Complex constraints management

Maria Isabel Alfonso[1] and Federico Barber[2]

[1] Department of Ciencia de la Computacin e Inteligencia Artificial,
University of Alicante, Spain
eli@dccia.ua.es
[2] Department of Sistemas Informaticos y Computacion,
University Polytechnique of Valencia, Spain
fbarber@dsic.upv.es

.

Abstract. The purpose of this paper is to show a new approximation to manage complex constraints in the framework of CSP problems. Concretely we propose the use of a labelled-CSP to specify complex temporal constraints, as a particular case of constraints. This framework allows us to specify and solve the constraint set associated to several types of CSP problems in an integrated manner. The complex constraints that we can manage are represented by a set of non-disjunctive and disjunctive constraints. We can also associate a cost with the constraints, that can affect to the obtained solution. So, the advantages of this framework are its expressiveness, that allows to specify very complex constraints which are present in real problems, and the fact that all constraints are processed in the same way.[3]

1 Introduction

A CSP problem (Constraint Satisfaction Problem) is specified providing a set of variables, a domain for each variable defining the values to which each variable may be assigned, and a set of constraints on the variables. Solving an instance of a CSP problem consists in assigning values to variables such that all constraints are satisfied simultaneously [1].

Each constraint is defined over some subset of the original set of variables and limits the combinations of values that the variables in this subset can take. The number of affected variables is the *arity* of the constraint. Any *n-ary* constraint can be expressed in terms of binary constraint. Hence, in some sense, binary CSPs are representative of all CSPs. For a comprehensive overview on the CSP see [2]. An exhaustive study can also be found in [3].

In a CSP problem we can stablish a neat distinction between:

- A precise definition of the constraints that define the problem to be solved.
- The algorithms and heuristics enabling the selection, ordering and cancellation of decisions to solve the problem.

[3] This work has been partially supported by the projects DPI2001-2094-C03-03 of the M.C.yT. (Spain) and UPV-20010980

The quality of the obtained solutions depends on the resolution method used, but it is also influenced by the representation model considered [4]. Several representations have been used to specify the constraints of a CSP problem depending on the method used to solve it. Van Hentenryck [5] and Cohen [6] tackle the CSP from a constraint logic programming viewpoint. Zhou [7] defines a constraint language, named NCL, and uses it to to solve several CSP problems, [8]. Dechter uses a temporal constraint network based representation [9].

In this paper we propose a new framework to specify and solve complex constraints in several CSP problems. We use a labelled temporal network representation (labelled TCSP) [10] that allows us to handle metric and disjunctive information (each edge can be labelled with multiple temporal intervals). The advantages of this representation over previous approaches are:

- It provides more expressiveness to specify complex constraints. So, we can specify constraints derived from disjunctive constraints and conditional constraints.
- Also, it can include cost information that can be managed in an integrated manner with the resolution method.
- It provides more flexible solutions, that is, the partial solutions are more easy to repair in the case of problem changes.

2 The TCSP Labelled Model

The labelled TCSP (Temporal Constraint Satisfaction Problem) model has been proposed in [10] . It extends the notion of Temporal Constraint Network from Dechter in [9], introducing the concept of labelled constraint, and the named *I-set* as the set of inconsistence-sets (*I-L-sets*). This results in a labelled point-based disjunctive metric temporal algebra, which gives rise to a labelled-TCN (LTCN).

The labelled disjunctive constraints and the *I-L-set* are defined as follows:

Definition 1 (*labelled canonical constraints*). A labelled canonical constraint $lec_{ij.k}$, between the temporal points t_i and t_j, is a canonical constraint $ec_{ij.k}$ associated to a set of labels $\{label_{ij.k}\}$, where each $ec_{ij.k}$ is an interval $[d_{ij.k}, D_{ij.k}]$, and each $label_{ij.k}$ is a symbol.

$lec_{ij.k} \equiv (ec_{ij.k}\{label_{ij.k}\})$, $ec_{ij.k} = [d_{ij.k}, D_{ij.k}], d_{ij.k} \leq D_{ij.k}$

The canonical constraint $ec_{ij.k}$ limits the temporal distance between the temporal points t_i and t_j, and indicates that $d_{ij.k} \leq t_j - t_i \leq D_{ij.k}$

Definition 2 (*labelled constraints*). A labelled constraint lc_{ij} is a disjunctive set of labelled canonical constraints $\{lec_{ij.k}\}$. That is, $lc_{ij} \equiv \{lec_{ij.1}, lec_{ij.2}, ..., lec_{ij.l}\}$

Each label in a labelled-TCN can be considered as a unique symbol.

Definition 3 (*Inconsistent-Label-Sets*). An *Inconsistent-Label-Set (I-L-Set)* is a set of labels $\{label_i\}$ and represents a set of inconsistent canonical constraints. That is, they cannot all simultaneously hold.

Definition 4 (*Inconsistence-Set*). An *Inconsistence-Set (I-Set)* is a set of *Inconsistent-Label-Sets* associated to a set of constraints C. It represents a set of overall inconsistent canonical constraints of C.

The general syntax for a constraint is:
$(t_i\{([d_1, D_1], \{label_1\}), ..., ([d_n, D_n], \{label_n\})\}t_j)$, with $d_i \leq D_i$, which means:
$(t_j - t_i \leq [d_1, D_1]) \vee ... \vee (t_j - t_i \leq [d_n, D_n])$.

we will use also the form $(t_i\{[d_1, D_1]_{\{label_1\}}, ..., [d_n, d_n]_{\{label_n\}}\}t_j)$. We refer to the constraint between the temporal points t_i and t_j as $t_i.t_j$.

2.1 Operations on Labelled Constraints

The main operations on labelled constraints are:

- Temporal Inclusion ($\subseteq_{lc}$), that takes into account the inclusion of temporal intervals and the inclusion of associated label sets.
- Temporal Union ($\cup_{lc}$), that performs the disjunctive temporal union of labelled constraints as the set-union of their canonical constraints. However, all labelled canonical constraints whose associated labels are *I-L-Sets* should be rejected.
- Temporal Composition ($\otimes_{lc}$), and Temporal Intersection ($\oplus_{lc}$), that are based on the operation $\otimes$ of the underlying disjunctive metric point-based algebra [9].

The operations outlined above are used into a total closure process that infers new constraints from those explicitly asserted. Given a minimal *LTCN* as input, and a new constraint to be asserted, the closure process outputs a new minimal *LTCN* [10].

3 Non-Disjunctive and Disjunctive Constraints

We refer as non-disjunctive constraints, those that can be specified using binary labelled temporal constraints with cardinality equal to 1. Some basic examples are: duration constraints (when they are done by a single temporal value or interval), precedence constraints (*"a is performed before b", "c is done after d*, and start/finish time constraints (when they are done by a single temporal value or interval, as in *"b must finish between 19:00 and 20:00"*.

We use the special temporal points T0 and TF to denote the beginning and the ending of the world, respectively. In the rest of the paper we use *on(a)*, and *off(a)* to denote the start and finish time of an action *a*.

As an example of non-disjunctive constraints, we can specify that *"the action a_2 can start x time units before a_1 finishes, moreover they must start at y time units and must finish after z time units"* as $\{(off(a_1)\{[-x, \infty[_{\{R0\}}\}on(a_2))$, $(T0\{[y, \infty[_{\{R0\}}\}on(a_1))$, $(T0\{[0, z]_{\{R0\}}\}off(a_3))\}$. If an *input* (or explicitly asserted) constraint lc_{ij} has only one canonical constraint, that is, only one disjunct, this canonical constraint has the label 'R0'. The labelled Universal Constraint is $\{U_{\{R0\}}\}$.

Disjunctive constraints are those with cardinality greater than 1. This allows us for example to specify multiple durations for a same action a. We can also refer to the fact that two actions a_1 and a_2 can be carried out in only two possible orders: a_1 before a_2, or a_2 before a_1, this occurs for example in a typical scheduling problem. In this case, we can use the general constraint:

$(on(a_1)\{[d_1, \infty[_{\{R1\}},]-\infty, d_2]_{\{R2\}}\}on(a_2))$, in which d_1 and d_2 are the durations of a_1 and a_2 respectively.If an input constraint lc_{ij} has more than one canonical constraint, each canonical constraint $lec_{ij.k} \in lc_{ij}$ has a single and exclusive label associated to it.

In the next section we explain how to specify more complex constraints using disjunctive ones.

4 COST AND MORE COMPLEX CONSTRAINTS

The labels of the model allow us incorporate several additional information that can be managed in an integrated manner with the reasoning algorithms. This is important in that we can specify more wide range of problems in the same way and solve it without change the reasoning algorithms applied. As an example of this feature, we show how to specify cost associated to resources required to carry out the corresponding actions depending on the time in which the actions are carried out. The form of a cost constraint would be: *"The resource r_k has a cost associated of x if it is used between temporal points t_1 and t_2"*.

We denote the cost of use of resource r_k by means the labelled temporal interval $[t_1, t_2]_{R_{ck}}$. In order to incorporate it into the network, we add the constraint $(t0\{[-\infty, t_1 - 1]_{\{R_a\}}, [t_1, t_2]_{\{R_{ck}\}}, [t_2 + 1, \infty]_{\{R_b\}}\}TF)$, in which R_a and R_b represent zero cost value. Thus, heuristics applied can use label R_{ck} in order to calculate the corresponding operation costs.

Due to closure process, each derived canonical constraint (it obtained by combining ($\otimes_{lc}$) or intersecting ($\oplus_{lc}$) two labelled canonical constraints) has a set of labels associated to it. This label set represents the conjunctive support-set of explicitly asserted canonical constraints. In consequence, this label set can also represent associated cost-labels. That is, the cost associated to each labelled temporal interval (by unit time) can be calculated as the sum of corresponding cost values associated to their labels: $cost(lec_{ij}) \Rightarrow \sum_{1..n} cost(l_i), \forall l_i \in labels(lec_{ij})$.

Heuristics applied must calculate the cost value of corresponding canonical constraints to select what of these are maintained or rejected by the resolution method. Note that we have incorporated the information cost in the constraints without any change over the syntax of labelled temporal constraints. This will allow us to use the same resolution method to solve the CSP problem, with or without associated costs to actions.

It is also possible to know the cost of an operation o_{ij},that uses the resource r_k, without needing of introduce any *cost label* R_{ck}. For example, suppose that o_{ij} have a duration denoted by dur_{ij} First, we retrieve the constraint between the temporal point T0 and $on(o_{ij})$ denoted by T0.$on(o_{ij})$, that represents the k possible start times $\{t_{st_1}, ..., t_{st_k}\}$ of o_{ij}. For each start time t_{st},

the corresponding cost value is calculated obtaining the temporal intersection: $[t_{st}, t_{st} + dur_{o_{ij}}] \oplus [t_1, t_2]$ and multiplying the length of resultant interval by the cost$x$ associated to temporal interval $[t_1, t_2]$

4.1 Complex disjunctive constraints

In this section, we refer to complex constraints that indicate that the duration of an action depends on a certain condition (*order dependent durations*). So, suppose that we have the non-overlapping actions a_x, a_y, and a_z, that can be carried out in whatever order. We can specify constraints such as: *"The duration of action a_x, denoted by $dur(a_x)$, is [5,8] if a_y is carried out before a_z, and $dur(a_x)$ is [12,15], if a_y is carried out after a_z"*. In this case, we need a set of constraints to specify this, (instead an unique constraint), plus extra *I-L-Sets*. The resulting constraint set is: $\{(off(a_x)\{[0,\infty[_{\{R1\}},]-\infty,-1]_{\{R2\}}\}on(a_y)),$
$(off(a_y)\{[0,\infty[_{\{R3\}},]-\infty,-1]_{\{R4\}}\}on(a_x)),$
$(off(a_x)\{[0,\infty[_{\{R5\}},]-\infty,-1]_{\{R6\}}\}on(a_z)),$
$(off(a_z)\{[0,\infty[_{\{R7\}},]-\infty,-1]_{\{R8\}}\}on(a_x)),$
$(off(a_y)\{[0,\infty[_{\{R9\}},]-\infty,-1]_{\{R10\}}\}on(a_z))$
$(off(a_z)\{[0,\infty[_{\{R11\}},]-\infty,-1]_{\{R12\}}\}on(a_y))\}$,
and the following *I-L-Sets*: {R9,R11},{R10,R12}. Note that R11 is associated to *" a_y is scheduled after a_z"* and R9 is associated to *"a_y is scheduled before a_z"*.

We can also overlap/not meet order-dependent durations between non overlapping actions that can be carried out in whatever order, such as: *"If the action a_x is executed after a_y, then a_x can be executed 10 minutes after a_y finishes"*. The constraint set that reflects this is: $\{(off(a_x)\{[0,\infty[_{\{R1\}},]-\infty,-11]_{\{R2\}}\}$ $on(a_y)), (off(a_y)\{[10,\infty[_{\{R3\}},]-\infty,-1]_{\{R4\}}\}on(a_x))$, and we need the extra *I-L-Sets*:{R2,R4},{R1,R3}. In this case R3 is associated to *"a_x is executed ten minutes after a_y"* and R2 is associated to *"a_x is carried out a_y"*.

4.2 Relationship to other temporal reasoning formalisms

The constraint class that we study is more expressive than the classes of STP constraints and TCSP constraints of [9]. For example, the constraints about multiple durations for a same action cannot be captured by the TCSP model but can be modelled naturally in our framework.

In [11], that extends the framework of simple temporal problems of [9], the constraints outlined above can be represented, but it cannot consider, for example, any information cost associated with the use of resources by the corresponding actions.

Other constraints solvers with similar expressivity are the LPSAT solver [12], which integrates propositional satisfiability and linear programming, and the formalism named TCSPP [13] which allow for reasoning about temporal preferences: it generalizes the TCSP with the addition of a semiring-based soft constraint formalism.

Both, LPSAT and TCSPP formalisms, assume that the problem constraint set is known in advance. So, the solutions obtained cannot easily changed when a new constraint have to be considered.

5 The Resolution Method

Recently, it has been proposed a new method to solve scheduling problems, that integrates effectively the CSP process into a limited closure process: not interleaving them but as a part of the same process [14]. It specifically handles a set of metric and disjunctive-based labelled temporal point constraints. A minimal LTCN is maintained as each constraint of the input constraint set is processed. It is an iterative algorithm, in which we add a new constraint each time. We generalize it in order to solve more general temporal constraint-based problems. The resultant model can be tailored by two parameters:

- The maximal number of indecisions (disjuncts) maintained in the network.
- The variable and value heuristics used to prune the search space.

The first parameter allows us to perform as a pure CSP process (if the number of indecisions is equal to zero), as a pure closure process (if this number is not limited), or as a convenient mixed closure-CSP method (obtaining a set of solutions, instead of a unique solution that results of a pure CSP process). The greater the number of indecisions maintained (disjuncts allowed), more solutions are also maintained, and fewer backtrackings are needed to obtain a solution. Moreover, we obtain a more incremental method in that we reduce the need to know in advance all constraints. This process varies itself automatically with the number of maintained disjunctions and which disjunctions are maintained in every moment.

The second parameter, the set of heuristics to apply, provides efficiency to the method in that they can take better decisions. Each input constraint acts as a variable, and each disjunction in a constraint represents a possible value for that variable. So, in each iteration we have to decide the next constraint to add (by means a variable heuristic), and which disjunctions (by means a value heuristic) have to be maintained in the network. We can create new heuristics (that we have named *mixed heuristics*), that combine the results of several ones, in order to take advantage of major information derived in the LTCN by the closure process applied. The value heuristics applied return a value set with a maximal cardinality indicated by the first parameter. So, we can maintain several disjunctions at the same time, and delay the corresponding decisions.

The result obtained applying this method allow us to obtain a set of solutions, instead of a unique solution that results of a pure CSP process.

5.1 *Closure-CSP* algorithm

The algorithm that implements the method outlined above is the following:

The Closure-CSP (C, ind, hvar, hval)

1. **For each** non-disjunctive $c_i \in$ C **do**
2. C $\leftarrow$ C $-c_i$; result $\leftarrow$ *closure*(c_i);
3. **If** not result **then** return *Inconsistent* **end-if;**
4. **End For**;
5. **While** C $\neq \emptyset$ do
6. $c_i \leftarrow$ *detect_non_disjunct*(C);
7. result $\leftarrow$ *closure*(c_i);
8. **If** not result **then** *backtracking* **end-if;**
9. C $\leftarrow$ C $-$ ND; nextDisj $\leftarrow$ *chooseVar* (C,hvar);
10. result $\leftarrow$ *closure*(nextDisj);
11. **If** not result **then** *backtracking* **end-if;**
12. decis $\leftarrow$ *chooseVal* (nextDisj, hval);
13. decis $\leftarrow$ *validate_decision* (decis, C);
14. decis $\leftarrow$ *revise_pending_decisions* (decis);
15. **If** |decis|> ind **then** decis $\leftarrow$ *restrict_dec* (decis) **end-if;**
16. result $\leftarrow$ *closure*(decis);
17. **end-while**

end Closure-CSP

C is the set of input constraints, *ind* is the maximal number of pending decisions in the network, *hvar* and *hval* are the variable and value heuristics respectively. Initially, we have an empty LTCN.

The *closure* process updates the initial LTCN, with the non-disjunctive constraints c_i (once at a time), obtaining a new minimal LTCN, or *false*, if it detects any inconsistency (in this case the problem has no solution). We record all c_i constraints added, and also we record some minimal LTCNs. When a dead-end is encountered, we returns to last minimal LTCN recorded, and add the corresponding c_i constraints to the C set (step 8). Initially, we have recorded one LTCN for each ten c_i constraints added to the network.

We apply a process named *detect_non_disjunct* (step 6), that detect disjunctive constraints from C that will convert in non disjunctive ones due to new inferred constraints obtained in step 7.

We use *chooseVar* in order to decide the next disjunctive constraint to be added to the network (*nextDisj*)from the constraint set C, and we use the variable heuristic *hvar* to choose it (step 9).

Once *nextDisj* is propagated by the *closure* process (step 10), it results a *false* value, a dead-end is encountered and we realize a chronological backtracking process.

Next, we choose a value set (that is, a set of disjunctions) from the constraint added (*nextDisj*) using the value heuristic *hval* (step 12). This heuristic decide which disjunctions have to be maintained in the network. If the heuristic decides more of one disjunction, then we have an indecision (that will correspond to several *pending decisions*). The resulting decision (*decis* has to be validated (step 13). The *validate_decision* process detects possible future inconsistences of *decis* with pending disjunctive constraints to add from the C set.

Next, we revise possible pending ordering decisions in *decis* from previous iterations. This is done by *revise_pending_decisions* (step 14). We modify the decision value, removing from *decis* possible *pending decisions*, that can be decided now. Moreover, we can maintain during a certain chosen time an indecision in the network, by fixing a limited number of iterations in order to reduce the propagation cost.

We can also modify (restrict) the decision value (*decis*) depending on maximal cardinality of the network constraints, indicated by the *ind* parameter, if it is necessary. This is done by the *restrict_dec* process in step 15.

Finally, we proceed to *closure* (step 16) the resultant decision value *decis*.

The computational cost of the *closure* process is $O(n^2l^{2e})$, in which n is the number of nodes in the network, l is the maximal number of disjunctions of input constraints, and e is the number of input constraints updated in the previous LTCN. This is the bounded cost of each iteration of the algorithm *Closure-CSP* [4]. This complexity makes infeasible to solve real problems, so we have introduced the parameter to maintain a maximal number of indecisions, and the use of convenient variable and value heuristics.

5.2 Mixed Heuristics

Due to the closure process used, that maintain a minimal LTCN, from a previous LTCN, more information is derived when we add a new constraint to the network. This allow us to use more informed heuristics.

Moreover, the value heuristics used can make a decision, several decisions, or not decide at all, in that we allow that several disjuncts can be maintained in the network (and decided later).

Considering the facts outlined above, we decide to use new heuristics that combine the results of several heuristics in order to reinforce or reject, the decision taken by them. Our idea is to take advantage of the major information propagated, in order to take better decisions avoiding backtrakings. So, we obtain more flexible partial solutions.

Therefore, we can create new heuristics that can be adapted to particular features of problem considered, or to set of solutions that we aim to obtain. For example, we can obtain a new heuristic that returns simply the intersection of the sets returned by two or more heuristics h_i. Examples of mixed heuristics can be found in [14] and [15].

6 Applying the Model

The general framework proposed can be applied to a great variety of problems that can be considered as CSP problems, such as vehicle routing, planning and scheduling. We have carried out some work with scheduling problems, and as

[4] The best case occurs when the algorithm acts as a pure CSP. Then, the closure process has a polynomial cost.

result we have been able of specifying a set of constraints not contemplated in previous approximations, such as setup and maintenance periods of the resources, consider the cost of use of resources, and others types of scheduling problems such as production lots [16].

Moreover, we have applied the Closure-CSP algorithm to solve several instances from a known benchmark of scheduling problems, and we have used also several randomly generated instances in order to analyze the resolution method. The results have shown that it is able to finding the optimal solution in most of cases, with a minimal number of backtrackings needed. Furthermore, the use of mixed heuristics, allows us to obtain better solutions than isolated heuristics.

In Figure 1 we show an experiment to demonstrate the capability of the proposed model to repair partial solutions when we add new constraints that are not known from the beginning.

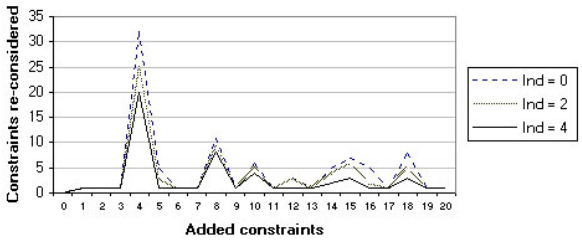

Fig. 1. Constraints re-considered in order to repair partial solutions.

We have used 15 randomly generated 10×15 job-shop instances (I_i). Each one of them has a constraint set C_i that contains 130 non-disjunctive and 225 disjunctive constraints. For each instance I_i, we choose 230 randomized constraints from the corresponding C_i set, and remove they from C_i, and solve the partial problem obtained. Next, we add successively 20 randomized constraints from the other 100 constraints of C_i set. We represent in Figure 1 the averaged number of constraints previously asserted that can be re-considered for all 15 instances, and for each constraint added. We can observe that this number is low, and it decreases when we maintain more indecisions in the network.

We have used Common Lisp with a Pentium III Computer and, for example, the time spent to solve the known ft06 (size 6×6) is 0,1 secs., and to solve ft10 (size 6×6) we spent 0,3 secs. The time spent to solve 30 randomly generated instances (size 10×5), with a bottleneck resource, and a limited makespan vary from 1,2 secs to 5 secs, allowing 0 indecisions and five indecisions respectively.

7 Conclusions

We have shown a new approximation to manage complex constraints in the framework of CSP problems. The labelled-TCSP model proposed handles metric

and disjunctive temporal constraints. This results in a more expressive model that allow us to:

The advantages of this representation over previous approaches are:

- Specify complex constraints derived from disjunctive constraints, and conditional constraints.
- Include cost information that can be managed in an integrated manner with the resolution method.
- Provide more flexible solutions: the partial solutions are easier to repair in the case of problem changes.

References

1. E. Tsang and A. Kwan. Mapping constraint satisfaction problems to algorithms and heuristics. Technical Report CSM-198, Dep. of Computer Science. Univ. of Essex, Colchester, December 1993.
2. V. Kumar. Algorithms for constraint satisfaction problems: a survey. *AI Magazine*, 13 1:32–44, 1992.
3. E. Tsang. *Foundations of constraint satisfaction*. Academic Press, Essex, 1993.
4. I. Gent, K. Stergiou, and T. Walsh. Decomposable constraints. *Artificial Intelligence*, 123:133–156, 2000.
5. P. van Hentenryck. *Constraint satisfaction in logic programming*. MIT Press. Cambridge, MA, 1992.
6. J. Cohen. Constraint logic programming languages. *Communications of ACM*, 33:52–68, 1990.
7. J. Zhou. Designing and implementing a natural constraint language for solving combinatorial problems. Technical Report 97-R-199, LORIA, Campus Scientifique, BP 239, France, 1997.
8. J. Zhou. A unified framework for solving boolean, integer and set constraints. In *Proc. of the 3th Int. Conf. on SSSE*, pages 559–570, August 1998.
9. R. Dechter, I. Meiri, and Perl J. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.
10. F. Barber. Reasoning on interval and point-based disjunctive metric constraints in temporal contexts. *Journal of Artificial Intelligence Research*, 122:35–86, 2000.
11. K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:81–117, 2000.
12. Steven A. Wolfman and Daniel S. Weld. The LPSAT engine its application to resource planning. In *IJCAI*, pages 310–317, 1999.
13. L. Khatib, P. Morris, R. Morris, and F. Rossi. Temporal constraint reasoning with preferences. In *IJCAI'2001*, August 2001.
14. M.I. Alfonso and F. Barber. A mixed closure-csp method to solve scheduling problems. In *Proceed. of the 14th IEA/AIE 2001*, Lecture Notes in A. I., pages 559–570, 2001.
15. M.I. Alfonso and F. Barber. Nuevas heursticas y medidas de textura para resolver problemas de scheduling mediante clausura y csp. In *Proceedings of the 9th CAEPIA'01*, volume 2, pages 833–842, Gijn, Spain, Nov 2001.
16. M.I. Alfonso. *Un modelo de integracion de tocnicas de clausura y CSP de restricciones temporales: aplicacion a problemas de scheduling*. PhD thesis, Dep. Ciencia de la Computacion e Inteligencia Artificial. Universidad de Alicante, Spain, 2001.

SUMMARY INFORMATION
IBERAMIA-2002 Summary Submission

Title: Complex constraints management

Authors: M. Isabel Alfonso, Federico Barber

M. Isabel Alfonso Galipienso
Dto. Ciencia de la Computacion e Intelignencia Artificial
Unversidad de Alicante
Ap. Correos, 99 E03330 Alicante Spain
Tel: +34 965 90 39 00x2516 Fax: +34 965 90 39 02
Email: eli@dccia.ua.es

Federico Barber Sanchis
Dto. Sistemas Informaticos y Computacion
Unversidad Politecnica de Valencia
Ap. Correos, 22012 E046020 Valencia Spain
Tel: +34 963 87 93 57 Fax: +34 963 87 73 59
Email: eli@dccia.ua.es

Abstract: The purpose of this paper is to show a new approximation to manage complex constraints in the framework of CSP problems. Concretely we propose the use of a labelled-CSP to specify complex temporal constraints, as a particular case of constraints. This framework allows us to specify and solve the constraint set associated to several types of CSP problems in an integrated manner. The complex constraints that we can manage are represented by a set of non-disjunctive and disjunctive constraints. We can also associate a cost with the constraints, that can affect to the obtained solution. So, the advantages of this framework are its expressiveness, that allows to specify very complex constraints which are present in real problems, and the fact that all constraints are processed in the same way

---

Keywords: Reasoning models, Temporal reasoning, Constraint satisfaction, disjunctive constraints

---

Topics: Satisfaccion de Restricciones, Planificacion y Optimizacion; Modelos de Razonamiento: Temporal

---

Section: Paper Track
