Heuristic Methods for Solving *Job-Shop* Scheduling Problems

A. Garrido, M. A. Salido, F. Barber and **M. A. López**[1]

Abstract. Solving scheduling problems with Constraint Satisfaction Problems (CSP's) techniques implies a wide space search with a large number of variables, each one of them with a wide interpretation domain. This paper discusses the application of CSP heuristic techniques (based on the concept of slack of activities) for variable and value ordering on a special type of job-shop scheduling problems in which the operations must schedule inside of temporal windows. These techniques are improved by introducing the concepts of slack probability and the find-hole method. Thus, a more flexible heuristic technique is obtained, which improves empirical efficiency and allows early detection of unfeasible problems.

1 INTRODUCTION

Scheduling is the problem of allocating limited resources to operations (activities) over time. Scheduling is a complex task that can be formulated using a constraint-based representation. Reasons for scheduling complexity include [4]:

- Scheduling is a feasibility problem. The final solution must accomplish all the problem constraints. Another objective to be satisfied is the optimization of an evaluation function, adjusting to certain criteria as cost, lateness, process time, inventory time, etc.
- Some scheduling problems have many constraints due to the unavailability of resources, due dates, etc.
- Constraint representation cannot express the importance of the value domains. The number and identity of tasks that require a resource over a particular time interval is a key piece of information that can suppose the basis for heuristic variable and value orderings.

Scheduling constraints are usually disjunctive ones (i.e.: two tasks cannot use the same resource at the same time). The consistence problem of metric disjunctive constraints is NP-hard [3], such that *CSP* techniques are used. However, inequality constraints generate a large search space that may have few (or no) feasible solutions. Thus, it becomes necessary to define techniques to empirically decrease this complexity and be able to solve real problems more efficiently (constraining value domains, relaxing some constraints, etc). We are interested in the relaxation of the heuristic variable and value orderings to obtain a more flexible method, which makes a better use of the knowledge the scheduler may have about each particular problem. Our work is focused on *job-shop scheduling* problems. In these problems, operations must be scheduled within their feasible time windows (i.e.: between its respective earliest start time and latest finish time). We propose heuristic techniques for variable and value orderings to be included in two known searching algorithms: *Basic-Depth-First Backtrack and Depth-First-with-DCE* [9, 10]. The former algorithm is the classical chronological backtracking procedure heuristically improved. The latter uses the additional heuristic *Dynamic Consistency Enforcement* (*DCE*), which dynamically focuses its effort on critical resource subproblems and learns from its previous faults.

We summarize the main concepts about the *job-shop scheduling* problem and the *CSP* approach in Section 2. In Section 3, we introduce the search method used and new heuristic concepts in this process. The proposed heuristics are empirically evaluated on a set of typical problems in Section 4. Conclusions and final remarks are discussed in Section 5.

2 THE *JOB-SHOP* SCHEDULING PROBLEM

A *job-shop scheduling* problem is represented by a set of jobs $J=\{J_1,...,J_n\}$ and a set of resources $R=\{R_1,...,R_m\}$. Each job J_i consists of a set of operations $O^i=\{O^i_1,...,O^i_n\}$ which must be performed between a *ready-time* (rt_i) and a *due-time* (dt_i). The execution of each operation (O^i_k) requires the use of a set of resources (R^i_k Í R) during a time interval (du^i_k). The start time st^i_k of operation O^i_k indicates when the operation may begin to use the resources R^i_k.

The problem of *job-shop scheduling* can be considered as a Constraint Satisfaction Problem [1], with the following elements:

- A set of variables $\{x_1,...,x_n\}$ associated with the start time of operations. These variables take values in finite domains $\{D_1,...,D_n\}$ that may be constrained by unary constraints over each variable. In these problems, time is usually assumed discrete, with a problem-dependent granularity.
- A set of constraints $\{c_1,...,c_m\}$ among variables which are predicates $c_k(x_i,...,x_j)$ defined on the Cartesian product D_i ´...´ D_j and restrict the variable domains.

[1] Dpto. Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n 46071, Spain, email: {agarridot, msalido, fbarber}@dsic.upv.es

Constraints of job-shop scheduling problems can be represented as binary, disjunctive, metric and point-based constraints [3]:

$$c_i(x_j, x_k) \equiv \{[d_1^- d_1^+][d_2^- d_2^+] \cdots [d_n^- d_n^+]\} \text{ where } d_i^- \le d_i^+ .$$

This constraint disjunctively restricts the temporal distance between x_j and x_k:

$$d_1^- \le x_k - x_j \le d_1^+ \vee \cdots \vee d_n^- \le x_k - x_j \le d_n^+$$

Moreover, unary constraints on a variable x_j may be represented as binary constraints between the variable and a special time-point $T0$, which represents *'the beginning of the world'* (usually, $T0 = 0$):

$$x_j - T0 \in \{[d_1^- d_1^+][d_2^- d_2^+] \cdots [d_n^- d_n^+]\}$$

meaning that: $x_j \in \{[d_1^- d_1^+][d_2^- d_2^+] \cdots [d_n^- d_n^+]\}$

Thus, we have a Temporal Constraint Problem or *TCP* [3]. An assignment of the variables $(x_i,...,x_j)$ in their domains is *consistent* with respect to c_k iff c_k is satisfied. A solution of a *CSP* is an assignment of a value to each variable within its respective value domain satisfying all constraints.

2.1 Constraints and *CSP* Algorithms

Two main constraints appear in this kind of *job-shop* problems:

- *Precedence constraints*: the operations O_j^i of each job J_i must be scheduled according to precedence constraints, i.e., there exists a partial ordering among the operations of each job and may be represented (Figure 1) by a precedence graph or *tree-like* structure [10]. Each precedence constraint O^i_k *BEFORE* O^i_l gives rise to the linear inequality $st^i_k + du^i_k < st^i_l$.
- *Capacity constraints:* resources cannot be used simultaneously by more than one operation. Thus, two different operations O^i_k and O^j_l cannot overlap unless they use different resources. Capacity constraints give rise to disjunctive linear inequalities:

$$\forall\, O^i_k, O^j_l: \quad R^i_k \cap R^j_l = \emptyset \vee ([O^i_k \text{ before } O^j_l] \vee [O^j_l \text{ before } O^i_k]$$

Additionally, other technological constraints may restrict the set of possible execution times for individual operations or availability times for resources, etc.

There are two main objectives in a *job-shop CSP*:

- *Feasibility*: if a solution can be found. We can later be interested in optimal solutions, according to several scheduling optimality criteria or cost functions to evaluate the optimality of each feasible solution.
- *Efficiency of the CSP process*. This objective implies minimizing backtracking processes and, consequently, the conflicts. When an operation requires a resource that is already being used, a conflict appears. We can anticipate possible conflicts and improve the algorithm efficiency by detecting resources with the highest contention.

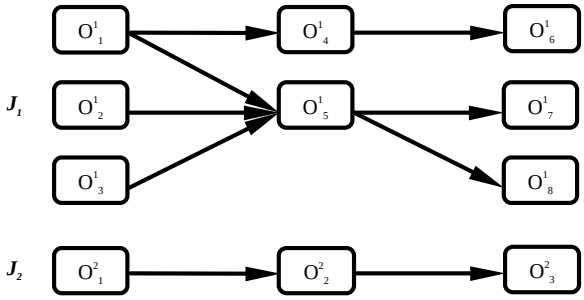

Figure 1. Example of tree-like (from [10])

The solving algorithm should be able to generate and evaluate all the possible assignments for each one of the problem variables. However, only a small fraction of these possible assignments will participate in a feasible solution. Thus, an efficient search, by means of an incremental method, should establish an appropriate order for instantiating the variables and obtain the order to select the values from the domains. This incremental method can be applied with two main techniques: retrospective and prospective techniques [1].

- *Retrospective techniques* that assign a value to a variable checking other variables with assigned values in order to avoid conflicts. If all constraints hold, another variable and value are selected. Otherwise, some constraint is not satisfied and backtracking occurs. The simplest retrospective technique is chronological backtracking. If the verification of consistency fails, it will select the variable most recently instantiated and it will test another value of its domain. When all values in the variable domain have been unsuccessfully tested, the backtracking goes back to the next most recently instantiated variable, and so on. If the procedure goes back to the initial state (i.e., the state with an empty schedule), the problem is unfeasible. Chronological backtracking may behave in an inefficient way, which is known as *thrashing* [6]. Thrashing may appear when the backtracking tries to recover from a *dead-end* state (a partial solution which cannot be completed). Since it tries to schedule the last scheduled operation, it may go back to similar *dead-end* states. However, the operation most recently scheduled is not usually the cause of the conflict. The search may fail due to some variable assignment previously performed in the search.
- *Prospective techniques* that propagate the effects of each variable instantiation to unassigned variables. This propagation is based on three levels of local consistency[2] which arise from the analysis of reasons for thrashing [1]:
  i) *Lack of node consistency*. No elements in the variable domain satisfy unary constraints. Assigning these values causes immediate failures.

[2] Local levels of *k-consistency* are empirically more efficient in backtracking processes than *total-consistency* [6].

ii) *Lack of arc consistency* which applies the previous concept to binary constraints.

iii) *Lack of path consistency*. For each value $x_i \hat{I} D_i$ and $x_k \hat{I} D_k$ such that $(x_i=v_i\ c_{ik}\ x_k=v_k)$ holds, a sequence of values does not exist

$x_{i+1} \hat{I} D_{i+1}, x_{i+2} \hat{I} D_{i+2},...,x_{k-1} \hat{I} D_{k-1}$, such as

$(x_i\ c_{i\,i+1}\ x_{i+1})$, $(x_{i+1}\ c_{i+1\,i+2}\ x_{i+2})$,..., and $(x_{k-1}\ c_{k-1\,k}\ x_k)$ hold.

2.2 Variable and Value Orderings

The order in which variables and domain values are selected in a *CSP* process is important to decrease the empirical computational time. An optimal variable/value ordering would produce a linear time solution for a feasible scheduling problem because no backtracking would be necessary. Thus, an aim is to minimize backtracking stages and, consequently, the conflicts. These conflicts appear when an operation requires a resource that is already being used. Therefore, it is necessary to use good *ordering* heuristics to efficiently solve practical problems and reduce the effective size of the search space [6]. In particular, *texture measurements* [8] can be used as a basis for heuristic decisions. A texture measure is an assessment of properties of a constraint graph and reflects the intrinsic structure of a particular problem. On the other hand, by detecting resources that have the highest contention, we can anticipate possible conflicts and improve algorithm efficiency [2, 10]. The most common heuristic is to instantiate the *most constrained* variable to *its least constraining* value. Intuitively, the earlier the most constrained variable is instantiated, the earlier the backtracking will take place (pruning the search space and minimizing thrashing). Furthermore, the probability of finding a solution without backtracking in this variable is increased by assigning the least constrained value.

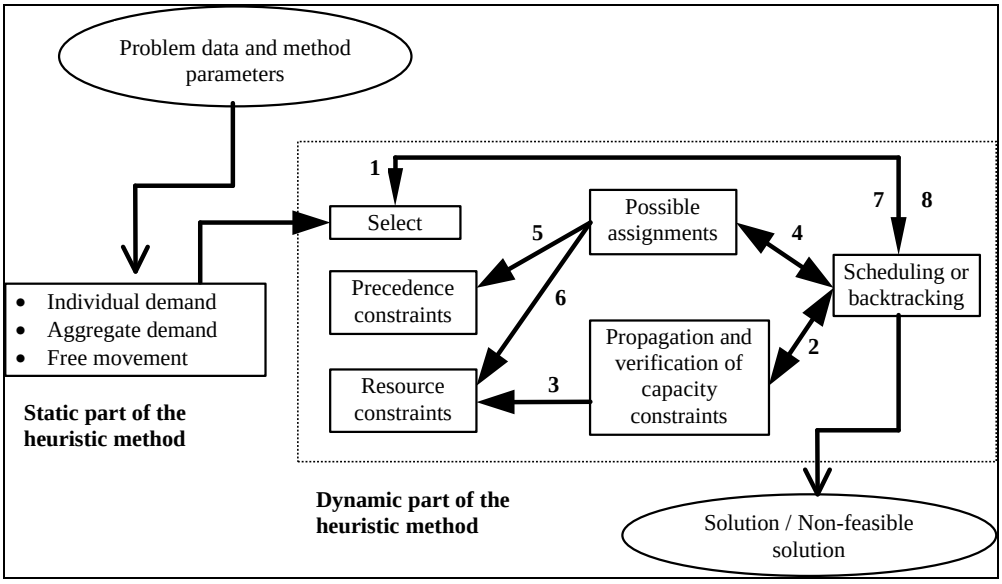

Figure 2. Heuristic Searching Method: Static part and Dynamic part

3 HEURISTIC SEARCH METHOD

We use the classic backtracking procedure, which is improved with specific heuristic criteria for variable ordering to avoid thrashing and to improve the efficiency of the process. Our work is based on the heuristic techniques developed by Sadeh [10]. However, we do not consider all the criteria because they vastly limit the search space. Moreover, the flexibility of the method is based on some additional considerations that allow us to extend the search space.

In order to study the calendar of utilization of each resource, we use the *Individual Demand* and *Aggregate Demand* techniques. These techniques and the concept of *slack probability* of an operation allow us to make better use of the temporal windows of the operations for scheduling. These techniques select the most critical operation and assign it its less-constrained value. We have mixed retrospective and prospective techniques in the verification of the consistency. Furthermore, we introduce the *find-hole* method that allows us to eliminate any doubt in the presence of a conflicting situation either: (i) according to operations not yet scheduled (prospective); or (ii) according to the maintenance of the consistency among the new operation and the operations already scheduled (retrospective).

3.1 Searching Process

Chronological backtracking is based on the incremental search of a solution. This incremental search can be represented as a transition between states. It starts in an initial state without any scheduled action. It finishes in a final state when a solution is found (all the actions have been scheduled) or when there is no feasible solution. Our algorithm uses some heuristics to improve the efficiency of the classical backtracking techniques. Figure 2 shows a schematic representation of the heuristic method used in

the search of a solution. The method has two parts: one static part (search anticipation), and another dynamic part, which is based on the techniques that are applied during the search process. The majority of the decisions are taken during the search process: consistency enforcing, which value to select, schedule, or unschedule, etc. However, all these decisions are conditioned by the decision adopted in the static part: the order of selecting the variables.

We can observe a possible sequence of decisions in the scheduling of an operation in Figure 2:

1. An operation is selected to be scheduled in (1).
2. Capacity constraints are verified according to the operations not yet scheduled and no conflict is detected (2, 3).
3. We look for a possible time for the start of the operation (4) guaranteeing the precedence and capacity constraints according to the operations that have already been scheduled (5, 6).
4. There is no conflict and the operation is scheduled (7).

On the other hand, if a conflicting situation is detected in step 3, a backtracking stage (8) will occur in step 4 and the procedure will go back to the most recently scheduled operation testing a new value of its domain.

3.2 *Slack* Probability of an Operation

The slack of an operation O^l_i is defined in Operational Research as the difference between the latest (lst^l_i) and earliest (est^l_i) start time:

$$S(O^l_i)=lst^l_i - est^l_i$$

This value indicates the number of units of time the operation's execution can be delayed, without delaying the project. In our context, we have a similar underlying idea: we are interested in finding which is the operation that admits more starting times, taking into account the precedence constraints. We define the *slack probability* (*SP*) of an operation as the probability of movement in the temporal window $[est^l_i,...,lft^l_i]$, where est^l_i is the earliest O^l_i start time and lft^l_i is the latest O^l_i finish time. Thus, the *slack probability* is defined as:

$$SP\left(O^l_i\right)=1-\frac{du^l_i}{\left(lft^l_i-est^l_i+1\right)}$$

If the operation has only one possible start time, the *slack probability* will be null. Otherwise, if the operation has a wide domain of possible start times, its *slack probability* will be the unit. Hence, an operation is more conflictive than another if its *slack probability* is minor.

A simple problem is shown in Figure 3. In this figure, there exist operations which share a same resource *R*. In addition, operation slacks are represented for each operation. If the criterion for selecting the next operation is the *slack probability*, the next operation to be scheduled will be the operation O^4_2 because its value $SP(O^4_2)$ is the least.

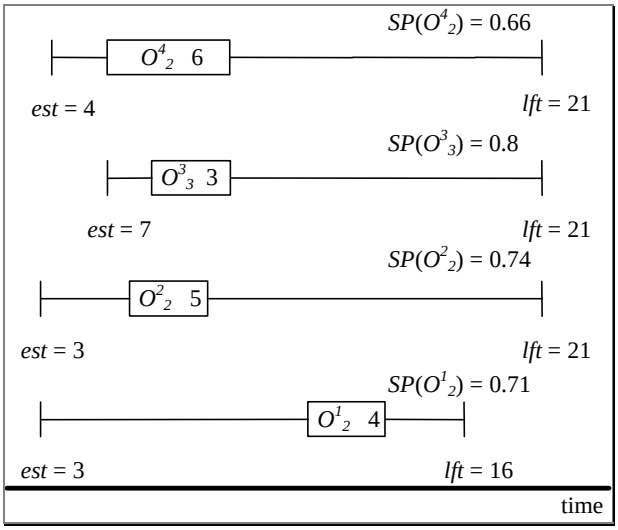

Figure 3. Representation of the slack probability for operations that share a resource *R* (boxes are labeled by the name of the operation and its duration)

3.3 Consistency Enforcing: the Heuristic *Find-Hole* Method

Once an operation has been selected to schedule, it is necessary to check that all the involved constraints are satisfied:

- *Consistency according to the precedence constraints*. The precedence constraints are defined among operations of a same job: the process must guarantee that two operations of a same job are not executed in the same instant of time. Essentially, the earliest start time is propagated downstream within the job whereas latest start time is propagated upstream. This propagation is applied before the *CSP* search process.
- *Consistency according to constraints about resource capacities*. Checking the consistency of capacity constraints is a difficult process due to the next constraints:
  - *Forward consistency checking*. When an operation is scheduled and a resource is allocated to the operation, a forward checking process analyzes the set of remaining possible reservations of other operations that requires the same resource and removes those conflicting reservations.
  - *Additional consistency check*ing. The process must check if two unscheduled operations that require the same resource are not overlapped. Two operations overlap when both require the same resource at the same time for every start time.
  - *Find-hole*. This method checks if an apparent conflicting situation is actually conflicting. Before indicating an operation has not any possible execution, it is necessary to check the resource usage calendar of this operation. If the shared resource is not used during the entire operation's execution, it can be used in another

operation. The *find-hole* method identifies the two extreme time points of the temporal line in which the action is executed (t^- and t^+, respectively) and searches some hole in which the resource is available and can be used by another operation. If a hole does not exist, an inconsistency will be detected because the resource is not available.

$$find\text{-}hole(O^l{}_i)\ checks\ if \begin{cases} lst^l{}_i + du^l{}_i > t^- \\ lst^l{}_i \le t^+ \\ est^l{}_i + du^l{}_i > t^- \end{cases}$$

4 EMPIRICAL EVALUATION

In this section, we study the empirical evaluation of the developed heuristic method. The empirical method performance is compared with the algorithms *Basic-Depth-First* and *Depth-First-with-DCE*. Both algorithms use the same variable/value ordering heuristics and the same techniques of consistency enforcing.

- *Basic-Depth-First*. This algorithm shows the efficiency of a chronological backtracking (it always goes back to the last scheduled operation).
- *Depth-First-with-DCE*. This algorithm behaves as the previous one, but it uses the added *DCE* heuristic.

4.1 Design of the Data Test

We have defined four types of *job-shop scheduling* problems. Each type has a different number of jobs, operations and resources (see Table 1). We have randomly generated 50 problems of each type. The number of operations may vary, but the number of jobs and resources cannot.

Table 1. Types of *job-shop* scheduling problems

	JOBS	RESOURCES	OPERATIONS
TYPE 1	10	6	60 max. 20 min.
TYPE 2	10	8	80 max. 50 min.
TYPE 3	10	10	100 max. 80 min.
TYPE 4	12	10	120 max. 108 min.

Two parameters allow us to deal with different scheduling conditions. The range parameter (*RG)* adjusts the distribution of the due dates and release dates of the jobs. The bottleneck parameter (*BK)* controls the number of bottleneck resources. In each problem type, we have three different values of the range parameter (*RG*) and two bottleneck configurations (*BK*) (see Table 2). Moreover, the operation durations were randomly obtained from two different distributions, depending on whether an operation requires a bottleneck resource or not.

Table 2. Comparison of the heuristic methods, which is used in the algorithms *Basic-Depth-First* and *Depth-First-with-DCE* vs. Chronological Backtracking (over 5 sets of 40 *job-shop* problems). Standard deviations appear in brackets

Performance of the heuristic method					
		CHRONOLOGICAL BACKTRACKING	**BASIC-DEPTH-FIRST**	**DEPTH-FIRST-WITH-DCE**	
				K=4	**K=8**
RG=0.2	**Search Efficiency (*)**	**0.12** (0.15)	**0.48** (0.44)	**0.68** (0.30)	**0.85** (0.33)
BK=1	**Solved Experiments**	**1**	**17**	**17**	**17**
RG=0.2	**Search Efficiency (*)**	**0.12** (0.15)	**0.64** (0.43)	**0.75** (0.32)	**0.86** (0.29)
BK=2	**Solved Experiments**	**1**	**24**	**24**	**24**
RG=0.1	**Search Efficiency (*)**	**0.17** (0.24)	**0.83** (0.33)	**0.89** (0.23)	**0.92** (0.19)
BK=1	**Solved Experiments**	**3**	**33**	**34**	**34**
RG=0.1	**Search Efficiency (*)**	**0.17** (0.24)	**0.91** (0.26)	**0.93** (0.20)	**0.94** (0.16)
BK=2	**Solved Experiments**	**3**	**36**	**36**	**36**
RG=0.0	**Search Efficiency (*)**	**0.15** (0.20)	**0.96** (0.20)	**0.96** (0.17)	**0.97** (0.14)
BK=1	**Solved Experiments**	**2**	**38**	**38**	**38**
TOTAL	**Search Efficiency (*)**	**0.15** (0.20)	**0.76** (0.34)	**0.84** (0.25)	**0.91** (0.23)
	Solved Experiments	**10**	**148**	**149**	**149**

(*) obtained by dividing the total number of problem's operations by the number of generated nodes in the solution search. In this way, the maximum efficiency is 1

4.2 Algorithm Comparison

Since a depth-bound was set in the solution search, when more than 800 states are generated, the search process stops. In this case, we assume the problem is probably nonfeasible.

The obtained results are summarized in Table 2. As we thought, the chronological backtracking method is not enough to solve complex *job-shop* problems. In spite of the low rate of found solutions, the efficiency is appropriate enough (it is 0.85 with *RG=0.2*, *BK=1* and consistency level *k=8*). This efficiency rate is due to the fact that we have also considered the unsatisfied problems in the efficiency calculation. Hence, the

proposed heuristic technique is capable of stopping the search when it deals with unfeasible problems (the process does not expect to find a feasible solution).

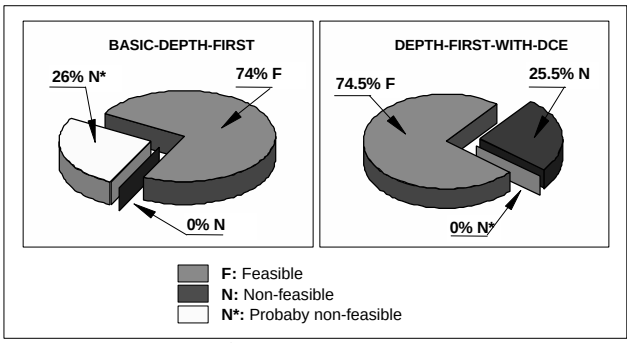

Figure 5. Comparison of the algorithms *BDF* and *DCE*

The results obtained by *Basic-Depth-First* (*BDF*) and *Depth-First-With-DCE* (*DCE*) algorithms are presented in Figure 4. 26% of the problems tested by *BDF* were declared probably nonfeasible (more than 800 states were generated), whereas *DCE* classified them as feasible and nonfeasible problems. Consequently, the *BDF* algorithm is not capable of properly determining whether the problem is feasible or not. However, the results are inverted in the *DCE* algorithm: all the problems for which the solution is not found are unfeasible.

5 CONCLUSIONS

In this paper, a variation of the *job-shop scheduling* problem, in which operations have to be performed within fixed temporal windows, has been analyzed. We refer to these problems as *job-shop CSP*s because operations must accomplish precedence and capacity constraints. *Job-shop CSP* problems cannot be solved with traditional scheduling techniques such as priority dispatch rules, one-pass scheduling techniques [5, 7] or traditional linear programming techniques of Operational Research.

Our studies are mainly based on Sadeh's work [8, 9, 10]. In his work, the *CSP* paradigm is applied to this kind of problems, demonstrating that *CSP* methods are promising alternatives to traditional scheduling methods for solving *job-shop CSP*s. We have proved propagation techniques and heuristics for variable and value ordering (*slack probability* and *find-hole* methods) are useful in solving *job-shop scheduling* problems. Moreover, these techniques are able to stop the solving process when a feasible solution cannot be found. Furthermore, the empirical results show these heuristic methods can efficiently solve different types of problems that could not be solved with the traditional *CSP*s.

ACKNOWLEDGMENTS

This work is proposed in the *Intelligent Planning & Scheduling Group* of the Polytechnic University of Valencia (http://www.dsic.upv.es/users/ia/gps) and partially supported by the grant CICYT/TAP98-0345 from the Spanish government.

REFERENCES

[1] J.C. Beck, *A Schema for Constraint Relaxation with Instantiations for Partial Constraint Satisfaction and Schedule Optimization*, Master's Thesis, Technical Report EIT-94-3, University of Toronto, (1994).

[2] J.C. Beck, A.J. Davenport, E.M. Sitarski and M.S. Fox, *Beyond Contention: Extending Texture-Based Scheduling Heuristics*, In National Conference On Artificial Intelligence (AAAI-97), (1997).

[3] R. Dechter, I. Meiri and J. Pearl, 'Temporal constraint networks', *Artificial Intelligence*, **49**, 61-95, (1991).

[4] M.S. Fox and N. Sadeh, *Why is Scheduling difficult? A CSP Perspective*, In 9th European Conference on Artificial Intelligence, (1990).

[5] S. French, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Wiley, (1982).

[6] V. Kumar, 'Algorithms for Constraint Satisfaction Problems: A Survey' *AI Magazine*, **13(1)**, 32-44, (1992).

[7] T.E. Morton and D.W. Pentico, *Heuristic Scheduling Systems*, Wiley, (1993).

[8] N.M. Sadeh, *Lookahead techniques for micro-opportunistic job-shop scheduling*, Ph.D., CMU-CS-91-102, School of Computer Science, Carnegie Mellon University, Pittsburgh, (1991).

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

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