Robustness, Stability, Recoverability and Reliability in Constraint Satisfaction Problems.

Federico Barber and Miguel A. Salido

Instituto de Automática e Informática Industrial
Universitat Politècnica de València
email {fbarber,msalido}@dsic.upv.es

Abstract. Many real-world problems in Artificial Intelligence (AI) as well as in other areas of computer science and engineering can be efficiently modeled and solved using constraint programming techniques. In many real-world scenarios the problem is partially known, imprecise, and dynamic such that some effects of actions are undesired and/or several un-foreseen incidences or changes can occur. Whereas expressivity, efficiency, and optimality have been the typical goals in the area, there are several issues regarding robustness that have a clear relevance in dynamic Constraint Satisfaction Problems (CSP). However, there is still no clear and common definition of robustness-related concepts in CSPs. In this paper, we propose two clearly differentiated definitions for *robustness* and *stability* in CSP solutions. We also introduce the concepts of *recoverability* and *reliability*, which arise in temporal CSPs. All these definitions are based on related well-known concepts, that are addressed in engineering and other related areas.

1 Introduction

Nowadays, many real problems can be modeled as Constraint Satisfaction Problems (CSP) that are solved using constraint programming techniques [3]. Much effort has been spent to increase the efficiency of constraint satisfaction algorithms: filtering, learning and distributed techniques, improved backtracking, use of efficient representations, heuristic search, etc. This effort has resulted in the design of constraint reasoning tools which have been used to solve numerous real problems. However, all these techniques assume that the set of variables and constraints, which compose the CSP, is completely known and fixed. This is a strong limitation when dealing with real situations where the CSP under consideration may evolve because of (i) changes in the environment or in its execution conditions, (ii) evolution of user requirements in the framework of an interactive design, and (iii) changes in other agents in the framework of a distributed system [23].

Since the nature of the real world is dynamic, techniques that attempt to model it should take this dynamicity into consideration [26]. A Dynamic Constraint Satisfaction Problem (DCSP) [8] is an extension to a static CSP that models addition and retraction of constraints and, hence, it is more appropriate for handling dynamic real-world problems. It is indeed easy to see that all possible changes to a CSP (constraint or domain modifications, addition or removal of variables) can be expressed in terms of addition or removal of constraints [23]. We remark that we only deal with aspects of pure satisfaction (i.e.: CSP). In the context of constraint optimization, it is well known that relaxations do not preserve optimality. This is an interesting, but much more complex issue.

Several proactive or reactive techniques have been developed to manage incidences in dynamic problems. Thus, computing a new solution from scratch after each problem change is possible (reactive technique), but it has two important drawbacks: inefficiency and instability of the successive solutions [23].

In [24], a proactive approach is presented to explore methods for finding solutions that are more likely to remain valid after changes that temporarily alter the set of valid assignments. In [5], a proactive approach uses the probability of change and the magnitude of change to model a CSP as a weighted CSP. In [6], a proactive approach assigns weights to each valid tuple based on its distance from the edge of the solution space to model the CSP as a weighted CSP. Other works are focused on searching for a new solution that minimizes the number of changes from the original solution. For instance, in [12], the concept of super-solution is introduced for constraint programming: 'A solution is a super solution if it is possible to repair the solution with only a few changes'. This is a generalization of both fault tolerance solutions in constraint programming [27] and supermodels in propositional satisfiability [17]. In [22], a method is proposed for reusing the previous solution to produce a new solution by means of local changes in the previous one.

By reading the research carried out in dynamic constraint satisfaction, we found that the terms robustness and stability are sometimes used interchangeably. Some authors refer to *robust solutions* with the same meaning that others use for *stable solutions*. For instance, one of the most recent papers regarding dynamic constraint satisfaction [25] states that the strategies that have been devised to handle CSPs are "methods for finding robust solutions that are either more likely to remain solutions after change or are guaranteed to produce a valid solution to the altered problem with a fixed number of assignment changes." In the Handbook of Constraint Programming [19] "There are three key concerns in solving dynamic CSPs. The first is to minimise the need for change, and thus to find robust solutions that are likely to remain solutions even after the change has occurred, or to need only minor 'repairs'"

In engineering, there is an agreement to distinguish between stable and robust concepts. Thus, the question that arise is: What is the difference between stable and robust CSP solutions?. Answering this question is not always easy since robustness has multiple, sometimes conflicting, interpretations [13]. In some areas, robustness has been assimilated to stability [29] and more appropriately, CSPs with temporal constraints has been related to noise tolerance [16], etc. Even in related areas such as Operation Research, the multiple meanings accorded to the term "robust" are open to debate [20]: *Robustness can be related to, or integrated into, the notions of flexibility, stability, sensitivity and even equity*. In constraint satisfaction, only a few works make a tiny distinction between robustness and stability [11]. However, we consider that robustness and stability terms should be clearly distinguished, since they represent different behavior of a CSP's solution after changes in the environment: Robust solutions refer to solutions that are either more likely to remain valid after change, whereas stable solutions are solutions that can adapt to a new valid solution with only few assignment changes to variables.

In this paper, we focus our attention on the 'robustness' and 'stability' concepts in CSPs. We propose general engineering-based and clearly different definitions for robust and stable CSP solutions. Moreover, we also introduce the concepts of 'recoverability' and 'reliability' which are relevant in real-world temporal-CSP domains. Clear and common definitions are needed to be able to evaluate different alternatives. Afterwards, new research lines will arise: How can we assess the robustness or stability of a solution? What does it guarantee? How can we get a more robust solution? What is the relationship between robustness and other problem parameters, such as optimality and constrainedness? Is it possible to obtain a model of robustness?

1.1 Definitions

Following some standard notations and definitions in the literature, we have summarized the basic definitions that will be used throughout this paper.

Definition 1. A Constraint Satisfaction Problem (CSP) is a triple $P =< X, D, C >$, where X is a finite set of variables $\{x_1, x_2, ..., x_n\}$, D is a set of domains $D = \{d_1, d_2, ..., d_n\}$ such that each variable $x_i \in X$ has a finite set of possible values d_i, and C is a finite set of constraints $C = \{C_1, C_2, ..., C_m\}$ that restrict the values that the variables can simultaneously take. C can be extensionally represented by a disjunctive set of feasible tuples or it can be intensionally represented by a conjunctive set of logical-mathematical formulae.

Definition 2. A dynamic constraint satisfaction problem (DCSP) is a a sequence of static CSPs, $(CSP_0, CSP_1, CSP_2, ..., CSP_i, ...)$, where each CSP_i is the result of changes in the preceding one [9]. In the original definition, changes could be due either to the addition or the removal of constraints.

Definition 3. The Solution Space is the portion of search space ($\prod_{i=1,n} d_i$) that satisfies all constraints. A solution S is a *feasible* instantiation of all variables, that is, it satisfies all constraints.

2 Incidences (or changes) in CSPs

Since many real problems are dynamic, unexpected incidences in the problem scenario occur due to its dynamism, spurious actions, lack of complete knowledge, etc. Let $Z = \{z_1, z_2, ..., z_i, ...\}$ the set of possible incidences that can occur in the future, which give rise to the set of possible changes in the CSP that models the problem. Let us also assume that each $z_i \in Z$ is independent and has a probability $p(z_i)$. This function $p(z_i)$ introduces a probability distribution P over Z (probability of change) such that $p(z_i)$ describes the relative likelihood for z_i to occur and $\sum_Z p(z_i) = 1$.

Each incidence $z_i \in Z$ can be modeled as a set of changes in variable domains or constraints. Since changes in domains can be represented as unary constraints, it can be assumed that each incidence z_i can be represented by a set of changes (restriction or relaxation) of constraints. In this paper, we are interested in robustness issues and how a CSP's solution maintains its feasibility after the set of possible incidences. Thus, we will only consider incidences that restrict the solution space (i.e.: restrict or add new constraints to the previous existing ones). The removal or relaxation of constraints is not considered here since it does not restrict the solution space. Therefore, each possible incidence $z_i \in Z$ is modeled as a new set of constrains Cz_i to be added to the previous set of constraints, making the problem more restricted, or even inconsistent.

In its dynamic evolution, a DCSP represents the successive CSPs that model the evolution of the problem, where each CSP_i represents the initial problem (modelled as CSP_0) after the successive occurrence of incidences $\{z_1, z_2, ..., z_i, ...\}$:

$$CSP_i =< X, D, C_{i-1} \cup Cz_i > \tag{1}$$

Note that CSP_{i+1} is equal to CSP_i if z_i does not occur.

We assume the incidences only restrict (but do not make empty) the initial solution space; otherwise the problem would become inconsistent. Therefore, some of the feasible solutions of the initial CSP_0 are also solutions of the final CSP.

Obviously, it is not possible to determine the robustness-related features of a system if no information about the incidences is given. In this case, we can obtain a rough estimation by means of the inclusion of random incidences (i.e.: random values for $p(z_i)$). However, it is important to remark that, in the same way that a CSP models the real-world problem, the set of incidences Z should also model the set of expected incidences that can occur in the real-world. Thus, Z should not be a set of randomly generated modifications of the constraints and domains of the CSP, but rather the result of modeling ($\{Cz_i\}$) the set of possible changes ($\{z_i\}$) that can occur in the real-world problem that it is modeled by the CSP.

For simplicity reasons, we will generalize our notation and denote Cz_i as z.

3 Robustness

Robustness is a common feature in our environment. Systems that belong to biological life, chemical compositions, physical structures, isolated objects, etc. [21] persist, remain running, and maintain their main features despite continuous perturbations, changes, incidences or aggressions. Thus, robustness is a concept related to the *persistence* of the system, its structure, its functionality, etc., against external interference: *"A system is robust, if it persists"*.

Thus, in a general way, "*robustness*" can be defined as the ability of a system to withstand stresses, pressures, perturbations, unpredictable changes, or variations in its operating environment without loss of functionality. A system that is designed to perform functionality in an expected environment is "*robust*" if it is able to maintain its functionality under a set of incidences. For example, *an algorithm is robust if it continues to operate despite unexpected inputs or erroneous calculations*.

Intuitively, the notion of robustness is easy to define, but its formalization depends on the system, on its expected functionality, and on the specific set of incidences to be confronted [18]. No general formal definition of robustness has been proposed, except a few exceptions or particular cases. Specifically, Kitano [14] mathematically defines the robustness (R) of a functional system (SYS) with regard to function (F) against a set of perturbations (Z) as (in a simplified way):

$$R^{SYS}_{F,Z} = \int_Z p(z) * F(SYS, z)dz \tag{2}$$

where, $p(z)$ is the probability for incidence $z \in Z$, and $F(SYS, z)$ is an evaluation function that returns zero when the system SYS fails under z or it returns a relative viability $]0, 1]$ otherwise. For instance, *if production drops 20% under a certain perturbation (z) compared with standard production, then 0.8 is returned.*

Expression (2) formalizes how a system (SYS) is able to maintain a certain level of its expected functionality (F) against a given set of perturbations (Z). According to (2), a system SYS_1 is more robust than SYS_2 with regard to an expected functionality F against a set of perturbations Z when:

$$R^{SYS_1}_{F,Z} > R^{SYS_2}_{F,Z} \tag{3}$$

The application of expression (2) is highly dependent on the system being assessed. Let us apply (2) to CSPs:

- S is a solution of the CSP, whose robustness we want to assess. Robustness is a concept related to CSP solutions, not to CSP itself. Thus, the system SYS in (2) can be related to the solution S in a CSP.
- Z is the discrete set of unexpected incidences (i.e.: changes in constraints).
- F is the expected functionality of the system. In CSP, the expected functionality of a solution is its feasibility.

Therefore, by applying (2), the robustness of a CSP solution (S) can be defined as follows:

Definition 3. A solution (S) of a CSP is $r{-}robust$ with respect to a set of incidences Z, each $z \in Z$ with a probability of occurrence p(z), when:

$$r(S, Z, P) = R^{S}_{F,Z,P} = \sum_Z p(z) * F(S, z) \tag{4}$$

where, in the case of a CSP, function $F(S, z)$ is the consistency of S after z:

- $F(S, z) = 1$ iff S also satisfies $C \cup z$.
- $F(S, z) = 0$, iff S does not satisfy $C \cup z$. More concretely, iff S does not satisfy z.

R-robustness of a solution represents the probability of remaining solution after Z and varies from 0 to 1 since $\sum_Z p(z) = 1$ and F(S,z) $\in \{0, 1\}$. The greater its r-robustness, the more robust a solution is and more likely will remain feasible after Z.

From expression (4), we can see that the algorithm for calculating the robustness $r - robust$ of a solution S against a set of incidences Z is trivial. It only requires to check whether S maintains its feasibility $F(S, z)$ for each incidence $z \in Z$. For each $z \in Z$, it costs O(n).

On the other hand, note that robustness does not require insensitiveness of the problem modeled by the CSP. For instance, the constraints of the problem could dramatically vary due to z, such that z could greatly reduce the solution space. However, a robust solution S with respect z would remain feasible after the incidence.

Also note that the robustness of a solution S does not depend on the behavior of S against an incidence z, but on how the feasibility of S is maintained over a set of unexpected incidences Z. Thus, the robustness of S depends on the probability $p(z)$ of each possible incidence $z \in Z$ and on how z affects to the feasibility of the solution $F(z)$. In other words, the only way to characterize the robustness level of a given CSP solution is to determine how its feasibility is maintained over several levels or probability of incidences.

Note that we do not take into account other aspects, that have usually been taken into account when the robustness of a CSP solution is assessed by other authors (e.g.: the number of variables that must change their values to make the initial solution feasible after the incidence, the number of unsatisfied constraints by the initial solution, etc.). In our approach, a solution is not more/less robust under a given incidence if the solution needs to be more/less repaired to deal with the incidence. We claim that robustness cannot be assessed on the basis that only small changes are necessary to obtain a new feasible solution. In problems related with satisfiability, robustness should be related to feasibility maintenance.

3.1 Example

Let us apply the above definition (4) to the following example. Let *P* be a CSP with two variables x_1 and x_2 with domains $D_1 : \{3..7\}$ and $D_2 : \{2..6\}$, respectively. The dynamic constraints are:

- $C_1 : x_1 + x_2 \leq 12$
- $C_2 : x_2 + x_1 \geq 6$
- $C_3 : x_2 - x_1 \leq 2$
- $C_4 : x_1 - x_2 \leq 4$

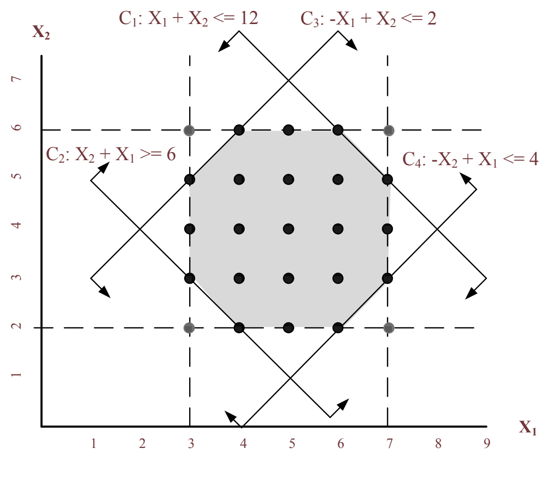

Fig. 1. CSP P and its solution space.

Figure 1 represents the solution space of the CSP, which is composed of 21 solutions. Let us suppose the following set Z of expected incidences (where $\sum_Z p(z_i) = 1$):

Incidence z_i	Likelihood $p(z_i)$	$z_i \rightarrow C_{z_i}$
z_1	0.15	$\{x_1 + x_2 <= 9, x_2 <= 5\}$
z_2	0.1	$\{x_1 + x_2 >= 10, x_1 >= 4\}$
z_3	0.25	$\{-x_1 + x_2 <= 0\}$
z_4	0.3	$\{x_1 - x_2 <= 2\}$
z_5	0.2	$\{x_1 > 4\}$

The robustness of each solution can be assessed according to expression 4. For instance, the solution $S = \{x_1 = 3, x_2 = 4\}$ is no longer valid when z_2, z_3, or z_5 occurs. Its robustness can be assessed as $R_Z^S = p(z_1) + p(z_4) = 0.15 + 0.3 = 0.45$.

Table 1. Robustness of each solution

Solution (x_1 x_2)	satisfies z_1?	satisfies z_2?	satisfies z_3?	satisfies z_4?	satisfies z_5?	Robustness
4,2	y	n	y	y	n	0,7
5,2	y	n	y	n	y	0,6
6,2	y	n	y	n	y	0,6
3,3	y	n	y	y	n	0,7
4,3	y	n	y	y	n	0,7
5,3	***y***	***n***	***y***	***y***	***y***	***0,9***
6,3	y	n	y	n	y	0,6
7,3	n	y	y	n	y	0,55
3,4	y	n	n	y	n	0,45
4,4	y	n	y	y	n	0,7
5,4	***y***	***n***	***y***	***y***	***y***	***0,9***
6,4	n	y	y	y	y	0,85
7,4	n	y	y	n	y	0,55
3,5	y	n	n	y	n	0,45
4,5	y	n	n	y	n	0,45
5,5	n	y	y	y	y	0,85
6,5	n	y	y	y	y	0,85
7,5	n	y	y	y	y	0,85
4,6	***n***	***y***	***n***	***y***	***n***	***0,4***
5,6	n	y	n	y	y	0,6
6,6	n	y	y	y	y	0,85

From Table 1, we can deduce that $\{x_1 = 5, x_2 = 3\}$ and $\{x_1 = 5, x_2 = 4\}$ are the most robust solutions, according to the above set of expected incidences. Likewise, $\{x_1 = 4, x_2 = 6\}$ is the least robust solution.

Even though the solution space of the above example is convex, note that it is not required for assessing the robustness of CSP solutions, nor is an implicit representation of the CSP necessary. Moreover, the robustness of each solution can be assessed independently of the assessment of other solutions.

3.2 What does r-Robustness guarantee?

The more robust solution is, the more likely it will remain valid after changes in the constraints. The following conclusions can be obtained from (4):

- A 1-robust solution is a solution that maintains its feasibility over the whole set of expected incidences.
- A 0-robust solution is a solution that becomes inconsistent with any expected incidence that may occur.
- An r-robust solution is a solution that maintains its feasibility over (100*r)% of probabilistically-pondered incidences. For instance, the solution $x_1 = 4, x_2 = 2$ of the above example (Table 1) is able to maintain robustness over 70% of the expected likelihood incidences. Specifically, this solution is robust against z_1, z_3 and z_4, which have an accumulated probability of 0.7.

4 Stability

Stability is an old concept that derives from astronomy and physics [28]. Loosely speaking, a solution (meaning an equilibrium state) of a dynamical system is said to be *stable* if small perturbations to the solution result in a new solution that stays "*close*" to the original solution. Perturbations can be viewed as small differences that occur in the actual state of the system [13]. Therefore, by applying this informal definition to CSPs, a solution is stable if small modifications of the constraint set allow a new solution (new consistent variable assignment) that remains close to the original solution:

$$\begin{gathered} Sol(X,D,C) \text{ is stable (with respect } z, C \cup z \cong C) \text{ iff} \\ \exists Sol(X,D,C \cup z) : Sol(X,D,C) \cong Sol(X,D,C \cup z) \end{gathered}$$

Definition 5. A solution S of a CSP is s-stable, with respect to an incidence z, if there exist a new feasible solution S in the s-neighborhood of S.

The neighborhood of solutions can be formally defined in terms of norms in the n-dimensional space [10]. Thus above definition can be detailed as follows.

Definition 6. A solution $S = (x_1 = v_1, x_2 = v_2, ..., x_n = v_n)$ is *s-stable* if, given an incidence z, there is a solution $S' = (x_1 = v'_1, x_2 = v'_2, ..., x_n = v'_n)$, such that: $\|S' - S\| < s$, where $\|.\|$ is some n-dimensional norm defined in the solution space to evaluate the difference between S and S'.

In relation to the implementation of n-dimensional norms, we have:

1. On metric domains, we can apply the Euclidean distance between S and S', with a optional weighted factor ρ_i for each variable x_i:

$$\sqrt{\sum_{i=1}^{n} \rho_i (x'_i - x_i)^2} \tag{5}$$

but normalizing with the maximum distance between any two tuples in the space of solutions. Note that the domains $\{d_i\}$ are finite in a CSP. Thus, the normalized relative distance in a metric domain between S and S' becomes:

$$\|S' - S\|_z = \frac{\sqrt{\sum_{i=1}^{n} \rho_i (x'_i - x_i)^2}}{\sqrt{\sum_{i=1}^{n} \rho_i \mid d_i \mid^2}} \tag{6}$$

Note that the similarity given in (6) between S and S' may be very low if the two solutions S and S' are very close in the n-dimensional space even though all the variables of S change their values. This n-dimensional norm measures the relative distance between S and S', such that $\|S' - S\|_z \in [0, 1]$ due to a change in the value of one or all variables.

2. On non-metric domains (like non-ordered sets of values), the Hamming distance (H) can be applied. This n-dimensional norm measures the number of variables that have different values in S and S'. Therefore, the distance between S and S' on non-metric domains can be defined as:

$$\|S' - S\|_z = \frac{\sum_{i=1}^{n} \rho_i H(x'_i, x_i)}{N} \tag{7}$$

where $H(x'_i, x_i)$ is equal to 0 iff $x'_i = x_i$, and 1 otherwise. The expression is normalized with respect to N, such that this criterion evaluates the relative number of variables that change their values and $\|S' - S\|_z \in [0, 1]$. Note that this concept is related to the super-solution concept given in [12].

These measures evaluate the closeness of solutions in the space of solutions. Therefore, given an incidence z, the s-stability for a solution S quantifies the s-proximity to S of the closest feasible solution S' in the n-dimensional space of the CSP. In other words, we should determine how much the new solution S' differs from the initial one S in order to address the incidence. A robust solution is a 0-stable solution.

The proposed measures of s-stability require finding a solution in the closest neighborhood of S, among the complete set of new feasible solutions, such that deviation with respect to the previous solutions S is minimized. Let us denote $N(S, z)$ as the value of $\|S' - S\|$ for the closest solution S' to S, after the occurrence of z:

$$N(S, z) = min_{S'} \|S' - S\| \tag{8}$$

Note that $N(S, z) = 0$ iff $F(S, z) = 1$ (i.e.: S satisfies z).

According to definition 6, we can define the s-stability (STA) of a solution (S) against a given set of perturbations (Z) as:

$$s(S, Z, P) = STA^S_{Z,P} = \sum_{Z} p(z) \cdot N(S, z) \tag{9}$$

where $p(z)$ is the probability of change. Thus STA^S_Z varies from 0 to n in the case of a non-metric CSP with n variables, and from 0 to 1 in the case of a CSP with metric domains. The lower is its s-stability, the more stable the solution is.

Computational cost for obtaining stability of a solution is NP-hard since the process for obtaining $N(S,z)$ for each $z \in Z$ derives in a Constraint Satisfaction and Optimization Problem (CSOP) whose constraints are $C \cup z$ and whose optimality criteria is to minimize $\|S' - S\|$ (Equation 8). However, this computational cost can be reduced by searching for a solution S' in the closest neighborhood of S. Note that the notion of distance between S' and S should take into account whether or not the domain is metric. Therefore, an incremental process can be defined (Algorithm 1 for metric-domains and Algorithm 2 for non-metric domains). In Algorithm 1, each iteration k in the process has a cost $(2 * k * \delta)^n$, such that the computational cost for obtaining stability of a solution can be decreased if a solution exits in the close neighborhood of S. In Algorithm 2, each iteration δ in the process has a cost $\binom{n}{\delta}(d_i)^\delta$. Thus, computational cost for obtaining stability of a solution can be decreased if a solution exists in the close neighborhood of S.

Algorithm 1 Incremental Stability Process for obtaining $N(S,z)$ in metric domains. Return $False$ if no solution.

```
Let CSP=<X, C, D>, let $S = (v_1, v_2, ..., v_n)$ be a solution $S$ of the CSP, and let $\delta$ be the granularity of the searching process.
k=1;
NSz=$False$;
repeat
  if ∃ S', a solution to CSP'=[X, C', D'], where
  $C' = C \cup z \wedge D' \subseteq D$: $d'_i = \{[v_i - k * \delta], [v_i + k * \delta]\} \cap d_i, \forall i \in 1..n$ then
    $NSz = \|S' - S\|$;
  else
    k=k+1;
  end if
until $D' \nsubseteq D$
return $NSz$;
```

Algorithm 2 Incremental Stability Process for obtaining $N(S,z)$ in non-metric domains. Return $False$ if no solution.

```
Let CSP=<X, C, D>, and let $S = (v_1, v_2, ..., v_n)$ be a solution $S$ of the CSP.
$\delta$=1;
NSz=$False$;
repeat
  if ∃ S', a solution to CSP'=[X, C', D'], where $C' = C \cup z \wedge D' \subseteq D$: $v'_i \in d_i$,
  $\sum_{i=1}^{n} H(v'_i, v_i) = \delta$ then
    $NSz = \delta$;
  else
    $\delta$ + 1;
  end if
until $\delta \neq n$
return $NSz$;
```

4.1 Example

Let's apply the above definition of stability (9) to the previous example (Figure 1) for the most and least robust solutions given in Table 1.

Table 2. Stability of some robust solutions {(5,3), (5,4)} and a non-robust solution (4,6).

Solution (x_1, x_2)	Closest sol. with z_1	Closest sol. with z_2	Closest sol. with z_3	Closest sol. with z_4	Closest sol. with z_5	Robustness	Stability
(5,3)	***satisfies***	***(6,4)***	***satisfies***	***satisfies***	***satisfies***	***0,9***	*0.14/*$\sqrt{50}$
(5,4)	***satisfies***	***(6,4)***	***satisfies***	***satisfies***	***satisfies***	***0,9***	*0.1/*$\sqrt{50}$
(4,6)	***(4,5)***	***satisfies***	***(5,5)***	***satisfies***	***(5,6)***	***0,4***	*0.7/*$\sqrt{50}$

For instance, the stability of solution $(4, 6)$, according to expression (9), is:

$$STA_Z^{(4,6)} = \frac{0.15 * 1 + 0.25 * \sqrt{2} + 0.2 * 1}{\sqrt{5^2 + 5^2}} = \frac{0.7}{\sqrt{50}} \tag{10}$$

Thus, following Table 2, $\{x_1 = 5, x_2 = 4\}$ is the most robust (0.9) and the most stable solution (0.1/$\sqrt{50}$) according to the given set of expected incidences.

4.2 What does s-stability guarantee?

Stability of a solution S represents the expected minimum normalized distance between the current solution (S) and a new feasible solution after Z. The more stable a solution is, the less needs to change in order to get a new solution after changes in the constraints. The following conclusions can be obtained from (Expression 9):

– A solution S, with $STA_Z^S = 0$, is a 1-robust solution. It is fully stable over the whole set of expected incidences.
– A solution S of a non-metric CSP, with $STA_Z^S = n$, requires changing the assignments of the whole set of variables to become consistent against any expected incidence that may occur. A solution S of a metric CSP, with $STA_Z^S = 1$, requires moving to the far extreme point of the solution space to become consistent against any expected incidence that may occur.
– A solution S of a non-metric CSP, with $STA_Z^S = k$, requires changing k variables, as average, to become consistent over the whole set of probabilistically-pondered incidences. A solution S of a metric CSP, with $STA_Z^S = k$, requires moving to a distance $(k * \sqrt{\sum_{i=1}^{n} d_i^2})$, as average, to become consistent over the whole set of probabilistically-pondered incidences.

5 Temporal Constraint Satisfaction Problems

A Temporal Constraint Satisfaction Problem (TCSP) is a subtype of CSPs, where variables represent temporal primitives (time points, temporal intervals, or temporal durations), such that solutions have a temporal interpretation [7], [2]. This is the typical case of scheduling problems, where variables can be instantiated on the time line (see Figure 2) so that they can be associated to starting or ending times of tasks (see Figure 3).

Besides $robustness$ and $stability$ concepts, in TCSP, not only is it important to know how different the new feasible solution S' is from the original one S, given an incidence z (i.e.: stability), but it is also important to know *(i) how long* the new solution S' differs from the initial solution S (recoverability), and *(ii) how long* the actual solution S can be maintained after the incidence (reliability). Therefore, two new properties appear in relation to the *temporal stability* or *temporal robustness*: *recoverability* and *reliability*.

An example of TCSP: A Scheduling Problem

Figure 3a shows a TCSP that represents a flow-shop scheduling problem with two jobs J_1, J_2, each of which has three activities $(x_{1i}, x_{2j}, i, j = 1..3)$ and one resource that should be shared by all activities. Each row corresponds to a job, and an activity (x_{ij}) is represented as a rectangle whose length corresponds to its duration. This problem can be modeled as a TCSP, where variables represent time points (starting or ending times) of different activities ($x_{ij.on}$, $x_{ij.off}$). There exist constraints that refer to non-overlap and precedence constraints among activities. Moreover, it is known that x_{23} should be performed at least k-units after x_{22} (Constraint C_{23-22}). The first solution (Figure 3a) minimizes the makespan and is considered to be the optimal solution. The projection of variables x_{ij} on time represents the optimal assignment of variables of the TCSP.

5.1 Recoverability

Recoverability refers to the ability to restore a system to the point at which a failure occurred. Despite proactive approaches, it is clear that robustness is not always completely guaranteed. Therefore, *recovery strategies* should be used once disturbing events occur in order to keep the feasibility of the pre-computed solution. Robustness and recoverability are closely related and, in some optimization frameworks, they have been unified into an integrated notion of *recoverable robustness* [15]. For TCSP, where solutions project over time, the recoverability of a solution can be measured by the required amount of time (δt) (after an incidence occurs) to restore part of the initial solution (Figure 2). Therefore, taking into account that temporal variables, in a solution of a TCSP, are distributed over time, we can define that a *δt-recovered* solution maintains the same initial values in the (temporal) variables that are related to times starting from δt after the incidence:

$$Sol_{\delta t}(X, D, C \cup z) \equiv Sol_{\delta t}(X, D, C)$$

where $Sol_{\delta t}$ covers the set of variables from δt after incidence (Figure 2). The objective of a recovery process is to minimize δt. Likewise, since the variables of TCSP are temporally ordered (i.e., they are instantiated over time), the objective of a recovery process is to minimize the set of variables (from the time t when the incidence occurs until $t + \delta t$) that require changing their values in order to obtain a new feasible solution.

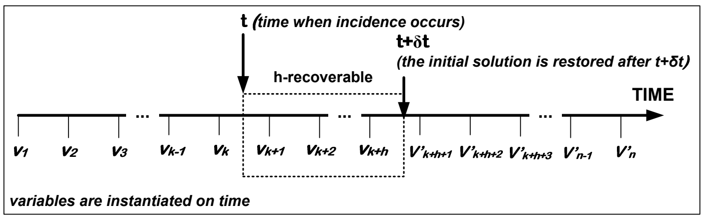
t (time when incidence occurs)
t+δt
(the initial solution is restored after t+δt)
h-recoverable
TIME
V1 V2 V3 Vk-1 Vk Vk+1 Vk+2 Vk+h V'k+h+1 V'k+h+2 V'k+h+3 V'n-1 V'n
variables are instantiated on time

Fig. 2. Recoverability of a solution in a TCSP.

Definition 7. A solution S is *h-recoverable* iff, at most, h variables (consecutive variables after the incidence occurs) require changing their values in order to obtain a new feasible solution S'.

Thus, a solution $S = (x_1 = v_1, x_2 = v_2, ..., x_n = v_n)$, whose variables are temporally ordered, is h-recoverable iff, given an incidence z that occurs in t, the incidence affects S in the temporal interval $[t, t + \delta t]$, such that the variables $v_1, ..., v_t$ and $v_{t+h+1}, ..., v_n$ can maintain their initial values, while the variables in the interval $[t, t + \delta t]$ $(v'_{t+1}, ..., v'_{t+h})$ must change their values (see Figure 2). The initial solution is recovered after x_{t+h}.

Note that the definition of h-recoverability is similar to the definition of $(h, 0)-$*super solutions* where if h variables lose their values, we can find another solution by reassigning these variables a new value. The only difference is that, in $h-$*recoverability*, the variables to be repaired are consecutive in time, while, in (h,0)-super solutions, the variables to be repaired are not consecutive.

From (Definition 7), it is immediate to conclude that $0 \leq h$-recoverability $\leq n$. The lower is h-recoverability, the more recoverable a solution is. A 0-recoverable solution S does not require changing any variables after incidence in order to maintain the feasibility of S (equivalent to 1-robust solution) An n-recoverable solution S does require changing all the variables after incidence. Thus, h-recoverability can be considered to be a temporal s-stability.

5.2 Reliability

In engineering, reliability is associated to the confidence that a system will perform its intended function during a specified period of time under stated conditions, as well as under unexpected circumstances. Mathematically it can be expressed as:

$$R(t) = \int_t^\infty f(x)dx \tag{11}$$

where $f(x)$ is the failure probability density function and t is the length of the period of time (which is assumed to start from time zero). There is always some chance for failure, but $R(t)$ means that the system has a specified probability that it will operate without failure before time t.

In TCSP, variables of solution are distributed on time. Thus, a solution found initially may be invalid for variables that are related to a time greater than δt after incidence. Thus, by applying the above concepts, we can assess that a TCSP solution is

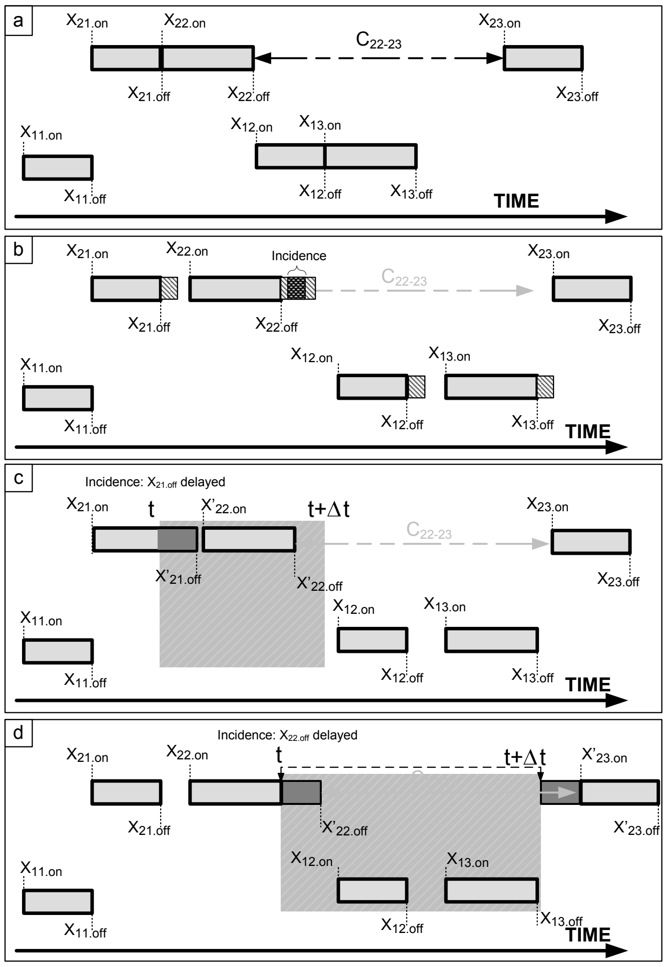

Fig. 3. A scheduling problem: four solutions.

δt-reliable, if given an incidence at time t, the solution remains valid until $t+\delta t$ (Figure 2. Thus, the set of variables that represent the solution of the problem from time t until $t+\delta t$ are not required to change their values:

$$Sol_{t \to \delta t}(X, D, C \cup z) \equiv Sol_{t \to \delta t}(X, D, C)$$

where $Sol_{\delta t}$ covers the set of CSP variables from time t until $t+\delta t$. The way to obtain a a reliable solution is maximize t, or alternatively, maximize the set of variables (from time t, when incidence occurs, until $t+\delta t$) that can maintain their values (Figure 2.

Similarly to recoverability, the reliability of a solution S can be defined in terms of the number of assignments in S that remain valid after the incidence occurs (i.e., they can take part of a solution of $TCSP_Z$).

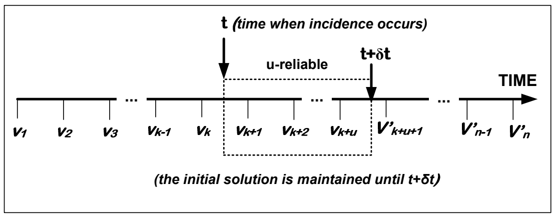
t (time when incidence occurs)
u-reliable
t+δt
TIME
v1
v2
v3
vk-1
vk
vk+1
vk+2
vk+u
V'k+u+1
V'n-1
V'n
(the initial solution is maintained until t+δt)

Fig. 4. Reliability of a solution in a TCSP.

Definition 8. A solution S is *u-reliable* if, at least, u assignments of variables in S (consecutive variables after the incidence occurs) can take part of a solution of the $TCSP_Z$. Thus, a solution $S = (x_1 = v_1, x_2 = v_2, ..., x_n = v_n)$ is u-reliable, iff given an incidence z that affects the problem from x_t, the assignments of variables $(x_1 = v_1, x_2 = v_2, ..., x_t = v_t, x_{t+1} = v_{t+1}, ..., x_{t+u} = v_{t+u})$, $1 \leq t+u \leq n$, can take part of a solution of $TCSP_Z$. The initial solution is maintained until x_{t+u}.

As in the above case, from (Definition 8) we can conclude that 0 $\leq$ u-reliability $\leq n$. The greater is u-reliability, the more reliable a solution is. A 0-reliable solution S cannot maintain values in any variable after incidence for maintaining feasibility of S. A n-reliable solution S can maintain values in all variables. Therefore, we can consider that u-reliability is a concept that is related to the temporary maintenance of robustness from time of occurrence. Moreover, a n-reliable solution is a 0-recoverable solution, which can also be considered to be a temporally 0-stable or 1-robust solution.

Note that *h-recoverability* and *u-reliability* of a solution S are not contradictory nor complementary concepts. If an incidence occurs at time t, (i) the initial solution S can be maintained feasible from t until $t + \delta_u$(u variables), and (ii) the initial solution S can be restored from $t+\delta_h$ (h variables), where $\delta_u \leq \delta_h$). Of course, $(h-recoverability + u-reliability) \leq n$.

Recoverability and Reliability in the example

With respect to the scheduling problem of Figure 3, Figure 3b shows a robust solution. To this end, some buffer times have been included between some activities in order to absorb incidences. For instance, if a resource is broken for a short time, (Incidence in Figure 3b), the solution is not affected by the incidence. Thus, all assignments to variables remain valid. Furthermore this solution is also stable. If variables $x_{21.off}, x_{22.off}, x_{12.off}$ or $x_{13.off}$ are minimally delayed, the rest of the variables maintain the same values. Moreover, the typical trade-off between robustness and optimality can be observed in Figure 3a/b.

Figure 3c shows a 3-recoverable solution for an incidence z: "$x_{21.off}$ is delayed to $x'_{21.off}$ in time t". In this case, only 3 variables must change their values ($x_{21.off}$, $x_{22.on}$, $x_{22.off}$), while assigned variables with assigned values greater than $t + \delta t$, ($x_{12.on}$, $x_{12.off}$, $x_{13.on}$, $x_{13.off}$, $x_{23.on}$, $x_{23.off}$), do not require change their values. On the other hand, Figure 3d shows a 4-reliability solution for an incidence z: $x_{22.off}$ is delayed to $x'_{22.off}$ in time t. In this case, the next 4 variables ($x_{12.on}$, $x_{12.off}$, $x_{13.on}$, $x_{13.off}$) do not change their values. The solution is maintained until $t + \delta t$. However, activity x_{23} must satisfy C_{23-22}, such that $x_{23.on}$ and further variables must change their values.

6 Generalizing concepts

In the previous sections, the concepts of robustness, stability, recoverability, and reliability have been defined by analyzing how a solution S absorbs or can be adapted to cope with an incidence z. These concepts can be generalized, such that we can assess the achievable levels of robustness, stability, recoverability, and reliability of solutions of a CSP for a given typology of incidences $z, p(z)$, a desired level of optimality of solution, and a given constrainedness of the CSP which is inherent to the problem. Here:

– Robustness guarantees that perturbations can be absorbed by the solution. Thus, robustness decreases as the level of incidences increases.
– Stability guarantee that the consequences of perturbations can be minimized by a new solution. Thus, stability decreases as the level of the incidences increases.
– A low-restricted CSP with a large solution space will usually allow more robust and stable solutions.
– A more optimized solution will usually be more sensitive to changes in the environment. Optimal solutions usually are in edges of solution's space where stability is lowest. There exists a clear trade-off between robustness and optimality/quality [4].

These ideas introduce the main concepts to which robustness, stability, recoverability, and reliability of solutions in CSP are related. These mutual relations are represented in Figure 5 and appear in many CSP's applications [1]. These relate the robustness, stability, recoverability, and reliability of solutions of CSPs with: *(i)* the constrainedness of CSPs (which is a problem-dependent feature); *(ii)* the incidences level (which is a feature of the problem and/or application scenario and is represented by the set Z; and *(iii)* optimality of S (which is a feature of each solution).

The evaluation of the robustness, stability, recoverability, and reliability of a solution S of a CSP can be viewed as a guarantee of the behavior of S with respect to Z. In this paper, we have proposed a definition and formalization of these concepts, on the basis of their meaning in other areas of science, as well as on how these concepts can be evaluated, and what it guarantees. A more detailed model of robustness would allow us to parameterize the implicit relations that exist in Figure 5, by relating the concepts of robustness with the characteristics of the problems or their application scenarios. From these definitions, efficient methods for obtaining more robust, stable, recoverable, and reliable solutions should be achieved.

7 Conclusions

While expressivity, efficiency, and optimality have been the typical goals in the development of CSP techniques, there are robustness-related issues that have received less

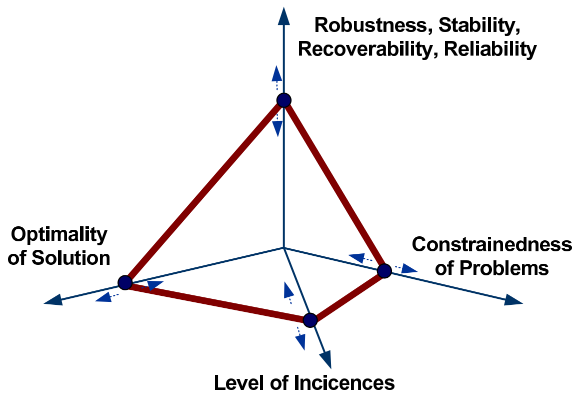
Robustness, Stability, Recoverability, Reliability
Optimality of Solution
Constrainedness of Problems
Level of Incicences

Fig. 5. Robustness and problem-related concepts.

attention. However, robustness requirements have a clear relevance in dynamic environments (usually with incomplete or imprecise knowledge). This work aims to review the concepts of robustness, stability, recoverability, and reliability in dynamic Constraint Satisfaction Problems. This suppose an advance in the state of the art in constraint programming, and new models and techniques can be developed to achieve this properties in CSP solutions.

The general notion of robustness includes several different concepts. Despite the existence of several works on dynamic CSP, there is still no clear and common definition of robustness-related concepts. In this paper, these concepts have been characterized and formalized, such that they can be used, in a general way, to assess robustness-related features of solutions in CSPs. These concepts have been applied to sample problems, which allows us to contrast the differences between them, as well as, the different ways that a solution can react to incidences: it can be *maintained*, it can be *adapted*, it can be *maintained during* (or *restored after*) a given time. This different behavior becomes relevant when a CSP is applied to solve real-world problems in a dynamic and partially unknown world.

On the basis of (Figure 5), we see that typology of expected incidences and their stochastic features, optimality of solutions, and constrainedness of problems are the main factors that limit the desired level of robustness, stability, recoverability, and reliability of solutions in CSPs. From this point, other relevant issues remain open: How can robustness-related concepts be efficiently measured?, How can robust, stable, recoverable or reliable solutions be obtained?, Which factors these concepts depend on?. The design of efficient algorithms for the evaluation of these concepts, and their extension to constrained optimization are the subject of relevant research lines.

Acknowledgments

This work has been partially supported by the research project TIN2010-20976-C02-01 (MEC, Spain-FEDER).

References

1. M. Abril, F. Barber, L. Ingolotti, M. A. Salido, P. Tormos, and A. Lova. An assessment of railway capacity. *Transportation Research Part E*, 44(5):774–806, 2008.
2. F. Barber. Reasoning on intervals and point-based disjunctive metric constraints in temporal contexts. *Journal of Artificial Intelligence Research*, 12:35–86, 2000.
3. R. Bartak and M. A. Salido. Constraint satisfaction for planning and scheduling problems. *Constraints*, 16(3):223–227, 2011.
4. D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, 2004.
5. L. Climent, M. A. Salido, and F. Barber. Robustness in dynamic constraint satisfaction problems. *Int. Journal of Innovative Computing Information and Control*, 8(4):2513–2532, 2012.
6. L. Climent, R. Wallace, M. Salido, and F. Barber. Modeling robustness in csps as weighted csps. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems CPAIOR 2013*, pages 44–60, 2013.
7. R. Dechter. Temporal constraint network. *Artificial Intelligence*, 49:61–295, 1991.
8. R. Dechter and A. Dechter. Dynamic constraint networks. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 37–42, 1988.
9. S. Gonzalez and P. Meseguer. Open, interactive and dynamic csp. In G. V. Ken Brown, editor, *Changes'05: International Workshop on Constraint Solving under Change and Uncertainty*, pages 13–17, 2005.
10. M. Hazewinkel. Encyclopaedia of mathematics. Springer, 2002.
11. E. Hebrard. Robust solutions for constraint satisfaction and optimisation under uncertainty. phd thesis. University of New South Wales, 2007.
12. E. Hebrard, B. Hnich, and T. Walsh. Super solutions in constraint programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-04)*, pages 157–172, 2004.
13. E. Jen. Stable or robust? what's the difference? *Complexity*, 8(3):12–18, 2003.
14. H. Kitano. Towards a theory of biological robustness. *Molecular Systems Biology*, 3(137), 2007.
15. C. Liebchen, M. Lbbecke, R. Mhring, and S. Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. *LNCS 5868*, 2009.
16. P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos. Mining frequent arrangements of temporal intervals. *Knowledge and Information Systems*, 21:133–171, 2009.
17. A. Parkes, M. Ginsberg, and A. Roy. Supermodels and robustness. *Proceedings The Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, (334-339), 1998.
18. A. Rizk, G. Batt, F. Fages, and S. Solima. A general computational method for robustness analysis withapplications to synthetic gene networks. *Bioinformatics*, 25(12):168–179, 2009.
19. F. Rossi, P. van Beek, and T. Walsh. Handbook of constraint programming. *Elsevier*, 2006.
20. B. Roy. Robustness in operational research and decision aiding: A multi-faceted issue. *European Journal of Operational Research*, 200:629–638, 2010.
21. E. Szathmary. A robust approach. *Nature*, 439:19–20, 2006.
22. G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: a survey. *Constraints*, 10(3):253–281, 2005.
23. G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proc. of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 307– 312, 1994.
24. R. Wallace and E. Freuder. Stable solutions for dynamic constraint satisfaction problems. In *Proc. 4th Int. Conf. on Principles and Practice of Constraint Programming (CP-98)*, pages 447–461, 1998.

25. R. Wallace, D. Grimes, and E. Freuder. Solving dynamic constraint satisfaction problems by identifying stable features. In *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI-09)*, pages 621–627, 2009.
26. D. Wang, Q. Tse, and Y. Zhou. A decentralized search engine for dynamic web communities. *Knowledge and Information Systems*, 26(1):105–125, 2011.
27. R. Weigel and C. Bliek. On re-formulation of constraint satisfaction problems. In *Proceedings European Conference on Artificial Intelligence (ECAI-98)*, pages 254–258, 1998.
28. S. Wiggins. Introduction to applied nonlinear dynamical systems and chaos. Springer, 1990.
29. Y. Zhou and W. Croft. Measuring ranked list robustness for query performance prediction. *Knowledge and Information Systems*, 16:155–171, 2008.