot of neighbors in a very fast manner. Some preliminary results have shown that the improvement achieved in this way makes up the time consumed by far.

5 Experimental Study

The purpose of the experimental study is to compare $GA+TS$ with other state-of-the-art algorithms. Firstly, we compare our algorithm with the GLS algorithm proposed in [3] and the MDL algorithm proposed in [7] to solve the JSP. We experimented across the 22 instances of size 10×10 proposed by Singer and Pinedo in [9] (ABZ5, ABZ6, LA16 to LA24, MT10, and ORB01 to ORB10). Weights and due dates are defined as follows: the first 20% of the jobs have a weight 4 (very important jobs), the next 60% have weight 2 (moderately important jobs) and the remaining jobs have weight 1 (not important jobs). The due date d_i for each job i is defined in this way:

$$d_i = f * \sum_{j=1}^{M} p_{ij},$$

where f is a parameter that controls the tightness of the due dates. In this benchmark three values are considered: $f = 1.3$, $f = 1.5$ and $f = 1.6$.

The algorithm proposed by Essafi et al. is implemented in C++ and the experiments are carried out in a PC with a 2.8 GHz processor and 512 MB RAM,

giving a maximum runtime of 18 seconds per run. The local search proposed by Mati et al. runs in a Pentium with a 2.6 GHz processor, and they use a maximum runtime of 18 seconds too. $GA+TS$ runs in a Windows XP in a Intel Core 2 Duo at 2.66GHz with 2Gb RAM. We choose the parameters /58/70/50/ (/GA population/GA generations/maximum number of iterations without improvement for TS/) for $GA + TS$ to obtain a similar runtime.

Table 1 summarizes the results of these experiments; 10 trials were done for each instance and the average weighted tardiness of the 10 solutions and the number of times that the best known solution (BKS) was found are reported, "-" indicates that BKS is reached in all 10 trials.

Table 1. Results from GLS, MDL and $GA+TS$ across Singer and Pinedo's instances

	$f = 1.3$				$f = 1.5$				$f = 1.6$			
Inst.	BKS	GLS	MDL	GA+TS	BKS	GLS	MDL	GA+TS	BKS	GLS	MDL	GA+TS
ABZ5	1403	-	1414(2)	1412(7)	69	-	-	-	0	-	-	-
ABZ6	436	-	-	-	0	-	-	-	0	-	-	-
LA16	1169	1175(9)	-	-	166	-	-	166(9)	0	-	-	-
LA17	899	-	-	-	260	-	-	-	65	-	-	-
LA18	929	933(6)	934(6)	-	34	-	-	-	0	-	-	-
LA19	948	949(8)	-	998(4)	21	-	-	-	0	-	-	-
LA20	805	-	-	834(3)	0	-	-	0(7)	0	-	-	-
LA21	463	-	-	-	0	-	-	-	0	-	-	-
LA22	1064	1087(1)	**1077**(**4**)	1079(3)	196	-	-	-	0	-	-	-
LA23	835	**865**(**2**)	**865**(**2**)	870(1)	2	-	-	-	0	-	-	-
LA24	835	-	-	-	82	**86**(**3**)	86(2)	88(1)	0	-	-	-
MT10	1363	1372(9)	-	1383(9)	394	-	-	-	141	162(1)	152(1)	**145**(**6**)
ORB1	2568	2651(0)	2639(3)	**2578**(**8**)	1098	1159(6)	1247(0)	-	566	688(0)	653(0)	**592**(**2**)
ORB2	1408	1444(2)	**1426**(**3**)	**1426**(**3**)	292	-	-	-	44	-	-	-
ORB3	2111	2170(4)	**2158**(1)	2160(**6**)	918	943(4)	961(0)	**939**(**4**)	422	514(1)	463(4)	**434**(**7**)
ORB4	1623	1643(**7**)	1690(2)	**1632**(6)	358	394(8)	435(4)	-	66	78(8)	68(8)	-
ORB5	1593	1659(1)	1775(0)	**1615**(**7**)	405	-	415(8)	428(7)	163	181(0)	**176**(**3**)	**176**(**3**)
ORB6	1790	-	1793(9)	1854(5)	426	440(5)	437(5)	**430**(**8**)	28	-	-	-
ORB7	590	592(9)	-	-	50	55(8)	-	-	0	-	-	-
ORB8	2429	2522(0)	2523(0)	**2477**(**4**)	1023	1059(**7**)	1036(6)	**1033**(2)	621	669(0)	643(1)	**639**(**3**)
ORB9	1316	-	-	-	297	311(7)	299(9)	**302**(**9**)	66	83(7)	80(4)	-
ORB10	1679	**1718**(5)	1774(1)	1731(**6**)	346	**400**(**4**)	436(0)	430(2)	76	142(0)	117(0)	**82**(**5**)

$GA + TS$ was the only algorithm capable of reaching the BKS in at least one run for all 66 instances. Globally, $GA + TS$ obtains the BKS in 517 of the total 660 runs (78.3%), GLS in 443 (67.1%) and MDL in 458 (69.4%). Regarding the average weighted tardiness, we have run two t-tests with alpha level at 0.05 to compare $GA + TS$ against GLS and MDL. With p-values of 0.016 and 0.010 respectively, both tests showed that there is good evidence that the mean average weighted tardiness obtained by $GA + TS$ is lower than the obtained by the other methods. Overall, $GA + TS$ is quite competitive with the state-of-the-art algorithms in solving the classic JSP with weighted tardiness minimization.

In the second series of experiments, we compared $GA+TS$ with ILOG CPLEX CP Optimizer (CP) in solving the SDST-JSP across the BT-set proposed in [2]. We define due dates and weights as before. BT instances are divided in three groups depending on its size: small instances, t2-ps01 to t2-ps05, are 10×5, medium instances, t2-ps06 to t2-ps10, are 15×5, and large instances, t2-ps11 to t2-ps15, are 20×5. These instances verify the triangular inequality for setup times. $GA + TS$ was parameterized as /100/200/50/. CP was run setting the

option "Extended" for parameter "NoOverlapInferenceLevel" as the results in this case were slightly better. Both methods were run 30 times for each instance. Table (2) summarizes the results of these experiments: for each method and instance, the average value of the weighted tardiness is reported. The time taken by $GA + TS$ in a single run is given in the last column. CP was given this time plus 20% more in each run. As we can observe, $GA + TS$ reaches much better solutions than CP. On average, the weighted tardiness obtained by CP is 13.6% worse than that obtained by $GA + TS$. We have run a t-test with alpha level at 0.05, and with a p-value of 0.000 the test showed that there is strong evidence that the mean average weighted tardiness obtained by $GA + TS$ is lower than the obtained by CP.

Table 2. Results from $GA+TS$ and CP in solving SDST-JSP with weighted tardiness minimization on the BT-set

Inst.	$f = 1.3$		$f = 1.5$		$f = 1.6$		
	GA+TS	CP	GA+TS	CP	GA+TS	CP	T(s)
t2-ps01	4454	4994	3361	3911	2852	3506	47
t2-ps02	3432	4143	2674	2957	2301	2558	48
t2-ps03	4001	4609	3120	3560	2677	3143	51
t2-ps04	3732	4021	2890	3050	2539	2632	48
t2-ps05	3806	4445	2996	3532	2620	3038	45
t2-ps06	9941	10533	8238	8997	7436	8148	121
t2-ps07	9508	10552	8079	8875	7425	8642	122
t2-ps08	9902	10834	8360	9317	7624	8521	120
t2-ps09	9998	11569	8215	9697	7317	8813	124
t2-ps10	10569	11999	8745	9968	7914	8848	116
t2-ps11	23052	26169	20816	24132	19764	22909	229
t2-ps12	23158	25331	21119	22309	20106	21039	241
t2-ps13	24026	25729	21821	23548	20618	22279	244
t2-ps14	25416	27569	23051	25580	21892	24079	250
t2-ps15	25427	27144	23028	25450	22049	23919	237

6 Conclusions

Our study of SDST-JSP has demonstrated that metaheuristics such as genetic algorithms and tabu search are very efficient in solving complex scheduling problems. These techniques are flexible and robust, so as they can be adapted to the particular characteristics of a given problem. Here, we have seen how a solution designed to cope with makespan can be adapted to cope with weighted tardiness, which is well-known that is harder to optimize. Also, we have demonstrated that for weighted tardiness minimization, a specific solution based on specific knowledge of the problem domain can be much more efficient than a solution built on a general purpose solver.

As future work, we plan to consider other scheduling problems and different objective functions, even non-regular objective functions, i.e. objective functions that could be decreasing on the completion times of the operations, such as robustness or stability measures.

Acknowledgments. This research has been supported by the Spanish Ministry of Science and Innovation under research project MICINN-FEDER TIN2010-20976-C02-02 and by the Principality of Asturias under grant FICYT-BP07-109.

References

1. Artigues, C., Lopez, P., Ayache, P.D.: Schedule generation schemes for the job shop problem with sequence-dependent setup times: Dominance properties and computational analysis. Annals of Operations Research 138, 21–52 (2005)
2. Brucker, P., Thiele, O.: A branch and bound method for the general-job shop problem with sequence-dependent setup times. Operations Research Spektrum 18, 145–161 (1996)
3. Essafi, I., Mati, Y., Dauzère-Pérès, S.: A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Computers and Operations Research 35, 2599–2616 (2008)
4. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers (1997)
5. González, M.A., Vela, C.R., Varela, R.: Genetic Algorithm Combined with Tabu Search for the Job Shop Scheduling Problem with Setup Times. In: Mira, J., Ferrández, J.M., Álvarez, J.R., de la Paz, F., Toledo, F.J. (eds.) IWINAC 2009. LNCS, vol. 5601, pp. 265–274. Springer, Heidelberg (2009)
6. González, M.A., Vela, C., Varela, R.: A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In: Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling (ICAPS 2008), pp. 116–123. AAAI Press, Sidney (2008)
7. Mati, Y., Dauzere-Peres, S., Lahlou, C.: A general approach for optimizing regular criteria in the job-shop scheduling problem. European Journal of Operational Research (2011), doi:10.1016/j.ejor.2011.01.046
8. Matsuo, H., Suh, C., Sullivan, R.: A controlled search simulated annealing method for the general jobshop scheduling problem. Working paper 03-44-88, Graduate School of Business, University of Texas (1988)
9. Singer, M., Pinedo, M.: A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. IIE Transactions 30, 109–118 (1998)
10. Sun, X., Noble, J.: An approach to job shop scheduling with sequence-dependent setups. Journal of Manufacturing Systems 18(6), 416–430 (1999)
11. Taillard, E.: Parallel taboo search techniques for the job shop scheduling problem. ORSA Journal on Computing 6, 108–117 (1993)
12. Van Laarhoven, P., Aarts, E., Lenstra, K.: Job shop scheduling by simulated annealing. Operations Research 40, 113–125 (1992)
13. Vela, C.R., Varela, R., González, M.A.: Local search and genetic algorithm for the job shop scheduling problem with sequence dependent setup times. Journal of Heuristics 16, 139–165 (2010)