# Generic CSP Techniques for the Job-Shop Problem*

**Javier Larrosa**[1] and **Pedro Meseguer**[2]

[1]Universitat Politècnica de Catalunya, Dep. Llenguatges i Sistemes Informàtics
Jordi Girona Salgado 1-3, 08034 Barcelona, SPAIN
E-mail: larrosa@lsi.upc.es

[2]Institut d'Investigació en Intel.ligència Artificial, CSIC
Campus UAB, 08193 Bellaterra, SPAIN.
E-mail: pedro@iiia.csic.es

**Abstract**. The job-shop is a classical problem in manufacturing, arising daily in factories and workshops. From an AI perspective, the job-shop is a constraint satisfaction problem (CSP), and many specific techniques have been developed to solve it efficiently. In this context, one may believe that generic search and CSP methods (which typically are better understood and easier to develop, codify and maintain than specific approaches) are not appropriated for this problem. In this paper, we contradict this belief. We show that generic search and CSP algorithms and heuristics can be successfully applied to job-shop problem instances that have been considered challenging by the job-shop community. In particular, we use forward checking with support-based heuristics, a combination of a generic CSP algorithm with generic heuristics. We improve this combination replacing the depth-first search strategy of forward checking by a discrepancy-based schema, a generic search strategy recently developed. Our approach obtains similar results to specific approaches in terms of the number of solved problems, with reasonable requirements in computational resources.

## 1. Introduction

The *job-shop problem* involves the temporal sincronization of the production of *n* jobs on *m* machines. Each job is composed by a sequence of *m* operations; each operation has a duration and it requires the exclusive use of a machine for its duration. Each job has a release date and a due date between which it should be accomplished. A solution of this problem can be formulated in different ways, as optimization or decision problems. In this paper, we consider the *job-shop with non-relaxable time windows*, for which a solution is a temporal assignment of operations to machines in such a way that jobs are perfomed timely, satisfying the sequence of its operations and respecting that any machine is used by at most one operation at any time.

The job-shop is a classical problem in manufacturing to which a considerable amount of research has been devoted in different fields of Computer Science. The interest for this problem is not purely academic, since it represents many real-world problems that arise daily in factories and workshops. Simpler, easier or more efficient solving methods are of great interest, because of the practical implications they may have.

From an Artificial Intelligence perspective, the job-shop has been treated as a *constraint*

* This research is supported by the Spanish CICYT project TIC96-0721-C02-02.

*satisfaction problem* (CSP) by several authors, who have developed efficient solving approaches. However, most of these approaches are specific for the job-shop problem. The purpose of this paper is to show that generic search and CSP algorithms and heuristics can be successfully applied to the job-shop problem, obtaining similar results which are close in performance to the results obtained with specific techniques. Our approach obtains similar results to specific approaches in terms of the number of solved problems. Our approach does not outperform specific approaches in terms of CPU time, but it reaches a reasonable performance using generic search and CSP methods which, in general, are easier to develop, codify and maintain than specialized approaches.

This paper is organized as follows. In Section 2, we revise the last approaches to the job-shop from AI. In Section 3, we express the job-shop as a CSP with two different formulations. In Section 4, we solve both formulations using depth-first based CSP algorithms. In Section 5, we solve again both formulations using discrepancy-based algorithms, a new type of search algorithms recently developed. In both Sections, we provide experimental results using a benchmark widely used in the literature. Finally, in Section 6, we summarize the contributions of this work.

## 2. Related Work

The job-shop problem has been object of intense research from different perspectives. In the following, we summarize some recent approaches from an AI point of view.

A very influential work was due to Sadeh and colleagues [Sadeh et al, 95; Sadeh and Fox, 96]. They formulate the job-shop as a CSP, where variables are associated with operations, variable domains are possible start times, and constraints involve precedence among operations and exclusivity in resource use. To solve this problem, they achieve initial local consistency on precedence constraints, and they use forward checking plus some extra local consistency on resource constraints as the basic algorithm. They develop two specific variable and value ordering heuristics based on resource contention, denominated ORR and FSS respectively. They use a benchmark of 60 randomly generated problems with job-shop structure. With this approach, they solve 52 out of 60 problems, visiting less than 500 nodes, outperforming existing approaches at year 1991. Next year they were able to solve all problems modifying the algorithms to enhance it with dynamic consistency enforcement, learning and a backjumping heuristic which renders the algorithm incomplete. Coincidentally, another approach [Muscettola, 94] was able to solve the 60 problems using a stochastic search procedure based on Monte-Carlo simulation using a bottleneck partitioning approach and a global heuristic based on resource contention.

Alternatively, Smith and Cheng [Smith and Cheng, 93] formulate the problem as a search in a binary decision tree, where each node represents two operations that compete for the same resource. A node has two successors, the two possible orderings for two operations. When one of the orderings is selected, this decision is propagated over the possible start and finish times of all other operations. In addition to this problem formulation, the main contribution of this approach is a dynamic variable ordering heuristic (which pair of operations consider next) and a dynamic value ordering heuristic (which operation post first), based on slacks (free period) left by two operations competing for the same resource, when they are scheduled from their earliest start time and consecutively. These heuristics are used in a very simple search method (PCP), which selects two

operations, determines which one is scheduled first and propagates the effect of this decision. If no operation appears unfeasible, the process iterates, otherwise it stops and returns false (no backtracking is done). With this approach they solve 56 out of 60 Sadeh problems. Using a more complex version of their heuristics, modified in a somehow ad-hoc manner, they are able to solve the 60 problems.

Finally, Crawford and Baker [Crawford and Baker, 94] codify Sadeh's benchmark as propositional satisfiability problems (SAT), and they solve them using a complete algorithm (Davis-Putnam procedure) and two incomplete ones (GSAT and ISAMP). Only ISAMP is able to solve the 60 problems, with a upper limit of 20,000 tries.

From this brief analysis, we can identify complex specialized algorithms such as a modified forward checking with incomplete backtracking [Sadeh et al, 95], complex heuristics based on global resource contention (ORR/FSS [Sadeh and Fox, 96], CPS [Muscettola, 94]), or specialized local heuristics such as the slack based [Smith & Cheng, 93]. Only the formulation of Crawford and Baker seems to be generic. However, this approach also presents some drawbacks, and it is the size of the SAT translation of a problem. According to [Harvey, 95], a typical benchmark problem is translated into a theory of 100,000 clauses and 20,000 literals, using more than 1 Mbyte of memory, which seems not to be very operational from a practical perspective. Regarding completeness, all approaches solving the whole benchmark are incomplete (although PCP can be completed easily). For all this, it seems to be a plausible goal to look for simple, generic solving methods for the job-shop, easy to implement and debug, which could solve the problem with a reasonable performance.

## 3. The Job-Shop as CSP

A CSP is defined by a set of variables taking values on finite domains, and a set of constraints disallowing combinations of values that cannot be simultaneously assigned. Considering the job-shop as a CSP, variables are operation start times ($st_i$), domains ($D_i$) are determined by each operation earliest start time ($est_i$), latest finish time ($lft_i$) and duration ($d_i$), $D_i=[est_i,\ lft_i-d_i]$. Constraints are either:

1. *Precedence* constraints (between consecutive operations of a job): if operation $i$ must be executed before operation $j$, then $st_i+d_i <= st_j$
2. *Resource* constraints: if operations $i$ and $j$ require the same machine, then $st_i+d_i<=st_j$ or $st_j+d_j<=st_i$

Most algorithms for CSP rely on the use of depth-first backtrack search. *Forward checking* [Haralick and Elliot, 80] is a simple, yet powerful general purpose algorithm for CSP. It traverses a search tree rooted with the initial problem. At each search state it selects an unassigned variable and sequentially considers the assignment of its feasible values. Each time a new value is assigned to the current variable, its effect is propagated toward future variables removing those values that are not consistent with it. A deadend takes place when an assignment propagation produces an empty domain. According to standard CSP terminology, assigned and unassigned variables are called *past* and *future* variables, respectively. We will refer to this formulation as CSP formulation 1.

Some authors have reported that the job-shop is more efficiently solved if it is formulated establishing precedences between pairs of operations competing for the same machine. Under this approach, pairs of operations requiring the same machine are CSP

variables. Each variable has two possible values in its domain, the two ways precedence can be established. We denote by $i \rightarrow j$ the fact that operation $i$ precedes operation $j$. Depth first backtracking search traverses a binary tree. At each search state, it selects a pair of operations competing for the same machine whose precedence has not yet been established, and sequentially attempt the two possible orders (*i.e.*: $i \rightarrow j$ and $j \rightarrow i$). After establishing an order, its effect is propagated toward each operation updating their time interval with the following rule:

$$\text{if } (k \rightarrow l): est_l = max\{est_l,\ est_k + d_k\};\ lft_k = min\{lft_k,\ lft_l - d_l\}$$

until a fixed point is reached. A deadend takes place when $est_i + d_i$ becomes greater than $lft_i$ for any operation $i$, then the algorithm backtracks to a previous decision. We will refer to this formulation as CSP formulation 2.

Independently of what approach is used, algorithms do not determine the order in which variables are selected and values are assigned. These orderings can have a dramatic effect in average search efficiency; for this reason heuristic orderings are of great importance in this context.

## 4. Solving the Job-Shop as CSP

In this section we show that general CSP techniques may be valid for the job-shop. Our claim is based on experimental results on a classical benchmark [Sadeh and Fox, 96]. The problem set consists of 60 randomly generated problems. Each problem contains 10 jobs and 5 resources. Each job has 5 operations. A controlling parameter was used to generate problems in three different deadline ranges: wide (w), median (m) and tight (t). A second parameter was used to generate problems with both 1 and 2 bottleneck resources. Combining these parameters, 6 different categories of problems were defined, and 10 problems were generated for each category. The problem categories were carefully defined to cover a variety of manufacturing scheduling circumstances. All problems have at least one solution.

### 4.1. CSP Formulation 1.

The first experiment aimed to show that the CSP formulation 1 described in Section 3 (a start time is associated to each operation at each search tree) using standard algorithms and heuristics produces competitive results. We used plain forward checking combined with *support-based* heuristics [Meseguer and Larrosa, 95; Larrosa and Meseguer, 95]. The only modification made to the problem description given in Section 3 was that implicit precedences between nonconsecutive operations of the same job were made explicit by additional precedence constraints. Heuristics were defined using the concept of support associated to each tree node:

1. The support that a feasible value $t$ of a future variable $i$ receives from another future variables $j$ is defined as,

$$s(i,t,j) = \frac{|D''_j|}{|D'_j|}$$

   where $D'_j$ is the current domain of variable $j$, and $D''_j$ is the set of values that would remain feasible if value $t$ were assigned to variable $i$.
2. The support that a feasible value $t$ of a future variable $i$ receives from all future

variables is,

$$s(i,t) = \sum_{j \in Future} s(i,t,j)$$

3. The support that a future variable *i* receives from all future variables is,

$$s(i) = \sum_{t \in Feasible} s(i,t)$$

The *lowest support* variable selection heuristic (*ls*) chooses the variable *k* with minimum support among future variables. The *highest support* value selection heuristic (*hs*) chooses the value *t* with maximum support, among the feasible values of the current variable. In previous experiments, support-based heuristics were found expensive to compute; for this reason an approximation was proposed. We define the *approximate support* that a value *t* of a variable *i* receives from another variable *j* as the support that it receives at the tree root

$$ap\_s(i,t,j) = \frac{|D'_j|}{|D_j|}$$

where $D_j$ is the initial domain and $D'_j$ is the set of values that would remain feasible if value *t* were assigned to variable *i* at the root. Accordingly, we define value and variable supports at each tree node as

$$ap\_s(i,t) = \sum_{j \in Future} ap\_s(i,t,j)$$

$$ap\_s(i) = \sum_{t \in Feasible} ap\_s(i,t)$$

Approximate heuristics (*ap_ls* and *ap_hs*) are defined similarly, but using approximate supports. The computational benefit of approximate heuristics is that individual supports $ap\_s(i,t,j)$ are only computed once before search starts.

Table 1 shows the results of running forward checking with exact and approximate heuristics. For each case we give two columns: number of solved problems (with a search limit of 500 visited nodes) and average search effort required as the number of visited nodes. As it can be observed, the results are quite good. Using the exact heuristics (*ls*/*hs*) 51 problems are solved. If we use the approximate lowest support for variable selection (*ap_ls*/*hs*), we still solve the same 51 problems. Interestingly, by approximating support for variable selection average time of solving a problem decreases from about 300 seconds in a Sun workstation to roughly 4 seconds. In addition, it should be noted that all solved problems are solved without any backtracking. If approximate heuristics are used for variable and value selection (*ap_ls*/*ap_hs*), only 39 problems can be solved. It illustrates the importance of value selection heuristic accuracy for scheduling problems. These results are compared with the corresponding ones of [Sadeh and Fox, 96], where the forward checking algorithm with ORR/FSS heuristics solved 52 problems.

| | ls/hw | | ap_ls/hw | | ap_ls/ap_hw | | ORR/FSS | |
|---|---|---|---|---|---|---|---|---|
| | solved | nodes | solved | nodes | solved | nodes | solved | nodes |
| w/1 | 10 | 50 | 10 | 50 | 9 | 55 | 10 | 52 |
| w/2 | 9 | 50 | 9 | 50 | 8 | 116 | 10 | 50 |
| m/1 | 9 | 50 | 9 | 50 | 5 | 53 | 8 | 64 |
| m/2 | 10 | 51 | 10 | 50 | 7 | 71 | 9 | 57 |
| t/1 | 6 | 50 | 6 | 50 | 4 | 50 | 7 | 68 |
| t/2 | 7 | 50 | 7 | 50 | 6 | 52 | 8 | 61 |
| sum | 51 | | 51 | | 39 | | 52 | |

Table 1. Results of forward checking with support-based heuristics, compared with results of forward checking with ORR/FSS heuristics of [Sadeh and Fox, 96].

### 4.2. CSP Formulation 2

The second experiment aimed to show that support-based heuristics are also general in the sense that they can be effectively applied to different algorithmic approaches. We use the CSP formulation 2 described in Section 3, where variables are pairs of operations and values are their two possible orders. In particular, we use the PCP algorithm presented in [Smith and Cheng, 93] combined to support-based heuristics. Unlike [Smith and Cheng, 93], our implementation allows backtracking when a deadend occurs.

Support-based heuristics are applied to this problem formulation in the following way. At a given search node, each operation $i$ has a time interval for its start time ($st_i \in D'_i = [est_i, lft_i - d_i]$) determined by previous decisions and the propagation rule. If $i$ and $j$ are two unordered operations competing for the same machine, we define the support that establishing $i \rightarrow j$ receives from the problem as

$$s(i \rightarrow j) = \sum_{j \in Operations} \frac{|D''_j|}{|D'_j|}$$

where $D'_k$ is the time interval $[est_k, lft_k - d_k]$ for operation $k$ before deciding the order between $i$ and $j$, and $D''_k$ is the time interval $[est_k, lft_k - d_k]$ after deciding $i \rightarrow j$ and propagating its effect.

With this definition, the lowest support variable selection heuristic for this algorithm selects the pair of operations $(i,j)$ with minimum sum of supports for the two possible orderings,

$$min_{k,l} \{s(k \rightarrow l) + s(l \rightarrow k)\}$$

In a similar way, the highest support value selection heuristic selects the ordering that ordering that receives the highest support.

Table 2 shows the results of this experiment: 56 problems were solved with a search limit of 1,000 nodes. All solved problem instances but one are solved without any backtracking (225 visited nodes). In average, our algorithm requires about 13 seconds to solve a problem. These results are compared to those of [Smith and Cheng, 93] where PCP combined to slack-based heuristics solved 56 problem, too.

| | ls/hw | | slack-based | |
|---|---|---|---|---|
| | solved | nodes | solved | nodes |
| w/1 | 10 | 225 | 10 | 225 |
| w/2 | 10 | 225 | 10 | 225 |
| m/1 | 10 | 225 | 10 | 225 |
| m/2 | 10 | 231 | 10 | 225 |
| t/1 | 10 | 225 | 10 | 225 |
| t/2 | 6 | 225 | 6 | 225 |
| sum | 56 | | 56 | |

Table 2. Results of PCP with support-based heuristics, compared with results of PCP with slack-based heuristics [Smith and Cheng, 93].

## 5. Using Discrepancy Algorithms

Experimental results presented in the previous Section show that any heuristic (ORR/FSS, slack-based, support-based) is not perfect and, in occasions, it may be wrong. When a wrong advice is made early in the search tree, any depth-first-based search procedure has to unsuccessfully traverse a large subtree without any solution. This is so because depth-first is strongly committed to the first heuristic advices. Trying to solve this problem for the job-shop, [Sadeh et al, 95] modified the forward checking algorithm adding an *incomplete backjumping heuristic*: when the system starts thrashing, the algorithm backjumps all the way to the first search state and simply tries the next best value. This approach renders the algorithm incomplete, and it has been implemented with a parameter (the maximum number of visited nodes between backjumps), which has to be adjusted to solve the whole benchmark. Alternatively, [Smith and Cheng, 93] modified the slack-based heuristics introducing a bias; this was implemented by two parameters $n_1$ and $n_2$, which should be adjusted manually to achieve the heuristic formulation able to solve the whole benchmark. In both cases, parameters are problem-dependent and they may change using a different benchmark, so manual tuning is always required.

To decrease the degree of dependency of depth-first search with initial decisions in the search tree, new search strategies have been recently proposed following the work of [Harvey and Ginsberg, 95] and further developed by [Korf, 96; Meseguer, 97; Walsh, 97]. These new algorithms are based in the concept of *discrepancy*. Regarding CSP, a search path has as many discrepancies as value assignments differing from the value ordering heuristic first choice. A discrepancy-based algorithm is not strictly committed to the first choices made early in the tree, which forces depth-first to search a sequence of nested subproblems, but it searches in several subtrees corresponding to subproblems which have little in common. This minimizes the negative performance impact of early wrong decisions. In particular, *limited discrepancy search* (LDS, [Harvey and Ginsberg, 95]) is a complete backtracking algorithm that searches the nodes of the tree in increasing order of discrepancies (*i.e.*: in its first iteration it searches all paths with less than 1 discrepancy, in its second iteration it searches all paths with less that 2 discrepancies, and so on). LDS is easily adapted to algorithms used in Section 4 and combined with support-based heuristics, producing complete procedures which are adaptable to problem difficulty, so no manual tuning of parameters is required. In the following, we provide experimental

results of these combinations for the two CSP formulations introduced in Section 3.

### 5.1. CSP Formulation 1

Our third experiment aimed to show that the reason for failure in 9 problems of Section 4.1. is the combination of two factors: occasional wrong heuristic advice and depth first commitment to early decisions. For this purpose, we combine LDS with forward checking and support-based heuristics (*ap_ls* and *hs*). Table 3 presents the results of the experiment where all problems are solved. 51 problems are solved with 0 discrepancies (and an average CPU time of 4 seconds), 8 problems require 1 discrepancy (and 115 seconds on average) and 1 problem requires 2 discrepancies (and 3,700 seconds). Tracing the execution we could verify our conjecture, because solution paths had their discrepancies in the first tree levels. As far as we know, this is the first paper reporting a complete algorithm in which all problems are solved using a standard CSP formulation.

| | ap_ls/hw | | inc. fc ORR/FSS | |
|---|---|---|---|---|
| | solved | nodes | solved | nodes |
| w/1 | 10 | 50 | 10 | 52 |
| w/2 | 10 | 53 | 10 | 50 |
| m/1 | 10 | 68 | 10 | 55 |
| m/2 | 10 | 50 | 10 | 54 |
| t/1 | 10 | 4,831 | 10 | 57 |
| t/2 | 10 | 813 | 10 | 60 |
| sum | | | 60 | |

Table 3. Results of LDS with forward checking and support-based heuristics, compared with results of incomplete forward checking with ORR/FSS heuristics [Sadeh et al, 95].

### 5.2. CSP Formulation 2

Our fourth and last experiment combined LDS with the PCP algorithm and support-based heuristics (Section 4.2.). Table 4 gives the results. Again, all problems are solved: 55 problems with 0 discrepancies (14 seconds on average), and 5 problems with 1 discrepancy (36 seconds on average).

| | ls/hw | | modif. slack-based | |
|---|---|---|---|---|
| | | | solved | nodes |
| w/1 | 10 | 225 | 10 | 225 |
| w/2 | 10 | 225 | 10 | 225 |
| m/1 | 10 | 225 | 10 | 225 |
| m/2 | 10 | 244 | 10 | 225 |
| t/1 | 10 | 225 | 10 | 225 |
| t/2 | 10 | 356 | 10 | 225 |
| sum | 60 | | 60 | |

Table 4. Results of LDS with PCP and support-based heuristics, compared with results of PCP with modified slack-based heuristics [Smith and Cheng, 93].

## 6. Conclusions

From this work we can conclude that search and CSP techniques motivated and developed in a generic context can be effectively applied the job-shop problem. Regarding performance, our generic approach has the same solving power than specific ones; although it does not outperform specific methods in CPU time, its computational requirements are quite reasonable. Regarding methodology, our approach is generic and it does not include domain-dependent elements such as bias or parameters which have to be manually adjusted for each problem set. This makes our approach more robust and more applicable to other problem instances. The solution proposed is a combination of three well known elements in the constraint community: constraint propagation (by forward checking), dynamic variable and value selection (by support-based heuristics) and early mistakes avoidance (by discrepancy-based search). heuristic advice. By the modular inclusion of each of these elements, we have assessed their relative importance and the role that each plays in the construction of the solution. Each of these elements has an intrinsic value for the search community and it has been independently analyzed and studied. This provides our approach a higher level of understanding than specific methods, which renders it more suitable for supporting the development of applications.

## References


[Crawford and Baker, 94] Crawford J. and Baker A. (1994). Experimental results on the Application of Satisfiability Algorithms to Scheduling Problems, *Proc of AAAI-94*, 1092-1097.

[Harvey, 95] Harvey W. (1995). *Nonsystematic Backtracking Search*, PhD thesis, Stanford University.

[Harvey and Ginsberg, 95] Harvey W. and Ginsberg M. (1995). Limited Discrepancy Search, *Proc. of IJCAI-95*, 607-613.

[Haralick and Elliot, 80] Haralick R. and Elliot G. (1980). Increasing tree search efficiency for constraint satisfaction problems, *Artificial Intelligence*, 14, 263-313.

[Korf, 96] Korf R. (1996). Improved Limited Discrepancy Search, *Proc. of AAAI-96*, 286-291.

[Larrosa and Meseguer, 95] Larrosa J. and Meseguer P. (1995). Optimization-based Heuristics for Maximal Constraint Satisfaction, *Proceedings of CP-95*, 103-120.

[Meseguer, 97] Meseguer P. (1997). Interleaved Depth-First Search, *Proc of IJCAI-97*, 1382-1387.

[Meseguer and Larrosa, 95] Meseguer P. and Larrosa J. (1995). Constraint Satisfaction as Global Optimization, *Proc. of IJCAI-95*, 579-584.

[Muscettola, 94] Muscettola N. (1994). On the Utility of Bottleneck Reasoning for Scheduling, *Proc. of AAAI-94*, 1105-1110.

[Sadeh et al, 95] Sadeh N., Sycara K., and Xiong Y. (1995). Backtracking techniques for the job shop scheduling constraint satisfaction problem, *Artificial Intelligence*, 76, 455-480, 1995.

[Sadeh and Fox, 96] Sadeh N. and Fox M. (1996). Variable and value ordering for the job shop constraint satisfaction problem, *Artificial Intelligence*, 86, 1-41.

[Smith and Cheng, 93] Smith S. and Cheng C. (1993). Slack-Based Heuristics for Constraint Satisfaction Scheduling, *Proc of AAAI-93*, 139-144.

[Walsh, 97] Walsh T. (1997). Depth-bounded Discrepancy Search, *Proc of IJCAI-97*, 1388-1393.