International Journal of Innovative
Computing, Information and Control
Volume **8**, Number **4**, April **2012**

ICIC International ©2012 ISSN 1349-4198
pp. **2513–2532**

ROBUSTNESS IN DYNAMIC CONSTRAINT SATISFACTION PROBLEMS

Laura Climent, Miguel A. Salido and Federico Barber

Instituto de Automática e Informática Industrial
Universidad Politécnica de Valencia
Camino de Vera s/n 46022, Valencia, Spain
{ lcliment; msalido; fbarber }@dsic.upv.es

Received January 2011; revised May 2011

Abstract. *Constraint programming is a successful technology for solving combinatorial problems modeled as constraint satisfaction problems (CSPs). Many real life problems are dynamic, which means that the initial description of the problem may change during its execution. These problems can be modeled as dynamic constraint satisfaction problems (DynCSPs), which are an important extension of the CSPs. In this paper, we focus our attention on the concept of robustness. Our aim is to find robust solutions which have a high probability of remaining valid faced with possible future changes in the constraints of the problem. We introduce the informed DynCSPs, proposing an approach to solve them by the weighted CSP (WCSP) modeling. Thus, the best solution for the modeled WCSP will be a robust solution for the original DynCSP. Furthermore, this technique has been evaluated in order to analyze the robustness of the solutions obtained.*
Keywords: Constraint satisfaction problem, Robustness, Dynamic constraints

1. **Introduction.** Constraint programming (CP) is a powerful paradigm for solving combinatorial problems [18]. CP was born as a multi-disciplinary research area that embeds techniques and notions coming from many other areas, among which artificial intelligence, computer science, databases, programming languages, and operations research play an important role. Constraint programming is currently applied with success to many domains such as scheduling, planning, vehicle routing, configuration, networks and bioinformatics. More information about constraint programming can be found in [1, 18, 22].

Many real life problems can be modeled as constraint satisfaction problems and are solved using constraint programming techniques. Much effort has been spent to increase the efficiency of the constraint satisfaction algorithms: filtering [18], learning and distributed techniques [20], the use of efficient representations and heuristics [4, 15, 16], etc. This effort resulted in the design of constraint reasoning tools which were used to solve numerous real problems.

However, many of these techniques assume that the set of variables, domains and constraints involved in the CSP is completely known and fixed when the problem is modeled. This is a strong limitation when we deal with real situations where the DynCSP under consideration may evolve because of the environment, the user or other agents [23]. Thus, the solution found for a problem can become invalid after changes in the parameters of the DynCSP.

We consider the importance of preventing the loss of a solution because it could entail several negative effects in the modeled problem. For example, in a scheduling problem composed of several machines, the loss of a solution could cause the stop of the production system, the breakage of machines, the loss of the material/object in production, etc. In

addition, these negative effects probably will have associated an economic loss. Thus, the main objective of this paper is to find robust solutions that have a high probability of remaining valid despite changes in the initial formulation of the modeled problem.

A DynCSP [5] is an extension of a static CSP that models addition and retraction of constraints and hence it is more appropriate for handling dynamic real-world problems. It is indeed easy to see that all possible modifications in constraints or domains of a DynCSP can be expressed in terms of constraint additions or removals [23].

2. Problem Statement and Preliminaries.

By following standard notations and definitions in the literature we summarize the basic definitions that will be used in the rest of the paper.

2.1. CSP preliminaries.

Definition 2.1. *A Constraint Satisfaction Problem (CSP) is represented by a triple* $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ *where:*

- $\mathcal{X}$ *is the finite set of variables* $\mathcal{X} = \{x_1, x_2, ..., x_n\}$.
- $\mathcal{D}$ *is a set of domains* $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ *such that for each variable* $x_i \in \mathcal{X}$ *there is a set of values that the variable can take.*
- $\mathcal{C}$ *is a finite set of constraints* $\mathcal{C} = \{C_1, C_2, ..., C_m\}$ *which restrict the values that the variables can simultaneously take.*

The number of tuples of a constraint C_i is composed by the elements of the Cartesian product of the domains of the variables involved in C_i. The *tightness* of a constraint is the relation of the number of forbidden tuples to the number of all possible tuples. The tightness is defined in the interval [0,1].

The constraints of the CSP determine the convex hull of the solution space (for linear constraints). Ideally, one would like to use as a relaxation the convex hull of the feasible solutions. However, in general, this polytope will have exponentially many facets and be difficult to construct [28].

The constraints of a CSP can be classified in two types: hard or soft constraints. The hard constraints have to be satisfied in all cases. However, the soft constraints do not have to be satisfied in all cases, so they can be relaxed but with a certain cost.

There are two main constraint representations, which are equivalent [2]:

- Intensional constraint representation: Constraints are represented as mathematic or logical function.

  Example 2.1. $x_1 \geq 3$ *is a unary intensional constraint.*

- Extensional constraint representation: Constraints are represented as a set of valid or invalid tuples.

  Example 2.2. *The set of tuples* $\{(3), (4), (5)\}$ *is the extensional representation of the constraint* $x_1 \geq 3$ *by means of valid tuples, considering the domain* $D_1 : \{0..5\}$ *for* x_1.

Definition 2.2. *A Dynamic Constraint Satisfaction Problem (DynCSP)* [5, 9] *is a sequence of static CSPs* $\langle CSP_{(0)}, CSP_{(1)}, ..., CSP_{(i+1)} \rangle$, *each* $CSP_{(i)}$ *resulting from a change in the previous one* ($CSP_{(i-1)}$), *representing new facts about the dynamic environment being modeled. As a result of such incremental change, the set of solutions of each* $CSP_{(i)}$ *can potentially decrease (in which case is considered a restriction) or increase (in which case is considered a relaxation).*

Restrictions occur when new constraints are imposed on a subset of existing variables (e.g., forcing a variable to assume a certain value). Relaxations occur when constraints that were assumed to hold are removed from the CSP.

Definition 2.3. *A Weighted Constraint Satisfaction Problem (WCSP) is a specific subclass of valued CSP* [8, 13, 21]. *A WCSP is defined as* $P = \langle \mathcal{X}, \mathcal{D}, S(k), \mathcal{C} \rangle$, *where:*

- $\mathcal{X}$ *and* $\mathcal{D}$ *are the set of variables and domains respectively, as in standard CSP.*
- $S(k)$ *is the valuation structure, where* $k \in \mathbb{N}^+$ *denotes the maximum cost.*
- $\mathcal{C}$ *is the set of constraints as cost functions (namely,* $C_i : \prod_{j \in var(C_i)} D_j \to \{0, 1, ..., k\}$*), where* $C_i \in \mathcal{C}$ *is defined over a subset of variables* $var(C_i)$.

Definition 2.4. *A valuation structure is defined as* $S(k) = (\{0, 1, ..., k\}, \oplus, >)$ *where*

- $\{0, 1, ..., k\}$ *is the set of costs, which are natural numbers bounded by* k *and totally ordered by* $>$.
- $\oplus$ *is the sum of costs.*
  $\forall a, b \in \{0, 1, ..., k\},\ a \oplus b = \min\{k, a + b\}$

A tuple t is an assignment to an ordered set of variables $\mathcal{X}_t \subseteq \mathcal{X}$. For a subset B of $\mathcal{X}_t$, the projection of t over B is noted as $t \downarrow_B$.

When C_i assigns a cost k to a tuple t (composed by $var(C_i)$), it means that t is a invalid tuple for C_i. Otherwise (the cost assigned is lower than k) t is a valid tuple for C_i with the corresponding associated cost.

The cost of a tuple t, noted $\mathcal{V}(t)$, is the sum of all the applicable costs:

$$\mathcal{V}(t) = \bigoplus_{C_i \in \mathcal{C}, var(C_i) \subseteq X_t} C_i(t \downarrow_{var(C_i)})$$

The tuple t is *consistent* if $\mathcal{V}(t) < k$. The main objective is to find a complete assignment with the minimum cost.

Example 2.3. *Figure 1 shows a WCSP* $P = \langle \{x, y\}, \{\{v_1, v_2\}, \{v_1, v_2\}\}, S(5), \{C_1, C_2\} \rangle$. *Observe that the set of costs is [0,...,5]. Let's assume* $var(C_1) = \{x, y\}$ *and* $var(C_2) = \{x, y\}$. *The costs assigned by the constraints are represented as labeled edges connecting the values of the tuples involved in the corresponding constraint. The cost assignment of the constraint* C_1 *is represented in Figure 1(a) and for the constraint* C_2, *it is represented in Figure 1(b).*

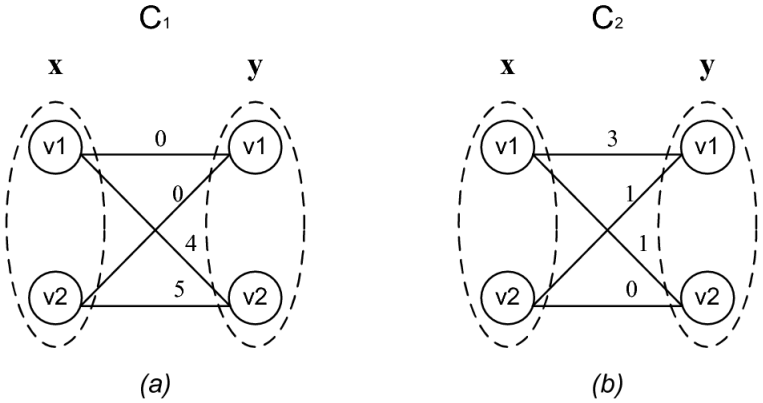

FIGURE 1. WCSP P

Table 1 shows the set of tuples of P with their corresponding assigned costs by the constraints and their $\mathcal{V}(t)$ values. In addition, it shows which tuples are solutions for P. The tuples $(x = v_1, y = v_2)$ and $(x = v_2, y = v_2)$ are not solutions of P due to their values of $\mathcal{V}(t)$ are not lower than 5. However, the tuples $(x = v_1, y = v_1)$ and $(x = v_2, y = v_1)$ are solutions of P. The best solution for P is $(x = v_2, y = v_1)$ because $\mathcal{V}(x = v_2, y = v_1) = 1$, which is the minimum global cost of the problem.

TABLE 1. Set of tuples of P and their corresponding costs

x	y	Cost C_1	Cost C_2	$\mathcal{V}(t)$	Solution?
v_1	v_1	0	3	3	Yes
v_1	v_2	4	1	5	No
v_2	v_1	0	1	1	Yes
v_2	v_2	5	0	5	No

2.2. **Stability and robustness.** Regarding the concept of robustness and stability of the solutions of the DynCSPs, there is a misunderstanding between the two concepts. Some researchers talk about stability and others about robustness. But, What is the difference between stable and robust? It is the first question that comes to mind, especially for researchers who work with quantitative models or mathematical theories.

In general, a solution is stable in a dynamic system, if by means of a few changes in the solution we can obtain a new solution that is similar to the original one. However, the robustness concept is broader than the stability concept. Robustness is a measure of feature persistence in systems that compels us to focus on perturbations because they represent changes in the composition or topology of the system. The perturbations are small differences in the actual state of the system [12].

Taking into account all these concepts, we can classify the nature of the solutions as:

Definition 2.5. *The stability (also called flexibility) of a solution is the ability of a solution to share as many values as possible with a new solution if a change occurs* [10]*. It is measured in terms of similarity of the new solution with the original one.*

Definition 2.6. *The robustness of a solution is the measure of the persistence of the solution after modifications in the original DynCSP. Thus, a solution of a DynCSP is robust if it has a high probability of remaining valid faced with changes in the problem. It is measured in terms of the persistence of the solution.*

2.3. **A toy example of a scheduling problem.**

Example 2.4. *Following, we present a toy example of a scheduling problem. It can be considered a dynamic problem since constraints may change. We model this problem as a DynCSP and we present different solutions with different robustness degree. In this scheduling problem, two activities A and B have to finish in 8 hours or sooner:* $A_{end} \leq 8$*;* $B_{end} \leq 8$*. In addition, the activity B has to start at least 3 hours after the end of the activity A:* $B_{begin} - A_{end} \geq 3$*. The duration of the activity A is greater than or equal to 2 hours:* $A_{end} - A_{begin} \geq 2$*. The duration of the activity B is greater than or equal to 1 hour:* $B_{end} - B_{begin} \geq 1$*. The original DynCSP that models the scheduling problem is* $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$*, where:*

- $\mathcal{X} = \{A_{begin}, A_{end}, B_{begin}, B_{end}\}$
- $\mathcal{D} = \{D_{A_{begin}} : [0, 8], D_{A_{end}} : [0, 8], D_{B_{begin}} : [0, 8], D_{B_{end}} : [0, 8]\}$

- $\mathcal{C} = \{C_1 : A_{end} \leq 8$
  $C_2 : B_{end} \leq 8$
  $C_3 : B_{begin} - A_{end} \geq 3$
  $C_4 : A_{end} - A_{begin} \geq 2$
  $C_5 : B_{end} - B_{begin} \geq 1\}$

A solution for P is: $S_1 = \{A_{begin} = 0, A_{end} = 2, B_{begin} = 5, B_{end} = 6\}$. Figure 2 shows the schedule that represents this solution.

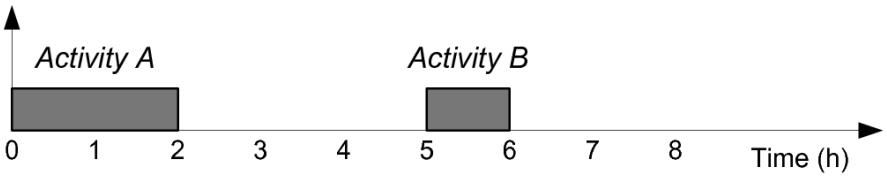

FIGURE 2. Schedule that represents the solution S_1

In real life problems, delays can occur in activities of the schedule. Let's consider a delay of 1 hour over the minimum expected duration of the first activity ($A_{end} - A_{begin} = 3$). After this change, the initial solution S_1 becomes invalid since it does not satisfy the constraint $C_3 : B_{begin} - A_{end} \geq 3$.

Following the initial modeling of the scheduling problem, and considering that the duration of all the activities might be extended, we can restrict the problem, by restricting the constraints associated to the duration of the activities. In this instance, we restrict the constraints C_4 and C_5 in: $C'_4 : A_{end} - A_{begin} \geq 3$; $C'_5 : B_{end} - B_{begin} \geq 2$. A solution for the restricted P could be $S_2 = \{A_{begin} = 0, A_{end} = 3, B_{begin} = 6, B_{end} = 8\}$. Figure 3 shows the schedule that represents this solution. The new solution obtained S_2 can be considered more robust than the original one S_1 because if a delay occurs (≤ 1 hour) in any activity, the solution S_2 remains valid.

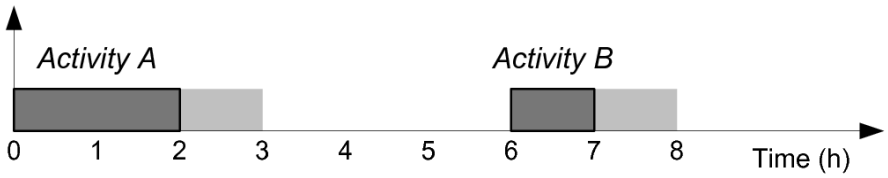

FIGURE 3. Schedule that represents the solution S_2

In scheduling problems, the most common way to generate robust schedules is by including buffers between activities. These buffers can absorb delays and avoid their propagation over the schedule [3]. However, the buffers also decrease the optimality of the schedule, due to they increase the makespan (the time difference between the start and finish of the sequence of activities).

Figure 2 shows the first schedule, whose makespan is 6 hours. Figure 3 shows the second schedule, whose makespan is 8 hours because it is composed of two buffers of one hour each one. Therefore, the second schedule is more robust than the first one, although the first one has the optimal makespan.

In many dynamic problems, there exists also an optimality criterion. That is why it is necessary to find a trade-off between robustness and optimality.

3. **Related Work.** Real world is dynamic and uncertain in its nature. It is common to find dynamic systems with time delays, failures, etc. [14, 27]. In addition, a complete knowledge about the dynamism in the system is unknown. In many cases, there exists uncertain data about possible changes from the real world [24]. The modeling of these systems is a hard task due to the uncertainty of these systems. However, the DynCSPs offer us the possibility of modeling this uncertainty. Thus, new techniques have been proposed to solve DynCSPs [24]. They can be classified in: reactive and proactive strategies.

3.1. **Reactive strategies.** The reactive methods reuse the original solutions once they are not valid because of the changes in the dynamic problem. These strategies are oriented to find a new solution which is as similar as possible to the invalid original solution or to repair it by making the minimum changes in it.

The reactive strategies can be classified in two groups:

- *Heuristic methods*, which use information about the affected parts of any previous consistent assignment (complete or not) as a heuristic for solving the current DynCSP [26].
- *Local repair methods*, which use any previous consistent assignment (complete or not) and repair it, by means of a sequence of local modifications [19, 23].

Some reactive strategies from the literature, focus their attention on the analysis of dynamic scheduling problems and they have been designed to minimally reconfigure schedules in response to a dynamic environment [19, 23]. In these environments, external factors have caused that the existent schedule becomes invalid, it may be due to the withdrawal of resources, the arrival of new resources or because of changes in the set of the scheduled activities. These techniques search a new schedule that differs minimally from the original one, since it is no longer valid after changes that have occurred in the problem. Thus, these methods firstly, model the problem as a DynCSP and secondly, apply heuristics or local repair methods. Some examples of this type of techniques are:

- An example of a technique applied to dynamic scheduling problems is based on the reduction of the *contention* in the constraints [19]. A constraint has *contention* when certain combinations of domain values of the variables may violate the constraint. For some constraints, *contention* can be measured, so that the search can be oriented to regions where the *contention* is small. Once the feasibility phase of the resources has been completed, the makespan optimization phase is executed.
- An example of a technique applied to dynamic scheduling problems in which new activities are introduced [23]. It is based on the idea that it is possible to enter a new task t iff there exists for t a location such that all the tasks whose location is incompatible with t's location can be removed and entered again one after another, without modifying t's location.

In addition, there are other reactive strategies that focus their attention on the DynCSPs in a general area of application. Thus, they simulate possible changes in the DynCSPs, trying to find solutions as similar as possible to the previous ones (solutions found before changes occur) [26].

The main advantage that our technique presents over the reactive strategies is that our technique prevents the loss of the original solution, by finding robust solutions. Limiting as much as possible the need of finding new solutions is an important issue, specially in dynamic problems that undergo frequent changes.

3.2. **Proactive strategies.** The proactive strategies use associated knowledge about the dynamism of the problem in order to find stable or robust solutions. They can be classified depending on the kind of solutions that they produce:

3.2.1. *Searching for stable solutions.* In the field of searching for stable solutions, there exist techniques [10, 11] that search for super-solutions. They perform a search task before changes occur in the problem, in order to be able to repair the invalid solution after changes occur. Informally, a solution is a super-solution if it is possible to repair the solution with only a few changes when a small number of variables lose their values. Formally, a solution is a *(a, b)*-super-solution if the loss of values of a variables at most, can be repaired by assigning other values to these variables, and changing the values of b variables at most. Deciding if a CSP has a *(a, b)*-super-solution is NP-complete for any a fixed. Mainly, the works developed in this area, focus their attention in the search of (1,0)-super-solutions.

Example 3.1. *Let's consider the following CSP:*

$x_0, x_1 \in \{1, 2, 3\}$
$C_1 : x_0 \leq x_1$

- The solution $(x_0 = 1, x_1 = 1)$ is not a (1,0)-super-solution, because if the variable x_0 loses its value 1, it is not possible to find another value for the variable that is consistent with x_1, since $(x_0 = 2, x_1 = 1)$ and $(x_0 = 3, x_1 = 1)$ are not solutions of the problem.
- The solution $(x_0 = 1, x_1 = 2)$ is a (1,0)-super-solution, because if any variable loses its value can be found at least one value to be compatible with the other variable. If x_0 loses its value 1, a value of 2 can be assigned to x_0, since $(x_0 = 2, x_1 = 2)$ is solution of the problem. If x_1 loses its value 2, a value of 1 or 3 can be assigned to x_1, since $(x_0 = 1, x_1 = 1)$ and $(x_0 = 1, x_1 = 3)$ are solutions of the problem.

In order to find super-solutions, these techniques develop search algorithms based on super-consistency (adaptations of arc-consistency techniques for super-solutions). However, finding (1,0)-super-solutions can be very difficult because (1) the existence of a *backbone variable* (a variable that takes the same value in all solutions) ensures that there not exist (1,0)-super-solutions, and (2) it is strange to find (1,0)-super-solutions where all the variables can be repaired. In this way, these points represent a disadvantage with respect to the technique developed in this paper.

3.2.2. *Searching for robust solutions.* The frameworks that have been proposed for finding robust solutions differ mainly in the assumption of the changes that may occur. On the one hand, some techniques focus their attention in possible changes in the variables and the values that they can take. On the other hand, other techniques focus their attention in possible changes in the constraints.

An example of the first set of approaches can be seen in [25]. This technique solves a restricted class of *recurrent DynCSPs* in which the values of the variables could be lost temporarily. A *recurrent DynCSP* is a DynCSP whose changes are temporal, so they may occur repeatedly and with different frequencies. The technique consists basically on penalizing the values that are no longer valid due to changes in the problem. Thus, although these values can be used later, the algorithm will try to find solutions that do not include them. This idea is incorporated in a technique of *hill-climbing*, where the penalties are used by *min-conflicts* to make the selection of values for the variables.

An example of the second set of approaches can be seen in [6], where there is a probability of existence associated with each constraint. The most robust solution is the solution that maximizes the probability of satisfying the constraints.

The technique explained in this paper, is a proactive strategy that searches for robust solutions and it focus its attention in possible future changes in the constraints. The main difference with respect to [6] is that we consider restrictive modifications over the

original constraints as possible changes. Our model is able to capture many changes that undergo real problems: incidences, delays, the decrease of resources, etc. Because these changes in the problem represent restrictive changes in the constraints. For example, in the railway timetabling field could be applied in order to generate timetables that can absorb incidences/delays of trains.

In this way, depending on the associated information about the dynamism to the problem, some approaches could be used and others not. Note that regarding the concept of robustness, there is not a general measure of it. Depending on the knowledge about the possible changes, the robustness measure will differ.

4. **Modeling Informed DynCSPs as WCSPs.** The aim of modeling DynCSPs as WCSPs is for obtaining robust solutions that have a high probability of remaining valid faced with future changes in the initial formulation of the modeled DynCSP. In this paper, we focus our attention on DynCSPs whose constraints are dynamic. As it has been explained previously, the DynCSP captures the constraint additions and removals over the original formulation of the problem. The constraint deletion can not invalidate a solution because it only can relax the problem. That is why this kind of changes in the constraints of the problem is not analyzed in the search process of robust solutions. However, a modification in a constraint could invalidate a solution if it is a restrictive modification over an original constraint. A modification in a constraint can be expressed as a deletion of the original constraint and the addition of a more restricted constraint.

In order to find robust solutions for these kind of DynCSPs, a set of new modified constraints is generated. They represent possible future changes in the original constraints. The new constraints are considered soft constraints, since they can be unsatisfied. However, the more new constraints satisfies a solution, the more robust it is. Because this solution will remain valid even if these restrictive modifications over the original constraints happen.

In order to clarify the explanation of our approach and without loss of generality, we will show a DynCSP with one dynamic constraint. Figure 4 shows a DynCSP called R, which it is composed of two variables x_0 and x_1 with domains $D_0 : \{3..7\}$ and $D_1 : \{2..6\}$ respectively (discontinuous lines), and four constraints (continuous lines):

- $C_1 : x_0 + x_1 - 12 \leq 0$ (dynamic constraint).
- $C_2 : x_1 - x_0 - 2 \leq 0$ (static constraint).
- $C_3 : x_1 + x_0 - 6 \geq 0$ (static constraint).
- $C_4 : x_0 - x_1 - 4 \leq 0$ (static constraint).

Since C_1 is a dynamic constraint, a set of new modified constraints over the constraint C_1, is generated. In this instance, this set is composed by C_{11} and C_{12} (see Figure 4). These constraints are considered as soft constraints, since they can be unsatisfied. However, the solutions that satisfy C_{11} and C_{12} have more probability of remaining valid faced with changes in C_1. Note that the solution $(x_0 = 5, x_1 = 4)$ is more robust than the solution $(x_0 = 6, x_1 = 6)$. Because $(x_0 = 5, x_1 = 4)$ satisfies C_{11} and C_{12}, but $(x_0 = 6, x_1 = 6)$ does not.

In dynamic problems, in which their constraints undergo changes, usually there is information associated to the problems that shows how much significant are these changes. We define this type of DynCSP as *informed DynCSPs*.

Definition 4.1. *An informed DynCSP is a dynamic constraint satisfaction problem with additional information associated to each constraint. This information is related to the dynamism of the constraints and it is provided by the user.*

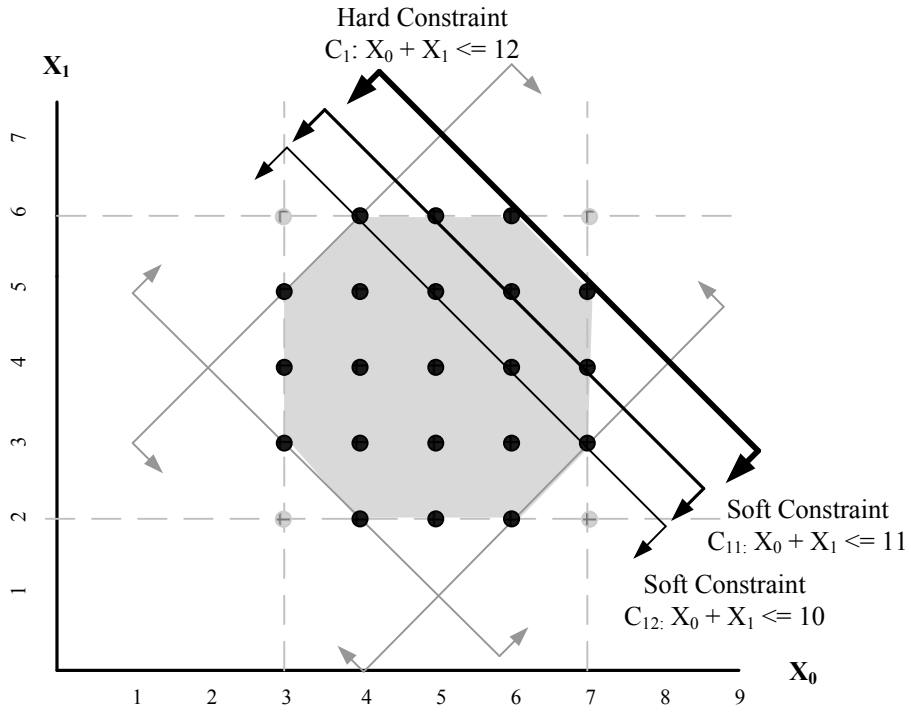

FIGURE 4. Example of the DynCSP R and new modified constraints generated over C_1

Figure 5 shows the necessary steps to model and solve an *informed DynCSP* as a WCSP. Initially, the dynamic problem is modeled as an *informed DynCSP* (P). Then, by using the additional information associated to each constraint, a WCSP is generated ($modP$). This modeled WCSP is composed of the original constraints of P and the sets of new modified constraints generated. The constraints are represented as costs functions that assign costs to the tuples involved in each constraint. Finally, $modP$ is solved by a general WCSP solver. The solutions obtained for $modP$ are the solutions of P. Furthermore, the best solution for $modP$ is considered to be one of the most robust solutions for the original informed DynCSP.

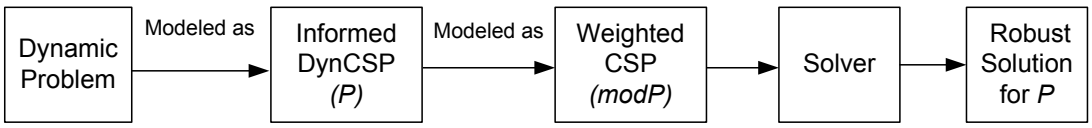

FIGURE 5. Modeling a DynCSP as a WCSP

4.1. **Dynamism functions.** We define two functions that model the dynamism of the constraints of the *informed DynCSPs*:

- $p(C_i)$: Each constraint C_i has a probability $p(C_i) \in [0, 1]$ that measures the probability of change of this constraint. It is called dynamism likelihood function. The minimum value ($p(C_i) = 0$) means that the constraint is static so there is no probability that it changes. The maximum value ($p(C_i) \approx 1$) means that the constraint is very dynamic so the probability that it changes is very high.
- $d(C_i)$: This function measures the magnitude of change for a dynamic constraint. Thus, $d(C_i) \in [0, 1]$ measures the percentage of tuples that will be invalid after a future change in the constraint. A value of $d(C_i) \approx 0$ means that the constraint C_i

would not change and almost all the valid tuples remain valid, meanwhile $d(C_i) \approx 1$ means that almost all valid tuples will be invalid after the change in the constraint. This function is not defined for static constraints (constraints with $p(C_i) = 0$).

There are several criterions for selecting the set of valid tuples that will be invalid for the new constraints generated. In this paper, without loss of generality, the criterion for intensional constraints is that the nearest tuples from the bounds of a constraint C_i are the tuples that will become invalid for the new constraints generated over C_i. Because they have a low probability of remaining valid after a change in C_i. Note that this criterion can not be used for extensional constraints, due to the nature of their representation.

The function that measures the magnitude of change $d(C_i)$ will be used for the generation of new modified constraints whereas the function that measures the probability of change $p(C_i)$ will be used in the cost assignment to the tuples.

4.2. **New constraint generation.** The algorithm for modeling an *informed DynCSP* as a WCSP is described in Algorithm 1. In this algorithm, (lines 6-7) is represented the generation of the set of new modified constraints ordered by their tightness $\{C_{i1}, C_{i2}, ..., C_{iw}\}$ for each original constraint $C_i \in \mathcal{C}$, where w is the number of constraints additions for each C_i of the original *informed DynCSP* P (P and w are provided by the user). The sets of new constraints represent possible modifications over each original constraint, based on their dynamism functions.

The set of valid tuples of a constraint C_i is denoted as $T(C_i)$. Each new generated constraint C_{ij} is composed of a subset of $T(C_i)$. In this way, C_{ij} is a more restricted version of C_i. Each original constraint C_i is considered as a hard constraint, meanwhile the set of new constraints generated $\{C_{i1}, C_{i2}, .., C_{iw}\}$ are considered as soft constraints.

4.2.1. *Properties of the new constraints.*

1. $\forall i \in \{1, .., m\}$ $T(C_{i1})$ is a subset of the set of $T(C_i)$, where m is the number of original constraints.
2. $\forall i \in \{1, .., m\}$ *if* C_{i1} is consistent then C_i is consistent.
3. $\forall i \in \{1, .., m\}$ $\{T(C_{ij}) : j \in \{2, .., w\}\}$ is a subset of $T(C_{i(j-1)})$.
4. $\forall i \in \{1, .., m\}$ *if* $\{C_{ij} : j \in \{2, .., w\}\}$ is consistent then $C_{i(j-1)}$ is consistent.

The number of tuples that become invalid for each generated C_{ij} regarding to $C_{i(j-1)}$ is $\lfloor (d(C_i) * T(C_i))/w \rfloor$.

4.3. **Cost assignment.** The associated cost to the tuples of the constraints is determined by the cost functions. There are two types of cost functions, depending on whether the constraint is hard (C_i) or soft (C_{ij}). Algorithm 1 shows the cost functions associated to hard constraints (lines 3-5) and soft constraints (lines 8-11).

The main utility of the cost function applied to an original constraint $C_i \in \mathcal{C}$ is to forbid the tuples that do not satisfy an original constraint. Thus, $C_i(t \downarrow_{var(C_i)})$ assigns a cost k to the tuple t if it does not satisfy C_i. The value k is an upper bound cost provided by the user. The cost assigned to a tuple $t \in T(C_i)$ is 0.

The cost function applied to the new constraints $C_{ij}(t \downarrow_{var(C_{ij})})$ allows us to prioritize among each set of new generated constraints based on $p(C_i)$. Thus, all the constraints belonging to an ordered set will have the same cost function. The cost assigned to a tuple $t \in T(C_{ij})$ is 0. Instead, for $t \notin T(C_{ij})$, $C_{ij}(t \downarrow_{var(C_{ij})})$ assigns to t a cost of $\lfloor p(C_i) * 100 \rfloor$, which is always lower than k, allowing t to be a solution of the problem (because C_{ij} is a soft constraint). Note that a high cost is assigned to the invalid tuples of the constraints C_{ij} whose $p(C_i)$ associated is high. Thus, we are penalizing strongly these tuples due to it is very probable that a constraint C_i undergo changes that invalidate them.

4.4. **Objective.** Given an *informed DynCSP* $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, the generation of new constraints and the cost assignment to the tuples of every constraint is carried out in order to generate the WCSP (*modP*), which can be solved by a generic WCSP solver. Algorithm 1 (line 12) shows the generation of $modP = \langle \mathcal{X}, \mathcal{D}, S(k), \mathcal{C}' \rangle$ which is composed of the original variables, the initial domains and the set of constraints $\mathcal{C}'$, being $\mathcal{C}'$ the union of the original constraints plus the new generated constraints ($\mathcal{C}' = \{C_i \cup \{C_{i1}, C_{i2}, .., C_{iw}\}\}, i \in \{1, .., m\}$).

We generate *modP* with the aim of finding solutions that satisfy the maximum number of new constraints of *modP* considering their priorities. Thus, these solutions have the highest probability of remaining valid after modifications in the original constraints of the *informed DynCSP*. The priority of each set of new generated constraints is expressed in terms of the cost assigned to their invalid tuples. A high cost means a high penalization for the tuples that do not satisfy such constraint.

The set of solutions obtained for *modP* is the set of solutions of P. In addition, the best solution for the WCSP modeled is the solution s with the minimum $\mathcal{V}(s)$ cost, which it means that s has the minimum global associated penalization. Algorithm 1 (lines 14-16) shows the best solution (if it exists) for *modP*. This solution is considered to be one of the most robust solutions for the original *informed DynCSP*, according to its dynamism functions.

Algorithm 1: Modeling an *informed DynCSP* as a WCSP and the solving process.

Data: An *informed DynCSP* $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, $p(C_i)$ and $d(C_i)$ functions, w parameter and k.
Result: Robust solution Sol and $\mathcal{V}(Sol)$.
begin

```
1   foreach C_i ∈ C do
2       foreach t ∈ C_i do
3           if t ∈ T(C_i) then
4               C_i(t ↓_var(C_i)) = 0;
            else
5               C_i(t ↓_var(C_i)) =k;
6       foreach  j ∈ {1, .., w} do
7           Generate {C_ij} based on d(C_i);
8           foreach t ∈ C_ij do
9               if t ∈ T(C_ij) then
10                  C_ij(t ↓_var(C_ij)) = 0;
                else
11                  C_ij(t ↓_var(C_ij)) = ⌊p(C_i) * 100⌋;
12  Generate modP = ⟨X, D, S(k), C'⟩ where,
    C' = {C_i ∪ {C_i1, C_i2, .., C_iw}}, i ∈ {1, .., m} ;
13  Sol ← Solve modP;
14  if ∃ Sol then
15      return (Sol, V(Sol));
    else
16      return (No Solution exists);
```

end

4.5. **WCSP file format specification.** The format used for the specification of the WCSPs is the WCSP file format. The following explanation of the format can be found

in [17]. The WCSP format is a simple format which should be easy to parse by WCSP solvers. It is composed of a list of numerical terms, except for the first one which defines the problem name, separated by space, tabulation or end of line (see Table 2 (right)). Instead of using names for making reference to variables, variable indexes are employed. The same for domain values. All indexes start at zero. All the constraints are defined in extension, by their list of tuples. A default cost value is defined per constraint in order to reduce the size of the list. Only tuples with a different cost value should be given. All the cost values must be positive. The structure of the format is: first, the problem name and dimensions, then the definition of the variables, and finally, the definition of the constraints. Files typically have the ending *.wcsp*.

A file in the wcsp format starts with the prologue:
<Problem name> $<N>$ $<K>$ $<C>$ $<UB>$, where

- $<N>$ is the number of variables (integer).
- $<K>$ is the maximum domain size (integer).
- $<C>$ is the total number of constraints (integer).
- $<UB>$ is the global upper bound of the problem (long integer).

The prologue is followed by the variable specifications:

<domain size of variable with index 0> ... <domain size of variable with index N-1>

The constraints are specified as follows (in one line):

<Arity of the constraint>
<Index of the first variable in the scope of the constraint>
...
<Index of the last variable in the scope of the constraint>
<Default cost value>
<Number of tuples with a cost different than the default cost>

And for every tuple (again in one line):

<Index of the value assigned to the first variable in the scope>
...
<Index of the value assigned to the last variable in the scope>
<Cost of the tuple>

There can be several constraints with the same scope (the solver should combine them into one constraint). The arity of a constraint may be equal to zero. In this case, there are no tuples and the default cost value is added to the total solution cost. This can be used to represent a global lower bound of the problem. The goal is to find an assignment of all the variables with minimum cost, strictly lower than the global upper bound *UB*. Tuples with a cost greater than or equal to *UB* are forbidden (hard constraint).

4.6. **Computational cost.** The complexity of the modeling phase of Algorithm 1 (lines 1-12) is related to the number of valid tuples of the constraints, the number of constraints of the DynCSP and the w parameter. The WCSP file format allows the assignment of a default cost to the invalid tuples of a constraint. Thus, it is just necessary to assign a cost to its valid tuples.

A set of w new modified constraints is generated over each original constraint $C_i \in \mathcal{C}$. The union of the original and new constraints is denoted as $\mathcal{C}'$. For each constraint of $\mathcal{C}'$, a cost is assigned to each valid tuple. Thus, the modeling phase of Algorithm 1 is $O(|T(C)| * |\mathcal{C}'|)$, which is equivalent to $O(|T(C)| * (w+1) * |\mathcal{C}|)$. The term $|\mathcal{C}'|$ denotes the number of constraints of $\mathcal{C}'$ and |T(C)| is the number of valid tuples of the constraints

(see Equation (1)). The arity of the original constraints is denoted as $|C|$ and the size of the domains is denoted as $|\mathcal{D}|$.

$$|T(C)| = (|\mathcal{D}|^{|C|}) * (1 - tightness) \tag{1}$$

Following, we present an example of an *informed DynCSP* and how it is modeled as a WCSP.

Example 4.1. *Let's P be a DynCSP with two variables x_0 and x_1 with domains $D_0 : \{3..7\}$ and $D_1 : \{2..6\}$, respectively. The dynamic constraints with their corresponding dynamism functions are:*

- $C_1(0.2, 0.2) : x_0 + x_1 \leq 12$
- $C_2(0.8, 0.4) : x_1 + x_0 \geq 6$
- $C_3(0.4, 0.3) : x_1 - x_0 \leq 2$
- $C_4(0.2, 0.4) : x_0 - x_1 \leq 4$

Each constraint is labeled with two real numbers. The first number between the parenthesis represents the dynamism likelihood $p(C_i)$ and the second the magnitude of change $d(C_i)$. We assume that $w = 1$ and $k = 100000000$.

Figure 6 shows the representation of P. In this specific example, the problem is composed by binary constraints. However, our method can be applied to problems of any arity.

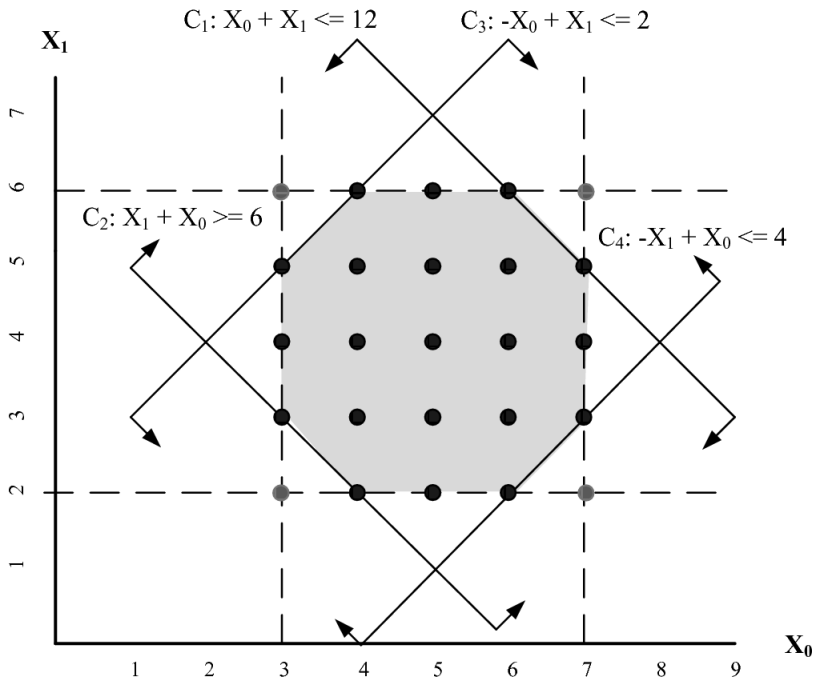

FIGURE 6. Example of the informed DynCSP P

Following the steps presented in Figure 5 and Algorithm 1, the initial representation of P (Table 2 (left)) is translated into a WCSP ($modP$), which it is composed of 8 constraints. Table 2 (right) shows the representation of $modP$. As it has been explained previously, all the valid tuples of the constraints have an associated cost of 0. The first constraint of the problem (C_1) has 2 variables, being the variable x_0 the first one and the variable x_1 the last one. This constraint has 24 valid tuples. The rest of the tuples are not allowed

due to their cost is k (default cost). The first valid tuple of C_1 is $(x_0 = 3, x_1 = 2)$ and the the last one is $(x_0 = 7, x_1 = 5)$. The last constraint (C_{41}) has also the same 2 variables. However, C_{41} has 14 tuples that satisfy it (with an associated cost of 0). The default cost of C_{41} is 20.

TABLE 2. Original informed DynCSP (P) (left) and $modP$ (right)

P	$modP$
Variables: $X_0..X_1$ Domain: $D_0 : 3-7$ $D_1 : 2-6$	filename.wcsp 2 2 8 8 100000000 8 7
$C_1(0.2, 0.2) : x_0 + x_1 \leq 12$	2 0 1 100000000 24 {(3 2 0), (3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0),(4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (5 6 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0)}
$T(C_{11}) = \lfloor(0.8 * T(C_1))\rfloor = 19$	2 0 1 20 19 {(3 2 0), (3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (6 2 0), (6 3 0), (6 4 0), (7 2 0), (7 3 0)}
$C_2(0.8, 0.4) : x_1 + x_0 \geq 6$	2 1 0 100000000 24 {(3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (5 6 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)}
$T(C_{21}) = \lfloor(0.6 * T(C_2))\rfloor = 14$	2 1 0 80 14 {(4 5 0), (4 6 0), (5 4 0), (5 5 0), (5 6 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)}
$C_3(0.4, 0.3) : x_1 - x_0 \leq 2$	2 1 0 100000000 24 {(3 2 0), (3 3 0), (3 4 0), (3 5 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (5 6 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)}
$T(C_{31}) = \lfloor(0.7 * T(C_3))\rfloor = 16$	2 1 0 40 16 {(3 2 0), (4 2 0), (4 3 0), (5 2 0), (5 3 0), (5 4 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)}
$C_4(0.2, 0.4) : x_0 - x_1 \leq 4$	2 0 1 100000000 24 {(3 2 0),(3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (5 6 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0),(6 6 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)}
$T(C_{41}) = \lfloor(0.6 * T(C_4))\rfloor = 14$	2 0 1 20 14 {(3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 4 0), (5 5 0), (5 6 0), (6 5 0),(6 6 0), (7 6 0)}

As we have pointed out, the WCSP solver finds as best solution, the solution s with the minimum value of $\mathcal{V}(s)$ associated. The best solution for $modP$ is $(x_0 = 5, x_1 = 4)$, because $\mathcal{V}(x_0 = 5, x_1 = 4) = 0$. This solution satisfies all the set of new modified constraints (4 constraints). For this reason, it is one of the most robust solutions for P. However, the solution $(x_0 = 3, x_1 = 3)$ that is one of the solutions provided by a usual CSP solver, just satisfies 2 of the 4 new modified constraints of $modP$. It does not satisfy the constraints C_{21} and C_{31}. The cost assigned to a tuple that does not satisfy C_{21} is 80. And the cost assigned to a tuple that does not satisfy C_{31} is 40. That is why, $\mathcal{V}(x_0 = 3, x_1 = 3) = 120$. This solution is an unrobust solution for P since it has a low probability of remaining valid faced with changes in the constraints C_2 and C_3.

5. **Evaluation.** The evaluation of the technique of modeling *informed DynCSPs* as WCSPs, consists in the analysis of the robustness of the solutions obtained for the WCSP modeled by our technique and solutions obtained by a usual CSP solver. The analyzed solution of the set of solutions provided by the CSP solver is the solution with the lowest values for its variables.

Furthermore, we have analyzed the effect of the *informed DynCSP* parameters and the dynamism functions in the robustness of the solutions obtained. We have developed the evaluation with random problems and with benchmarks from the literature.

The level of robustness of a solution s of an *informed DynCSP*, is indirectly related to the value of $\mathcal{V}(s)$. In addition, the number of new modified constraints satisfied by the solutions is another robustness measure. The higher this percentage is, the more robust the solution is, due to the solution is able to remain valid after a high number of changes in the constraints of the problem.

We have used the WCSP solver ToulBar2[1] for solving the modeled WCSPs. And the cut-off time has been 600 seconds for each instance.

5.1. **Random problems.** The evaluation has been developed by generating 100 random instances for each problem, by a fixed $w = 1$. The problems were randomly generated by increasing the number of variables, the domain size, the number of constraints and the tightness of the constraints.

Table 3 shows that the percentages of modified constraints satisfied by the solutions for the WCSPs modeled (WCSP) in comparison with percentages for the solutions found by a usual CSP solver (Original CSP), increase directly with the number of variables due to the number of valid tuples per constraint increases also. The technique of modeling DynCSPs as WCSPs achieves its best robustness results in problems in which the number of valid tuples of the constraints is high. Nevertheless, we cannot observe another significant relation with other parameters of the problems.

TABLE 3. Percentages of modified constraints satisfied for the solutions of the modeled WCSP and for the solutions found by a usual CSP solver

				Modified Constraints Satisfied	
Variables	Domain Size	Constraints	Tightness	WCSP	Original CSP
15	60	30	0.4	70%	46.6%
30	60	30	0.4	93.3%	56%
60	60	30	0.4	96%	50%
30	15	60	0.9	96.6%	65%
30	30	60	0.9	95%	65%
30	60	60	0.9	95%	65%
60	30	15	0.4	93.3%	46.6%
60	30	30	0.4	93.3%	50%
60	30	60	0.4	93.3%	46.6%
60	60	60	0.9	95%	58.3%
60	60	60	0.7	90%	53.3%
60	60	60	0.4	90%	50%

In all the cases, the solutions found by our technique are more robust than the solutions found by a usual CSP solver, because the solutions found for the WCSP modeled satisfy a higher number of modified constraints.

[1] *http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/ToolBarIntro*

5.2. **Benchmarks.** In the literature there are not *informed DynCSPs* benchmarks. Therefore, we have considered WCSPs benchmarks and we have added dynamism functions to the constraints of the problems. The used instances are from two WCSPs benchmarks: Academics and Planning. The benchmarks can be download from [7]. The format of the benchmarks is the WCSP file format. In this format, the constraints are represented extensionally. For this reason, the criterion for choosing the valid tuples that become invalid for a new restricted constraint has been to choose randomly the tuples.

The problems that only have one solution were skipped, because it is only possible to choose this solution as the most robust solution of the problem. In addition, the original soft constraints were removed from the problems due to our technique has not been developed for constraint satisfaction and optimization problems (CSOPs).

5.2.1. *Robustness analysis based on the number of constraints additions.* In the first developed analysis our aim has been to determine the effectiveness of our technique and to analyze the influence of the parameter w (number of new generated constraints over each original constraint) in the robustness of the solution obtained. For each problem, 10 random instances were generated by choosing random dynamism functions.

Table 4 shows the number of hard constraints (h_C) that belong to the problems analyzed. In addition, it can be observed how many new modified constraints have been generated (m_C). This number is directly related to the parameter w. However, there is a saturation point for the number of new modified constraints that it is possible to create for each original constraint since it is not allowed to create a new modified constraint which has the same valid tuples than other ones. For each analyzed problem, it is shown the number of modified constraints satisfied (s_C) for the solutions obtained by a usual CSP solver (Ori) and for the WCSP modeled by our technique ($WCSP$), and the difference (D) between these values.

TABLE 4. Robustness analysis based on w

		$w=1$			$w=2$			$w=4$			$w=8$			$w=16$			$w=32$		
Problem	Sol	m_C	s_C	D	m_C	s_C	D	m_C	s_C	D	m_C	s_C	D	m_C	s_C	D	m_C	s_C	D
4queens	Ori	6	2	1	9	2	1	14	4	0	17	4	1	21	4	1	25	4	2
(h_C=4)	WCSP		3			3			4			5			5			6	
8queens	Ori	28	14	5	56	20	9	112	34	15	218	56	32	392	94	58	590	102	99
(h_C=28)	WCSP		19			29			49			32			152			201	
langford_2_4	Ori	29	12	2	54	17	2	87	21	3	103	23	6	106	26	1	97	31	2
(h_C=32)	WCSP		14			19			24			29			27			33	
langford_3_9	Ori	369	176	10	737	275	17	1441	453	19	2643	693	44	4197	957	88	4755	981	89
(h_C=369)	WCSP		186			292			472			737			1045			1070	
16wqueens	Ori	120	61	22	240	87	41	476	142	80	903	245	141	1555	351	245	2067	432	314
(h_C=120)	WCSP		83			128			222			386			596			746	
slangford_3_11	Ori	550	270	75	1100	408	117	2180	680	203	4106	1112	351	6960	1656	578	9167	1999	720
(h_C=550)	WCSP		345			525			883			1463			2234			2719	
driverlog01cc	Ori	39	14	10	66	16	11	93	17	11	93	15	14	92	15	12	87	16	13
(h_C=472)	WCSP		24			27			28			29			27			29	
logistics01bc	Ori	140	41	8	214	52	9	272	52	10	283	52	11	272	48	15	280	51	11
(h_C=2222)	WCSP		49			61			62			63			63			62	
mprime01ac	Ori	141	59	12	255	79	20	433	110	29	646	138	40	792	155	60	818	153	63
(h_C=12264)	WCSP		71			99			139			178			215			216	
rovers02ac	Ori	63	21	2	98	21	3	130	25	4	138	21	4	138	23	3	134	25	6
(h_C=5029)	WCSP		23			24			29			25			26			31	

The robustness results obtained for the analysis show that for all the problems, the number of new modified constraints satisfied (s_C) for the solutions found for the WCSP modeled is always bigger than it is for the solutions found by a usual CSP solver. Therefore, the solutions obtained by our technique are more robust. The difference of the number of modified constraints satisfied (D) increases in problems where the number of

solutions and the number of valid tuples of the constraints is high. For example, for the problem *slangford_3_11* our technique obtains solutions with the biggest D. Figure 7 shows a chart with four of the most representative problems of the Table 4.

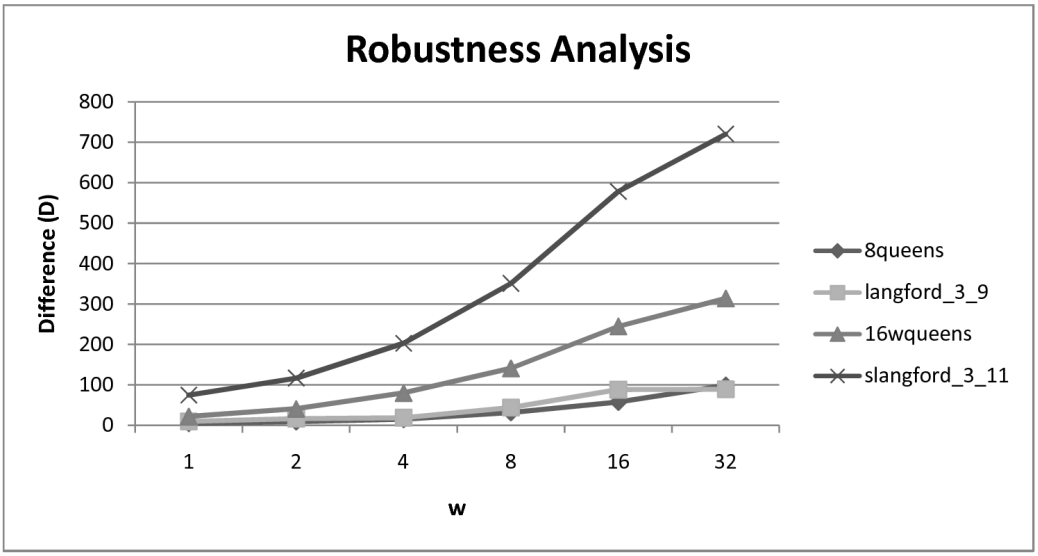

FIGURE 7. Robustness analysis of the benchmarks, based on w

Furthermore, in Table 4 and Figure 7, it can be observed that D is directly related to w. The bigger w is, the bigger D is. For example, for the problem *slangford_3_11* with $w = 32$, the solutions obtained by our technique satisfy 720 new modified constraints more than the solutions obtained by a usual CSP solver.

However, when w takes the value of the saturation point, it is not possible to create new modified constraints (m_C). Thus, in this saturation point, D stops increasing. Each problem has a different saturation point, depending on the number of valid tuples for the constraints of the problem. For example, the problem *langford_3_9* has a saturation point of $w = 16$, because D stops increasing for $w > 16$ (see Figure 7). Nevertheless, Figure 7 shows that the other three problems do not have a saturation point lower than or equal to $w = 32$, because in this point, D has not stopped increasing.

5.2.2. *Robustness analysis based on the dynamism functions.* In the second developed evaluation, our aim has been to analyze the effect of the dynamism functions in the robustness of the solutions obtained. For this purpose, we have chosen 2 problems: *driverlog01cc* and *slangford_3_11*, with a fixed $w = 1$. For each problem, 10 random instances have been generated by choosing different random valid tuples for the new modified constraints.

Table 5 shows the number of new modified constraints generated (m_C), the number of modified constraints satisfied (s_C) and the difference between these values (D). The dynamism functions $p(C_i)$ and $d(C_i)$ are represented as p and d.

It can be observed in Table 5 that $p(C_i)$ is not related to D. The function $p(C_i)$ is used in the cost assignment to the invalid tuples of the new modified constraints. Thus, the function $p(C_i)$ affects in the cost of the solutions obtained ($\mathcal{V}(s)$) but not in the number of modified constraints satisfied (s_C).

Figure 8 shows the difference of new satisfied constraints (D) for the solutions obtained by a usual CSP solver (*Ori*) and for the WCSP modeled, for the two problems (*driverlog01cc* and *slangford_3_11*) with a fixed $p(C_i) = 0.2$.

TABLE 5. Robustness analysis based on $p(C_i)$ and $d(C_i)$

		driverlog01cc	p=0.2		p=0.4		p=0.6		p=0.8	
d	Sol	m_C	s_C	D	s_C	D	s_C	D	s_C	D
0.2	*Ori*	26	16	9	18	6	17	7	18	7
	WCSP		25		24		24		25	
0.4	*Ori*	43	19	11	20	11	22	8	19	12
	WCSP		30		31		30		31	
0.6	*Ori*	49	11	13	14	11	14	10	13	11
	WCSP		24		25		24		24	
0.8	*Ori*	49	3	8	4	7	3	8	3	8
	WCSP		11		11		11		11	
		slangford_3_11	p=0.2		p=0.4		p=0.6		p=0.8	
d	Sol	m_C	s_C	D	s_C	D	s_C	D	s_C	D
0.2	*Ori*	550	437	76	437	75	440	72	435	78
	WCSP		513		512		512		513	
0.4	*Ori*	550	330	91	331	93	331	89	333	91
	WCSP		421		427		420		424	
0.6	*Ori*	550	215	94	217	95	219	91	224	86
	WCSP		309		312		310		310	
0.8	*Ori*	550	107	75	108	73	112	65	178	71
	WCSP		182		181		117		107	

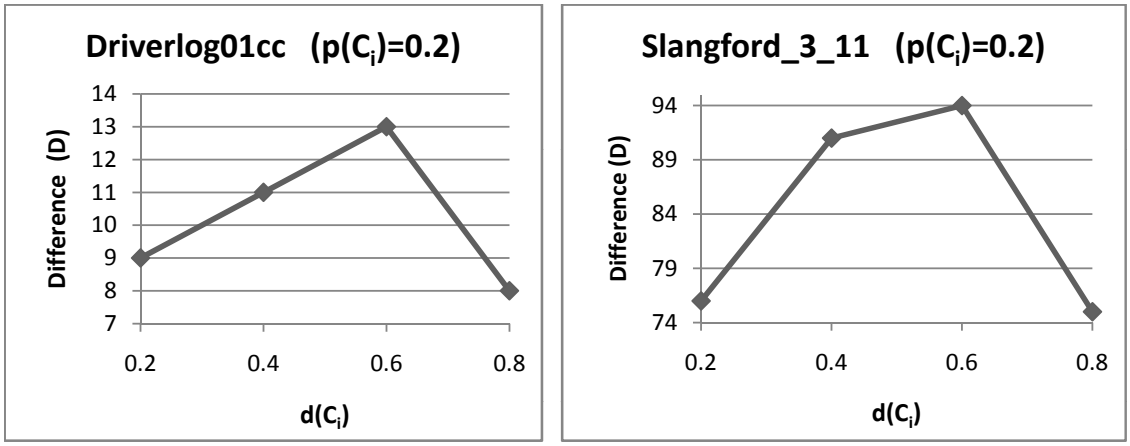

FIGURE 8. Robustness analysis for the benchmarks, based on $d(C_i)$ with a fixed $p(C_i) = 0.2$

The difference of new satisfied constraints (D) increases when $d(C_i)$ increases because the percentage of invalid tuples for a new modified constraint increases also. However, there is a saturation point which is $d(C_i) = 0.8$ for both problems. At this point the number of new satisfied constraints (s_C) decreases due to the tightness of the new modified constraints is so high that it is unlikely that the solutions satisfy a high number of these constraints. For both problems the maximum level of D is achieved for $d(C_i) = 0.6$.

6. **Conclusions.** In this paper, we have presented the *informed DynCSP* which is a DynCSP with additional information about the dynamism of its constraints. The main objective has been to find robust solutions that have a high probability of remaining valid faced with possible future changes in the constraints. In order to deal with *informed DynCSPs*, we have introduced a technique which consists on model *informed DynCSPs*

as WCSPs, by generating new modified constraints and assigning costs to the tuples of each constraint.

From the robustness evaluation, we can conclude that the solutions obtained for the WCSP modeled by our technique were more robust than the solutions obtained by a usual CSP solver.

The method of modeling *informed DynCSPs* as WCSPs has the main advantage over the reactive strategies that it prevents the loss of solutions in dynamic problems. In addition, the *informed DynCSP*, defined in this paper, has the advantage that is able to capture many changes that undergo real problems and they can be represented as restrictive modifications over the original constraints.

Summarizing, the benefits of the *informed DynCSPs* and the developed method are:

- *Reliable.* The method of modeling *informed DynCSPs* as WCSPs ensures obtaining robust solutions.
- *Practical.* The proposal uses an efficient algorithm to model *informed DynCSPs* as WCSPs, with a low computational cost with respect to the solving phase.
- *Specific.* The *informed DynCSPs* allow to specify in the modeling of dynamic problems that have associated information about possible changes in their constraints. Thus, the search of robust solutions is oriented according to this information, obtaining better results.

Acknowledgments. This work has been partially supported by the research projects TIN2010-20976-C02-01 (Min. de Ciencia e Innovación, Spain) and P19/08 (Min. de Fomento, Spain-FEDER), and the fellowship program FPU.

REFERENCES

[1] K. R. Apt, *Principles of Constraint Programming*, Cambridge University Press, 2003.
[2] F. Barber and M. Salido, Introducción a la programación de restricciones, *Inteligencia Artificial: Revista Iberoamericana de Inteligencia Artificial*, vol.20, pp.13-29, 2003.
[3] L. Climent, F. Barber, M. Salido and L. Ingolotti, Robustness and capacity in scheduling: Application to railway timetabling, *Workshop on Planning, Scheduling and Constraint Satisfaction*, pp.87-96, 2008.
[4] R. Dechter and J. Pearl, Network-based heuristics for constraint satisfaction problems, *Artificial Intelligence*, vol.34, pp.1-38, 1988.
[5] R. Dechter and A. Dechter, Belief maintenance in dynamic constraint networks, *Proc. of the 7th National Conference on Artificial Intelligence*, pp.37-42, 1988.
[6] H. Fargier and J. Lang, Uncertainty in constraint satisfaction problems: A probabilistic approach, *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pp.97-104, 1993.
[7] S. Givry, *Soft CSP Benchmarks*, http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP.
[8] S. D. Givry, J. Larrosa, P. Meseguer and T. Schiex, Solving max-sat as weighted CSP, *Principles and Practice of Constraint Programming – CP 2003, LNCS*, vol.2833, pp.363-376, 2003.
[9] S. González and P. Meseguer, *Open, Interactive and Dynamic CSP*, 2008.
[10] E. Hebrard, B. Hnich and T. Walsh, Super CSPs, *Technical Report*, 2003.
[11] E. Hebrard, B. Hnich and T. Walsh, Super solutions in constraint programming, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pp.157-172, 2004.
[12] E. Jen, Stable or robust? what's the difference? *Complexity*, vol.8, pp.12-18, 2003.
[13] J. Larrosa and T. Schiex, Solving weighted csp by maintaining arc consistency, *Artificial Intelligence*, vol.159, pp.1-26, 2004.
[14] Z. Lin, Y. Xia, P. Shi and H. Wu, Robust sliding mode control for uncertain linear discrete systems independent of time-delay, *International Journal of Innovative Computing, Information and Control*, vol.7, no.2, pp.869-880, 2011.
[15] C. Liu, An evolutionary algorithm for solving dynamic nonlinear constrained optimization, *ICIC Express Letters*, vol.4. no.3(B), pp.1039-1044, 2010.

[16] T. Nakano and M. Nagamatu, Lagrange neural network for solving constraint satisfaction problem, *International Journal of Innovative Computing, Information and Control*, vol.1, no.4, pp.659-671, 2005.
[17] L. Otten, *Wcsp File Format Specification*, http://graphmod.ics.uci.edu/group/WCSP_file_format.
[18] F. Rossi, P. Van Beek and T. Walsh, *Handbook of Constraint Programming*, Elsevier, 2006.
[19] H. Sakkout and M. Wallace, Probe backtrack search for minimal perturbation in dynamic scheduling, vol.5, pp.359-388, 2000.
[20] M. A. Salido and F. Barber, Distributed CSPs by graph partitioning, *Applied Mathematics and Computation*, vol.183, pp.491-498, 2006.
[21] T. Schiex, H. Fargier and G. Verfaillie, Valued constraint satisfaction problems: Hard and easy problems, *Proc. of the 14th IJCAI*, pp.631-637, 1995.
[22] E. Tsang, *Foundation of Constraint Satisfaction*, Academic Press, 1993.
[23] G. Verfaillie and T. Schiex, Solution reuse in dynamic constraint satisfaction problems, *Proc. of the 12th National Conference on Artificial Intelligence*, pp.307-312, 1994.
[24] G. Verfaillie and N. Jussien, Constraint solving in uncertain and dynamic environments: A survey, *Constraints*, vol.10, no.3, pp.253-281, 2005.
[25] R. Wallace and E. Freuder, Stable solutions for dynamic constraint satisfaction problems, *Proc. of the 4th International Conference on Principles and Practice of Constraint Programming*, pp.447-461, 1998.
[26] R. Wallace, D. Grimes and E. Freuder, Solving dynamic constraint satisfaction problems by identifying stable features, *International Joint Conferences on Artificial Intelligence*, pp.621-627, 2009.
[27] L. Wang and C. Shao, State feedback controller design for a class of uncertain systems with time-varying delays and controller failures, *International Journal of Innovative Computing, Information and Control*, vol.6, no.5, pp.2055-2064, 2010.
[28] X. Zhang, Z. Tang, Y. Cui and L. Jiang, A real-time convex hull algorithm, *ICIC Express Letters*, vol.4, no.3(A), pp.623-628, 2010.