ai-review manuscript No.
(will be inserted by the editor)

Finding Robust Solutions for Constraint Satisfaction Problems with Discrete and Ordered Domains by *Coverings*

Laura Climent · Richard J. Wallace · Miguel A. Salido · Federico Barber

Received: date / Accepted: date

Abstract Constraint programming is a paradigm wherein relations between variables are stated in the form of constraints. Many real life problems come from uncertain and dynamic environments, where the initial constraints and domains may change during its execution. Thus, the solution found for the problem may become invalid. The search for *robust* solutions for Constraint Satisfaction Problems (CSPs) has become an important issue in the field of constraint programming. In some cases, there exists knowledge about the uncertain and dynamic environment. In other cases, this information is unknown or hard to obtain. In this paper, we consider CSPs with discrete and ordered domains where changes only involve restrictions or expansions

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.

L. Climent
Instituto de Automática e Informática Industrial. Universitat Politècnica de València, Spain.
Tel.: +34-96-3873550
Fax: +34-96-3877359
E-mail: lcliment@dsic.upv.es

R. J. Wallace
Cork Constraint Computation Centre and Department of Computer Science. Western Gateway Building. University College Cork, Ireland.
Tel.: +353-21-4205954
Fax: +353-21-4205369
E-mail: r.wallace@4c.ucc.ie

M. A. Salido
Instituto de Automática e Informática Industrial. Universitat Politècnica de València, Spain.
Tel.: +34-96-3883512
Fax: +34-96-3877359
E-mail: msalido@dsic.upv.es

F. Barber
Instituto de Automática e Informática Industrial. Universitat Politècnica de València, Spain.
Tel.: +34-96-3879357
Fax: +34-96-3877359
E-mail: fbarber@dsic.upv.es

of domains or constraints. To this end, we model CSPs as weighted CSPs (WCSPs) by assigning weights to each valid tuple of the problem constraints and domains. The weight of each valid tuple is based on its distance from the borders of the space of valid tuples in the corresponding constraint/domain. This distance is estimated by a new concept introduced in this paper: *coverings*. Thus, the best solution for the modeled WCSP can be considered as a most robust solution for the original CSP according to these assumptions.

Keywords Robustness · Uncertainty · Dynamism · Dynamic Constraint Satisfaction Problems (DynCSPs)

1 Introduction

It is well-known that many Artificial Intelligence problems can be modeled as CSPs. Much effort has been spent to increase the efficiency of algorithms for solving CSPs. However, many of these techniques assume that the set of variables, domains and constraints involved in the CSP are known and fixed when the problem is modeled. This is a strong limitation when we deal with real-life situations where both the original problem and the corresponding CSP model may evolve because of the environment, the user or other agents. In such situations, a solution that holds for the original model can become invalid after changes in the original problem.

There are two main approaches for dealing with these situations: (i) *reactive approaches*, whose main objectives are to obtain a new solution that is as similar as possible with respect to the previous solution (the solution found before the changes occurred) as efficiently as possible and (ii) *proactive approaches*, which use knowledge about possible future changes in order to avoid or minimize their effects (see Verfaillie and Jussien (2005) for a survey). Note that proactive approaches are applied before the changes occur, in order to prevent their effects. On the other hand, reactive approaches are applied when the changes invalidate the original solution, in order to repair the solution or obtain a new one. Both approaches are compatible and therefore a proactive approach can be initially applied and a reactive approach would only be needed if the previously obtained solution is lost.

Using reactive approaches entails re-solving the CSP after each solution loss, which consumes computational time. That is a clear disadvantage, especially when we deal with short-term changes, where solution loss is very frequent. In addition, in many applications, such as online planning and scheduling, the delivery time of a new solution may be too long for actions to be taken on time (Verfaillie and Jussien (2005)). Furthermore, a solution loss can produce several negative effects in the problem modeled. For example, in a real problem of task assignment in a production system with several machines, it could cause the shutdown of the production system, the breakage of machines, the loss of the material/object in production, etc. In a transport timetabling problem, the solution loss, due to some disruption at a point, may produce a delay that propagates through the entire schedule. In addition, all the negative effects stated above will probably entail an economic loss. Since we strongly value solution loss prevention in uncertain and dynamic environments, we have taken a proactive approach, focusing on the search for robust solutions. Robust solutions are those that have a high probability of remaining valid when faced with possible future changes.

Most approaches that have been proposed for finding robust solutions assume the existence of knowledge about the uncertain and dynamic environment (see Section 3.2). However, it is difficult to characterize the robustness of solutions when detailed information about possible future changes is scarce. In this paper, we consider situations in which only limited (and intuitively reasonable) assumptions are made about possible changes that can occur in CSPs with ordered and discrete domains: namely that changes always take the form of restrictions at the borders of a domain or constraint. This is motivated in Section 2. In this paper we present a new concept called coverings for estimating the distances from borders of the domains/constraints. Informally, a covering represents a set of partial or complete solutions that surround another solution and therefore they confer certain level of robustness. Figure 1 shows the necessary steps to model and solve a CSP in order to find robust solutions. After calculating the coverings, the original CSP (*P*) is modeled as a WCSP (*modP*) by assigning weights to the valid tuples based on their coverings. Finally, the modeled WCSP (*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 CSP (*P*).

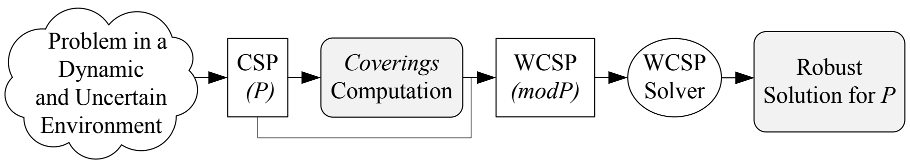


Fig. 1 Modeling robustness in a CSP as a weighted CSP (WCSP).

The next section discusses the motivation of the little assumptions of dynamism done in this paper. Stability and robustness concepts are explained in Section 3, as well as approaches related with these solution features. We describe some technical background in Section 4 and we present the covering framework (Section 5) and an algorithm for calculating coverings (Section 6). Then, in Section 7, we describe an enumeration-based technique for modeling robustness in a CSP as a weighted CSP (WCSP). We show a simple example in Section 8 and a case study in Section 9. Experimental results are presented in Section 10 and finally we present conclusions in Section 11.

2 Motivation

The approach presented in this paper does not require extra associated information about future changes in the problem. However, it considers little assumptions of the dynamism that are inherent to the structure of CSPs with ordered domains. Due to the fact that no additional knowledge of the dynamism is known, it is reasonable to assume that it is more likely that any bound of the solution space undergoes restrictive or relaxed modifications.

Some real life problems that support this assumption are temporal reasoning-based problems (Dechter et al (1991)), which consists on the reasoning of relationships between time and actions/events. These include natural language understanding, simulation, diagnosis, scheduling, planning, etc. When dealing with temporal problems, the dynamism and uncertainty is practically inherent to them. Thus, modifications are more likely to occur in the bounds of the solution space due to the time relationships. Planning and scheduling is a rich context where uncertainties and changes cannot be easily avoided (Verfaillie and Jussien (2005)). For instance, arrival times of employees/transports or finalization of tasks/events may undergo delays/advances. These temporal changes are translated into restrictions/relaxations of the borders of the resultant CSPs constraints. For instance, if the end time of a task is 16, consequently, all of the subsequent tasks in the same job must have a start time in the interval $[17, max]$. In this case, a delay of 3 time units in the aforesaid task is propagated to the subsequent tasks, so that the new domains must be restricted to $[20, max]$.

Other real life problems supporting the aforementioned dynamism assumptions are the spatial and geometric reasoning problems. In such problems, measurement errors can unleash in a partially incorrect representation of the real life problem. Distance constraints are generally modified by relaxing or restricting the constraints in boundaries.

This condition is extended to design problems, where the resultant CSP is not completely determined before the solving process. These CSPs are modeled by assigning the design elements to variables, and constraints represent the properties that these elements must satisfy. For these problems, the resources and conditions of the environment may undergo changes. For instance, consider the example introduced in Sam (1995) that consists on a design problem of floor damping by means of beams and conducts traversing them for avoiding vibrations. Consider a variable representing the number of conducts to introduce in the beams, whose upper bound is 10 conducts. In the future, it is more likely to happen that there are less/more available quantity of conducts (modifications of domain borders) than a certain intermediate domain value becomes invalid (for example, 5 conducts). The authors mention possible environmental changes, for instance, the system floor may become more and more slender, producing a increase in the susceptibility of the vibrations. The effect of this change in the original problem, is translated into a restrictive modification of the constraints that involve vibrations in the modeled CSP.

Following, we present an example to illustrate the main objective when facing changes.

Example 1 Figure 2 shows a solution space of a CSP, which is composed of two variables x_0 and x_1. It can be observed that it has 29 solutions (black points). If no specific information is given about the dynamics of the problem, i.e. the kinds and likelihoods of changes in the problem, it is not easy to decide which solution is considered the most robust.

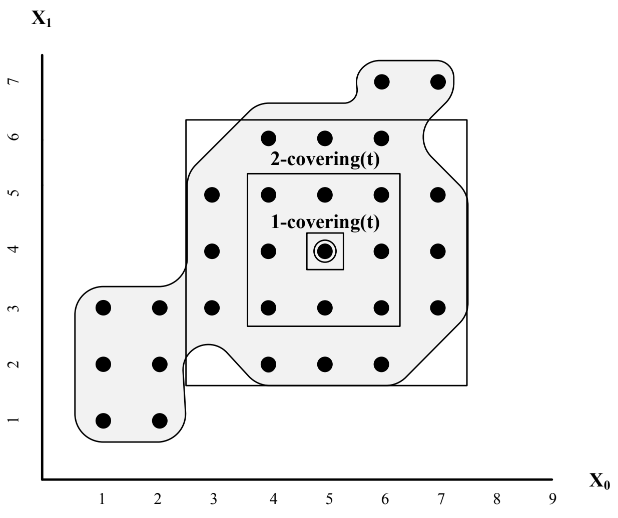


Fig. 2 Solution space of the CSP.

In Figure 2, a smaller solution space is represented (smallest square), which is the result of restrictive modifications over the original constraints and domains of the CSP. If there is no information about unexpected changes in the CSP, even if it does not cover all the possible changes, it is reasonable to assume that the original constraints and/or domains undergo modifications in the form of range reductions or expansions (as motivated above). Note that the possibility of solution loss only exists when changes to the original bounds of the solution space are restrictive.

In this situation, since larger restrictions always include (some) smaller ones, we will assume that values affected by larger restrictions are, in general, less likely to be removed. This is shown in Figure 2, where solutions in the smallest square area have a higher probability of remaining valid. Given these assumptions, *the most robust solution of a CSP with discrete and ordered domains is the solution that is located as far away as possible from the bounds of the solution space*. Consequently, our main goal is focused on searching these solutions.

In the above example, with a 2-dimensional non-convex solution space, the most robust solution would be $(x_0 = 5, x_1 = 4)$, because it is the solution located furthest away from the search space bounds. However, in a n-dimensional CSP (n is the number of variables), selecting the most robust solution is not straightforward, especially if we are dealing with non-convex CSP solution spaces. Furthermore, many real life problems involve non-linear constraints, for instance, in the design problems (Sam (1995)). Thus, our aim is to develop approaches that can be applied to both, convex (Climent et al (2011)) and non-convex spaces.

3 Stability and Robustness

We first clarify the distinction between the concepts of robustness and stability for CSPs and subsequently several proactive approaches are introduced. In Jen (2003), the authors introduce the general concepts of stability and robustness: A solution is *stable* in a dynamic system, if by means of a few changes in the original solution, we can obtain a new solution that is similar to the original one. Robustness, on the other hand, is a measure of persistence in systems, which compels us to focus on perturbations. Perturbations are small differences in the current state of the system. In Verfaillie and Jussien (2005), the authors define the concept of robust solution for the specific case of CSPs:

Definition 1 A robust solution has every chance to resist changes, i.e., to remain a solution in spite of these changes (Verfaillie and Jussien (2005)).

Note that this concept has a strong dependency with respect to the assumption of the future possible changes that may occur. Furthermore, regarding Definition 1, there are cases in which is not possible to find a solution which is able to resist all the changes, especially when information about future changes is limited. In order to consider these cases, Definition 1 can be slightly modified to define the most robust solution for CSPs.

Definition 2 The most robust solution within a set of solutions is the one with the highest likelihood of remaining valid after any type of change.

The concept of a stable solution has also been defined for the specific case of CSPs (Hebrard (2006), Verfaillie and Schiex (1994)):

Definition 3 A solution f is more stable than another solution g if and only if, in the event of a change, a closer alternative to f than to g exists (Hebrard (2006)).

This definition considers the easiness with which the original solution can be modified to produce a solution to the new problem and also how much “closer” the new solution is to the original solution. Furthermore, we would like to note that Definitions 1 and 2 do not consider alterations in the original solution but only its resistance to changes in the problem. On the other hand, Definition 3 does consider changes to the original solution when a new solution is produced after a change in the problem.

Several techniques have been proposed in the past for handling with problems that come from uncertain and dynamic environments, which can be classified based on the kind of solutions that they obtain (see Verfaillie and Jussien (2005) for a survey). In this section we only focus on the explanation of super solutions, earlier techniques that search for robust solutions and their limitations.

3.1 Searching for Stable Solutions

In Hebrard (2006), the author presents techniques that search for stable solutions of a certain type, called super-solutions. The goal is to be able to repair an invalid solution after changes occur, with minimal changes that can be specified in advance. For CSPs, the focus has been on finding (1,0)-super-solutions.

Definition 4 A solution is a (1, 0)-super-solution if the loss of the value of one variable at most can be repaired by assigning another value to this variable without changing the value of any other variable (Hebrard (2006)).

However, finding (1,0)-super-solutions is problematic because, (1) if there is a *backbone variable* (a variable that takes the same value in all solutions), this ensures that there are no (1,0)-super-solutions, (2) in general, it is unusual to find (1,0)-super-solutions where all variables can be repaired. For these reasons, in Hebrard (2006) the author also developed a *branch and bound*-based algorithm for finding solutions that are close to (1,0)-super-solutions, i.e., where the number of repairable variables is maximized (also called maximizing the (1-0)-repairability).

3.2 Searching for robust solutions

Most earlier approaches that search for robust solutions use additional information about the uncertain and dynamic environment in which the problem occurs, and usually involve probabilistic methodologies.

Thus, information is gathered in the form of penalties, in which values that are no longer valid after a change in the problem are penalized (Wallace and Freuder (1998)). Thereafter, the algorithm tries to find solutions that do not include these penalized values. Other techniques are based on associated information about the dynamism of the constraints. In the Probabilistic CSP model (PCSP) (Fargier and Lang (1993)), each constraint is associated with a probability of existence. The most robust solution is the solution that maximizes the probability of satisfying the constraints. In Climent et al (2010), the authors proposed a different probabilistic approach that considers two parameters associated with each constraint: the probability of change of the constraint and its amount of change. According to these parameters, several restricted CSPs are generated. The most robust solution is the solution that satisfies the highest number of restricted CSPs.

Other techniques focus on the dynamism of the variables of the CSP. For instance, the Mixed CSP model (MCSP) (Fargier et al (1996)), considers the dynamism of certain *uncontrollable* variables that can take on different values of their uncertain domains. Here, the objective is to find an assignment of decision variables (which are the usual variables of the CSP model) that satisfies all possible values that the uncontrollable variables can take. The Uncertain CSP model (UCSP) is an extension of MCSP, whose main innovation is that it considers continuous domains (Yorke-Smith and Gervet (2009)). The Stochastic CSP model (SCSP) (Walsh (2002)) assumes a probability distribution associated with the uncertain domain of each uncontrollable variable; here the objective is to find a solution with the maximum probability of validity. The Branching CSP model (BCSP) considers the possible addition of variables to the current problem (Fowler and Brown (2003)). For each variable, there is a gain associated with an assignment. The objective is to find a solution that maximizes the gain by considering the possible additional future variables that are compatible with the normal variables, and also considering their probability of future existence.

In most of these models, the form of the algorithm is dependent on detailed knowledge about the dynamic environment. Therefore, a list of the possible changes or the representation of uncertainty is required, often in the form of an associated probability distribution. As a result, these approaches cannot be used if the required information is not known. In many real problems, however, knowledge about possible further changes is either limited or nonexistent. Hence, there is an important need for techniques that find robust solutions in this kind of environment.

4 Technical Background

Here, we give some basic definitions that are used in the rest of the paper, following standard notation and definitions from the literature.

Definition 5 A *Constraint Satisfaction Problem* (CSP) is represented as a triple $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ where:

- $\mathcal{X}$ is a 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 $x_i \in \mathcal{X}$, D_i is a set of values that the variable can take.
- $\mathcal{C}$ is a finite set of constraints $\mathcal{C} = \{C_1, C_2, ..., C_e\}$ which restrict the values that the variables can simultaneously take. We use $\mathcal{DC}$ to denote the set of unary constraints associated with $\mathcal{D}$.

Definition 6 A *tuple t* is an assignment of values to a subset of variables $\mathcal{X}_t \subseteq \mathcal{X}$.

For a subset B of $\mathcal{X}_t$, the projection of t over B is denoted as $t \downarrow_B$. For a variable $x_i \in \mathcal{X}_t$, the projection of t over x_i is denoted as t_i.

The number of possible tuples of a constraint $C_i \in \mathcal{C}$ is composed of the elements of the Cartesian product of the domains of $var(C_i)$: $\prod_{x_j \in var(C_i)} D_j$, where $var(C_i) \subseteq \mathcal{X}$ are the variables involved in C_i.

Definition 7 The *tightness* of a constraint is the ratio of the number of forbidden tuples over the number of possible tuples. The tightness is defined within the interval [0,1].

Definition 8 We denote the set of valid tuples of a constraint $C_i \in (\mathcal{C} \cup \mathcal{DC})$ as $T(C_i)$ and its size $|T(C_i)|$ is:

$$|T(C_i)| = \begin{cases} (\prod_{x_j \in var(C_i)} |D_j|) * (1 - tightness) & \text{if } C_i \in \mathcal{C} \\ |D_j| & \text{if } C_i \in \mathcal{DC} \text{ and } x_j \in var(C_i) \end{cases} \tag{1}$$

where $|D_j|$ is the domain size of the variable x_j.

The original and static model of a CSP is not able to capture further modifications over the constraints, so we will use a variant of the static CSP model.

Definition 9 A *Dynamic Constraint Satisfaction Problem* (DynCSP) (Dechter and Dechter (1988)) is a sequence of static CSPs $\langle CSP_{(0)}, CSP_{(1)}, \ldots, CSP_{(l)} \rangle$, each $CSP_{(i)}$ resulting from a change in the previous one ($CSP_{(i-1)}$) and 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 it is considered a restriction) or increase (in which case it is considered a relaxation).

We focus our attention on DynCSPs in which the solution space of each $CSP_{(i)}$ decreases over $CSP_{(i-1)}$ (restriction). We do not analyze DynCSPs in which there exist relaxations of the $CSPs$ since these changes cannot invalidate a previously found solution. As noted earlier, our technique is applied before the changes occur. Thus, it is applied to the original CSP ($CSP_{(0)}$) of the DynCSP.

For modelling robustness of a complete solution, we use the weighted CSP formalism. This motivates the next two definitions and the accompanying example.

Definition 10 A *Valued Constraint Satisfaction Problem* (Valued CSP) is defined by a classical CSP $\langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, a valuation structure $S = (E, \circledast, \succ)$, and an application φ from $\mathcal{C}$ to E. It is noted $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, S, \varphi \rangle$, $\varphi(C_i)$ is called the valuation of C_i (Schiex et al (1995)).

Definition 11 A *Weighted Constraint Satisfaction Problem* (WCSP) is a specific subclass of Valued CSP. Here, we consider a variant of WCSP, formalized in Larrosa and Schiex (2004). This variant of WCSP is defined as $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, S(W) \rangle$, where:

- $\mathcal{X}$ and $\mathcal{D}$ are the set of variables and domains, respectively, as in standard CSPs.
- $S(W) = \langle \{0, 1, ..., W\}, \oplus, > \rangle$ is the valuation structure, where:
  - $\{0, 1, ..., W\}$ is the set of costs bounded by the maximum cost $W \in \mathbb{N}^+$.
  - $\oplus$ is the sum of costs. Thus $\forall a, b \in \{0, 1, ..., W\}$, $a \oplus b = min\{W, a + b\}$
  - $>$ is the standard order among natural numbers.
- $\mathcal{C}$ is the set of constraints as cost functions ($C_i : \prod_{x_j \in var(C_i)} D_j \to \{0, 1, ..., W\}$).

Assigning the maximum cost W to a tuple *t*, means that *t* is an invalid tuple for C_i. Otherwise (the cost assigned is lower than W) *t* is a valid tuple for C_i with the corresponding cost. The cost of a tuple *t*, denoted $\mathcal{V}(t)$, is the sum of all the applicable costs:

$$\mathcal{V}(t) = \bigoplus_{C_i \in \mathcal{C}, var(C_i) \subseteq X_t} C_i(t \downarrow_{var(C_i)}) \tag{2}$$

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

Example 2 Figure 3 shows an example of WCSP $P = \langle \{x, y\}, \{\{v_1, v_2\}, \{v_1, v_2\}\}, S(5), \{C_1, C_2\} \rangle$. It can be observed that the set of costs is $\{0, ..., 5\}$. Let us 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 3 (a) and the constraint C_2 is represented in Figure 3 (b).

Table 1 shows the set of tuples of P with their corresponding costs assigned 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 because

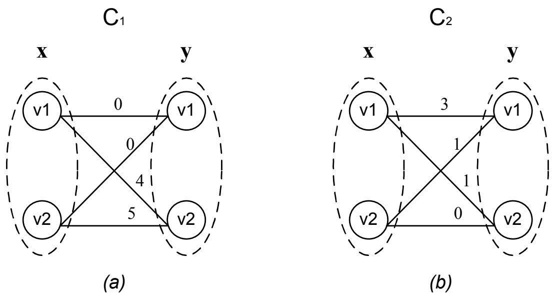


Fig. 3 WCSP P

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

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 for the problem.

5 Finding coverings

In order to search for robust solutions, we have developed a technique for calculating coverings for valid tuples. The covering of a tuple measures the protection of the tuple against perturbations. It is based on the 'onion topology'. A valid tuple with more layers (its distance to the bounds is higher) is presumed to have a higher probability of remaining valid than a tuple with fewer layers. Here, a layer of a tuple is a convex hull of valid tuples.

To the best of our knowledge, the application of the 'onion structure' to CSPs is a novel idea, although it has been used in dynamic networks. In Herrmann et al (2011), the authors analyze how the structure of a network can affect its robustness to targeted attacks and random failures. After a robustness analysis, the authors stated: "Our results show that robust networks have a novel "onion-like" topology consisting of a core of highly connected nodes surrounded by rings of nodes with decreasing degree".

We first introduce the concept of *topology* as it provides formulations about proximity relations between the elements that compose it. Subsequently, our own definitions and the developed technique will be explained.

5.1 Topology of the CSPs

In the rest of the paper, we consider CSPs with discrete and ordered domains. Each search space of this CSP typology is a n-dimensional hyperpolyhedron, where the set of valid tuples of the hyperpolyhedra is denoted by T. There are several distance functions $d(x, y) : T \times T \rightarrow \mathbb{R}^+$ that can be defined over each pair of tuples x and y. We will use the Chebyshev distance, which measures the maximum absolute differences along any coordinate dimension of two vectors (see Equation 3). This distance metric is selected in order to distinguish among hyperpolyhedra in n-dimensional space, which are analogous to squares for $n = 2$ (see Figure 2) and cubes for $n = 3$. In particular, the corners of a cube are located at the same distance from the central point than the edges, a feature which differ from Euclidean distance metric. By checking areas of satisfiability inside these hyperpolyhedra, we can ensure minimum distances to the bounds, which is used for the robustness computation (see Section 5.2).

$$d(x, y)_{Chebyshev} = \max_i (|x_i - y_i|) \tag{3}$$

Depending on the nature of the modeled problem, the above distance may or may not be applicable. For instance, in CSPs with symbolic domains, it cannot be applied unless there is an order relationship between their values. In this case, a monotonic function has to be applied to map the elements by preserving their order. To this end, a monotonic mapping function f is defined over the elements of the CSP domain: $f(x) : \mathcal{D} \rightarrow \mathbb{Z}$.

Example 3 We consider a CSP with a symbolic and ordered domain $\mathcal{D}$ which represents clothing sizes: $\{$extra-small, small, medium, large, extra-large$\}$. In this case, a monotonic function that assigns greater values to bigger clothing sizes can be defined. For example, $f(extra{-}small) = 1$, $f(small) = 2$, $f(medium) = 3$, $f(large) = 4$ and $f(extra - large) = 5$.

As we have pointed out, a CSP with discrete and ordered domains (both numeric and symbolic) can have a metric function defined over their set of valid tuples T. Therefore, T is a topological space, and the following topological definitions presented in William (2006) can be applied to CSPs:

Definition 12 The neighborhood N ($N \subseteq T$) of a valid tuple t is an open set containing all the valid tuples close to t.

5.2 The Concept of Coverings

Under the assumptions presented in Section 2, the core of the 'onion structure' is the most robust part of a structure, since the surrounding layers protect it against perturbations. The core is located at the furthest point from the outer layer of the 'onion structure'. The same philosophy can be applied to solutions of a CSP: the further away a solution is from the bounds of the solution space, the more robust the solution is. In order to determine how far a valid tuple is from the bounds, we analyze its coverings ('onion layers').

Definition 13 We define the *k-covering(t)* of a valid tuple $t \in T$ as its neighbourhood $\{y \in T : y \neq t \wedge d(t,y)_{Chebyshev} \leq k\}$, where $k \in \mathbb{N}$.

From Definition 13, the following property can be deduced: k-covering(t)$\supseteq$(k-1)-covering(t).

To calculate the coverings of a CSP where variable domains are not ordered in $\mathbb{Z}$, a monotonic function has to be applied to map the values (it must also be an order preserving function). In the previous subsection, we presented an example of mapping function for symbolic domains. Below, we give an example for rational domains.

Example 4 We consider a CSP with a rational and ordered domain $\mathcal{D}$: $\{0.156, 0.205, 0.212, 0.854\}$. Therefore, a monotonic function that preserves the order of the set of values could be defined. For example, $f(0.156) = 1, f(0.205) = 2, f(0.212) = 3$ and $f(0.854) = 4$.

Definition 14 $maxTup(k,|t|)$ denotes the maximum number of tuples that can make up a k-covering(t), where k is the k-covering and $|t|$ represents the arity of t.

Proposition 1 *The maximum possible number of tuples inside a k-covering(t) is:*

$$maxTup(k,|t|) = (2k+1)^{|t|} - 1 \tag{4}$$

Proof: The exclusion of the central tuple t from the n-dimensional generalization of Moore neighbourhood (an adaptation from **?**) is equivalent to $maxTup(k,|t|)$. The number of elements of Moore neighbourhood in a 2-dimensional space is $(2k+1)^2$. For the generalization of the latest formula, let's suppose that $k = 1$, so we are working on 1-covering and we consider the number of tuples including t (that will be removed later). For $|t| = 2$, maxTup$(1,1) = (2 \cdot 1 + 1)^1 = 3$ (according to Moore neighbourhood). It is straightforward that for t dimensions $\mathbb{R} \times ...^t ... \times \mathbb{R}$: maxTup$(1,|t|) = (2k+1)^{|t|}$. Thus, if we remove the tuple $(p_1, p_2, ..., p_t)$, the number of neighbours of this tuple is: maxTup$(1,|t|) = (2k+1)^{|t|} - 1$. So, we can conclude that maxTup$(k,|t|) = (2k+1)^{|t|} - 1.\square$

Note that domains bounds are 1-dimensional spaces ($|t| = 1$), so the maximum possible tuples composing a domain bound covering is $(2k+1)-1$. From Proposition 1, it is obvious that |k-covering(t)| $\leq$maxTup(k, $|t|$), where |k-covering(t)| is the covering cardinality.

Definition 15 A k-covering(t) is called *complete* if |k-covering(t)| =maxTup(k, $|t|$).

If k-covering(t) is complete, it means that t is located at least k Chebyshev distance from the bounds, because inside the k-covering(t) all the tuples are valid. On the other hand, if at least one invalid tuple is inside of the k-covering(t), the unsatisfiability space is not completely outside of k-covering(t) and the minimum distance of t from the bounds of the solution space is the distance to the closest invalid tuple.

Note that if at least one of the closest neighbors of t is invalid, it means that t is located on a bound of the solution space.

If there are several tuples with the same number of complete coverings, are they equally robust? The answer is obtained by calculating the number of valid tuples of the minimum incomplete covering (the next covering to the maximum complete covering). Considering the 'onion topology', if there are holes in an 'onion layer', it is preferable that they are as small as possible. The same happens with tuples that do not have any complete covering. In these cases, we cannot ensure even a minimum distance of 1 from the bounds (very low robustness), but the tuples with higher $|$1-covering(t)$|$ are more robust.

In Figure 2 (see Example 1) the 1-covering(t) and 2-covering(t) for $t = (x_0 = 5, x_1 = 4)$ are represented. It can be seen that the 1-covering(t) is complete because $|$1-covering(t)$|$ =maxTup(1,$|t|$)$= 8$. However, the 2-covering(t) is not complete because $maxTup(2, |t|) = 24$ and $|$2-covering(t)$| = 20$. Thus, we can only ensure that $(x_0 = 5, x_1 = 4)$ is located at a distance of at least 1 from the bounds (it has only one completed layer). Note that some bounds of the solution space are located inside the 2-covering(t). For those tuples whose 1-covering is incomplete, their robustness can be distinguished by the cardinality of their 1-covering. For instance, the tuple $v = (x_0 = 3, x_1 = 4)$ is more robust than $w = (x_0 = 7, x_1 = 7)$ because $|$1-covering(v)$| = 5$ and $|$1-covering(w)$| = 2$. This information is equivalent to the closeness of both tuples to the bounds of the solution space. Note that tuple v only has bounds on its left and left-up side. On the other hand, tuple w has bounds on its right, right-up, right-down, left-up, up and down sides.

5.3 Coverings and Robustness

In this work, we are assuming that restrictions in the tuple space start at borders. This means that for a restriction in a given direction, a tuple located further from the border is less likely to be invalid than one located close to the border. This is simply because a larger restriction, one that involves tuple t, must also include any tuples closer to the given border. However, when we consider restrictions in any direction, this assumption can no be longer made. For example, consider the case where $|t| = 1$ and there are three values, a, b and c, where $a < b < c$. Suppose that the probability of a restriction involving a is 0, while one involving c is 1/5, and one involving b and c is 1/10. (In this case, these are the only possible restrictions.) Note that this example is consistent with our assumption that larger restrictions in one direction necessarily have a lower probability of occurrence than smaller restrictions. For this problem, the border value a is the most robust value, although its $|$k-covering(a)$| = 1$, while the $|$k-covering(b)$| = 2$.

Despite the existence of such extreme cases, our model will hold over a wide variety of probability distributions. In fact, the only requirement is that the probability associated with a given tuple must be lower than any of the probabilities for tuples closer to a border. This will always be the case when the probabilities of losing a border value are roughly equal for all such values. But it is also true under many other cases. Thus, using a variation on the previous example, if the probability of losing a,c and b is 1/2, 1/10, and 1/10, respectively, given the loss of either a or b, then b is still the most robust value.

Moreover, in these cases, if we combine a choice of one tuple with the highest k-covering with another such tuple from a different constraint, then we will have chosen the tuple with the lowest probability of being lost in each case. Hence, this combination will be associated with the lowest probability product, i.e. with the greatest robustness. Obviously, this argument can be extended, which gives us a rationale for the method of aggregation described in Section 5, where we define a type of weighted CSP whose solution gives us the most robust solution under our assumptions.

6 Algorithm for Calculating Coverings

Searching for the solution that is completely surrounded for the greatest number of solutions (core of the 'onion') requires determining the complete solution space of the CSP, which is NP-hard in general. An approach based on calculating all the CSP solutions for providing the most robust one is not generally viable due to the fact that it would be extremely time consuming task. Consequently, we have developed an approach for finding robust solutions. It is based on modeling the coverings of each valid tuple of constraints/domains of the CSP in a WCSP. In this section we present an algorithm for calculating the coverings. It must be taken into account that solving WCSPs is NP-hard in general (due to the fact that it is an optimization problem). However, this can be alleviated by using heuristics and Branch & Bound techniques. For instance, in Larrosa et al (1999), the authors developed a $O(ed^3)$ algorithm for solving WCSPs. In addition, a Branch & Bound algorithm is an 'anytime' technique, so we can limit the search time by fixing a timeout.

Algorithm 1 calculates the coverings of the valid tuples, after first carrying out a global arc-consistency (GAC3) process (line 1). In this preliminary step, it searches for a support of each domain value in order to detect tuples that are not globally consistent. For this purpose, we have implemented the well known GAC3 (Mackworth (1977)), but any other consistency techniques could also be applied. For calculating coverings, Algorithm 1 begins with $k = 1$ (1-covering(t)), by increasing this value by one unit in each iteration until the maximum covering of a CSP is reached (or optionally, a lower bound U). In each iteration, $\forall t \in T(C_i), \forall C_i \in (\mathcal{C} \cup \mathcal{DC})$, k-covering($t$) is computed, if and only if $k = 1$ or (k-1)-covering(t) is complete (see Definition 15).

Definition 16 The maximum covering of a CSP is denoted as *max-covering(CSP)* and it is reached when there is no tuple whose k-covering is complete for some $C_i \in (\mathcal{C} \cup \mathcal{DC})$.

Furthermore, if the user desires to obtain a lower k-covering(t), she/he can optionally fix a lower bound U. In this case, the algorithm stops after calculating $min(U$, max-covering$(CSP))$.

Definition 17 We define last-covering(t) to be the last k-covering(t) computed by the algorithm for the tuple t. The value of 'last' in last-covering(t) term is equal to $min(U$, max-covering$(CSP), (k + 1))$, if k-covering(t) is the highest completed covering of t.

Algorithm 1 returns the size of last-covering(t) computed for each valid tuple t of each constraint and domain of the CSP. It is a measure of the robustness of each tuple of each constraint. In addition, the value of max-covering(CSP) is returned (unless U is provided). The algorithm updates the boolean variable $C_i.completeness$ according to if C_i has at least a valid tuple t whose k-covering(t) is complete (lines $18 - 19$). In addition, it fixes the value of max-covering(CSP) when a constraint does not have any valid tuple t whose k-covering(t) is complete (line $22 - 23$).

Since k-covering(t) $\supseteq$ (k-1)-covering(t), the algorithm only analyzes the new possible neighbours of t, which are the neighbours that belong to it but do not belong to (k-1)-covering(t) (the neighbours placed in the k 'onion layer'). This is due to the fact that the neighbours of the lower coverings have already been calculated in previous iterations and stored in last-covering(t). If the tuple t has an incomplete covering, fact that is reflected in the boolean variable $t.incomplete$ (lines 12 and 21), the algorithm does not compute its following coverings.

Algorithm 1: calculateCoverings (CSP P)

```
Data: A CSP $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ and $U$ (optional)
Result: |last-covering($t$)| $\forall t \in T(C_i) \forall C_i \in (\mathcal{C} \cup \mathcal{DC})$ and max-covering($P$)
1  GAC3(P);
2  $k \leftarrow 1$;
3  foreach $C_i \in (\mathcal{C} \cup \mathcal{DC})$ do
4      max-covering($P$) $\leftarrow \infty$;
5      $T(C_i) \leftarrow$ Ordered list of valid tuples of $C_i$;
6      foreach $t \in T(C_i)$ do
7          |last-covering($t$)| $\leftarrow 0$;
8          $t.incomplete \leftarrow False$;
9  repeat
10     foreach $C_i \in (\mathcal{C} \cup \mathcal{DC})$ do
11         $C_i.completeness \leftarrow False$;
12         foreach $t \in T(C_i)$ : $t.incomplete = False$ do
13             foreach $\{y \in T(C_i)$: $y < t \wedge d(t_1, y_1)_{Chebyshev} \leq k \}$ do
14                 if isNewneighbour ($k$,$t$,$y$) then
15                     |last-covering($t$)| $\leftarrow$ |last-covering($t$)| $+ 1$;
16                     if $y.incomplete$=$False$ then
17                         |last-covering($y$)| $\leftarrow$ |last-covering($y$)| $+ 1$;
18                     if isComplete ($k$,|last-covering($y$)|,$|y|$) then
19                         $C_i.completeness = True$;
20                     else if $\forall i \in [1, |y|], d(t_i, y_i)_{Chebyshev} = k$ then
21                         $y.incomplete = True$;
22         if $C_i.completeness = False$ then
23             max-covering($P$)= $k$;
24     $k \leftarrow k + 1$;
25 until $k$ >max-covering($P$) or $k > U$ (if $U$ is provided) ;
26 return |last-coverings|, max-covering(P) (if $U$ is not provided)
```

Procedure `isNewneighbour` (k,t,y) : Boolean

```
1 $equalDist \leftarrow False$;
2 for $i \leftarrow 1$ to $|t|$ do
3     if $d(t_i, y_i)_{Chebyshev} > k$ then
4         return False
5     if $d(t_i, y_i)_{Chebyshev} = k$ then
6         $equalDist = True$;
7 return equalDist
```

Procedure `isComplete` (k,|last-covering(t)|,$|t|$) : Boolean

```
1 maxTup($k, |t|$) := $(2k+1)^{|t|}$-1;
2 if |last-covering(t)| $\geq$ maxTup($k, |t|$) then
3     return True
4 else
5     return False
```

Firstly, Algorithm 1 initializes some necessary structures (lines $2-8$). Then the sets of valid tuples $T(C_i)$ of each constraint $C_i \in (\mathcal{C} \cup \mathcal{DC})$ are ordered by the value of the first variable of the valid tuples. In this way, a tuple a can only be located in a lower position than a tuple b if $a_1 \leq b_1$ (considering that the subindex 1 indicates the first variable that makes up the tuple). Thus, the tuples whose first variable has the minimum possible value will be placed in the lowest positions. For expressing the order of the tuples, we use the notation $a < b$, which means that the tuple a is located in a lower position than b in the list of ordered tuples.

The implementation of an algorithm for calculating coverings does not strictly require an ordered list of $T(C_i)$, but this ordering allows for the reduction of the computation time, because it avoids checking a pair of tuples twice by means of exploiting the symmetry of the neighbours relation. This temporal benefit is achieved by selecting a reduced subset of possible neighbours of each valid tuple $t \in T(C_i)$ which are located in a lower position of t. If we do not select this reduced set, the algorithm must check all the valid tuples for each valid tuple. The reduced set is a subset composed of the valid tuples that are ordered in a lower position than t in the ordered list of $T(C_i)$ and whose difference between the value of their first variable with respect to t is lower or equal to k. Thus, the reduced set of t is composed of $y \in T(C_i) : y < t \wedge d(t_1, y_1)_{Chebyshev} \leq k$ (line 13). Furthermore, other value orderings and therefore other reduced sets can be selected. Here, we select this one because it is quite discriminative and it only requires the time of computing the difference of two values for each tuple ordering.

The procedure `isNewneighbour` checks if a valid tuple y is a new neighbour in k-covering(t). This condition is determined by checking that at least one of the variables of y has a value difference of k with respect to t (line 5) and the rest of the variables have a value difference lower or equal to k (line 3). Algorithm 1 increases in one unit the value of the |last-covering| of each of the two tuples if procedure `isNewneighbour` returns $True$, but only if the tuple does not have any incomplete covering. The procedure that checks the latest is the procedure `isComplete`, which determines if a k-covering(t) is complete. Firstly, it calculates the maximum number of tuples of k-covering(t): maxTup$(k, |t|)$ (see Equation 4). Subsequently, it checks if |last-covering(t)| is equal to maxTup$(k, |t|)$. Algorithm 1 fixes the value of $y.incomplete$ to $True$ in line 21 if its analyzed covering is incomplete and y has had all the possible neigbours already analyzed. This condition is checked in line 20 by checking if all the variable values differences with respect t are equal to k (this means that t is the greatest neighbour, which is placed in the greatest corner according to the value ordering). For the rest of tuples the completeness is not checked because there are possible new neighbours of them that are ordered in bigger positions and therefore have not been analyzed yet.

As mentioned previously, Algorithm 1 does not compute k-covering(t) if (k-1)-covering(t) is not complete. However, this restriction could be deleted by skipping the completeness check of the tuples, that is to say, the boolean variables $t.incomplete$. In this case, we could not use the principle of minimum distance from the constraint (see Section 5.2). In this paper, this principle is necessary due to the main objective of finding solutions located far away from the constraint boundaries. Nevertheless, the coverings computation does not require the completeness property. Thus, both the covering concepts for CSPs and Algorithm 1 (with the modification discussed above) can be also applied to other areas that do not require this property.

6.1 Computational Cost

The complexity of Algorithm 1 is directly related to the number of coverings computed, the number of valid tuples of the constraints, the number of variables and the number of constraints and their arity. However, not all these parameters have the same impact in the computational cost. In the following, we analyze the impact of this factors in the steps that the algorithm carries out.

The first step of the algorithm is to develop a GAC3 process (line 1), which runs in $O(er^3d^{r+1})$, where d is the largest domain size, r is the greatest arity of the constraints and $e = |\mathcal{C}|$. In the second step Algorithm 1 sorts the set of valid tuples by the value of their first variable for each constraint $C_i \in (\mathcal{C} \cup \mathcal{DC})$. The value of the cardinality of the maximum possible set of valid tuples has an upper worst case bound, which is $d^r * (1 - t_M)$, where t_M is the minimum constraint tightness. The cost of sorting a set of p elements with the *quicksort* algorithm is $O(p * log(p))$. Thus, sorting the maximum set of valid tuples for all the constraints and domains is $O((e + n) * d^r * (1 - t_M) * log(d^r * (1 - t_M)))$, where $n = |\mathcal{X}|$.

Finally, the k-coverings are computed for each valid tuple t of each $C_i \in (\mathcal{C} \cup \mathcal{DC})$. The algorithm analyzes a reduced subset of possible new neighbours of t. The size of the subset depends on k and the number of valid tuples, being in the worst case equal to $((k + 1) * d^{r-1} * (1 - t_M))$. Note that the reason of subtracting one unit to r in d^{r-1} is due to one value variable is fixed to the value difference of k (the first variable), however the values of the rest of variables can have all the combinations in their domains. For calculating a covering of a valid tuple, all the variables of all the tuples of its reduced subset are checked. In the worst case, the algorithm calculates max-covering(CSP) (the user does not fix a U parameter), where max $= \lfloor \frac{d}{2} \rfloor$ (all the tuples are valid for all the coverings). The computation time of the second step of the algorithm is obtained by multiplying the maximum number of tuples in the reduced set, the arity of the tuples (r), the maximum possible number of tuples for each $C_i \in (\mathcal{C} \cup \mathcal{DC})$, the cardinality of the latest set ($e + n$) and the worst case of max-covering(CSP). The result (deleting the constant terms) results on: $O(d^{2r} * (1 - t_M)^2 * k * r * (e + n) * \frac{d}{2})$.

For calculating the total computational cost of Algorithm 1, we sum the three parallel steps: GAC3 process, ordering of the reduced set of tuples and calculating the coverings and we group common terms, which results on: $O((er^3 d^{r+1}) + (e + n) * d^r * (1 - t_M) * ((d^r * (1 - t_M) * k * r * \frac{d}{2}) + log(d^r * (1 - t_M))))$. Note that the number of valid tuples of the constraints is crucial in the computational cost of Algorithm 1. The number of valid tuples of each constraint is determined by the domain size of the variables and the tightness and arity of the constraints. For this reason, the GAC3 process (line 1), which restricts the search space by deleting inconsistent values, has a high impact on the reduction of the computational cost.

7 Modeling Robustness in a CSP as a WCSP

In this section we introduce an enumeration-based technique for modeling a CSP as a WCSP. In this way, we obtain a CSP model based on the |last-coverings| of the solutions for each constraint/domain. As we have pointed out before, under some conditions (Section 3), |last-covering($s \downarrow_{var(C_i)}$)| for a solution s can be considered a measure of its robustness for C_i, and the sum of |last-covering($s \downarrow_{var(C_i)}$)| for each $C_i \in (\mathcal{C} \cup \mathcal{DC})$ is an approximation of the robustness of s for the CSP. The WCSP model considers that the sum of the costs assigned to the tuples of each constraint determines how good a solution is for the WCSP. Thus, we model a CSP as a WCSP after obtaining its |last-coverings| by Algorithm 1.

Although there are other valued CSPs that could conceivably be used to model robustness, these models either involve operations that are questionable (e.g. probabilistic CSP, where valuations based on |last-coverings| would be multiplied) or are insufficiently discriminating (e.g. fuzzy CSP (FCSP), leximin FCSP). In addition, the WCSP adequately incorporates the enumeration aspect of coverings, unlike the other valued CSP models.

The WCSP modeling begins by assigning a cost to each valid tuple t involved in each constraint that represents its penalty as a function of its |last-covering(t)|. Tuples with the highest last-covering for C_i will have the lowest associated cost, because this value indicates the minimum distance of t from the bounds of C_i. Given the assumptions outlined in Sections 1 and 3, the further away the solution is located from the bounds of a constraint, more robust it is, because it is more resistant to possible changes.

Definition 18 We define *max-|last-covering(C_i)|* = max {|last-covering(t)|, $\forall t \in T(C_i)$}.

The penalty of a valid tuple t for a constraint C_i without considering the rest of the constraints of the CSP is denoted as $p_i(t)$ (see Equation 5).

$$p_i(t) = \text{max-}|\text{last-covering}(C_i)| - |\text{last-covering}(t)| \tag{5}$$

Nevertheless, this is not the final cost assigned to t. We must normalize the penalties of each C_i with respect to the other constraints, because the maximum possible size of the coverings depends on the arity of the tuples (see Equation 4). Otherwise, constraints with higher arity would have higher cost ranges and therefore higher penalties. In this case, we would be assuming that these constraints have a higher likelihood of undergoing restrictive modifications, which is not necessarily true according to the limited assumptions we are making for CSPs with discrete and ordered domains. By using a normalization process, we can achieve the same cost range for all the constraints. To obtain normalized scores, we use the maximum penalization assigned to the tuples of each constraint and the maximum penalization assigned to the tuples across all constraints (e is the number of constraints of the CSP according to Definition 5). In this way, the cost function for the tuples of C_i assigns a normalized cost to every tuple of C_i and it is denoted as $C_i(t \downarrow_{var(C_i)})$ (see Equation 6).

$$C_i(t \downarrow_{var(C_i)}) = \begin{cases} 0 & \text{if } t \in T(C_i) \text{ and } p_i(t) = 0 \\ \left\lfloor \frac{p_i(t) * max\{p_j(x), \forall j \in [1 \ldots e] \forall x \in T(C_j)\}}{max\{p_i(y), \forall y \in T(C_i)\}} \right\rfloor & \text{if } t \in T(C_i) \text{ and } p_i(t) \neq 0 \\ W, (W \approx \infty) & \text{if } t \notin T(C_i) \end{cases} \tag{6}$$

In line with the version of WCSP that we are using, $C_i(t \downarrow_{var(C_i)})$ assigns a cost of W to each tuple *t* that does not satisfy the constraint C_i because it is not a partial solution. Note that for valid tuples $C_i(t \downarrow_{var(C_i)}) \in [0, max\{p_j(x), \forall j \in [1 \ldots e] \forall x \in T(C_j)\}]$. A valid tuple *t* whose |last-covering(t)| = max-|last-covering(C_i)| ($p_i(t) = 0$) has an associated cost of 0. This tuple is not penalized because it has the highest likelihood of remaining valid when faced with future changes in C_i. On the other hand, if |last-covering(t)| $= 0$ for C_i, which means that t does not have any neighbour in its 1-covering(t) (t is completely non-robust for C_i), t has the maximum possible cost associated: $max\{p_j(x), \forall j \in [1 \ldots e] \forall x \in T(C_j)\}$.

Once the costs have been assigned, the WCSP is generated and solved using a WCSP solver. The solutions obtained for the modeled WCSP are also solutions of the original CSP. In addition, the best solution s with the minimum $\mathcal{V}(s)$ (see Equation 2) is taken to be one of the most robust solutions for the original CSP according to our assumptions.

8 Example

In order to clarify the concepts of coverings, we present an example with two variables (2-dimensional hyperpolyhedron CSP with non-convex constraints). We show only the costs assigned to the complete solutions.

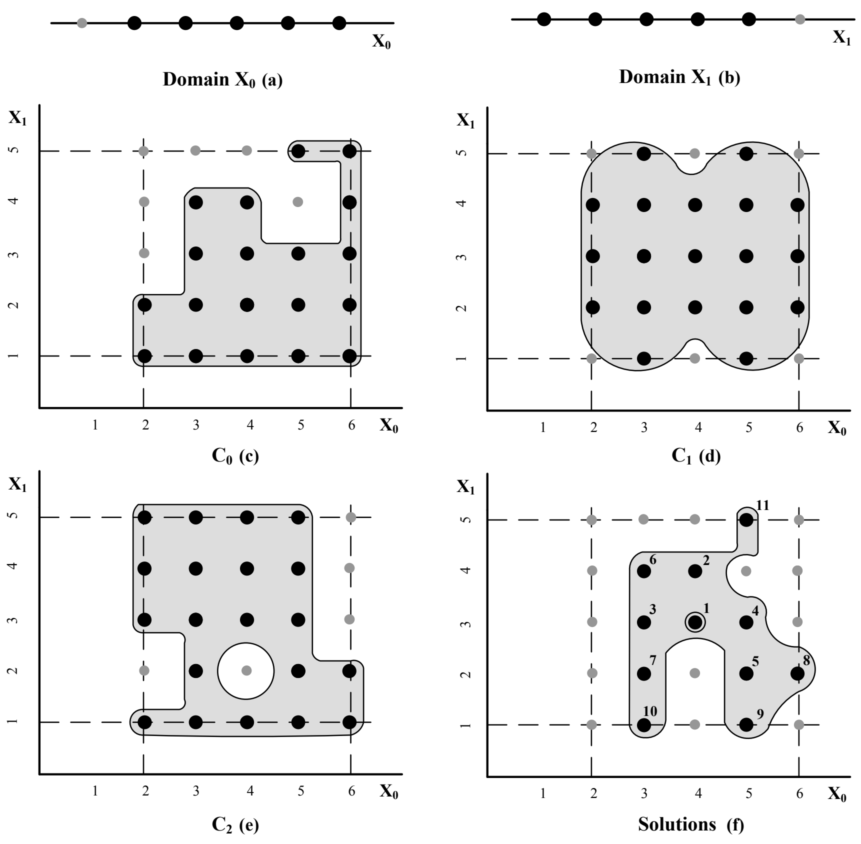


Fig. 4 Example of a extensional CSP *R*.

Example 5 Figure 4 shows a CSP *R* extensionally represented, which is composed of two variables x_0 and x_1 with domains $D_0 : \{2..6\}$ and $D_1 : \{1..5\}$, respectively. The domains associated with x_0 and x_1 are represented in Figures 4(a) and 4(b), respectively. There are three extensional constraints C_0, C_1 and C_2 (Figures 4(c), 4(d) and 4(e), respectively). The valid tuples of the constraints and domains are represented with big black points; the invalid tuples are represented with small grey points. Figure 4(f) shows the solutions of *R* labelled by their robustness levels as assessed by our technique for calculating coverings.

Table 2 shows the solutions of the CSP *R* in decreasing order of robustness (s_i). This order is inversely related to the $\mathcal{V}(s)$ obtained after solving the modeled WCSP. In addition, we present |last-covering(s)| for the solution space of *R*. Note that its calculation is viable because we are analyzing a toy problem. As expected, the obtained robustness results match with the |last-coverings| in the solution space: the solutions with higher |last-coverings| in the solution space are identified as more robust by our technique (see the highlighted columns in Table 2).

Table 2 Solutions ordered by their robustness.

s_i	solution(s)	|last-covering(s)|	$\mathcal{V}(s)$	$X_0(s)$	$X_1(s)$	$C_0(s)$	$C_1(s)$	$C_2(s)$
1	(4,3)	6	17	0	0	9	0	8
2	(4,4)	5	27	0	5	11	11	0
3	(3,3)	4	28	5	0	10	3	10
4	(5,3)	4	28	5	0	9	3	11
5	(5,2)	4	34	5	5	2	12	10
6	(3,4)	3	36	5	5	13	12	1
7	(3,2)	3	41	5	5	9	12	10
8	(6,2)	3	57	15	5	11	14	12
9	(5,1)	2	58	5	15	11	15	12
10	(3,1)	1	59	5	15	11	15	13
11	(5,5)	1	61	5	15	13	15	13

Table 2 also shows the costs assigned by the constraints (C_i) and domains (X_i), whose sum is $\mathcal{V}(s)$ (see Equation 2). To clarify the process of cost assignment, we explain one case, $C_0(s_3)$, in detail. Taking into account that max-covering(R)= 2, the penalty for $s_3 = (3,3)$ (before the normalization process) is $p_0(s3) = 16 - 6 = 10$ (see Equation 5), since max-|last-covering(C_0)| $= 16$ (|last-covering($(4,2)$)| $= 16$ for C_0, which is the maximum for C_0) and |last-covering(s_3)| $= 6$ for C_0. The associated cost (considering the normalization) is: $C_0(s_3) = \lfloor \frac{10*15}{14} \rfloor = 10$ (see Equation 6), since $max\{p_j(x), \forall j \in [1 \dots e] \forall x \in T(C_j)\} = 15$ ($p_1((3,1)) = 18 - 3 = 15$, which is the maximum for *R*) and $max\{p_0(y), \forall y \in T(C_0)\} = 14$ ($p_0((6,5)) = 16 - 2 = 14$, which is the maximum for C_0).

The best solution found is the solution $s1 = (x_0 = 4, x_1 = 3)$ and its |last-covering($s1$)| $= 6$ in the solution space, which is the highest for R (see Figure 4(f)). As previously mentioned, the solution s with the highest |last-covering(s)| in the solution space is the solution located furthest away from the search space bounds. As a result, its likelihood of remaining valid when faced with future restrictive modifications over the bounds of the solution space, is higher. Therefore, it is considered the most robust solution for the original CSP.

In contrast, the solutions $s10 = (x_0 = 3, x_1 = 1)$ and $s11 = (x_0 = 5, x_1 = 5)$ are the least robust solutions since their |last-covering| $= 1$, which is the lowest value for R. These solutions only have one neighbour in their 1-covering in the solution space (see Figure 4(f)). Thus, it is very probable that they will become invalid after restrictive modifications over the original constraints of the CSP.

9 Case Study

In this section, a case study of a scheduling problem is analyzed to graphically show the robustness of solutions with and without the use of our technique. To this end, we analyze two problems derived from Taillard optimization problems (Taillard (1993)): “os-taillard-4-100-0” and “os-taillard-4-105-0”. Both are well-known problems and were used in the CSP solver competition[1]. These problems were also used by Wallace

[1] *http://www.cril.univ-artois.fr/ lecoutre/benchmarks.html*

and Grimes in their reasoning-reuse technique for finding new similar solutions to the original solutions lost for DynCSPs (Wallace and Grimes (2010)). We would like to point out that our technique is not developed specifically for scheduling problems but for general CSPs.

Both problems come from the same scheduling instance and were converted to satisfaction problems by fixing the maximum makespan allowed (latest finishing time). For the "os-taillard-4-100-0" problem the makespan is set to its best known makespan. Thus, all the schedules for this problem are optimal. However, the maximum makespan allowed for the "os-taillard-4-105-0" problem is set to 105% of the best makespan. Therefore, the schedules obtained for this problem may be, and usually will be, non optimal. The motivation for analysing this problem is the well known trade-off between robustness and stability, and optimality. Since the open shop benchmark is composed of 4 machines, 4 jobs and 4 tasks per job, the resultant CSP model contains 16 variables and 48 constraints; the latter prevent two tasks from using the same machine at the same time as well as ensuring that two tasks of the same job do not overlap.

Figure 5 shows a non-robust optimal solution in which no covering analysis was carried out. The jobs are represented on the vertical axis and the time is represented on the horizontal axis. Tasks are shown in light grey with the number of tasks and the machine assigned to each task. The striped buffer times represent natural buffer times produced because the earliest starting time for the next related task in any job depends on the release of a machine or the end of another task, because of a gap between the last task of a job and the makespan. It can be observed that the optimal solution has only 7 natural buffer times between tasks, so a delay at any other place in the schedule will invalidate the obtained solution. Only tasks 3, 4, 10 and 15 have a large buffer time, which it is able to absorb a longer delay.

For searching for robust solutions for this problem, since most solutions in scheduling problems start at 0, we have included this condition, in order to exclude the first tasks of every job in the robustness computation. We have modeled these problems as well as the problems presented in Section 10 as WCSPs by following the WCSP file format[2]. In addition, ToulBar2[3] has been used for solving the resultant WCSPs.

Figure 6 shows an optimal solution obtained for the "os-taillard-4-100-0" problem using our technique for $U = 1$. It can be observed that it has additional buffer times (in dark grey). These buffer times are not produced because the earlier starting next task related with a task is not able to start before; for this reason, they are distinguished from the natural buffer times. Because of them, this optimal schedule is more robust than the previous one, since it is able to withstand a greater variety of delays. In this schedule, 5 new buffer times were generated but most of them were small. The total robust buffer times summed to 7 time units.

Figure 7 shows an optimal solution obtained for "os-taillard-4-100-0" problem by using our technique for max-covering(CSP). Here, Algorithm 1 calculates the maximum covering, which is 4-covering. It can be observed that it also has 5 robust buffer times; however two of them are larger. As before, the number of robust buffer

[2] *http://graphmod.ics.uci.edu/group/WCSP_file_format*

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

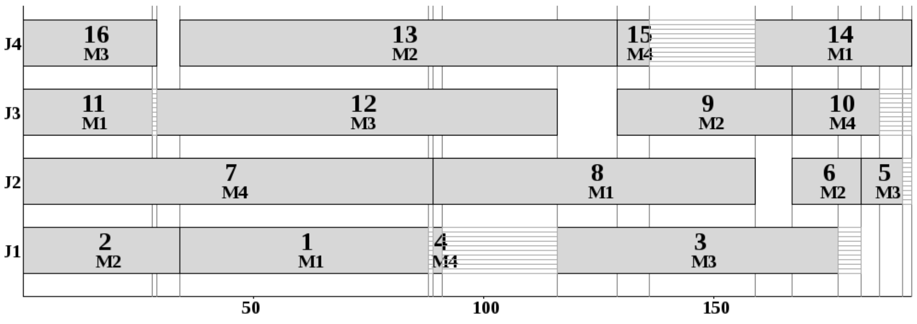


Fig. 5 Non-robust Schedule (makespan=193s)

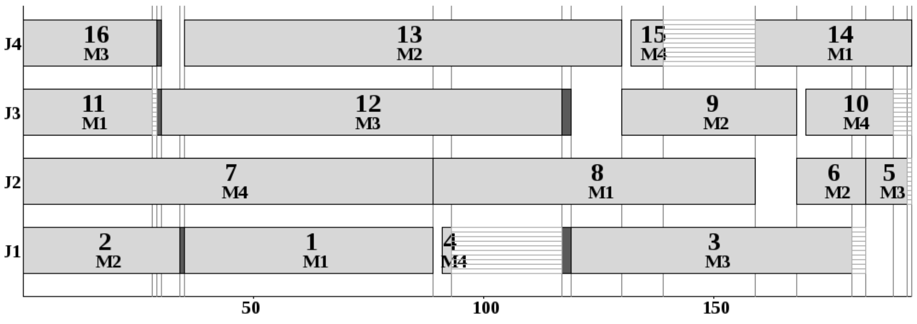


Fig. 6 1-covering Schedule for “os-taillard-4-100-0” (nbRobustBuffers=5, robustBuffersSum=7, makespan=193)

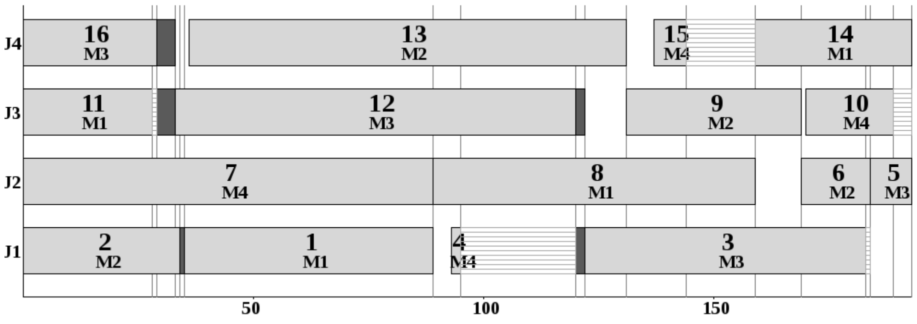


Fig. 7 4-covering (max-covering(CSP)) Schedule for “os-taillard-4-100-0” (nbRobustBuffers=5, robustBuffersSum=13, makespan=193)

times and their duration offer a robustness measure of the schedule, since they are able to absorb more delays than the previous schedules. The more buffer times there are and the bigger they are, the higher the probability that the rest of the schedule remains satisfiable after changes. For instance, in Figure 7 tasks 11 and 16 can afford a delay up to 4 time units. However, in Figure 6 they only can afford a delay of 1 time unit. The total sum of buffers of the schedule in Figure 7 is 13 time units.

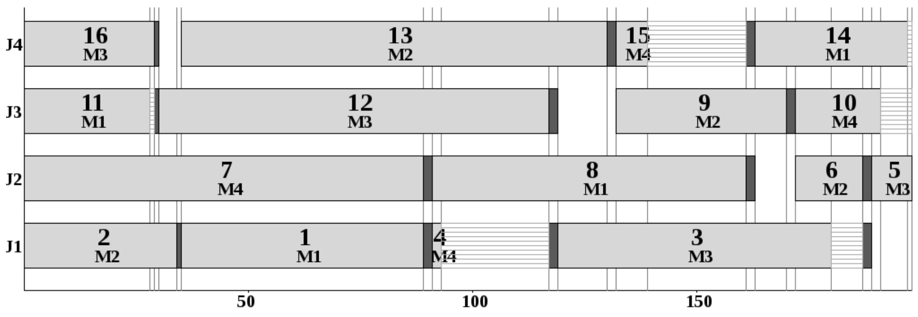


Fig. 8 1-covering Schedule for "os-taillard-4-105-0" (nbRobustBuffers=13, robustBuffersSum=23, makespan=198)

Figure 8 shows the schedule obtained for "os-taillard-4-105-0" problem by using our technique for $U = 1$. As expected, the resulting schedule was more robust than the previous ones, since it has more robust buffer times and their sum was bigger (13 robust buffer times whose sum was 23 time units). However, it is not optimal because the makespan increased from 193 to 198. In Section 10.1 there is a deeper evaluation of this trade-off between robustness and optimality for the aforementioned problem.

Table 3 presents the computational cost (in seconds) to obtaining these schedules. For calculating simple solutions (it is considered only the satisfiability of the modeled CSP) we have used the same WCSP solver by associating no penalisation to all the tuples. All the tests were executed on a Intel Core i5-650 Processor (3.20 Ghz). The table is divided in 3 processes: GAC3, WCSP modeling (including the time requiring for calculating the coverings with Algorithm 1) and the solving time required by ToulBar2. Note that for these small instances, the solving time of our modeled WCSPs and simple CSPs is similar. However, for greater instances, they are not comparable, because the solving time of the modeled WCSPs is much higher because it is an optimization problem (see its complexity in Section 6) in contrast wa satisfiability problem of CSP, which is NP-complete generally.

Table 3 Computational time (in seconds).

		GAC3	WCSP modeling	solving
os-taillard-4-100-0	simple solution	0.046	-	0.076
	1-covering solution	0.046	0.588	0.078
	4-covering solution	0.046	2.517	0.078
os-taillard-4-105-0	simple solution	0.048	-	0.095
	1-covering solution	0.048	1.444	0.104

10 Experimental Results

In this section, we perform several experiments to evaluate the performance of our technique for calculating coverings. Experiments were done with random problems and benchmarks presented in the literature. The random instances generator (RBGenerator 2.0), the benchmarks and the parser for the XCSP instances can be found on Christophe Lecoutre's web page [4].

10.1 Trade-off Robustness-Optimality in Scheduling

In Section 9 we studied the benchmark "os-taillard-4-100-0" by finding optimal and non-optimal robust solutions. As it is well known in the literature, there exists a trade-off between robustness and optimality. In this subsection we evaluate this trade-off more extensively by fixing the makespan over a range of different percentages of the best makespan. The solutions were obtained by our technique for $U = 4$, since max-covering$(CSP) = 4$ for "os-taillard-4-100-0", which is the most constrained instance. Figure 9 shows the makespan of the schedules on the horizontal axis, the number of robust buffers on the left vertical axis and the sum of the robust buffers on the right vertical axis. Similar computational times to the schedule represented in Figure 7 have been obtained, consuming an extra time of 2.5 s approximately with respect obtaining a simple schedule.

As expected, the robustness of the solutions obtained is positively correlated with the makespan. It can be observed that for a solution whose makespan is near the optimal makespan (makespan = 199), there is a large proportional increase in the number of robust buffers with respect to the number for the previous solution (makespan = 193). There is also a point (makespan = 214) where the solution has the same number of buffers as the previous one (makespan = 209). The reason is that by this point almost all tasks (or all of them) have one robust buffer associated, so finding further buffers becomes more difficult. On the other hand, the sum of buffer times increases in a more uniform fashion, which shows that it is more difficult to find new buffers than to increase the size of a buffer.

10.2 Robustness Analysis with DynCSPs

To the best of our knowledge, there are no benchmarks of DynCSPs (see Definition 9) in the literature, so authors working in this field simulate changes in CSPs in order to generate DynCSPs. For instance, in Wallace and Grimes (2010) a type of dynamism simulation over random CSPs was to randomly change tuples constituting particular relations. Thus, the number of constraints remains the same after each alteration. For CSPs with ordered domains, the authors reduced the maximum domain values and restricted the constraints (when they were restricting the CSPs, since they also consider CSP relaxations).

[4] *http://www.cril.univ-artois.fr/ lecoutre/index.html*

We have generated DynCSPs that are composed of $l+1$ static CSPs: $\langle CSP_{(0)}, CSP_{(1)}, ..., CSP_{(l)}\rangle$, where $CSP_{(0)}$ is the original CSP. We have created two types of DynCSPs: ones with dependent changes and ones with independent changes. For the DynCSPs whose changes are dependent, each $CSP_{(i)}$ is generated from $CSP_{(i-1)}$ by making a restrictive modification with respect to some bound (constraint or domain) of the solution space of $CSP_{(i-1)}$. On the other hand, if the changes are independent, each $CSP_{(i)}$ includes a restrictive modification with respect to some bound of the solution space of $CSP_{(0)}$. Notice that in the first case the changes are cumulative but in the second case they are not. Since dependent changes are cumulative, for this type of change we have selected constraints and domains for restriction based on their relative frequency. Thus, if there are x times more constraints than domains for a CSP, x constraints are restricted for each domain that is restricted. The reason of making this 'equality' distribution is restricting in the same proportion instances with different number of variables and/or constraints.

We simulated restrictive changes over the bounds with different magnitude of change d. To do this, we randomly selected an invalid tuple located next to any bound of the CSP and invalidated all the tuples surrounding it to a distance of d. For DynCSPs whose changes are dependent we fixed $d=1$. (The changes are cumulative; therefore, a big d would very markedly restrict the CSP). In this case, only valid tuples located on a bound became invalid. As stated, a tuple is located on a bound if at least one of its closest neighbour tuples is invalid. Note that if a valid tuple becomes invalid for $CSP_{(i)}$, a valid tuple located next to the new invalid tuple can become invalid for $CSP_{(j)}, \forall j > i$. For DynCSPs whose changes are independent we have selected bigger magnitudes of change since the changes are non-cumulative. Thus, d has been randomly fixed to a different value in the interval $[1,\ldots,max_d]$ for each DynCSP evaluated. Note that if $d>1$ valid tuples that are not located on a bound can became invalid also (but they have to be at distance lower or equal to d from some bound).

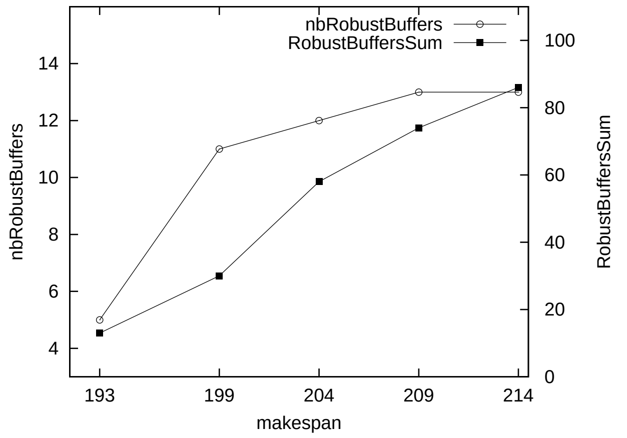


Fig. 9 Robustness analysis for the benchmark "os-taillard-4-100-0" by setting its makespan to 100%, 105%, 108%, 110% and 113% of the best makespan.

For each DynCSP, new restrictive CSPs were generated until the obtained solution became invalid. Thus, l indicates the number of restrictive changes that the solution was able to resist for a DynCSP. We have generated 500 random different DynCSPs of each type for each problem and computed the mean number of changes satisfied, which can be considered a measure of the robustness of the solutions according to the limited assumptions made for the type of CSP analyzed.

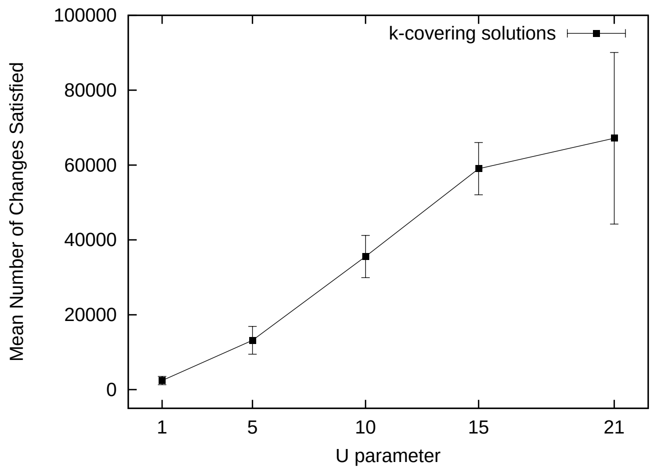
10.2.1 U Parameter Analysis

We have analyzed the "lard-84-84" benchmark, which is a large benchmark problem proposed by Marc Van Dongen for the 2006 CSP Solver Competition. This problem is composed of 84 variables, each with a domain size of 85, and 3486 binary constraints. We have chosen this benchmark because it has a large set of solutions, which it makes possible to graphically show the solution robustness for a wide range of U values.

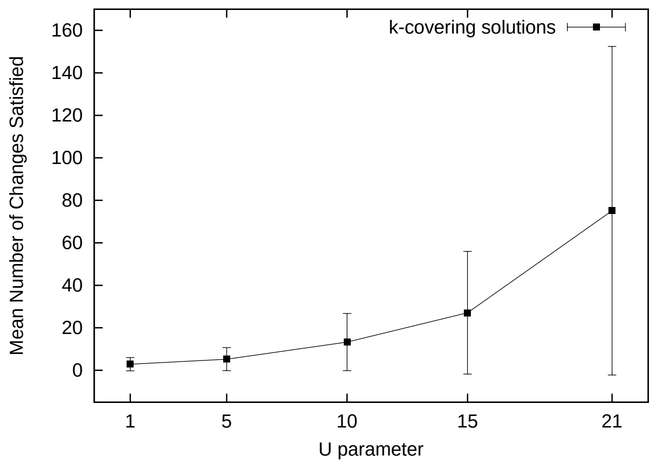
In Figure 10, the left vertical axis measures the number of supported changes. As one might expect, we can see in Figures 10(a) and 10(b) that the mean number of cumulative or independent changes that can be satisfied both increase markedly as U increases. For independent changes, we have fixed $max_d = 25$, which represents a large maximum magnitude of change because it is almost 30% of the domain size of the problem. Because max_d has a relatively high value, the mean number of independent changes is much lower than the mean number of dependent changes (see Figure 10). For instance, Figure 10(b) shows that the maximum mean of independent changes satisfied is 75.13, which is reached when Algorithm 1 calculates the maximum covering, which is the 21-covering for this problem. On the other hand, Figure 10(a) shows that 67159.09 dependent changes are satisfied by the solution found for the same covering. We would like to note that the magnitude of dependent changes is 1; therefore, for loosely constrained large benchmarks such as the one analyzed in this subsection, this kind of change has only a minor impact.

From this analysis we can conclude that by increasing U in Algorithm 1, we can obtain several grades of robustness in our solutions. However there is a point after which is difficult or impossible to improve the robustness results because the solution found by our technique is one of the solutions located as far as possible from the bounds of the solution space. Thus, in Figure 10(a) we see that the slope for mean number of changes satisfied becomes less marked for higher values of the maximum U parameter.

With respect to the standard deviation, it can be observed in Figure 10 that it is higher for solutions that are able to support a higher mean number of changes (more robust solutions). It must be taken into account that the restrictive changes are completely random. Therefore, the variability in the sequences of changes is very high for every developed simulation. The lower the average robustness of a solution is, the shorter these sequences are, and therefore the variability between the sequences is also lower. In Figure 10 we can see that the standard deviation of independent changes is bigger than the standard deviation of dependent changes. The main reason is that for independent changes the magnitude of change varies randomly according to max_d and for this analysis max_d has been fixed at a high value.



(a) Dependent Changes



(b) Independent Changes

Fig. 10 Robustness analysis based on U parameter for the benchmark *lard-84-84*.

For this benchmark and the following random CSPs analyzed, we also have analyzed another proactive technique in the literature: finding super solutions (see Section 3.1 and Hebrard (2006)). The main reason of choosing this technique is that the assumptions made concerning possible future changes are limited, as with the assumptions made by our technique. Therefore, both techniques do not require specific additional information about the future possible changes, as for example probabilities of change or a list of possible changes. Note that this technique does not search for robust solutions as such, but it does search for stable solutions. The algorithm implemented for the super-solutions is the Branch and Bound that maximizes the number of repairable values for (1,0)-super-solutions by using MAC+ (Hebrard (2006)). The time cutoff is fixed to 200 seconds. In addition, we analyzed the robustness of a simple solution obtained by a CSP solver. This analysis has not been included as an alternative to our technique obviously, but in order to detect whether there are cases in which all solutions have similar robustness. For both techniques, values were selected in lexicographical order.

Regarding the stability of the solutions obtained by the k-covering technique for the "lard-84-84" benchmark, all the obtained solutions are (1,0)-super-solutions (see Definition 4), which means that all the variables are repairable faced with the loss of one value at most. Note that due to this fact, any of these solutions could be chosen as the most stable solution by the search algorithm for finding (1,0)-super-solutions developed in Hebrard (2006). If the values are explored in lexicographical order, the solution provided by the algorithm for finding (1,0)-super-solutions for this benchmark is the same as the simple solution provided by a usual CSP solver in which the values are also explored in lexicographical order. This solution satisfies a smaller mean number of changes than any of the solutions obtained by our technique: 44.45 for dependent changes and 2.73 for independent changes. Therefore, all the solutions obtained by our technique for this benchmark are (1,0)-super-solutions and in addition, they are more robust than solutions found by the technique for finding (1,0)-super-solutions.

10.2.2 CSP Parameters Analysis

The main goal of the work presented in this subsection is to analyze the robustness of the solutions obtained for a wide interval of values of the CSP parameters. For this purpose, random CSPs were generated with RBGenerator 2.0 whose format is XCSP 2.1[5]. In addition, the previous experiments developed are composed of intensional convex constraints, and since our approach is also working for extensional and non-convex constraints, we have used this random generator because produces this type of random CSPs. The values of the domains are integer discrete values in the interval $[0, |D| - 1]$, where $|D|$ is the domain size. In generating these problems, we used the option of the RBGenerator 2.0 that merges the constraints of similar scope (this makes changes over the constraints more uniform).

In addition to assessing our technique for max-covering(CSP), we analyzed the robustness of solutions obtained with the (1,0)-super-solutions technique and those obtained by a solver that searches for a simple solution (as in previous experiments). However, for the simulation of independent changes, we have chosen a smaller max_d value than the used for "lard-84-84" benchmark, commensurate with the smaller domain sizes of these problems. Thus, we have fixed $max_d = 5$. We considered the following CSP parameters: number of variables, domain size, constraint graph density (number of constraints with number of variables held constant), tightness of constraints, and constraint arity.

Regardless of the parameter examined, the robustness results show a general pattern related to the overall level of constrainedness of the CSPs. First we describe this

[5] *http://www.cril.univ-artois.fr/ lecoutre*

general pattern, and subsequently we will give more detailed results for specific CSP parameters.

For all CSPs that are not highly restricted, the mean number of changes before solution breakage for solutions obtained by the k-covering technique is higher than the number of changes for solutions obtained by the (1,0)-super-solution technique or for simple solutions. For CSPs that are very highly restricted, the number of changes allowed by solutions obtained by the three methods is similar. This is because in these cases the CSPs have very few solutions and consequently the distances of *all* solutions from the bounds is very low. For most of these instances, the number of solutions is so low that the solutions are scattered within the tuple-space, so the likelihood of a solution being located on the bounds of the solution space is very high. It can even be the case that none of the solutions has a neighbour solution inside its 1-covering. But if there are no solutions with at least one neighbour solution, all the solutions are equally robust under the present assumptions regarding change.

For problems of low constrainedness, the differences in robustness between solutions obtained by our technique and those obtained by other methods are accentuated. For these CSPs, the solution space is greater and therefore the likelihood that there are solutions surrounded by neighbour solutions is also higher. On the other hand, this feature presents a disadvantage for techniques that search for (1,0)-super-solutions or for simple search, because the less restricted the CSP is, the closer the values obtained by either of these techniques are to the minimum values of the domains. In consequence, the likelihood of these solutions remaining valid after changes to the domains of the CSP is very low. Although both techniques find similar solutions for this type of CSP, the reasons for this fact are different in each case. Obviously, selection of values close to the smallest domain value for the algorithm that searches for a simple solution occurs because its value ordering is lexicographical. With the super-solution technique, since the solution space is higher for slightly restricted CSPs, the likelihood that there is a large percentage of solutions with a high number of repairable values is greater also. Thus, among all equally stable solutions this technique also finds the solution according to the lexicographical value ordering.

With respect to the two types of DynCSPs generated, we found that the mean number of dependent changes satisfied is higher (and less variable) than the mean number of independent changes satisfied, for all of the parameters analyzed. This is largely due to the magnitude of the changes simulated. Although the dependent changes are cumulative, each change simulated has magnitude 1. However for independent changes the magnitude can be up to 5. These stronger restrictions are, of course, more likely to render the solution invalid.

In the following we present more detailed results for three of the parameters: variables, domain size and tightness. The results for number of constraints were similar to those for tightness (although for cumulative changes the differences were smaller), while results for arity showed marked differences for the highest arity tested (4 as opposed to 2 and 3) that corresponded to differences found in other experiments. In subsequent figures, problems are represented as: $< arity$, number of variables: $|\mathcal{X}|$, domain size: $|D|$, number of constraints: $|\mathcal{C}|$, $tightness >$. In each figure, the left vertical axis shows the mean number of supported changes with the standard deviation (continuous line) or the modeling time required by our technique (discontinuous line). The modeling time includes the time required by Algorithm 1 for calculating the coverings and also the time necessary for modeling the CSP as a WCSP (by means of the cost function of Equation 6). Recall that all tests were executed on a Intel Core i5-650 Processor (3.20 Ghz). Data for dependent and independent changes are shown separately.

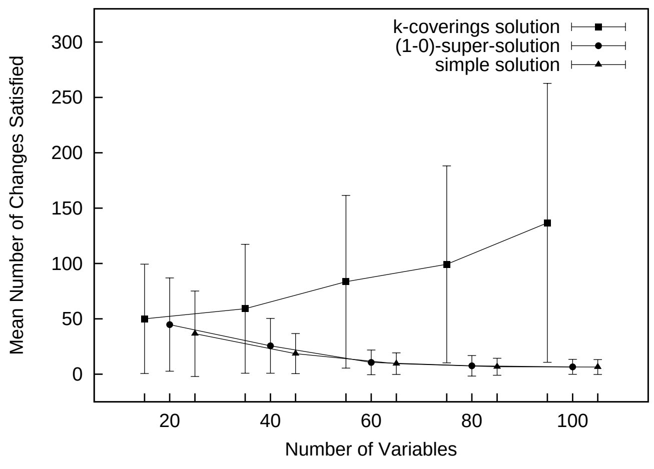
Figure 11 shows robustness results for the number of variables of the CSP is varied. In Figure 11(a) we see that the k-covering solutions satisfied a higher mean number of dependent changes for higher numbers of variables. (This is because the larger CSPs are less constrained and therefore the number of solutions is higher.) The maximum number of cumulative changes required to invalidate the k-covering solutions is 136.72, which was found for the maximum number of variables evaluated (100 variables). On the other hand, for the same condition, the simple solution and the (1,0)-super-solution had their worst robustness results (just 6.49 and 6.63 for mean number of dependent changes without breakage, respectively). For the smallest number of variables evaluated (20 variables), the corresponding differences between the k-covering solution and the other two solutions is much lower because this problem is highly constrained.

On the other hand, the robustness results obtained for our solutions for DynCSPs with independent changes (see Figure 11(b)) are similar over all the problems. This is because increasing the number of variables does not increase the number of valid tuples associated with each constraint, which is the information that our algorithm uses for penalizing the valid tuples in order to find robust tuples.

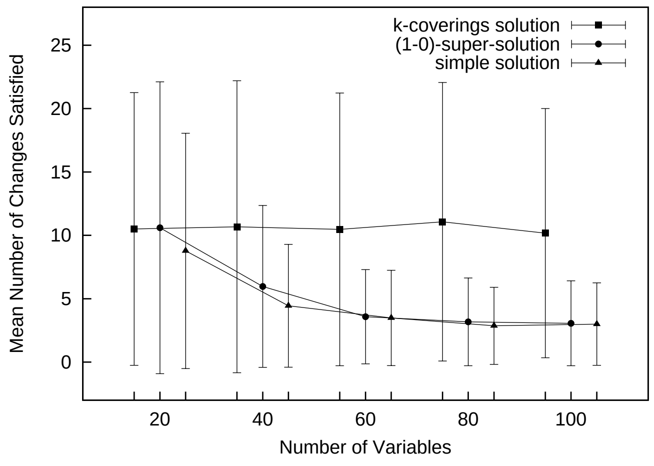
Figure 11(c) shows the modeling time required by our technique under each condition. We see that although there is an abrupt increase in time for the instance that has 80 variables, this increase is only $0.016s$. Increasing the number of variables only generates new unary constraints associated with the domains (whose number of new valid tuples is their domain size), and for this reason the increase of the time is not very significant in comparison with other CSPs parameters.

These results were evaluated statistically, first with a two-factor Analysis of Variance (ANOVA), followed by the Tukey HSD test for differences between pairs of individual means (Hays (1973); Winer (1971)). For the experiment with cumulative changes, the main effects (algorithm and problem) were highly significant; for algorithms, F(2,7485)=1948.1, $p <<$ 0.001; for problems, F(4,7485)=137.3, $p <<$ 0.001. In addition, the interaction was statistically significant: F(8,7485)=149.3, p $<<$ 0.001. Because of the large number of tests made under each condition, the Tukey HSD statistic was very small (approx. 0.1), so all but one of the differences between means were statistically significant. For the experiment with independent changes, the F statistics were somewhat smaller, but still greater than 100, so both the main effects and interaction were highly significant statistically. Here HSD = 0.08, so almost all differences between means were statistically significant.

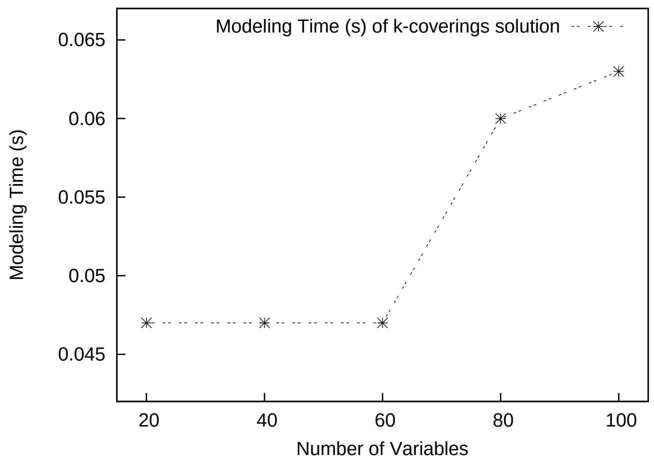
Figure 12 shows the robustness analysis for changes in domain size. In Figures 12(a) and 12(b), we observe that with our technique the mean number of cumulative or independent changes that leave the solution satisfied is markedly increased as domain size increases. In these cases, the CSPs are much less constrained and in addition the number of valid tuples associated with each constraint is greater. (This means that the likelihood of finding a tuple surrounded by a high number of valid



(a) Dependent Changes



(b) Independent Changes



(c) Modeling Time (s)

Fig. 11 Robustness analysis based on the number of variables ($<2, |\mathcal{X}|, 18, 200, 0.22>$).

tuples is greater.) The largest mean for cumulative changes before solution breakage is 869.12, while for independent changes it is 311.06. These best robustness results for solutions computed by the k-covering were obtained for the maximum domain size analyzed, which was 90. For the same condition, the other two methods found solutions that were distinctly less robust. Thus, the mean number of changes before solution breakage was less than 10% of the mean with our method for cumulative changes and less than 22% of the mean for sequences of independent changes.

As shown in Figure 12(c), the maximum difference in modeling time required for the different types of instances is $0.78s$. The main reason for the increase with increasing domain size is that this parameter is directly related to the number of valid tuples of the constraints, which has a high impact on the computational time of Algorithm 1.

For both domain size experiments, the ANOVA gave F values that were highly significant statistically, for both of the main effects and for the interaction. Given the small value for HSD (again, about 0.1 in each case), all differences between individual means were statistically significant with $p = 0.01$.

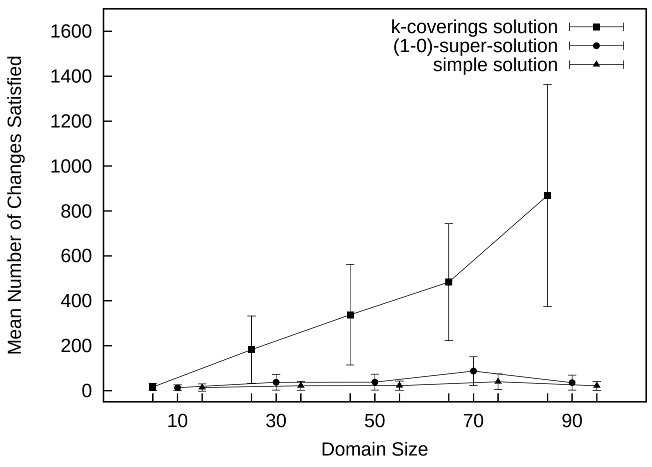
Figure 13 shows the robustness analysis when tightness of the constraints is varied. In Figure 13(a) we see that the mean number of cumulative changes before solution breakage decreases as tightness increases and the problems become more constrained overall. The maximum number of cumulative changes allowed by the the k-covering solutions is 289.83, which was found for the lowest tightness value (0.1). For this instance, the simple solution and the (1,0)-super-solution had their worst results since their solutions became invalid after fewer than 8 changes. For the maximum tightness evaluated (0.9), even when the problem had only 5 solutions the k-covering algorithm was able to find solutions that remained valid for a few more dependent changes than solutions found by the other algorithms (the mean difference was 6).

Figure 13(b) shows that the mean number of independent changes allowed by solutions found by the k-covering algorithm also decreased as tightness increased. Although this decrease is not as marked as for cumulative changes, there was a difference of almost 10 between the means for the most and the least constrained problems.

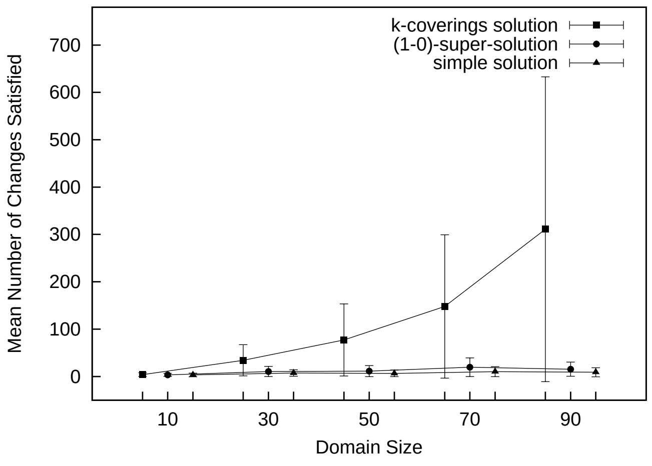
Figure 13(c) shows the modeling time required by our technique for each instance. The maximum difference between instances was $0.155s$. As mentioned, tightness is related to the number of valid tuples of the constraints. For this reason, Algorithm 1 spends more time finding the coverings for low tightness values.

For both tightness experiments, the ANOVA gave F values that were highly significant statistically, for both of the main effects and for the interaction. Again, given the small values for HSD (about 0.1 and 0.08), nearly all differences between individual means were statistically significant for $p = 0.01$.

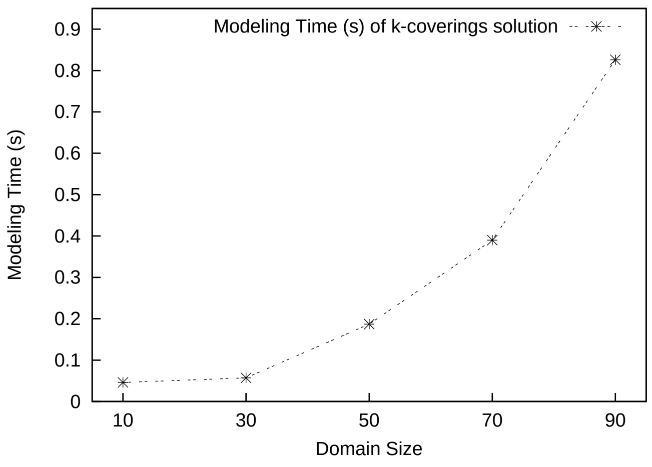
Collectively these experiments show that our method gives a striking improvement in solution robustness for problems that are not highly constrained, in particular those with a larger number of variables (when other features are held constant), with large domain sizes and/or high arity, or with low tightness and/or a low number of constraints. The differences are stronger for domain size, arity and tightness parameters, since a variation in these parameters has a higher impact on the size of the solution space and also on the number of valid tuples associated to each constraint.



(a) Dependent Changes

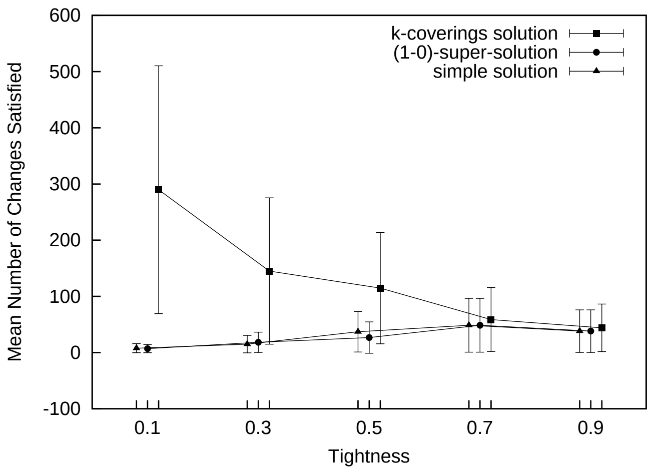


(b) Independent Changes

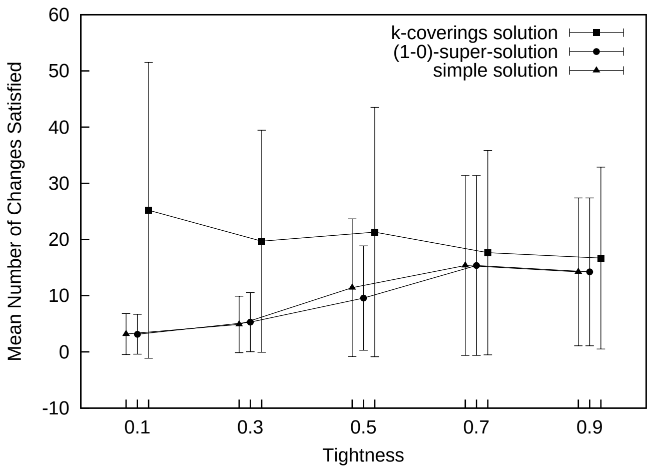


(c) Modeling Time (s)

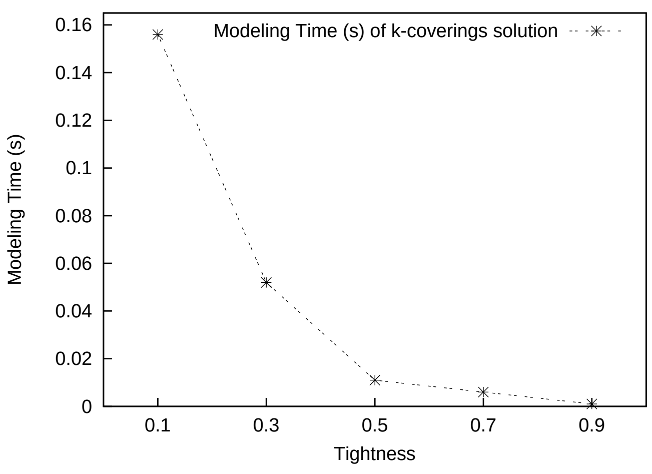
Fig. 12 Robustness analysis based on the domain size ($< 2, 25, |D|, 75, 0.45 >$).



(a) Dependent Changes



(b) Independent Changes



(c) Modeling Time (s)

Fig. 13 Robustness analysis based on the tightness ($<$ 2, 40, 25, 120, $tightness >$).

On the basis of our experiments with cumulative changes of small magnitude and independent changes of varying magnitude, we can conclude that solutions found with our methods have a higher likelihood of remaining valid than solutions found with other methods when faced with changes of the either type, although the effects are more dramatic in the first case. As mentioned earlier, it is often more likely that restrictions on the bounds of the solution space will be of smaller than larger magnitude. It is therefore encouraging that this is where we find the greatest improvements with our methods.

11 Conclusions

This paper presents an approach to solution robustness for Dynamic CSPs with discrete and ordered domains where only limited assumptions are made about the changes that can occur, commensurate with the structure of these problems. In this context, it is reasonable to assume that, when problems change, the original bounds of the solution space may undergo restrictive modifications. Therefore, the main objective in searching for robust solutions is to find solutions located as far away as possible from these bounds.

Our main objective is encapsulated in the framework introduced in Section 5, in which the concept of an 'onion topology' is applied to CSPs. Using this framework, we develop the basic concept of coverings, which we then apply to the problem of finding robust solutions. We also present an algorithm that is, to the best of our knowledge, the first practical method for calculating coverings for CSPs.

In order to find robust solutions, we have developed an enumeration-based technique, which is based on the covering concept, and which models CSPs as WCSPs by assigning a cost to each valid tuple of every constraint. The cost of each valid tuple t is obtained by calculating $|\text{last-covering}(t)|$ for the corresponding constraint. The obtained solution for the WCSP is a solution for the original CSP that has a high probability of remaining valid after restrictive modifications of the constraints and domains of original CSP.

We have evaluated these new techniques for finding robust solutions in experiments (and case studies) on both structured and random CSPs. We have shown that they can dramatically outperform both ordinary CSP algorithms and algorithms that find (1,0)-super-solutions (or maximize the number of repairable variables in case that there does not exist a (1,0)-super-solution) under many conditions where there are real differences in the robustness of solutions that might be obtained. The latter occurs under conditions where the constraints on the problem are not so great that there are only a few valid solutions.

In particular, we simulated two types of change: a sequence of cumulative changes where each step is of small magnitude, and a sequence of independent changes, each starting with the same base problem, where individual changes could vary in magnitude. Although differences in robustness were more dependable in the former situation (cumulative small changes), in both situations comparisons with the other search methods yielded results that were highly significant statistically.

In these experiments, search time depends strongly on the domain size of the variables, the tightness and arity of the constraints. However, it is possible to reduce this time by fixing a lower limit U, if the user is willing to possibly sacrifice a certain level of robustness in the solutions obtained in order to obtain a solution more quickly. This was demonstrated by tests using a large benchmark problem, where we showed that we can obtain more robust solutions for higher values of the U parameter.

After applying our technique to a well-known form of scheduling problem, we obtained schedules that are optimal *and* robust, because they have more extensive buffer times, and so are able to absorb greater delays in the tasks. Furthermore, as expected, we obtained more robust schedules by sacrificing the optimality of the schedule (trade-off between robustness and optimality).

These techniques can be applied to many real-world problems in which the order over elements of domains is significant. In many of these problems there is an added difficulty in that the environment is not only dynamic but also highly uncertain, because information about the possible future changes is limited or nonexistent. Since the techniques presented in this paper require only assumptions about the nature of change that are largely inherent in the structure of these problems, they are able to provide robust solutions even under these difficult conditions.

In the future, we plan to extend the techniques proposed here to handle constraint satisfaction and optimization problems (CSOP). Thus, our model could be extended by including other kinds of optimization (for example, minimizing the time, maximizing the profit, etc.). We consider that this step would increase the versatility of our technique, since we could potentially find robust solutions while meeting several other optimality criteria.

References

Climent L, Salido M, Barber F (2010) Robust Solutions in Changing Constraint Satisfaction Problems. In: Proceedings of the 23rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE-2010), pp 752–761, Part 1

Climent L, Salido M, Barber F (2011) Reformulating dynamic linear constraint satisfaction problems as weighted csps for searching robust solutions. In: Ninth Symposium of Abstraction, Reformulation, and Approximation (SARA-11), pp 34–41

Dechter R, Dechter A (1988) Belief maintenance in dynamic constraint networks. In: Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88), pp 37–42

Dechter R, Meiri I, Pearl J (1991) Temporal constraint networks. Artificial intelligence 49(1):61–95

Fargier H, Lang J (1993) Uncertainty in constraint satisfaction problems: a probabilistic approach. In: Proceedings of the Symbolic and Quantitative Approaches to Reasoning and Uncertainty (EC-SQARU-93), pp 97–104

Fargier H, Lang J, Schiex T (1996) Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In: Proceedings of the 13th National Conference on Artificial Intelligence, pp 175–180

Fowler DW, Brown KN (2003) Branching constraint satisfaction problems and markov decision problems compared. Annals of Operations Research 118(1-4):85–100

Hays W (1973) Statistics for the social sciences (2nd. edit.), vol 410. Holt, Rinehart and Winston New York

Hebrard E (2006) Robust Solutions for Constraint Satisfaction and Optimisation under Uncertainty. PhD thesis, University of New South Wales

Herrmann H, Schneider C, Moreira A, Andrade Jr J, Havlin S (2011) Onion-like network topology enhances robustness against malicious attacks. Journal of Statistical Mechanics: Theory and Experiment 2011(1):P01,027

Jen E (2003) Stable or robust? what's the difference? Journal of Complexity 8(3):12–18

Larrosa J, Schiex T (2004) Solving weighted CSP by maintaining arc consistency. Artificial Intelligence 159:1–26

Larrosa J, Meseguer P, Schiex T (1999) Maintaining reversible DAC for Max-CSP. Journal of Artificial Intelligence 107(1):149–163

Mackworth A (1977) On reading sketch maps. In: Proceedings of IJCAI'77, pp 598–606

Sam J (1995) Constraint consistency techniques for continuous domains. These de doctorat, École polytechnique fédérale de Lausanne

Schiex T, Fargier H, Verfaillie G (1995) Valued constraint satisfaction problems: hard and easy problems. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-95), pp 631–637

Taillard E (1993) Benchmarks for basic scheduling problems. European Journal of Operational Research 64(2):278–285

Verfaillie G, Jussien N (2005) Constraint solving in uncertain and dynamic environments: A survey. Constraints 10(3):253–281

Verfaillie G, Schiex T (1994) Solution reuse in dynamic constraint satisfaction problems. In: Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), pp 307–312

Wallace R, Freuder E (1998) Stable solutions for dynamic constraint satisfaction problems. In: Proceedings 4th International Conference on Principles and Practice of Constraint Programming (CP-98), pp 447–461

Wallace RJ, Grimes D (2010) Problem-structure vs. solution-based methods for solving dynamic constraint satisfaction problems. In: Proceedings of the 22nd International Conference on Tools with Artificial Intelligence (ICTAI-10), IEEE

Walsh T (2002) Stochastic constraint programming. In: Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02), pp 111–115

William F (2006) Topology and its applications. John Wiley & Sons

Winer B (1971) Statistical principles in experimental design (2nd. edit.). McGraw-Hill Book Company

Yorke-Smith N, Gervet C (2009) Certainty closure: Reliable constraint reasoning with incomplete or erroneous data. Journal of ACM Transactions on Computational Logic (TOCL) 10(1):3