Job shop scheduling with setup times and maximal time-lags: A simple constraint programming approach

Diarmuid Grimes[1] and Emmanuel Hebrard[1,2]

[1] Cork Constraint Computation Centre & University College Cork, Ireland
{d.grimes|e.hebrard}@4c.ucc.ie
[2] LAAS-CNRS, Toulouse, France
hebrard@laas.fr

Abstract. In previous work we introduced a simple constraint model that combined generic AI strategies and techniques (weighted degree heuristic, geometric restarts, nogood learning from restarts) with naive propagation for job shop and open shop scheduling problems. Here, we extend our model to handle two variants of the job shop scheduling problem: job shop problems with setup times; and job shop problems with maximal time lags. We also make some important additions to our original model, including a solution guidance component for search. We show empirically that our new models often outperform the state of the art techniques on a number of known benchmarks for these two variants, finding a number of new best solutions and proving optimality for the first time on some problems. We provide some insight into the performance of our approach through analysis of the constraint weighting procedure.

1 Introduction

Scheduling problems have proven fertile research ground for constraint programming and other combinatorial optimization techniques. There are numerous such problems occurring in industry, and whilst relatively simple in their formulation - they typically involve only *Sequencing* and *Resource* constraints - they remain extremely challenging to solve. After such a long period as an active research topic (more than half a century back to Johnson's seminal work [18]) it is natural to think that methods specifically engineered for each class of problems would dominate approaches with a broader spectrum. However, it was recently shown [27, 15, 26] that generic SAT or constraint programming models can approach or even outperform state of the art algorithms for open shop scheduling and job shop scheduling. In particular, in a previous work [15] we introduced a constraint model that advantageously trades inference strength for brute-force search speed and adaptive learning-based search heuristics combined with randomized restarts and a form of nogood learning.

Local search algorithms are generally the most efficient approach for solving job shop scheduling problems. The best algorithms are based on tabu search [**?**], or on a CP/local search hybrid [29]. Pure CP approaches can also be efficient, especially when guided by powerful search strategies that can be thought of as metaheuristics [4]. The best CP approach uses inference from the *Edge-finding* algorithm [8, 22] and dedicated variable ordering heuristics such as *Texture* [3]. On the other hand, we take a minimalistic approach to modelling the problem. In particular, whilst most algorithms consider resource constraints as global constraints, devising specific algorithms to filter them, we simply decompose them into primitive disjunctive constraints ensuring that two tasks sharing a resource do not run concurrently. To this naive propagation framework, we combine slightly more sophisticated, although generic heuristics and restart policies. In this work, we have also incorporated the idea of solution guided search [4].

We showed recently that this approach can be very effective with respect to the state of the art. However, it is even more evident on variants of these archetypal problems where dedicated algorithms cannot be applied in a straightforward manner. In the first variant, running a task on a machine requires a setup time, dependent on the task itself, and also on the previous task that ran on the same machine. In the second variant, maximum time lags between the starting times of successive tasks of each job are imposed. In both cases, most approaches decompose the problem into two subproblems, for the former the traveling salesman problem with time windows [1, 2] is used, while the latter can be decomposed into sequencing and timetabling subproblems [10]. On the other hand, our approach can be easily adapted to handle these additional constraints. Indeed, it found a number of new best solutions and proved optimality for the first time on some instances from a set of known benchmarks.

It may appear surprising that such a method, not reliant on domain specific knowledge, and whose components are known techniques in discrete optimization, could be so effective. We therefore devised some experiments to better understand how the key component of our approach, the constraint weighting, affects search on these problems. These empirical results reveal that although the use of constraint weighting is generally extremely important to our approach, it is not always so. In particular on *no-wait* job shop scheduling problems (i.e. problems with maximal time-lag of 0 between tasks), where our approach often outperforms the state of the art, the weight even seems to be detrimental to the algorithm.

In Section 2, we describe our approach. In Section 3, after outlining the experimental setup, we provide an experimental comparison of our approach with the state-of-the-art on standard benchmarks for these two problems. Finally we detail the results of our analysis of the impact of weight learning in these instances in Section 4.

2 A Simple Constraint Programming Approach

In this section we describe the common ground of constraint models we used to model the variants of JSP tackled in this paper. We shall consider the minimization of the total makespan (C_{max}) as the objective function in all cases.

2.1 Job Shop Scheduling Problem

An $n \times m$ job shop problem (JSP) involves a set of nm *tasks* $\mathcal{T} = \{t_i \mid 1 \leq i \leq nm\}$, partitioned into n *jobs* $\mathcal{J} = \{J_x \mid 1 \leq x \leq n\}$, that need to be scheduled on m *machines* $\mathcal{M} = \{M_y \mid 1 \leq y \leq m\}$. Each job $J_x \in \mathcal{J}$ is a set of m tasks $J_x =$

$\{t_{(x-1)*m+y} \mid 1 \leq y \leq m\}$. Conversely, each machine $M_y \in \mathcal{M}$ denotes a set of n tasks (to run on this machine) such that: $\mathcal{T} = (\bigcup_{1 \leq x \leq n} J_x) = (\bigcup_{1 \leq y \leq m} M_y)$.

Each task t_i has an associated duration, or processing time, p_i. A *schedule* is a mapping of tasks to time points consistent with: sequencing constraints which ensure that the tasks of each job run in a predefined order; and *resource* constraints which ensure that no two tasks run simultaneously on any given machine.

In this paper we consider the standard objective function defined as the minimization of the *makespan* C_{max}, that is, the total duration to run all tasks. If we identify each task t_i with its start time in the schedule, the job shop scheduling problem (JSP) can thus be written as follow:

$$(JSP)\ \ minimise\, C_{max}\ \text{ subject to :}$$

$$C_{max} \geq t_i + p_i \qquad \forall t_i \in \mathcal{T} \tag{2.1}$$

$$t_i + p_i \leq t_{i+1} \qquad \forall J_x \in \mathcal{J},\ \forall t_i, t_{i+1} \in J_x \tag{2.2}$$

$$t_i + p_i \leq t_j \ \vee\ t_j + p_j \leq t_i \qquad \forall M_y \in \mathcal{M},\ t_i \neq t_j \in M_y \tag{2.3}$$

2.2 Constraint model

The objective to minimise (total makespan) is represented by a variable C_{max} and the start time of each task t_i is represented by a variable $t_i \in [0, \ldots, max(C_{max}) - p_i]$. Next, for every pair of tasks t_i, t_j sharing a machine, we introduce a Boolean variable b_{ij} which represents the relative ordering between t_i and t_j. A value of 0 for b_{ij} means that task t_i precedes task t_j, whilst a value of 1 stands for the opposite ordering. The variables t_i, t_j and b_{ij} are linked by the following constraint:

$$b_{ij} = \begin{cases} 0 \Leftrightarrow t_i + p_i \leq t_j \\ 1 \Leftrightarrow t_j + p_j \leq t_i \end{cases}$$

Bounds consistency (BC) is maintained on these constraints. A *range support* of a constraint $C(x_1, \ldots, x_k)$ is an assignment of $\{x_1, \ldots, x_k\}$ satisfying C, and where the value assigned to each variable x_i is an integer taken in the interval $[min(x_i)..max(x_i)]$. A constraint $C(x_1, \ldots, x_k)$ is *bounds consistent* (BC) iff, for every variable x_i in the scope of C, $min(x_i)$ and $max(x_i)$ have a range support. Here, the scope of the constraint involves three variables, b_{ij}, t_i and t_j, therefore BC can be achieved in constant time for a single constraint, by applying simple rules. For n jobs and m machines, this model involves $nm(n-1)/2$ Boolean variables and as many ternary disjunctive constraints. Using an AC3 type constraint queue, the wort case time complexity for achieving bounds consistency on the whole network is therefore $O(C_{max}{*}nm(n{-}1)/2)$ since in the worst case bounds can be reduced by one unit at a time. For instance, consider three tasks t_i, t_j and t_k such that $p_i = p_j = p_k = 1$ and assume that $b_{ij} = b_{jk} = 0$ (hence $t_i \leq t_j \leq t_k$). Moreover, suppose that the domain of b_{ik} is reduced to the value 1, so that the cycle is closed. Since the domains are reduced by a constant amount at each propagation, the number of iterations necessary to obtain a failure is in $O(C_{max})$. However, it rarely reaches this bound in practice. Observe, moreover, that artificially increasing the size of the instance by a fixed amount will not affect the propagation loop as long as the durations increase proportionally to the horizon.

2.3 Search Strategy

We use the model described above in two different ways. Initially the lower bound on C_{max} is set to the duration of the longest job/machine, whilst the upper bound ub is initialised by a greedy algorithm in one case (Section 3.1), or by simply summing the durations of every task (Section 3.2). Since this starting upper bound is often very poor, especially in the latter case, we reduce the gap by performing a dichotomic search. We repeatedly solve the decision problem with a makespan fixed to $\frac{ub+lb}{2}$, updating lb and ub accordingly, until they have collapsed. Each dichotomic step has a fixed time cutoff, if the problem is unsolved the lb is updated, although not stored as the best proven lb. Moreover, we observed that in many cases, the initial upper bound is so overestimated that it helps to slightly bias the dichotomic pivot toward lower values until a first solution is found.

If the problem has not been solved to optimality during the dichotomic search, we perform a branch & bound search with the best makespan from the dichotmic search as our upper bound, and the best proven lb as our lower bound. Branch & bound search is performed until either optimality is proven or an overall cutoff is reached.

Branching: Instead of searching by assigning a starting time to a single value on the left branches, and forbidding this value on the right branches, it is common to branch on *precedences*. An unresolved pair of tasks t_i, t_j is selected and the constraint $t_i + p_i \leq t_j$ is posted on the left branch whilst $t_j + p_j \leq t_i$ is posted on the right branch. In our model, branching on the Boolean variables precisely simulates this branching strategy and thus significantly reduces the search space. Indeed, the existence of a partial ordering of the tasks (compatible with start times and durations, and such that its projection on any job or machine is a total order) is equivalent to the existence of a solution. In other words, if we successfully assign all Boolean variables in our model, the existence of a solution is guaranteed. Assigning each task variable to its lowest domain value gives the minimum C_{max} for this solution.

Variable Selection: We use the domain/weighted-degree heuristic [5], which chooses the variable minimising the ratio of current domain size to total weight of its neighboring constraints (initialised to 1). A constraint's weight is incremented by one each time the constraint causes a failure during search. It is important to stress that the behaviour of this heuristic is dependent on the modelling choices. Indeed, two different, yet logically equivalent, sets of constraints may distribute the weights differently. In this model, every constraint involves at most one search variable. Moreover, the relative light weight of the model allows the search engine to explore many more nodes than would a method relying on stronger inference, thus learning weights quicker.

However, at the start of the search, this heuristic is completely uninformed since every Boolean variable has the same domain size and the same degree. We therefore use an augmented version of the heuristic, where, instead of the domain size of b_{ij}, we use the domain size of the two associated task variables t_i, t_j. We denote $dom(t_i) = (max(t_i) - min(t_i) + 1)$ the domain size of task t_i, that is, the residual time windows of its starting time. Moreover, we denote $w(i, j)$ the number of times the search failed while propagating the constraint between t_i, t_j and b_{ij}. We choose the variable minimising the sum of the tasks' domain size divided by the weighted degree:

$$\frac{dom(t_i) + dom(t_j)}{w(i,j)} \tag{2.4}$$

Moreover, one can also use the weighted degree associated with the task variables. Let $\Gamma(t_j)$ denote the set of tasks sharing a resource with t_j. We call $w(t_j) = \sum_{t_i \in \Gamma(t_j)} w(i,j)$ the sum of the weights of every ternary disjunctive constraint involving t_j. Now we can define an alternative variable ordering as follows:

$$\frac{dom(t_i) + dom(t_j)}{w(t_i) + w(t_j)} \tag{2.5}$$

We refer to these heuristics as $tdom/bweight$ and $tdom/tweight$, $tdom$ refers to the sum of the domain sizes of the tasks associated with the Boolean variable, and $bweight$ ($tweight$) refers to the weighted degree of the Boolean (tasks). Ties were broken randomly.

Value Selection: Our value ordering is based on the solution guided approach (SGMPCS) proposed by Beck for JSPs [4]. This approach involves using previous solution(s) as guidance for the current search, intensifying search around a previous solution in a similar manner to i-TSAB [21]. In SGMPCS, a set of elite solutions is initially generated. Then, at the start of each search attempt, a solution is randomly chosen from the set and is used as a value ordering heuristic for search. When an improving solution is found, it replaces the solution in the elite set that was used for guidance. The logic behind this approach is its combination of intensification (through solution guidance) and diversification (through maintaining a set of diverse solutions).

Interestingly Beck found that the intensification aspect was more important than the diversification. Indeed, for the JSPs studied, there was little difference in performance between an elite set of size 1 and larger elite sets (although too large a set did result in a deterioration in performance). We use an elite set of 1 for our approach, i.e. once an initial solution has been found this solution is used, and updated, throughout our search.

Furthermore, up until the first solution is found during dichotomic search, we use a value ordering working on the principle of best *promise* [11]. The value 0 for b_{ij} is visited first iff the domain reduction directly induced by the corresponding precendence ($t_i + p_i \leq t_j$) is less than that of the opposite precedence ($t_j + p_j \leq t_i$).

Restart policy: It has previously been shown that randomization and restarts can greatly improve systematic search performance on combinatorial problems [12]. We use a geometric restarting strategy [28] with random tie-breaking. The geometric strategy is of the form $s, sr, sr^2, sr^3, ...$ where s is the base and r is the multiplicative factor. In our experiments the base was 64 failures and the multiplicative factor was 1.3. We also incorporate the nogood recording from restarts strategy of Lecoutre et al. [19], where nogoods are generated from the final search state when the cutoff has been reached. To that effect, we use a global constraint which essentially simulates the unit propagation procedure of a SAT solver. After every restart, for every minimal subset of decisions leading to a failure, the clause that prevents exploring the same path on subsequent restarts is added to the base. This constraint is not weighted when a conflict occurs.

3 Experimental Evaluation

We compare our model with state-of-the-art solvers (both systematic and non-sysytematic) on 2 variants of the JSP, job shop problems with sequence dependent setup times and job shop problems with time lags. All our experiments were run on an Intel Xeon 2.66GHz machine with 12GB of ram on Fedora 9. Due to the random component of our algorithm, each instance was solved ten times and we report our results in terms of both best and average makespan found per problem. Each algorithm run on a problem had an overall time limit of 3600s.

The number of algorithms we need to compare against makes it extremely difficult to run all experiments on a common setting.[3] We therefore decided to compare with the results taken from their associated papers. Since they were obtained on different machines with overall cutoffs based on different criteria, a direct comparison of cpu time is not possible. However, an improvement on the best known makespan is sufficient to observe that our approach is competitive. Therefore, we focus our analysis of the results on the objective value (although we do include average cpu time over the 10 runs for problems where we proved optimality).

3.1 Job Shop Scheduling Problem with Sequence Dependent Setup-times

A job shop problem with sequence-dependent setup times, involves, as in a regular JSP, m machines and nm tasks, partitioned into n Jobs of m tasks. As for a JSP, the tasks have to run in a predefined order for every job and two tasks sharing a machine cannot run concurrently, that is, the starting times of these tasks should be separated by at least the duration of the first. However, for each machine and each pair of tasks running on this machine, the machine needs to be setup to accommodate the new task. During this setup the machine must stand idle. The duration of this operation depends on the sequence of tasks, that is, for every pair of tasks (t_i, t_j) running on the same machine we are given the setup time $s(i, j)$ for t_j following t_i and the setup time $s(j, i)$ for t_i following t_j. The setup times respect the triangular inequality, that is $\forall i, j, k \; s(i, j) + s(j, k) \geq s(i, k)$. The objective is to minimise the *makespan*. More formally:

$$\begin{aligned}
(SDST - JSP) \;\; minimise \, C_{max} & \;\; \text{subject to :} \\
C_{max} \geq t_i + p_i & \qquad \forall t_i \in \mathcal{T} & (3.1) \\
t_i + p_i \leq t_{i+1} & \qquad \forall J_x \in \mathcal{J}, \; \forall t_i, t_{i+1} \in J_x & (3.2) \\
t_i + p_i + s_{i,j,y} \leq t_j \; \vee \; t_j + p_j + s_{j,i,y} \leq t_i & \qquad \forall M_y \in \mathcal{M}, \; \forall t_i \neq t_j \in M_y & (3.3)
\end{aligned}$$

State of the art: This problem represents a challenge for CP and systematic approaches in general, since the inference from the Edge-finding algorithm is seriously weakened as it cannot easily take into account the setup times. Therefore there are two main approaches to this problem. The first by Artigues *et al.* [1] (denoted AF08 in Table 1) tries to adapt the reasoning for simple unary resources to unary resources with setup

[3] The code may be written for different OS, not publicly available, or not open source.

Table 1: SDST-JSP: Comparison vs state-of-the-art (best & mean C_{max}, 10 runs).

Instance	AF08	BSV08	GVV08		GVV09		*tdom/bweight*		
	Best	Best	Best	Avg	Best	Avg	Best	Avg	Time
t2-ps01	**798**	**798**	**798**	**798**			**798**	798.0	0.1
t2-ps02	**784**	**784**	**784**	**784**			**784**	784.0	0.2
t2-ps03	**749**	**749**	**749**	**749**			**749**	749.0	0.2
t2-ps04	**730**	**730**	**730**	**730**			**730**	730.0	0.1
t2-ps05	**691**	693	**691**	692			**691**	691.0	0.1
t2-ps06	**1009**	1018	1026	1026			**1009**	1009.0	20.3
t2-ps07	**970**	1003	**970**	971			**970**	970.0	46.1
t2-ps08	**963**	975	**963**	966			**963**	963.0	86.1
t2-ps09	1061	**1060**	**1060**	**1060**			**1060***	1060.0	1025.1
t2-ps10	**1018**	**1018**	**1018**	**1018**			**1018**	1018.0	11.0
t2-ps11	1494	1470	**1438**	1439	**1438**	1441	1443	1463.6	-
t2-ps12	1381	1305	**1269**	1291	**1269**	1277	**1269**	1322.2	-
t2-ps13	1457	1439	**1406**	1415	1415	1416	1415	1428.8	-
t2-ps14	1483	1485	**1452**	1489	**1452**	1489	**1452**	1470.5	-
t2-ps15	1661	1527	**1485**	1502	**1485**	1496	1486	1495.8	-
t2-pss06		1126					**1114***	1114.0	600.9
t2-pss07		1075					**1070***	1070.0	274.1
t2-pss08		1087					**1072***	1073.0	-
t2-pss09		1181					**1161***	1161.0	-
t2-pss10		1121					**1118***	1118.0	47.2
t2-pss11		1442					**1412***	1425.9	-
t2-pss12		1290			**1258**	1266	1269	1287.6	-
t2-pss13		1398			**1361**	1379	1365	1388.0	-
t2-pss14		1453					**1452***	1453.0	-
t2-pss15		1435					**1417***	1427.4	-

times. The approach relies on solving a TSP with time windows to find the shortest permutation of tasks, and is therefore computationally expensive.

The second type of approach relies on metaheuristics. Balas *et al.* [2] proposed combining a shifting bottleneck algorithm with guided local search (denoted BSV08 in Table 1[4]), where the problem is also decomposed into a TSP with time windows. Hybrid genetic algorithms have also been proposed by González *et al.* for this problem, firstly a hybrid GA with local search [13] and more recently GA combined with tabu search [14] (denoted GVV08 and GVV09 *resp.* in Table 1). For both GA hybrids, the problem is modeled using the disjunctive graph representation.

Specific Implementation Choices: Our model is basically identical to the generic scheduling model introduced in Section 2. However, the setup time between two tasks is added to the duration within the disjunctive constraints. That is, given two tasks t_i and t_j sharing a machine, let $s_{i,j}$ (resp. $s_{j,i}$) be the setup time for the transition between t_i and t_j (resp. between t_j and t_i), we replace the usual disjunctive constraint with:

$$b_{ij} = \begin{cases} 0 \Leftrightarrow t_i + p_i + s_{i,j} \leq t_j \\ 1 \Leftrightarrow t_j + p_j + s_{j,i} \leq t_i \end{cases}$$

Evaluation: Table 1 summarizes the results of the state-of-the-art and our approach on a set of benchmarks proposed by Brucker and Thiele [7]. The problems are grouped based on the number of jobs and machines (*n*x*m*), *01-05 are of size 10x5, *06-10 are

[4] Results for t2-pss-*06-11 and 14-15 are from http://www.andrew.cmu.edu/user/neils/tsp/outt2.txt

of size 15x5, while *11-15 are of size 20x5. Each step of the dichotomic search had a 30 second cutoff, the search heuristic used was *tdom/bweight*. We use the following notation for Table 1 (we shall reuse it for Tables 3 and 4): underlined values denote the fact that optimality was proven, bold face **values** denote the best value achieved by any method and finally, values* marked with a star denote instances where our approach improved on the best known solution or built the first proof of optimality. We also include the average time over the 10 runs when optimality was proven (a dash means optimality wasn't proven before reaching the 1 hour cutoff).

We report the first proof of optimality for four instances (`t2-ps09, t2-pss06, t2-pss07, t2-pss10`) and 8 new upper bounds for t2-pss* instances (however it should be noted that there is no comparison available for GVV09 on these 8 instances). In general, our approach is competitive with the state-of-the-art (GVV09) and outperforms both dedicated systematic and non-systematic solvers.

3.2 Job Shop Scheduling Problem with Time Lags

An $n \times m$ job shop problem with time lags (JTL) involves the same variables and constraints as a JSP of the same order. However, there is an additional upper bound on the time lag between every pair of successive tasks in every job. Let l_i denote the maximum amount of time allowed between the completion of task t_i and the start of task t_j. More formally:

$$(TL - JSP) \;\; minimise\, C_{max} \;\; \text{subject to :}$$

$$C_{max} \geq t_i + p_i \qquad \forall t_i \in \mathcal{T} \tag{3.4}$$

$$t_i + p_i \leq t_{i+1} \qquad \forall J_x \in \mathcal{J},\, \forall t_i, t_{i+1} \in J_x \tag{3.5}$$

$$t_{i+1} - (p_i + l_i) \leq t_i \qquad \forall J_x \in \mathcal{J},\, \forall t_i, t_{i+1} \in J_x \tag{3.6}$$

$$t_i + p_i \leq t_j \;\vee\; t_j + p_j \leq t_i \qquad \forall M_y \in \mathcal{M},\, \forall t_i \neq t_j \in M_y \tag{3.7}$$

This type of constraint arises in many situations. For instance, in the steel industry, the time lag between the heating of a piece of steel and its moulding should be small. Similarly when scheduling chemical reactions, the reactives often cannot be stored for a long period of time between two stages of a process to avoid interactions with external elements. This problem was not only studied within the steel and chemical industries [24] but also in the food industry and the pharmaceutical industry.

State of the art: Caumond *et al.* introduced in 2008 a genetic algorithm able to deal with general time lag constraints [9]. However most of the algorithms introduced in the literature have been designed for a particular case of this problem: the *no-wait* job shop. In this case, the maximum time-lag is null, i.e. each task of a job must start directly after its preceding task has finished.

For the no-wait job shop problem, the best methods are a tabu search method by Schuster (TS [25]), another metaheuristic introduced by Framinian and Schuster (CLM [10]) and a hybrid constructive/tabu search algorithm introduced by Bożejko and Makuchowski in 2009 (HTS [6]). We report the best results of each paper. It should be noted that for HTS, the authors reported two sets of results, the ones we report for the "hard" instances were "without limit of computation time".

Table 2: Results summary for JTL- and NW-JSP.

(a) JTL-JSP: C_{max} & Time.

Instance Sets	CLT		$tdom/bweight$	
	C_{max}	Time	C_{max}	Time
car[5-8]_0_0,5	**7883.25**	322.19	**7883.25**	2.16
car[5-8]_0_1	**7731.25**	273.75	**7731.25**	4.16
car[5-8]_0_2	**7709.25**	297.06	**7709.25**	6.31
la[06-08]_0_0,5	1173.67	2359.33	**980.00**	2044.77
la[06-08]_0_1	1055.33	1870.92	**905.33**	2052.41
la[06-08]_0_2	1064.33	1853.67	**904.67**	2054.81

(b) NW-JSP: Summary of APRD per problem set

Instance	TS	HTS	CLM	CLT	tdom/ twdeg	tdom
ft	-8.75	**-10.58**			**-10.58**	-9.79
abz	-20.77	-25.58			**-25.89**	-25.1
orb	2.42	0.77	1.44		**0.00**	**0.00**
la01-10	4.43	1.77	3.31	4.53	**0.00**	**0.00**
la11-20	9.52	-5.40	5.14	29.14	-6.32	**-6.36**
la21-30	-33.93	**-39.96**	-34.62		-39.85	-39.04
la31-40	-36.69	**-42.39**	-36.87		-41.65	-40.36
swv01-10	-34.41	**-37.22**	-34.39		-36.88	-35.33
swv11-20	-40.62	**-42.25**			-39.17	-33.87
yn	-34.87	**-41.84**			-38.78	-39.03

Specific Implementation Choices: The constraint to represent time lags between two tasks of a job are simple precedences in our model. For instance, a time lag l_i between t_i and t_{i+1}, will be represented by the following constraint: $t_{i+1} - (p_i + l_i) \leq t_i$.

Although our generic model was relatively efficient on these problems, we made a simple improvement for the no-wait class based on the following observation: if no delay is allowed between any two consecutive tasks of a job, then the start time of every task is functionally dependent on the start time of any other task in the job. The tasks of each job can thus be viewed as one block. In other words we really need only one task in our model to represent all the tasks of a job. We therefore use only n variables standing for the jobs: $\{J_x \mid 1 \leq x \leq n\}$.

Let h_i be the total duration of the tasks coming before task t_i in its job. That is, if job $J = \{t_1, \dots, t_m\}$, we have: $h_i = \sum_{k<i} p_k$. For every pair of tasks $t_i \in J_x, t_j \in J_y$ sharing a machine, we use the same Boolean variables to represent disjuncts as in the original model, however linked by the following constraints:

$$b_{ij} = \begin{cases} 0 \Leftrightarrow J_x + h_i + p_i - h_j \leq J_y \\ 1 \Leftrightarrow J_y + h_j + p_j - h_i \leq J_x \end{cases}$$

Notice that while the variables and constants are different, these are still exactly the same ternary disjuncts used in the original model.

The no-wait job shop scheduling problem can therefore be reformulated as follows, where the variables $J_1, \dots, J_n$ represent the start time of the jobs, $J_{x(i)}$ stands for the job of task t_i, and $f(i,j) = h_i + p_i - h_j$.

$$(NW-JSP) \;\; minimise \, C_{max} \;\; \text{subject to :}$$

$$C_{max} \geq J_x + \sum_{t_i \in J_x} p_i \qquad \forall J_x \in \mathcal{J} \tag{3.8}$$

$$J_{x(i)} + f(i,j) \leq J_{x(j)} \vee J_{x(j)} + f(j,i) \leq J_{x(i)} \qquad \forall M_y \in \mathcal{M},\; t_i, t_j \in M_y \tag{3.9}$$

Evaluation: On general JTL problems, it is difficult to find comparable results in the literature. To the best of our knowledge, the only one available is the genetic algorithm by Caumond *et al.* [9] that we shall denote CLT. In Table 2a, we report the results from our model on the instances used in that paper, where instances are grouped based on type (car (4 instances) / la (3 instances)) and maximum time lag (0.5 / 1 / 2).

For the no-wait job shop problem, we first present our results in terms of each solver's average percentage relative deviation (PRD) from the reference values given in [6] per problem set in Table 2b. The PRD is given by the following formula:

$$PRD = ((C_{Alg} - C_{Ref})/C_{Ref}) * 100 \tag{3.10}$$

where C_{Alg} is the best makespan found by the algorithm and C_{Ref} is the reference makespan for the instance given in [6]. There are 82 instances overall.

Interestingly, the search heuristic $tdom/tweight$ performed much better with our no-wait model than $tdom/bweight$, thus we report the results for this heuristic. This was somewhat surprising because this heuristic is less discriminatory as the task weights for a Boolean are the weights of the two jobs, which will be the same for all Booleans between these two jobs. Further investigation revealed that ignoring the weight yielded better results on a number of problems. Thus we also include the heuristic $tdom$.

Table 3: NW-JSP: Comparison vs state-of-the-art on *easy* instances (best & mean C_{max}, 10 runs).

Instance	Size	Ref	TS	HTS	CLM	CLT	$tdom/tweight$			$tdom$		
	nxm		Best	Best	Best	Best	Best	Avg	Time	Best	Avg	Time
ft06	6x6	73	**73**	**73**	**73**		**73**	73	0.01	**73**	73	0.02
ft10	10x10	1607	1620	**1607**	1619		**1607**	1607	4.08	**1607**	1607	2.49
abz5		2150	2233	2182			**2150**	2150	9.28	**2150**	2150	8.87
abz6		1718	1758	1760			**1718**	1718	1.25	**1718**	1718	0.71
orb01		1615	1663	1615	1646		**1615**	1615	1.65	**1615**	1615	1.45
orb02		1485	1555	1518	1518		**1485**	1485	1.16	**1485**	1485	1.12
orb03		1599	1603	**1599**	1603		**1599**	1599	4.22	**1599**	1599	3.10
orb04		1653	**1653**	**1653**	**1653**		**1653**	1653	1.56	**1653**	1653	1.11
orb05		1365	1415	1367	1371		**1365**	1365	3.91	**1365**	1365	4.43
orb06		1555	**1555**	1557	**1555**		**1555**	1555	0.31	**1555**	1555	0.26
orb07		689	706	717	706		**689**	689	6.10	**689**	689	3.34
orb08		1319	**1319**	**1319**	**1319**		**1319**	1319	2.22	**1319**	1319	2.12
orb09		1445	1535	1449	1515		**1445**	1445	1.02	**1445**	1445	0.68
orb10		1557	1618	1571	1592		**1557**	1557	4.55	**1557**	1557	4.78
la01	10x5	971	1043	975	1031	975	**971**	971	0.13	**971**	971	0.11
la02		937	990	975	**937**	**937**	**937**	937	0.24	**937**	937	0.19
la03		820	832	**820**	832	**820**	**820**	820	0.14	**820**	820	0.15
la04		887	889	889	889	911	**887**	887	0.28	**887**	887	0.17
la05		777	817	**777**	797	818	**777**	777	0.30	**777**	777	0.22
la06	15x5	1248	1299	**1248**	1256	1305	**1248**	1248	115.19	**1248**	1248	81.70
la07		1172	1227	**1172**	1253	1282	**1172**	1172	66.96	**1172**	1172	57.30
la08		1244	1305	1298	1307	1312	**1244**	1244	50.35	**1244**	1244	38.63
la09		1358	1450	1415	1451	1547	**1358**	1358	181.55	**1358**	1358	102.10
la10		1287	1338	1345	1328	1333	**1287**	1287	54.14	**1287**	1287	30.78
la16	10x10	1575	1637	**1575**	1637	1833	**1575**	1575	2.09	**1575**	1575	1.37
la17		1371	1430	1384	1389	1591	**1371**	1371	2.34	**1371**	1371	1.70
la18		1417	1555	**1417**	1555	1790	**1417**	1417	1.38	**1417**	1417	1.31
la19		1482	1610	1491	1572	1831	**1482**	1482	3.14	**1482**	1482	3.08
la20		1526	1705	**1526**	1580	1828	**1526**	1526	0.70	**1526**	1526	0.66

Our approach was better than the local search approaches on the smaller problem sets, and remained competitive on the larger problem sets. In Table 3 we provide results for the instances regarded as easy in [6], these had been proven optimal by Mascis [20]. We proved optimality on all these instances, in under 10s for most cases. It is of interest to note that $tdom$ was nearly always quicker than $tdom/tweight$ at proving optimality. In Table 4, we report results for the "hard" instances where our approach found an improving solution, and the first proofs of optimality for 10 (la12, la21-25, la36 and la38-40) of the 53 open problems.

Table 4: NW-JSP: Improvement on *hard* instances (best & mean C_{max}, 10 runs).

Instance	Size	Ref	TS	HTS	CLM	$tdom/tweight$			$tdom$		
	nxm		Best	Best	Best	Best	Avg	Time	Best	Avg	Time
swv06	20x15	3291	3502	3290	3291	**3278***	3378.0	-	3391	3500.4	-
la11	20x5	2821	1737	1621	1714	**1619***	1646.9	-	1622	1632.2	-
la12		2434	1550	1434	1507	**1414**	1432.7	-	**1414***	1414.0	2892.37
la14		2662	1771	1610	1773	**1578***	1628.5	-	**1578***	1611.1	-
la15		2765	1808	1686	1771	**1679***	1693.2	-	1681	1691.9	-
la21	15x10	2092	2242	**2030**	2149	**2030**	2030.0	-	**2030***	2030.0	579.69
la22		1928	2008	**1852**	1979	**1852**	1854.3	-	**1852**	1852.0	1013.45
la23		2038	2093	**2021**	2038	**2021**	2033.2	-	**2021**	2021.0	1160.13
la24		2061	2061	**1972**	2133	**1972**	1982.7	-	**1972**	1972.0	1128.55
la25	20x10	2034	2072	**1906**	2050	**1906**	1906.0	1336.92	**1906**	1906.0	218.60
la27		2933	2968	2675	2933	**2671***	2750.3	-	2675	2743.0	-
la36	15x15	2810	2993	**2685**	2810	**2685**	2715.5	-	**2685**	2685.0	1530.39
la37		3044	3171	**2831**	3161	2937	2974.0	-	**2831**	2930.4	-
la38		2726	2734	**2525**	2726	**2525**	2556.9	-	**2525**	2525.0	2898.77
la39		2752	2804	2687	2784	**2660***	2686.0	-	**2660***	2662.7	3564.28
la40		2838	2977	2580	2880	**2564***	2660.8	-	**2564***	2591.9	2879.08

4 Weight learning analysis

We have previously shown that the *weighted degree* is a key element of our approach [16]. In particular the gap in performance between $tdom/bwdeg$ and $tdom$ was quite large for open shop scheduling problems. Here we try to give a more precise characterization of the importance of learning weights, by gradually reducing the influence of these weights in the variable selection heuristic. We observe that the impact of the weights is very much problem-dependent. It is extremely important for job shop with setup times model and for the standard model for job shop with time lags. However, for the specific model for no-wait job shop problems, it can be detrimental in some cases.

4.1 Evaluation of weighted degree

In order to evaluate the effect of weight learning on search, we devised the following variable ordering heuristic, that we denote $tdom/(K + bweight)$, and that selects first the variable b_{ij} minimising the value of:

$$\frac{dom(t_i) + dom(t_j)}{w(i,j) + K} \tag{4.1}$$

Observe that when $K = 0$, this heuristic is equivalent to $tdom/(bweight)$, whereas, when K tends toward infinity, the weights become insignificant in the variable selection. For $K = \infty$ the next variable is selected with respect to $tdom$ only.

We can therefore tune the impact of the weights in the variable choice, by setting the constant K. As K increases, the role of the weights is increasingly restricted to a tie breaker. We selected a subset of instances small enough to be solved by $tdom/(\infty + bweight)$. For the selected subset of small instances, we ran each version of the heuristic ten times with different random seeds. We report the average cpu time across the ten runs in Table 5. When the run went over a one hour time cutoff, we report the deviation to the optimal solution (in percentage) instead.

Table 5: Weight evaluation: cpu-time or deviation to the optimal for increasing values of K.

Instance	$tdom/(K + bweight)$						
	$K = 0$	$K = 10$	$K = 100$	$K = 1000$	$K = 10000$	$K = 100000$	$K = \infty$
`t2-ps07`	26.55	**23.33**	26.67	41.60	77.27	403.90	*+12.9%*
`t2-ps08`	41.08	**35.85**	93.60	128.96	194.96	665.28	*+9.8%*
`t2-ps09`	971.83	956.63	**948.28**	957.85	1164.94	1649.19	*+8.8%*
`t2-ps10`	**13.04**	13.95	13.63	19.44	100.25	422.24	*+15.7%*
`la07_0_3`	*+0.0%*	*+0.0%*	*+0.0%*	*+0.0%*	*+0.0%*	*+0.0%*	*+5.8%*
`la08_0_3`	15.63	**12.45**	23.03	30.22	117.50	391.99	3098.87
`la09_0_3`	1.61	**0.51**	1.44	10.16	129.62	169.02	2115.98
`la10_0_3`	3.42	2.25	**0.41**	0.69	1.39	3.44	39.66
`la07_0_0`	1751.16	549.58	392.71	151.70	66.18	**49.67**	57.28
`la08_0_0`	2231.18	575.44	309.04	113.95	42.04	**35.74**	38.63
`la09_0_0`	2402.76	1291.29	691.96	407.68	147.73	**89.28**	102.03
`la10_0_0`	3274.86	833.28	214.51	53.75	26.85	**26.51**	30.82

For job shop with setup times, the best compromise is for $K = 10$. For very large values of K, the domain size of the tasks takes complete precedence on the weights, and the performance degrades. However, as long as the weights are present in the selection process, even simply as tie breaker, the cpu time stays within one order of magnitude from the best value for K. On the other hand, when the weights are completely ignored, the algorithm is not able to solve any of the instances. Indeed the gap to optimality is quite large, around 9% to 15%.

For job shop with time lags, the situation is a little bit different. As in the previous case, the best compromise is for $K = 10$ and the performance degrades slowly when K increases. However, even when the weights are completely ignored, the gap stays within a few orders of magnitude from the best case. Finally, for the no-wait job shop, we observe that the opposite is true. Rather than increasing with K, the cpu time actually *decreases* when K grows.

One important feature of a heuristic is its capacity to focus the search on a small subset of variables that would constitute a backdoor of the problem. It is therefore interesting to find out if there is a correlation between a high level of inequality in the weight distribution and the capacity to find small backdoors. We used the *Gini* coefficient to characterize the weight distribution. The Gini coefficient is a metric of inequality, used for instance to analyse distribution of wealth in social science.

The Gini coefficient is based on the *Lorenz* curve, mapping the cumulative proportion of income y of a fraction x of the poorest population. When the distribution is perfectly fair, the Lorenz curve is $y = x$. The Gini coefficient is the ratio of the area lying between the Lorenz curve and $x = y$, over the total area below $x = y$. We consider only search trees for unsatisfiable instances. In an ideal situation, when the search converges immediately toward a given set of variables from which a short proof of unsatisfiability can be extracted, the Gini coefficient of the weight distribution typically increases rapidly and monotonically. In Figure 1 we plot the Gini coefficient of the proofs for the instance `t2-ps07`; for an instance of random CSP with 100 variables, a domain size of 15, 250 binary constraints of tightness 0.53 uniformly distributed; and a pigeon holes instance. After each geometric restart, the Gini coefficient is computed and plotted against the current number of explored nodes. We observe that the weight distribution is quickly and significantly biased on the job shop instance. On the other hand, there is much less discrimination on the random CSP instance, where constraints are uniformly distributed, and almost no discrimination at all on the pigeon hole problem. We were

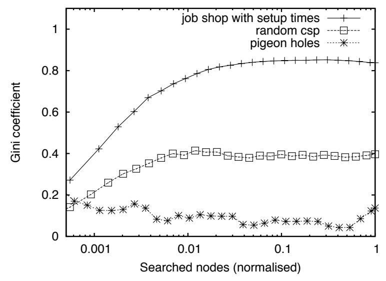
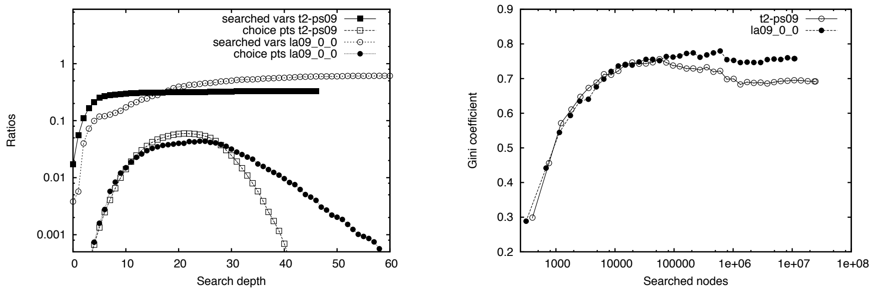


Fig. 1: Weight distribution bias: Gini coefficient over the (normalised) number of searched nodes.



(a) Search statistics

(b) Evolution of the Gini coef

Fig. 2: Search tree and weight distribution for `t2-ps09` and `la09_0_0`.

interested in checking if one could predict, from the fairness of the weight distribution, how beneficial the weighted degree heuristic is for the considered problem. However, when comparing two proofs that required a comparably large amount of search, but for which we showed that, in one case the weights are beneficial, and in the other case detrimental, it is in fact extremely difficult to differentiate the evolution of the coefficient. It took 11 million nodes to prove that $C_{max} = 1357$ is unsatisfiable for `la09_0_0` and 24 million nodes to prove that $C_{max} = 1059$ is unsatisfiable for `t2-ps09`. It is clear from the results in Table 5 however, that the weights helped in the latter case, whereas they did not in the former case. We report two statistics collected during search showing some clear differences: the ratio of (Boolean) variables that are selected at a choice point up to each depth in the search tree, over the total number of (Boolean) variables; the ratio of the number of choice points, that is nodes of the search tree, at each depth, over the total number of explored nodes.

Clearly for `t2-ps09`, where the weights are useful, the search is more focused on lower depth, and on a smaller ratio of variables. Indeed, the cumulative ratio of searched variables tops at 0.3 (See Figure 2a). On the other hand, for `la09_0_0`, even very deep in the tree, new choice points are opened (the ratio of choice points is more spread out), and they involve a large proportion of new variables (the cumulative number of searched variables increases almost linearly up to 0.6). The evolution of the Gini coefficient during search is, however, very similar in both cases (See Figure 2b).

One possibility is that the build up of contention is more important for the no wait problems due to the stronger propagation between tasks of the one job. Preliminary results suggest that both $tdom$ $tdom/bweight$ initially select Booleans between the same pair of jobs once a pair has been selected. The heuristics diverge when search backs up from deep in search, $tdom$ will still often choose Booleans from the same pair of jobs as the variable above the choice point, while the weights learnt deep in search may result in the heuristics that use $bweight$ and $tweight$ choosing variables associated with a different pair of jobs. Obviously, this effect will be stronger for $bweight$ as the weights are individual.

5 Conclusions

We have shown how our constraint model can be easily extended to handle two variants of the job shop scheduling problem. In both cases we found our approach to be competitive with the state-of-the-art, most notably in proving optimality on some of the open problems of both problem types.

Whereas it appeared to uniformly improve search efficiency for standard job shop and open shop scheduling problems, our analysis of constraint weighting revealed that it can actually be detrimental for some variants of these problems.

References

1. C. Artigues and D. Feillet. A branch and bound method for the job-shop problem with sequence-dependent setup times. *Annals OR*, 159(1):135–159, 2008.
2. Egon Balas, Neil Simonetti, and Alkis Vazacopoulos. Job shop scheduling with setup times, deadlines and precedence constraints. *J. of Scheduling*, 11(4):253–262, 2008.
3. J. C. Beck, A. J. Davenport, E. M. Sitarski, and M. S. Fox. Texture-Based Heuristics for Scheduling Revisited. In *AAAI'97*, pages 241–248, 1997.
4. J. Christopher Beck. Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling. *Journal of Artificial Intelligence Research*, 29:49–77, 2007.

5. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting Systematic Search by Weighting Constraints. In *ECAI'04*, pages 482–486, 2004.
6. W. Bozejko and M. Makuchowski. A fast hybrid tabu search algorithm for the no-wait job shop problem. *Computers & Industrial Engineering*, 56(4):1502–1509, 2009.
7. P. Brucker and O. Thiele. A branch and bound method for the general- shop problem with sequence-dependent setup times. *Operation Research Spektrum*, 18:145–161, 1996.
8. J. Carlier and E. Pinson. An Algorithm for Solving the Job-shop Problem. *Management Science*, 35(2):164–176, 1989.
9. Anthony Caumond, Philippe Lacomme, and Nikolay Tchernev. A memetic algorithm for the job-shop with time-lags. *Computers & OR*, 35(7):2331–2356, 2008.
10. J. M. Framinan and C. J. Schuster. An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach. *Computers & OR*, 33:1200–1213, 2006.
11. P. A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proc. Tenth European Conference on Artificial Intelligence-ECAI'92*, pages 31–35, 1992.
12. Carla P. Gomes, Bart Selman, and Henry Kautz. Boosting combinatorial search through randomization. In *AAAI'98*, pages 431–437, 1998.
13. M. A. González, C. R. Vela, and R. Varela. A new hybrid genetic algorithm for the job shop scheduling problem with setup times. In *ICAPS*, pages 116–123. AAAI, 2008.
14. M. A. González, C. R. Vela, and R. Varela. Genetic algorithm combined with tabu search for the job shop scheduling problem with setup times. In *IWINAC (1)*, volume 5601 of *Lecture Notes in Computer Science*, pages 265–274. Springer, 2009.
15. D. Grimes, E. Hebrard, and A. Malapert. Closing the Open Shop: Contradicting Conventional Wisdom. In *CP'09*, pages 400–408, 2009.
16. D. Grimes, E. Hebrard, and A. Malapert. Closing the Open Shop: Contradicting Conventional Wisdom on Disjunctive Temporal Problems. In *14th ERCIM International Workshop on Constraint Solving and Constraint Logic Programming (CSCLP'09)*, 2009.
17. A. Hodson, A. P. Muhlemann, and D. H. R. Price. A microcomputer based solution to a practical scheduling problem. *The Journal of the Operational Research Society*, 36(10):903–914, 1985.
18. S. M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68, 1954.
19. C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Nogood Recording from Restarts. In *IJCAI'07*, pages 131–136, 2007.
20. Alessandro Mascis and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143(3):498–517, 2002.
21. E. Nowicki and C. Smutnicki. An Advanced Tabu Search Algorithm for the Job Shop Problem. *Journal of Scheduling*, 8(2):145–159, 2005.
22. W. Nuijten. *Time and Resource Constraint Scheduling: A Constraint Satisfaction Approach*. PhD thesis, Eindhoven University of Technology, 1994.
23. W. H. M. Raaymakers and J. A. Hoogeveen. Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing. *European Journal of Operational Research*, 126(1):131 – 151, 2000.
24. C. Rajendran. A no-wait flowshop scheduling heuristic to minimize makespan. *The Journal of the Operational Research Society*, 45(4):472–478, 1994.
25. C. J. Schuster. No-wait job shop scheduling: Tabu search and complexity of problems. *Math Meth Oper Res*, 63:473–491, 2006.
26. A. Schutt, T. Feydy, P. J. Stuckey, and M. Wallace. Why cumulative decomposition is not as bad as it sounds. In *CP'09*, pages 746–761, 2009.
27. N. Tamura, A. Taga, S. Kitagawa, and M. Banbara. Compiling finite linear csp into sat. In *CP'06*, pages 590–603, 2006.
28. T. Walsh. Search in a Small World. In *IJCAI'99*, pages 1172–1177, 1999.
29. J-P. Watson and J. C. Beck. A Hybrid Constraint Programming / Local Search Approach to the Job-Shop Scheduling Problem. In *CPAIOR'08*, pages 263–277, 2008.