ROBUSTNESS AND STABILITY IN DYNAMIC CONSTRAINT SATISFACTION PROBLEMS

Laura Isabel Climent Aunés

A dissertation submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in the Departament de Sistemes Informàtics i Computació at Universitat Politècnica de València

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Advisors: Dr. Miguel A. Salido and Prof. Federico Barber

December, 2013

Als meus pares.
A la meua germana.

“Unexpected changes in life are what increase its complexity, but also its beauty.”

“Cambios inesperados en la vida son los que incrementan su complejidad, pero también su belleza.”

“Canvis inesperats en la vida són els que incrementen la seua complexitat, però també la seua bellesa.”

Laura Isabel Climent Aunés

Acknowledgements

Before starting the acknowledgments of this thesis, I would like to mention that they can be found in two different places. The first place is the city were I was born: Valencia, Spain. The second is Cork, an Irish city where I did an internship and also the place where I am currently working.

First of all, I would like to thank my advisors: Miguel Ángel Salido and Federico Barber. I would like to thank Miguel for his constant dedication and his advice. I would also like to thank Federico for introducing me to his research group and to this project. Furthermore, I would like to thank my "international" advisor, Richard Wallace for teaching me how to improve the formalization of many ideas presented in this thesis and for his crucial cooperation in this work.

My second (but not less important) words of acknowledgments are directed to my colleagues. However, I must say that the word "colleagues" is not enough to describe what they have been to me. After all of these years in which I have been working on this thesis, my colleagues have become my friends, and even my second family. I have plenty of wonderful memories with them. Thank you all for these moments and for creating such a nice working environment in both places, DSIC in Valencia, and INSIGHT center in Cork. Moreover, your advice and help has been extremely valuable to me. I would like to thank in particular the staff members of Lab 207 in the DSIC: Laura, Marlene, Mario, César, Joan, Flabio, Valentina, etc., and also the colleagues of our neighbour laboratories. In addition, I would like to thank to the staff members of INSIGHT center. Specially to Milan, Yuri, Diarmuid, Ena, Luis and Deepak.

In regard to the writing of this thesis, I have received a lot of help with the English language and also with its content. The following persons have contributed in such delicate task: Milan, Yuri, Luis, Pierre and Teresa. Thanks a million, I truly appreciate all the effort that you have put into it.

Finally, the last words of acknowledgments are dedicated to the many people outside of the research computer science area, from whom I have received a lot of support: my family and my friends. The unconditional and constant dedication of my parents and my sister has definitely contribute to the completion of this work. Thank you for being the great family that anyone would wish to have. I would also like to thank all the other members of my family: my grandparents, my uncles and aunts, my cousins, etc. In conclusion, I would like to thank, also, all of my friends: the ones from Benicalap, from university, from Erasmus, from high school, from school, etc. Thanks for being always loyal to me and for staying with me in good times as well as in bad times.

Abstract

Constraint programming is a paradigm wherein relations between variables are stated in the form of constraints. It is well-known that many real life problems can be modeled as Constraint Satisfaction Problems (CSPs). Due to their pervasive use in solving these problems, 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 completely known and fixed when the problem is modeled. This is a strong limitation because many problems come from uncertain and dynamic environments, where the original problem, and consequently its associated CSP model, may evolve due to the environment, the user or other agents. In such situations, a solution that holds for the original problem can become invalid after changes occur in the problem.

There exist two main approaches for dealing with these situations: reactive and proactive. Using reactive approaches entails re-solving the CSP after a solution is no longer a solution, which is time consuming. That is a clear disadvantage, especially when we deal with short-term changes. This is also motivated in Verfaillie and Jussien (2005), which is an important survey on constraint solving in uncertain and dynamic environments. In this survey, the authors state that a desirable objective in dynamic and uncertain frameworks is:

First *"Limit as much as possible the need for successive online problem solvings."*

Motivated by the **first** statement, in this dissertation we develop proactive approaches, which try to offer a resistance to the possible future alterations of the problem. Therefore, these approaches are applied before changes in the original problem occur. There exist two main types of proactive approaches, which can be distinguished on the basis of the characteristics of the solutions that they obtain: *robust* and *flexible*. The flexibility concept implies modifications over the original solution whilst the robustness concept does not. In Verfaillie and Jussien (2005), the authors make the **second** statement as another desirable objective.

Second *"Limit as much as possible changes in the produced solution."*

For this reason, this thesis mainly focuses on the search of robust solutions, which have a high probability of remaining solutions after changes over the CSP. Furthermore, in Verfaillie and Jussien (2005) the authors also mention as future interesting work the possibility of developing proactive strategies that combine the solution features of robustness and flexibility.

Third *"The production of solutions that are at the same time robust and flexible, that have every chance to resist changes and can be easily adapted when they did not resist, is obviously a desirable objective."*

According to this objective, expressed in the **third** statement, in this thesis we develop approaches that meet the condition of combining solution robustness and *stability* (the stability is a special case of flexibility). To the best of our knowledge, this combination has not yet been developed for CSPs.

Proactive approaches use the available knowledge about possible future changes in order to avoid or minimize their effects. However, for many real life problems the information about the uncertain and dynamic environment is unknown or hard to obtain. In many proactive approaches that search for robust solutions, the form of the algorithm is dependent on detailed knowledge about the dynamic environment. As a result, in the lack of such a priori information, these methods can not be applied. For this reason, the author of Hebrard (2006) makes the **fourth** statement for a desirable characteristic of approaches dealing with such an uncertain and dynamic framework.

Fourth *"Ideally, no additional knowledge over the data used to build the classical constraint network is required and no more expertise than for solving the problem without taking uncertainty into account."*

In this dissertation, we try to answer an interesting question in dynamic and uncertain environments: when additional information about the possible changes in the problem is unknown, is it possible to define what is the robustness of a solution of a CSP and to build appropriate algorithms? We found that it is possible and justifiable to extract some limited (and intuitively reasonable) assumptions about the dynamism of problems for which the order over the domain elements is significant. Therefore, in this work we present approaches for dealing with types of problems that are, therefore, consistent with the **fourth** statement. The fulfillment of these four statements has been the motivation and the main objective of the work presented in this thesis.

Resumen

La programación con restricciones es un paradigma en el que las relaciones entre las variables se expresan en forma de restricciones. Es bien sabido que muchos de los problemas de la vida real se pueden modelar como Problemas de Satisfacción de Restricciones (CSPs). Debido a su amplia utilización en la resolución de estos problemas, se ha invertido mucho esfuerzo en incrementar la eficiencia de los algorítmos que resuelven CSPs. Sin embargo, muchas de estas técnicas asumen que el conjunto de variables, dominios y las restricciones involucrados en el CSP son completamente conocidos y fijos cuando el problema se modela. Esta es una fuerte limitación porque muchos problemas provienen de entornos inciertos y dinámicos, en los que el problema original, y en consecuencia su modelo CSP, pueden evolucionar debido al entorno, al usuario u otros agentes. En tales situaciones, una solución del problema original puede convertirse en inválida después de que se produzcan cambios en el problema.

Existen dos estrategias principales para hacer frente a estas situaciones: reactivas y proactivas. Utilizar estrategias reactivas implica volver a resolver el CSP después de que una solución deje de ser una solución, lo cual conlleva un consumo de tiempo. Esta es una desventaja obvia, especialmente cuando tratamos con cambios a corto plazo. Esto también está motivado en Verfaillie and Jussien (2005), que es un importante estudio de la resolución de problemas con restricciones en entornos dinámicos e inciertos. En este trabajo, los autores afirman que un objetivo deseable en los marcos dinámicos e inciertos es:

Primera *"Limitar en la medida de lo posible la necesidad de sucesivas resoluciones online."*

Debido a esta **primera** declaración, en esta tesis se desarrollan estrategias proactivas, que tratan de ofrecer una resistencia a las posibles modificaciones futuras del problema. Por lo tanto, estos métodos se aplican antes de que ocurran cambios en el problema original. Existen dos tipos principales de estrategias proactivas, las cuales pueden distinguirse en base a las características de las soluciones que se obtienen: *robustez* y *flexibilidad*. El concepto de flexibilidad implica modificaciones en la solución original, mientras que el concepto de robustez no. En Verfaillie and Jussien (2005), los autores hacen la **segunda** declaración como otro objetivo deseable.

Segunda *"Limitar en la medida de lo posible, los cambios en la solución producida."*

Por esta razón, esta tesis se centra principalmente en la búsqueda de soluciones robustas, las cuales tienen una alta probabilidad de continuar siendo soluciones después de cambios en el CSP. Además, en Verfaillie and Jussien (2005) los autores también mencionan como un interesante trabajo futuro la posibilidad de desarrollar

estrategias proactivas que combinen las características de una solución de robustez y flexibilidad.

Tercera *"La producción de soluciones que sean a la vez robustas y flexibles, que tengan todas las posibilidades para resistir los cambios y puedan ser adaptadas fácilmente cuando no los resisten, es obviamente un objetivo deseable."*

De acuerdo con este objetivo, expresado en la **tercera** declaración, en esta tesis se desarrollan estrategias que cumplen la condición de combinar la robustez y *estabilidad* (la estabilidad es un caso especial de flexibilidad). Por lo que nosotros conocemos, esta combinación todavía no se ha desarrollado para CSPs.

Las estrategias proactivas utilizan el conocimiento disponible acerca de los posibles cambios futuros con el fin de evitar o reducir al mínimo sus efectos. Sin embargo, para muchos de los problemas de la vida real la información sobre el entorno incierto y dinámico es desconocida o difícil de obtener. Para muchas estrategias proactivas que buscan soluciones robustas, la forma del algoritmo depende de un conocimiento detallado sobre el entorno dinámico. Como resultado, estos métodos no se pueden aplicar si la información requerida es desconocida. Por esta razón, el autor de Hebrard (2006) hace la **cuarta** declaración como una característica deseable de las estrategias que se enfrentan a dicho incierto y dinámico marco.

Cuarta *"Idealmente, no se requiere ningún conocimiento adicional sobre los datos utilizados para construir la clásica red de restricciones ni más experiencia para resolver el problema sin tener en cuenta la incertidumbre."*

En esta tesis, tratamos de responder a una pregunta interesante en entornos dinámicos e inciertos: cuando no se conoce información adicional sobre los posibles cambios en el problema, ¿es posible definir lo que es la robustez de una solución de un CSP y desarrollar algoritmos apropiados?. Nosotros creemos que es posible y justificable extraer algunas suposiciones limitadas (e intuitivamente razonables) sobre el dinamismo de los problemas para los que el orden sobre los elementos del dominio es importante. Por lo tanto, en este trabajo presentamos estrategias para hacer frente a tipos de problemas que son, por lo tanto, consistentes con la **cuarta** declaración. El cumplimiento de estas cuatro declaraciones ha sido la motivación y el objetivo principal del trabajo presentado en esta tesis.

Resum

La programació amb restriccions és un paradigma en què les relacions entre les variables s'expressen en forma de restriccions. És ben sabut que molts dels problemes de la vida real es poden modelar com a Problemes de Satisfacció de Restriccions (CSPs). Degut la seua àmplia utilització en la resolució d'aquestos problemes, s'ha invertit molt esforç en incrementar l'eficiència dels algoritmes que resolen CSPs. No obstant això, moltes d'aquestes tècniques assumixen que el conjunt de variables, dominis i les restriccions involucrats en el CSP són completament coneguts i fixos quan el problema es modela. Esta és una forta limitació perquè molts problemes provenen d'entorns incerts i dinàmics, en els que el problema original, i en conseqüència el seu model CSP poden evolucionar a causa de l'entorn, a l'usuari o altres agents. En tals situacions, una solució del problema original pot convertir-se en invàlida després que es produïsquen canvis en el problema.

Hi ha dos estratègies principals per a fer front a estes situacions: reactives i proactives. Utilitzar estratègies reactives implica tornar a resoldre el CSP després que una solució deixe de ser una solució, la qual cosa comporta un consum de temps. Este és un desavantatge obvi, especialment quan tractem amb canvis a curt termini. Açò també està motivat en Verfaillie and Jussien (2005), que és un important estudi de la resolució de problemes amb restriccions en entorns dinàmics i incerts. En este treball, els autors afirmen que un objectiu desitjable en els marcs dinàmics i incerts és:

Primera *"Limitar en la mesura que siga possible la necessitat de successives resolucions online."*

Motivat per la **primera** declaració, en esta tesi es desenvolupen estratègies proactives, que tracten d'oferir una resistència a les possibles modificacions futures del problema. Per tant, estos mètodes s'apliquen abans de que ocórreguen canvis en el problema original. Hi ha dos tipus principals d'estratègies proactives, les quals poden distingir-se basant-se en les característiques de les solucions que s'obtenen: *robustesa* i *flexibilitat*. El concepte de flexibilitat implica modificacions en la solució original, mentres que el concepte de robustesa no. En Verfaillie and Jussien (2005), els autors fan la **segona** declaració com un altre objectiu desitjable.

Segona *"Limitar en la mesura que siga possible, els canvis en la solució produïda."*

Per aquesta raó, esta tesi es centra principalment en la busca de solucions robustes, les quals tenen una alta probabilitat de continuar sent solucions després de canvis en el CSP. A més a més, en Verfaillie and Jussien (2005) els autors també mencionen com un interessant treball futur la possibilitat de desenvolupar estratègies proactives que combinen les característiques d'una solució de robustesa i flexibilitat.

Tercera *"La producció de solucions que siguen al mateix temps robustes i flexibles, que tinguen totes les possibilitats de resistir els canvis i puguen ser adaptades fàcilment quan no els resistisquen, és òbviament un objectiu desitjable."*

D'acord amb este objectiu, expressat en la **tercera** declaració, en esta tesi es desenvolupen estratègies que complixen la condició de combinar la robustesa i *estabilitat* (l'estabilitat és un cas especial de flexibilitat). Pel que nosaltres coneixem, esta combinació encara no s'ha desenvolupat per a CSPs.

Les estratègies proactives utilitzen el coneixement disponible sobre dels possibles canvis futurs a fi d'evitar o reduir al mínim els seus efectes. No obstant això, per a molts dels problemes de la vida real la informació sobre l'entorn incert i dinàmic és desconeguda o difícil d'obtindre. Per a moltes estratègies proactives que busquen solucions robustes, la forma de l'algoritme depén d'un coneixement detallat sobre l'entorn dinàmic. Com resultat, estos mètodes no es poden aplicar si la informació requerida és desconeguda. Per aquesta raó, l'autor de Hebrard (2006) fa la **cuarta** declaració com una característica desitjable de les estratègies que s'enfronten a aquest incert i dinàmic marc.

Cuarta *"Idealment, no es requerix cap coneixement addicional sobre les dades utilitzades per a construir la clàssica xarxa de restriccions ni més experiència per a resoldre el problema sense tindre en compte la incertesa."*

En esta tesi, tractem de respondre a una pregunta interessant en entorns dinàmics i incerts: quan no es coneix informació addicional sobre els possibles canvis en el problema, és possible definir el que és la robustesa d'una solució d'un CSP i desenvolupar algoritmes apropiats?. Nosaltres creiem que és possible i justificable extraure algunes suposicions limitades (i intuïtivament raonables) sobre el dinamisme dels problemes per als que l'orde sobre els elements del domini és important. Per tant, en este treball presentem estratègies per a fer front a tipus de problemes que són, per tant, consistents amb la **cuarta** declaració. El compliment d'estes quatre declaracions ha sigut la motivació i l'objectiu principal del treball presentat en esta tesi.

Contents

List of Algorithms

List of Tables

List of Figures

Chapter 1

Introduction

This thesis introduces algorithms and techniques for finding solutions with certain level of robustness and stability for problems that come from uncertain and dynamic environments. Robust solutions have a high likelihood of remaining solutions when faced with further possible changes over the original problem. This dissertation focuses on difficult situations when the problems do not have extra detailed information associated about the dynamism. Thus, the techniques presented consider the robustness criterion according to a new dynamism environment, which it is presented in this dissertation, for problems whose elements have order relationships between them.

This chapter first introduces concepts related constraint programming. Secondly, the problematic situations associated with uncertain and dynamic environments, as well as previous work that deals with problems that come from these environments, are explained. Since the level of knowledge about the possible future changes is highly significant, a detailed explanation about this issue is also provided. Finally we mention the contributions of the work developed, as well as a brief overview of the contents that are further explained in detail throughout this dissertation.

1.1 Constraint Programming

Constraint programming (CP) is a powerful paradigm for modeling and solving combinatorial problems. CP was born as a multi-disciplinary research area that embeds techniques and notions coming from many other areas, among which are artificial intelligence, computer science, databases, programming languages, and operations research. Constraint programming is currently applied with success to many domains such as scheduling, planning, vehicle routing, configuration, networks, bioinformatics, etc.

In CP the relations between variables are stated in the form of constraints. Many real life problems are explicitly composed by constraints and therefore, if they are properly modeled, the framework and algorithms of CP can be used in order to solve them. These problems can be modeled as Constraint Satisfaction Problems (CSPs),

which are composed of a finite set of variables, a finite set of values associated with each variable and a finite set of constraints that restrict the values that the variables can simultaneously take. Thus, if there exist at least one combination of domain values that satisfy all the constraints, the CSP is satisfiable. The set of all the domain values combinations of the variables that satisfy all the constraints are the complete set of solutions of the problem. If there does not exist any solution, the CSP is unsatisfiable.

As an example of a CSP modeling, a toy scheduling problem is roughly modeled as a CSP. In following chapters, a formal definition of a CSP (see Section 2.1) and the formal model of this toy example (see Section 8.1) are introduced. Scheduling problems consist in scheduling a set of finite tasks that compose a finite set of jobs and assigning to them the resources that each task needs over time. A common objective for these problems is to minimize the makespan. The makespan is the last finishing time of all the tasks of all the jobs of the schedule. These problems are well-known real life problems and have been vastly used in both research and application frameworks. Furthermore, throughout this thesis, the scheduling problems have been analyzed and evaluated with the proposed approaches.

Example 1.1.1. *In this example a toy scheduling problem with only* 2 *tasks is presented. Tasks* T_0 *and* T_1 *are both of duration* 3 *time units. Both tasks use the same resource, and therefore both tasks can not be executed in parallel. Furthermore, there is a task precedence requirement, which states that task* T_0 *has to be executed before* T_1*. For this example we assume that there is not a restrictive maximum desired makespan. This problem can be modeled as a CSP by defining a set of variables, domains and constraints that properly represent the problem. For modeling the problem as a CSP, the start time of each task can be modeled as a variable whose domain values are the time units available. For this toy scheduling problem, there exist only one constraint that predefines the task order.*

This CSP is satisfiable and one of its solutions is represented in Figure 1.1. In this figure, T_0 is represented in green and T_1 in orange. The start time of task T_0 is 0 and the start time of task T_1 is 3 time units later, that is, T_1 starts after the finalization of task T_0. Note that this solution satisfies the task order requirement.

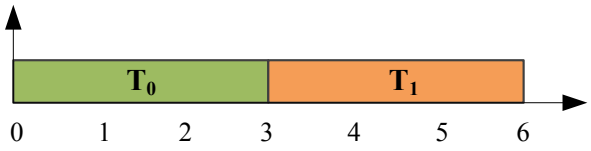


Figure 1.1: Valid schedule for Example 1.1.1.

Example 1.1.1 can be easily solved because it has a small number of variables, domain values and constraints. However, typically, real life problems have a high complexity due to the increased number of variables, domain values and constraints. In these cases, CP techniques are useful in order to solve the CSP in a reasonable

time. Among the CP techniques that deal with these problems, there exist: filtering techniques, learning and distributed techniques, the use of efficient representations and heuristics, etc. This effort resulted in the design of constraint reasoning tools which are used to solve numerous real problems. More detailed information about constraint programming can be seen in (Tsang, 1993; Rossi et al., 2006; Lecoutre, 2013).

1.2 Dynamism and Uncertainty

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. For real life problems, this represents a strong limitation because usually such problems come from dynamic environments. Hence, both the original problem and its corresponding modeled CSP may evolve over time because of changes in the environment, in the user or in other agents. For instance, in a bus transport system, an incident may occur due to some technical component failure, making the bus unavailable until repairs can be made. Another example of a possible incidence could be due to weather conditions, for instance, the buses could require lowering the speed due to the dangerous road conditions when it is storming.

The changes that the original problem undergoes also affect to its corresponding CSP model. However, the original and static model of a CSP is not able to capture further modifications over the original problem. To solve this shortcoming, in (Dechter and Dechter, 1988) the authors define a variant of the mathematical CSP model originally called Dynamic Constraint Network (DCN) and later Dynamic Constraint Satisfaction Problems (DynCSPs). Unlike CSPs, DynCSPs are able to capture the changes that the original problem undergoes. A DynCSP is composed by a sequence of static CSPs, where each one is a result of a change in the previous one. In this way, each static CSP represents new facts about the dynamic environment being modeled. In Section 2.3 a formal definition of a DynCSP is introduced.

We would like to highlight that for modeling a problem that comes from a dynamic environment as a DynCSP, it is necessary to have some knowledge about the future facts that the dynamic environment may undergo. Nevertheless, the real world is uncertain in its nature and therefore, information about the dynamism of the environment may be incomplete, erroneous or may even not exist. For instance, in the bus transport system example explained above, the weather conditions, such as rain or storm, could be approximated with certain likelihood. However, this data may not be accurate enough or might be incorrect. Moreover, there are a number of future events that are completely unforeseen (e.g., the crash of a bus or the breakdown of its engine). These partially/totally unexpected events cannot be predicted and therefore can not be modeled as a DynCSP. In such cases, the original CSP refers to the first static CSP and only when future modifications occur, these changes could be modeled

as further static CSPs in a DynCSP. In this dissertation we deal with these situations, thus, we refer as original CSP to the first static CSP that models the original problem before that any future uncertain change occurs.

Because of the possible future changes in the original problem, a solution that holds for the original CSP model can become invalid after some changes over the original problem. The solution loss could produce several negative effects in the modeled problem. In a real problem of task assignment in a production system with several machines, it could cause the shutdown of the system, the breakage of machines, the loss of the material/object in production, etc. Another example of the solution loss consequences is focused on the transport timetabling problem, where a disruption at some point may produce a delay that propagates through the entire schedule. In addition, all the negative effects stated above can also entail an economic loss. For instance, in a bread production system, if an incidence occurs that is resulting in a delay in the bread kneading process, the bread will not be properly kneaded when the bread baking process starts. In this case, the bread would not have the proper consistency and could not be sold. Due to all these negative effects derived by the solution loss, authors working in this dynamic and uncertain framework (including the authors of this dissertation) strongly value the solution loss prevention in dynamic environments.

Considering again the toy scheduling problem of the Example 1.1.1, a problem that frequently occurs in real life, the tasks of the problem may undergo delays in the future. Here, for instance, we assume that task T_0 undergoes a future delay of one time unit due to some disruption. Thus, the valid schedule represented in Figure 1.1 will become invalid after this incident. Note that task T_1 is not able to start in its assigned time due to the extension of T_0 duration. This fact is shown in Figure 1.2 by a superposition of one time unit of both tasks.

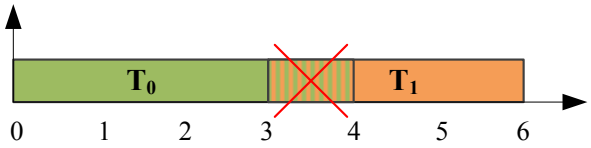


Figure 1.2: Invalid schedule for Example 1.1.1 after a delay in T_0.

The research community interest in real life problems that may evolve in the future has expanded over time. For these problems, the objective is beyond simply finding an optimal solution because it is also important to consider the dynamism and uncertainty of the environment. We recommend an insightful survey of approaches related to this area: (Verfaillie and Jussien, 2005).

Following is a brief overview of different approaches and their limitations, with special remarks on those that are of the same form as those dealt with in this thesis. There exist two main approaches for dealing with problems that come from dynamic and uncertain environments: *reactive* and *proactive* approaches.

1. **Reactive Approaches**: These approaches are applied when the computed solution becomes invalid (note that in this case the term 'solution' does not hold anymore) after changes in the problem. Their main objective is to obtain a new solution, which generally should be as similar as possible with respect to the previous solution found before the changes occurred, as efficiently as possible.

2. **Proactive Approaches**: These approaches are applied before the possible future changes over the original problem occur. Thus, they use the available knowledge about the possible future dynamism in order to avoid or minimize their effects in the original solution that they compute.

Regarding proactive approaches, there exist two main properties that are associated with the solutions obtained: *flexibility* and *robustness*. Furthermore, there exists another characteristic called *stability*, which is a more restricted concept of the flexibility characteristic. Hence, the definitions of these concepts are introduced in Section 1.4.1.

Since proactive approaches search for an original solution for the original dynamic problem and the reactive approaches search for a new solution after the computed solution becomes invalid, both approaches are compatible and can be applied simultaneously. A proactive approach can be initially applied and a reactive approach would only be needed if the previously obtained solution is lost after changes in the problem. For instance, in (Wallace and Freuder, 1998) dynamic problems are solved by combining both types of approaches.

1.3 Reactive Approaches

Reactive approaches are applied when the solutions are no longer solutions because of the changes in the dynamic problem. Thus, they are applied to the last CSP of a DynCSP that models the changes that have invalidated the previous solution. These strategies are oriented to find a new solution or to repair previous unfeasible assignments. Moreover, most reactive approaches try to minimize the number of needed changes for such a reparation. According to (Verfaillie and Jussien, 2005), reactive strategies can be classified into two main groups:

1. **Solution Reuse Techniques**: These approaches use any previous complete or incomplete consistent assignment (before changes over the previous CSP) and repair it, until the new assignment is consistent with the current DynCSP.

2. **Reasoning Reuse Techniques**: These approaches use information about the affected parts of any previous consistent assignment (complete or not) for solving the current DynCSP and finding a new solution.

Some solution reuse techniques are local search-based, which use the solution that has become invalid for starting the local search by repairing its inconsistent values.

For instance, the version *min-conflicts* heuristic (Minton et al., 1992) tries to minimize the number of constraints that are violated by the current assignment. These kind of approaches have also been applied to scheduling problems in order to reconfigure schedules in response to a dynamic environment. In these environments, external factors invalidated the existent schedule through the withdrawal of resources, the arrival of new resources or because of changes in the set of the scheduled activities. Thus, these methods firstly, model the problem as a DynCSP and secondly, apply heuristics, local repair methods, etc. For instance, in (Verfaillie and Schiex, 1994) a technique applied to dynamic scheduling problems is presented. It is based on the idea that it is possible to enter a new task *t* iff there exists for *t* a location such that all the tasks whose location is incompatible with t's location can be removed and entered again one after another, without modifying t's location.

In further work of solution reuse techniques, the goal is beyond repairing the solution because the main objective is to the differences of the new repaired solutions with regard to previous solutions. This is called minimum perturbation/change and it has been applied to constraint programming among other works in (Bellicha, 1993; Ran et al., 2002). Furthermore, there exist many minimum perturbation approaches applied to scheduling. These techniques search for a new schedule that differs minimally from the previous one, since it is no longer a solution after changes in the problem. For instance, a technique applied to dynamic scheduling problems that is based on the reduction of the *contention* in the constraints (Sakkout and Wallace, 2000). A constraint has *contention* when certain combinations of domain values of the variables may violate the constraint. For some constraints, *contention* can be measured, so that the search can be oriented to regions where the *contention* is small. Once the feasibility phase of the resources has been completed, the makespan optimization phase is executed.

Reasoning reuse techniques employ information about the inconsistencies between the static CSPs that form the DynCSP. The main idea is based on the assumption that changes between one static CSP and the previous one are small and therefore both CSPs show similarities that can be exploited. In addition, since the previous solution has been invalidated, the next CSP is more restricted. For this reason, most of the implied constraints, which are deduced from the consequences of the constraints, will remain valid. These techniques first determine the validity of the implied constraints and subsequently compose the new implied constraints. This information, which is extracted from the inconsistencies produced in the DynCSP, is subsequently used for solving the last static CSP of the DynCSP. An example of a reasoning reuse technique can be found in (Schiex and Verfaillie, 1994), which is an explanation-based method that records information about the implied constraints. By means of this data, the technique removes the implied constraints whose justification is based on another invalid implied constraint. Another example of this type of techniques can be found in (Verfaillie and Schiex, 1994), where the changes can occur while solving a CSP and the solving algorithms are adapted to avoid restarting the search from scratch.

As mentioned, reactive approaches re-solve the CSP after each solution loss, which consumes computational time. That is a clear disadvantage, especially when we deal with short-term changes. This idea is also shared by the **first** statement mentioned in the abstract, which comes from (Verfaillie and Jussien, 2005).

First *"Limit as much as possible the need for successive online problem solvings"*.

Another important issue to be considered in uncertain and dynamic situations are the temporary changes. This kind of changes may occur repeatedly and with varying frequencies thus resulting in frequent solution loss. Furthermore, in many applications, such as on-line planning and scheduling, the time required to compute a new solution may be too long for actions to be taken on time. In addition, the solution loss (independently if a new solution is provided on time or not) has several disadvantages, which have been mentioned in Section 1.2.

1.4 Proactive Approaches

Limiting the need to resolve a problem instance as much as possible is an important issue for dynamic problems, specially for those that undergo frequent changes. Proactive approaches try to avoid the aforementioned drawbacks of re-solving the problem after changes occur. Hence these approaches are highly valued for dealing with problems in uncertain and dynamic environments. Several proactive approaches have been proposed in the literature for handling this type of problem, which can be classified based on the kind of solutions that they obtain. Following, this section introduces the solution properties: robustness, flexibility and stability. In addition, we explain several earlier proactive approaches that deal with problems that come from uncertain and dynamic environments.

1.4.1 Robustness, Flexibility and Stabiity

We first clarify general concepts about these terms and subsequently they are specifically defined for CSP solutions. There is a thin line that separates these concepts, and in some occasions, they have been incorrectly interpreted in the literature. For instance, some researchers mix up the terms stability and robustness, or use them interchangeably.

In general, a solution is stable in a dynamic system, if by means of a few changes in the solution we can obtain a new solution that is similar to the original one. However the robustness concept is broader than the stability concept. Robustness is a measure of feature persistence in systems that compels us to focus on perturbations because they represent changes in the composition or topology of the system. Here, a perturbation is a small difference in the actual state of the system (Jen, 2003). In the field of Constraint Programming, the authors of (Verfaillie and Jussien, 2005) define the concepts of flexible and robust solutions for the specific case of CSPs:

Definition 1.4.1. *A robust solution has every chance to resist changes, i.e., to remain a solution in spite of these changes (Verfaillie and Jussien, 2005).*

Definition 1.4.2. *A flexible solution is anything (a partial solution, complete solution, conditional solution, set of solutions, etc.) that, in case of change, can be easily modified to produce a solution to the new problem (Verfaillie and Jussien, 2005).*

Note that both concepts have a strong dependency with respect to the assumptions of the possible future changes that may occur. Furthermore, regarding Definition 1.4.1, there are cases in which it is not possible to find a solution that is able to resist all the changes, especially when information about possible future changes is limited. In order to consider these cases, Definition 1.4.1 can be slightly modified to define the most robust solution for CSPs.

Definition 1.4.3. *The most robust solution to a CSP within a set of solutions is the one with the highest likelihood of remaining a solution after changes in the CSP.*

Regarding Definition 1.4.2, a more specific concept was defined for CSPs: that of stable solutions ((Hebrard, 2006), cf.(Verfaillie and Schiex, 1994)):

Definition 1.4.4. *A solution $s1$ is more stable than another solution $s2$ iff, in the event of a change that invalidates them, a closer alternative to $s1$ than to $s2$ exists (modified from (Hebrard, 2006)).*

The main difference between the definition of stability (Definition 1.4.4) and flexibility (Definition 1.4.2) is that the former introduces the concept of 'closer' solution. Thus, both Definitions 1.4.2 and 1.4.4 consider the ease with which the original solution can be modified to produce a solution to the new problem, but only Definition 1.4.4 specifies how much closer the new solution is to the original solution. The measurement of this closeness will be explained in Section 1.4.3.

Furthermore, note that the definitions of robustness (Definition 1.4.1 and Definition 1.4.3) do not consider the alterations in the original solution but only its resistance to changes in the problem. In contrast, the definitions of flexibility and stability (Definitions 1.4.2 and 1.4.4) do consider changes to the original solution when a new solution is produced after a change in the problem.

As mentioned in Section 1.2, the loss of the solution may cause several negative effects. This is also motivated in (Verfaillie and Jussien, 2005) as a desirable objective, as mentioned in the **second** statement of the abstract.

Second *"Limit as much as possible changes in the produced solution".*

From hence, it is very important to avoid the loss of the solution and if it is lost, it is very valued that the newly found solution is as similar as possible to the previous one. Thus, in this dissertation, we first prioritize robustness and secondly stability.

We reconsider again the toy scheduling problem of Example 1.1.1. Figure 1.3 represents a valid schedule for this example. Furthermore, this solution is robust

when faced with the kind of incidence explained in Section 1.2 (delay of 1 time unit in task T_0). This is due to this schedule is able to remain valid when faced with this type of dynamism. Figure 1.4 shows how the idle time after task T_0 absorbs the delay in this task and therefore the start time of the next task (T_1) remains valid.

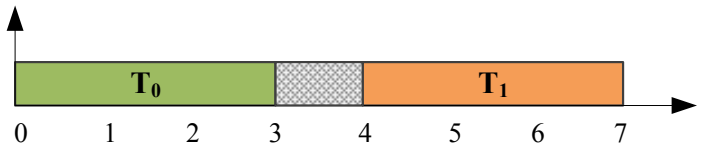


Figure 1.3: Valid schedule for example Example 1.1.1.

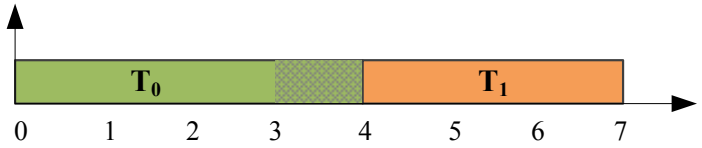


Figure 1.4: Valid schedule for example Example 1.1.1 after a delay in T_0.

1.4.2 Searching for Robust Solutions

Many proactive approaches search for robust solutions according to the information of the uncertain and dynamic environment that they consider. Following, some representative techniques of this type are categorized according to their dynamism assumptions. Note that the detailed information that these approaches require is directly related to the assumptions made about the possible future changes.

1. Dynamism in the solution values: These approaches consider that some values of the solution might become invalid in the future. Thus, they use insights about previous invalidations to gather statistics on which values in the solution should be preferred. In (Wallace and Freuder, 1998) a technique that gathers the data in the form of penalties is proposed, in which values that are no longer valid after a change in the problem are penalized. Thereafter, the algorithm tries to find solutions that do not include these penalized values.

2. Dynamism in the constraints: These approaches consider as possible future changes constraint additions, restrictions and/or deletions. For dealing with these uncertain constraints, a variant of the original CSP model can be used: Flexible CSP (see Section 2.4.1 for a detailed explanation). The main difference that this extension of the original model includes, is that constraints can be violated to some extent. There exist several types of Flexible CSPs according to the manner of representing the range of violation. Among these, we following

introduce two models, in addition, in Section 2.4.2 another model is explained in detail: the Weighted Constraint Satisfaction Problem (WCSP). This model has a specific section in Chapter 2 due to it is used by several approaches presented in this thesis and therefore, a more detailed and technical explanation is necessary.

- The Probabilistic CSP (PCSP) (Fargier and Lang, 1993) has an associated likelihood of existence of each constraint. Thus, according to these probabilities, the most robust solution is the solution that maximizes the probability of satisfying the constraints.
- The Fuzzy CSP (FCSP) (Rosenfeld et al., 1976) has associated functions that express the satisfiability/unsatisfiability of the values for each constraint. The most robust solution of a FCSP is the solution that maximizes the value of the minimum satisfiability of all the constraints.

3. Dynamism in the variables and their domain values: The approaches classified in this group consider uncertainty about the domain values associated with certain variables or that the variables are uncertain theirselves. Some of the main types of approaches are:

   - 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 satisfy all possible values that the uncontrollable variables can take.
   - The Uncertain CSP model (UCSP) (Yorke-Smith and Gervet, 2009) is an extension of the MCSP, whose main innovation is that it considers continuous domains.
   - 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) (Fowler and Brown, 2003) considers the possible addition of variables to the current problem. 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 original variables, and also considers their probability of future existence.

In most of these aforementioned 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 are useless if the required information is unknown. In many real problems, however, knowledge about possible further changes is either limited or non-existent. Hence, there is an important need for techniques that search for robust solutions in this kind of environment. Therefore, this is one of the main motivations of this dissertation.

1.4.3 Searching for Stable Solutions

One of the most known approaches for the search of stable solutions for CSPs is the super-solutions approach. 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. Here, the measurement of how 'close' the two solutions are is measured as the number of variables that take the same value in both solutions. That is to say, the dissimilarity is measured as the number of variables that do not take the same value in both solutions.

Definition 1.4.5. *A solution is an (a, b)-super-solution if the loss of values of at most a variables can be repaired by assigning other values to these variables and changing the values of at most b other variables (Hebrard et al., 2003).*

For CSPs, a major focus has been on finding $(1, 0)$-super-solutions (due to the high computational cost of computing $a > 1$ or $b > 0$). This is one of the reasons for which we analyze this particular super-solution case in this thesis. The other reason is that limiting the changes in the original solution as much as possible, motivates the search of (a, 0)-super-solutions.

Definition 1.4.6. *A solution is (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 et al., 2004).*

The (1, 0)-super-solutions can be simply explained by the following example:

Example 1.4.1. *Let us consider the following CSP:*
$x_0, x_1 \in \{1, 2, 3\}$
$C_1 : x_0 \leq x_1$

- *The solution $(x_0 = 1, x_1 = 1)$ is not a (1, 0)-super-solution, because if the variable x_0 loses the value 1, it is not possible to find another value for x_0 that is consistent with C_1, since $(x_0 = 2, x_1 = 1)$ and $(x_0 = 3, x_1 = 1)$ are not solutions to the problem.*

- *The solution $(x_0 = 1, x_1 = 2)$ is a (1, 0)-super-solution, because if any variable loses its value, at least one value can be found that is compatible with the assignment of the other variable. If x_0 loses its value 1, a value of 2 can be*

assigned to x_0, since $(x_0 = 2, x_1 = 2)$ is solution of the problem. If x_1 loses its value 2, a value of 1 or 3 can be assigned to x_1, since $(x_0 = 1, x_1 = 1)$ and $(x_0 = 1, x_1 = 3)$ are solutions of the problem.

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 the 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).

1.5 Dynamism Knowledge

The dynamism knowledge associated with dynamic problems is a very important issue to take into account. Ideally, all the information about the future changes would be known and certain. In this case, DynCSPs could be modeled with complete certainty prior to any changes taking place. Nevertheless, as mentioned in Section 1.2, the real world is uncertain in its nature and therefore, the information about the dynamism of the environment may be incomplete, erroneous or may even not exist. Under these circumstances, it is desirable to consider approaches that do not require extra information about the future changes. Indeed, these techniques should be able to solve the problem only with the data needed for modeling and solving the original classical CSP without considering the possible future changes.

As previously mentioned, fully reactive approaches are applied when changes to the original problem invalidate the partial/complete solution. For this reason, dynamism knowledge does not concern them. In contrast, proactive approaches are applied before changes in the original problem happen, and therefore have a strong dependency with the dynamism knowledge. For many of these approaches, the information is gathered in the form of probabilities distributions and/or a list of the possible changes. As a result, these approaches are specific to problems with extra data, and are therefore useless if this information is unknown. In many real problems, however, knowledge about possible further changes is either limited or non-existent. Hence, there is an important need for techniques that find robust solutions in this kind of environment. Note that the latter is very difficult to achieve due to the very definition of a proactive approach, which involves techniques that try to achieve a resistance to future perturbations. Therefore, at least some little assumptions about the future perturbations is required in order to avoid or minimize their effects. The less detailed the dynamism knowledge is required by an approach, the more difficult to characterize the robustness and stability of its solutions, but the greater the approach usability is. This is also expressed in the **fourth** statement of the abstract, which comes from (Hebrard, 2006).

Fourth *"Ideally, no additional knowledge over the data used to build the classical constraint network is required and no more expertise than for solving the problem without taking uncertainty into account."*.

This dissertation provides an answer to an interesting question in dynamic and uncertain environments: when we have no information about the possible changes in the problem definition, is it possible to define what is the robustness of a solution of a CSP and to build algorithms able to find the most robust solutions? We found that it is possible, reasonable and justifiable to extract some limited (and intuitively reasonable) assumptions about the dynamism of problems for which the order over the domain elements is significant. In this framework, it is common that any bound of the solution space undergoes restrictive or relaxed modifications in the future (this is motivated in Section 3.1). These little assumptions are inherent to the structure of such problems and based on them we present algorithms that search for robust solutions under these aforementioned difficult conditions in which no extra detailed dynamism data is available. Furthermore, we also present another approach that is informed. The data required by this technique, is related to the same dynamism assumptions for CSPs with ordered domains.

It must be taken into account that for such a dynamism environment, the possibility of solution loss only exists when changes over the original bounds of the solution space are restrictive. Recall that minimizing the risk of losing the solution of the CSP is the main objective of proactive approaches that search for robust solutions. In Chapter 3 we will fully explain how the latter concept is related to the feasible neighbours assignments surrounding the solution. At this stage, the solutions covered by these neighbours, under certain circumstances, are introduced as 'covered solutions'.

Figure 1.5 is an adaptation from a figure in (Hebrard, 2006) and it shows several existent approaches for problems in dynamic and uncertain environments represented in black. Furthermore, our two different approximations described above, which search for covered solutions with and without extra dynamism information, are represented with red. This figure represents a 3-dimensional space that categorizes the features of these approaches in terms of: reactivity/proactivity, robustness/stability and knowledge intensity. Observing the existing approaches and their distribution over the space, we can notice that one of the missing points is the search for robust solutions with low dynamism knowledge. This dissertation focus on this region (see the red point 'covered solutions'). In addition, even if it has not been our chief work, we also analyzed the search of robust solutions when there exists a high level of dynamism knowledge of future bounds restrictions of the solution space of CSPs with ordered domains (see the red point 'covered solutions with dynamism data'). This specific type of dynamism knowledge had not been analyzed in the literature before.

Regarding the robustness/stability axis in Figure 1.5, we believe that separate these concepts in two axis would be more appropriated. The fact that a solution is robust does not necessarily entail a reduction of its stability. This is also expressed in the **third** statement of the abstract, which comes from (Verfaillie and Jussien, 2005).

Even so, most approaches in the literature focused only in one of these features. As contrary, in this dissertation we analyze these both concepts and their influence. Our approaches prioritize the search of robust solutions, but these solutions also possess some stability properties (Chapter 3). For representing this features combination, the work proposed is placed closer to the stability point than other robust approaches (see red points in Figure 1.5).

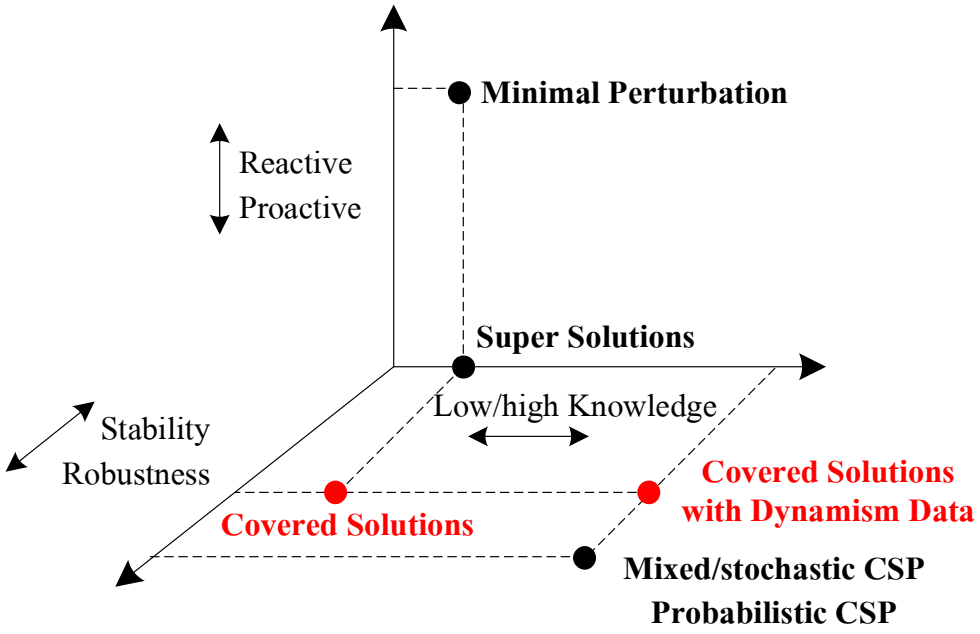


Figure 1.5: Six proactive and reactive approaches and their characteristics.

1.6 Contributions and Outline

In this dissertation we present proactive approaches that search for robust and stable solutions by taking profit of inherent limited assumptions to problems in which the order of the elements is significant. This order feature has an impact on the possible future changes of the problem. To the best of our knowledge, this inherent dynamism information had not been analyzed in the constraint programming literature. This thesis contributes toward better understanding, modeling and solving of problems that come from this kind of dynamic and uncertain environments. Furthermore, we categorize and extend the robustness and stability of the solutions in these difficult and common situations.

Previous work exists that explores the search of stable solutions in situations in which there does not exist additional dynamism information (Hebrard, 2006). Nevertheless, the dynamism assumptions of this work are not order-related and we recall

that stability of solutions considers modifications over the solutions, which produces several negatives effects (see Section 1.2). Robust solutions, however, maximize the likelihood of remaining solutions in the absence of further modifications over them. In addition, the solutions obtained by our approaches not only posses robustness characteristics, but also a specific type of stability.

Furthermore, based on the same dynamism assumptions of problems for which the order is significant, we also analyzed the search of robust solutions for CSPs with ordered domains with extra data about the dynamism of the solution space bounds. The main advantage that all the approaches introduced in this thesis (with and without extra information) present over the reactive strategies, is that they try to prevent the loss of the original solution, by finding robust solutions. Following, we enumerate the main contributions of this thesis:

1. We introduce a new dynamism framework for CSPs with ordered domains that model real life problems in which the order is significant, introducing theoretical approaches for dealing with such environment. Moreover, we further extend the definitions of robustness and stability associated with the solutions in such framework.

2. We present an informed approach that considers extra dynamism data associated with the bounds of CSPs and an algorithm that models such information in order to find solutions that present a resistance to these possible future changes.

3. We present three uninformed approaches that do not consider extra data about the dynamism but only consider the existence of an order over the elements of the domains of the CSPs. Among them, we present a modeling technique that is specific for finite linear CSPs. The other two techniques are not limited to this type of CSPs. The first approach is a modeling technique and the second approach is a search algorithm. These three techniques share the main objective of searching for robust solutions for CSPs with ordered domains. Further detailed explanation about their differences is explained in the chapters associated with these approaches.

This dissertation is structured in 10 chapters organized as follows:

- **Chapter 1, Introduction:** In this chapter the constraint programming framework is introduced. Furthermore, the concept of dynamic and uncertain environment is explained in detail. Subsequently, we explain the related work and an overview of the approaches proposed in this thesis for dealing with these situations.

- **Chapter 2, Technical Background:** This chapter provides several standard notations and definitions from the literature. This formalized technical information will be used in this thesis and is therefore necessary for a correct understanding of the new concepts and approaches presented. These concepts belong

to Constraint Programming. Among them, can be found the formal CSP definition as well as some CSP extensions: DynCSPs and Constraint Satisfaction and Optimizations Problems (CSOPs). Furthermore, we explain concepts related to the search and consistency methods for solving CSPs and CSOPs.

- **Chapter 3, New Dynamism Framework for CSPs with Ordered Domains:** This chapter focuses on the introduction and motivation of a dynamism framework associated with CSPs that model problems whose elements are related by a certain order relationship (Climent et al., 2013b). Moreover, we extend the robustness and stability concepts to this dynamism environment (Climent et al., 2014). For such framework, we propose two new theoretical approaches (Climent et al., 2011, 2013c). Finally, two technical approaches are compared: enumeration-based techniques and a search algorithm.

- **Chapter 4, Probabilistic Enumeration-based Technique:** We propose a new probabilistic approach (Climent et al., 2012a). This enumeration-based technique models CSPs with extra information about the dynamism of their bounds as WCSPs, with the objective of finding robust solutions according to this dynamism knowledge. The information about the dynamism is gathered in two parameters associated with the dynamic bounds of the CSP. One of them measures the likelihood of a constraint of undergoing a restrictive modification. The other parameter measures the magnitude of change of the aforementioned constraint.

- **Chapter 5, Enumeration-based Technique for Finite Linear CSPs:** In this chapter an enumeration-based technique designed specifically for finite linear CSPs is introduced (Climent et al., 2011). Unlike the approach presented in the previous chapter, this enumeration-based technique is non-probabilistic. The only assumptions made about the dynamism are those inherent to the structure of problems with ordered domains. This approach searches for solutions close to the centroid of the geometric figure representing the dynamic bounds of Linear CSPs.

- **Chapter 6, Enumeration-based Technique by Coverings:** An enumeration-based technique without extra detailed information about future changes (Climent et al., 2012b, 2013c,b) is presented. Unlike the technique for Linear CSPs, this approach is for general purpose and therefore it can be applied to CSPs with non-convex solution spaces. The main idea of this approach is to search for solutions that are covered by feasible assignments for each bound. Since this means that the solution is located far from the bounds.

- **Chapter 7, Search Algorithm for Finding Robust and Stable Solutions:** Hitherto all the approaches introduced were enumeration-based approaches. In this chapter a search algorithm that directly searches for robust and stable solutions is introduced (Climent et al., 2012c, 2014). Thus, it is not necessary an

intermediate step of WCSP modeling. Furthermore, the main advantage with respect to the enumeration-based technique for CSPs by coverings is that the checking of the feasibility is done directly in the solution space, instead of each bound. The search algorithm searches for solutions that maximize an objective function, which is the sum of the feasible contiguous surrounding neighbours in each variable axis of the tentative assignment.

- **Chapter 8, A Case Study: Scheduling Problems:** There exist real life problems that can provide more information about their dynamism due to their special characteristics. In this chapter we analyze the scheduling problems, which are well-known real life problems. The special feature that these problems have is that the domain values represent time units. By using this knowledge, we adapt the two general uniformed approaches introduced in this dissertation: the enumeration-based technique for CSPs by coverings and the search algorithm (Climent et al., 2013a). In this way, better robustness and stability results can be obtained. Furthermore, a toy benchmark is analyzed in order to show the robustness and stability results, graphically.

- **Chapter 9, Evaluation:** In this chapter we describe the experimental results performed for the analysis of the approaches presented in this dissertation. We compare some of our techniques with another proactive approach that does not require extra detailed dynamism data. Since there do not exist DynCSPs benchmarks in the literature, we explain the process of simulating dynamism that we perform in order to compute the robustness of the solutions obtained (Climent et al., 2009a,c,b; Salido et al., 2009). The experiments are carried out with random CSPs and benchmarks from the literature, among them can be found several scheduling problems. Finally, we analyze the advantages and limitations of the approaches presented in this thesis.

- **Chapter 10, Conclusions:** Finally, we discuss about the strengths and limitations of the approaches presented in this thesis, as well as their contributions to the literature. Future lines of work are discussed. In addition, the list of publications associated with the work presented is mentioned.

Chapter 2

Technical Background

This chapter introduces formal definitions and explanations of Constraint Programming concepts that are used throughout this dissertation, and follows the standard notations in the literature. First, we focus on explaining in more detail some concepts introduced in Chapter 1 about Constraint Satisfaction Problems (CSPs) and Dynamic Constraint Satisfaction Problems (DynCSPs). Secondly, the Constraint Satisfaction and Optimization framework is introduced, paying special attention to the model that will be repeatedly used by three approaches presented in this thesis: the WCSP model. Finally, Section 2.5 contains general concepts of inference techniques and Section 2.6 explains some hybrid methods. Furthermore, several heuristics and strategies that can have a huge impact on the reduction of the computation time are explained. These techniques are divided into variable ordering heuristics and restarting strategies.

The definitions and techniques described in this chapter belong to a vast area of Constraint Programming. We only focus on those concepts that are necessary for understanding the framework and approaches presented in this dissertation. For more extended information, we strongly recommend the following books: (Rossi et al., 2006), (Lecoutre, 2013) and (Dechter, 2003).

2.1 Constraint Satisfaction Problems (CSPs)

In this section, we introduce the formal definition of a Constraint Satisfaction Problem and several concepts related with this model. Most of the concepts presented can be found in the literature, however, some of them are specific to this thesis.

Definition 2.1.1. *A Constraint Satisfaction Problem (Mackworth, 1977b), abbreviated as CSP, is represented as a triple* $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ *where:*

- $\mathcal{X}$ *is a finite set of variables* $\mathcal{X} = \{x_0, x_1, \ldots, x_n\}$.

- *$\mathcal{D}$ is a finite set of domains $\mathcal{D} = \{D(x_0), D(x_1), \ldots, D(x_n)\}$ such that for each variable $x_i \in \mathcal{X}$, $D(x_i)$ is a set of values that the variable can take.*

- *$\mathcal{C}$ is a finite set of constraints $\mathcal{C} = \{C_0, C_1, \ldots, C_e\}$ which restrict the values that the variables can simultaneously take.*

Subsequently, we present a simple CSP example that will be referenced in order to explain several concepts throughout this chapter.

Example 2.1.1. *We consider a CSP named E_1 that it is composed of 3 variables, their corresponding domains (ordered respectively) and 2 constraints. $E_1 = \langle\{x_0, x_1, x_2\}$, $\{\{1,2,3,4,5\},\{1,2\},\{1,2,3\}\},\{\{x_0 - x_1 > 0\},\{x_1 + x_2 < 4\}\}\rangle$. Thus, $D(x_0) = \{1,2,3,4,5\}$, $D(x_1) = \{1,2\}$ and $D(x_2) = \{1,2,3\}$.*

2.1.1 Domains

There are several types of domains of CSPs. Here, we only explain the types of domains that have been used in this dissertation. Following, we explain two types of domains related to the nature of their values.

- *Numeric Domains*: The domains are composed of values in the set of the real numbers ($\mathbb{R}$) (Lhomme, 1993).

  For instance, integer values (e.g., $\{-1, 0, 1, \ldots\}$) and rational numbers (e.g., $\{0.5, 0.51, \ldots\}$) are numeric.

- *Symbolic Domains*: The domains are composed of symbols. Typically, the domain values that are not numeric are symbolic.

  For example, the colours, the clothing sizes, the names, etc.

Other two types of domains are related to the amount of elements that compose them (Lhomme, 1993).

- *Continuous Domains*: A set of values that consists in all the existent numbers in an interval.

  For instance, all the values in the interval $[1, 5]$ (including all the real numbers) represent a continuous domain.

- *Discrete Domains* (also called discontinuous domains): A set of concrete values within an interval.

  For example the integers $\{1, 3, 7\}$ and the rational numbers $\{0.3, 0.55\}$ represent discrete domains.

Unlike in discrete domains, in continuous domains it is impossible to enumerate all the elements that compose the domain due to the sampling space being defined over the real numbers ($\mathbb{R}$), which is infinite. Hence, it is not possible to enumerate all the solutions that compose a CSP with continuous domains.

2.1.2 Assignments

Following, we define concepts related to variable assignments associated with CSPs.

Definition 2.1.2. *A variable assignment is the association of a domain value to the corresponding variable.*

Among the different representations of such assignments, we use the pair variable-value $(x_i = a)$, where $a \in D(x_i)$. For instance, all the possible variable assignments for variable x_1 of the CSP of Example 2.1.1 are $(x_1 = 1)$ and $(x_1 = 2)$.

Definition 2.1.3. *A tuple t is an ordered assignment of domain values to a subset of variables of $\mathcal{X}$.*

Definition 2.1.4. *Given a tuple t, $\mathcal{X}_t \subseteq \mathcal{X}$ is the subset of variables that compose t.*

Some examples of possible tuples associated with the CSP of Example 2.1.1 are: $(x_0 = 1), (x_2 = 3), (x_1 = 2, x_2 = 3), (x_0 = 5, x_1 = 2, x_2 = 1), (x_0 = 1, x_1 = 2, x_2 = 1)$, etc. If a tuple is composed by the complete set of variables of a CSP, it is also called *complete* assignment. From the previous set of tuples, only tuples $(x_0 = 5, x_1 = 2, x_2 = 1)$ and $(x_0 = 1, x_1 = 2, x_2 = 1)$ are complete assignments. However, if a tuple is composed by a lower number of variables, it is called *partial* assignment. From the previous set of tuples the partial assignments are: $(x_0 = 1), (x_2 = 3)$ and $(x_1 = 2, x_2 = 3)$.

Definition 2.1.5. *For a subset B of $\mathcal{X}_t$, the projection of t over B is denoted as $t \downarrow_B$.*

For instance, considering $t = (x_0 = 5, x_1 = 2, x_2 = 1)$ and $B = \{x_0, x_2\}$, $t \downarrow_B = (x_0 = 5, x_2 = 1)$. Another projection involving only one variable is denoted as $t(x_i)$, and it represents the projection of t over $x_i \in \mathcal{X}_t$. For the previous example, $t(x_0) = 5$, $t(x_1) = 2$ and $t(x_2) = 1$.

2.1.3 Constraints

As mentioned earlier, in the CSP paradigm, constraints are linking variables and restricting the values that these variables can simultaneously take. Subsequently, specific concepts and properties related to the CSP constraints are described.

Definition 2.1.6. *A constraint of a CSP is a condition that restrict the values that the variables can simultaneously take.*

Definition 2.1.7. *The arity of a constraint is the number of variables involved in the constraint.*

Typically, constraints with only one variable are called unary constraints. Constraints with two variables are called binary constraints. However, constraints with three or more variables are called "n-ary" constraints, where n is their arity.

The domains of the CSP can be expressed as unary constraints. In the rest of this thesis when we refer to constraints we also refer to these unary constraints. In order to differentiate these constraints associated with the domains, we denote them by $\mathcal{DC}$. Note that $\mathcal{C} \cap \mathcal{DC} = \emptyset$.

The number of possible tuples of a constraint $C_i \in (\mathcal{C} \cup \mathcal{DC})$ 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 $var(C_i) \subseteq \mathcal{X}$ is the set of variables involved in C_i (scope of C_i). For instance, considering the constraint $C_1 = \{x_1 + x_2 < 4\}$, $var(C_1) = \{x_1, x_2\}$.

Definition 2.1.8. *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 2.1.9. *We denote the set of valid tuples of a constraint $C_i \in (\mathcal{C} \cup \mathcal{DC})$ as* $T(C_i)$.

Following, the cardinalities of the set of valid tuples for constraints are introduced.

$$|T(C_i)| = \begin{cases} |D(x_j)| & \text{if } C_i \in \mathcal{DC} \text{ and } x_j \in var(C_i) \\ \prod_{x_j \in var(C_i)} |D(x_j)| * (1 - tigthness) & \text{if } C_i \in \mathcal{C} \end{cases} \tag{2.1}$$

For further complexity analysis of the algorithms presented in this thesis, we will use the maximum number of valid tuples of the complete set of constraints, which is defined below.

Definition 2.1.10. *We denote the maximum set of valid tuples associated with the constraints of a CSP as* $T(C)_{Max}$.

$$T(C)_{Max} = max\{|T(C_i)|, \forall C_i \in (\mathcal{C} \cup \mathcal{DC})\} \tag{2.2}$$

There are two equivalent representations for expressing constraints:

- *Intensional constraint representation*: Constraints are represented as mathematical or logical functions.

  For instance, the two constraints of Example 2.1.1 ($\{x_0 - x_1 > 0\}, \{x_1 + x_2 < 4\}$) are intensional constraints.

- *Extensional constraint representation*: Constraints are represented as the set of all the valid/ invalid tuples.

  For Example 2.1.1, the set of tuples $((x_0 = 1, x_1 = 1), (x_0 = 1, x_1 = 2), (x_0 = 2, x_1 = 2))$ express extensionally the invalid tuples of the constraint $\{x_0 - x_1 > 0\}$.

Note that if the domains are discrete, all the intensional constraints can be extensionally represented, but this fact is not true in the opposite direction, since not all the extensional constraints can also be intensionally represented. Furthermore, extensional constraints can not represent continuous sets of valid tuples due to their own nature of representing each tuple one by one.

Regarding the types of constraints, we only focus on a type that is of interest for the work presented in this thesis. One of the most basic types of constraint is the linear constraint. Thus, by default, the rest of the constraints are called non-linear constraints.

Definition 2.1.11. *A linear constraint is a constraint in the form of Equation 2.3, where* $\mathcal{X}$ *represents the variables,* A *represents the coefficients of the variables and* c *is a constant.*

$$A\mathcal{X} + c\{<, \leq, =, \neq, >, \geq\}0 \tag{2.3}$$

Definition 2.1.12. *A finite linear CSP is fully composed by integer variables and linear constraints. An integer variable is defined in the set of the integer values.*

The search and solution spaces associated with finite linear CSPs are *convex spaces*, a property that we use in the approach developed in Chapter 5 and that it is explained in Section 2.1.5.

An additional categorization of constraints embraced throughout this thesis are the concepts of *hard* and *soft* constraints. Hard constraints have to be satisfied in all cases. However, soft constraints do not, so they can be relaxed but with a certain cost. The latter type of constraints is associated with constraint optimization and therefore explained at further depth in Section 2.4.

2.1.4 Solutions

Following, we define concepts of CSP solutions. In this thesis, in order to differentiate a tuple that is known to be feasible/valid, we name it s. The value assigned to a variable x_i in s is denoted as $s(x_i)$. If s is a partial assignment, it is also called partial solution. If s is a complete assignment, then it is a solution of the CSP.

Definition 2.1.13. *A solution of a CSP is an assignment of the domain values to each of the variables that does not violate any constraint.*

If such an assignment does not exist, the CSP is unsatisfiable. If there exists at least one solution, the CSP is satisfiable. Regarding Example 2.1.1, the complete assignment $(x_0 = 1, x_1 = 2, x_2 = 1)$ is not a solution of E_1 because it does not satisfy the constraint $\{x_0 - x_1 > 0\}$. However, the complete assignment $(x_0 = 5, x_1 = 2, x_2 = 1)$ satisfies the two constraints $\{x_0 - x_1 > 0\}$ and $\{x_1 + x_2 < 4\}$. That is why, it is a solution of E_1 and therefore E_1 is satisfiable.

The set of solutions of a CSP P is denoted as $\mathcal{S}(P)$. As it is known in the literature, the cardinality of a set A is usually denoted as $|A|$. Therefore, the number of solutions of a CSP P is denoted as $|\mathcal{S}(P)|$.

According to Definition 2.1.4, $\mathcal{X}_s$ is the subset of variables that compose a feasible tuple s. Thus, $\mathcal{X} \backslash \mathcal{X}_s$ represents the set of unassigned variables. Considering the feasible partial assignment $s = (x_0 = 5, x_1 = 2)$ for E_1 (CSP of Example 2.1.1), $\mathcal{X}_s = \{x_0, x_1\}$ and $\mathcal{X} \backslash \mathcal{X}_s = \{x_2\}$.

2.1.5 Spaces associated with CSPs

There exist two main spaces associated with a CSP: the search space and the solution space. The Cartesian product of the domains of all the variables compose the search space, whilst only those candidates that are certainly solutions (because they satisfy all the CSP constraints) compose the solution space. If the solution space of a CSP is empty, this CSP is unsatisfiable. Figure 2.1 left shows a hyper-polyhedron search

space with three variables. Once the constraints have been applied, the set of candidates that satisfy all the constraints are solutions for this CSP and they compose the solution space (see Figure 2.1 right). We would like to point out that the number of variables of a CSP fixes the dimensionality of the search and solution spaces. Thus, for a CSP with n variables, the associated spaces are n-dimensional.

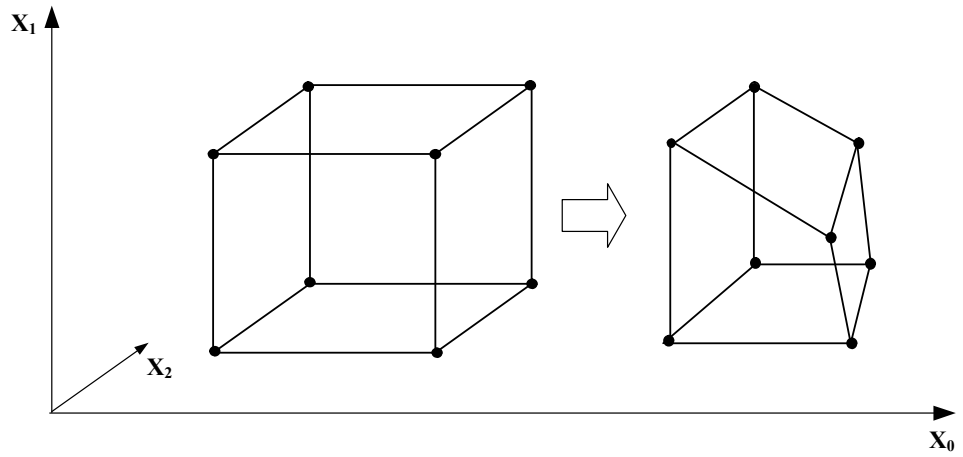


Figure 2.1: Hyper-polyhedron generated by the constraints of the CSP.

It is also of interest for this work to distinguish between convex and non-convex spaces corresponding to sets of candidates/solutions of CSPs.

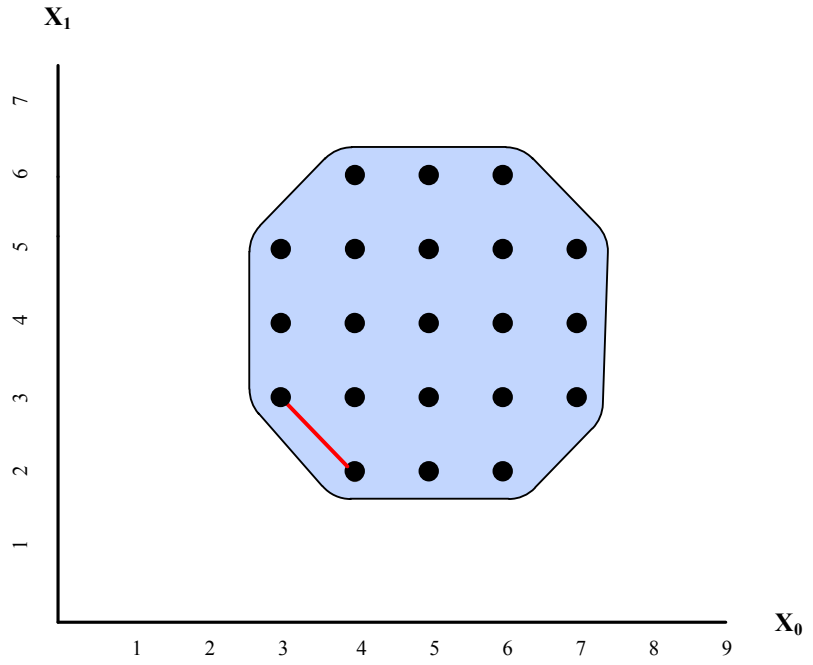
Definition 2.1.14. *A space is convex if for every pair of points within the object, every point on the straight line segment that joins them is also inside the space.*

Definition 2.1.15. *A space is non-convex if there exists a pair of points within the object, for which there exists a point on the straight line segment that joins them that is outside the space.*

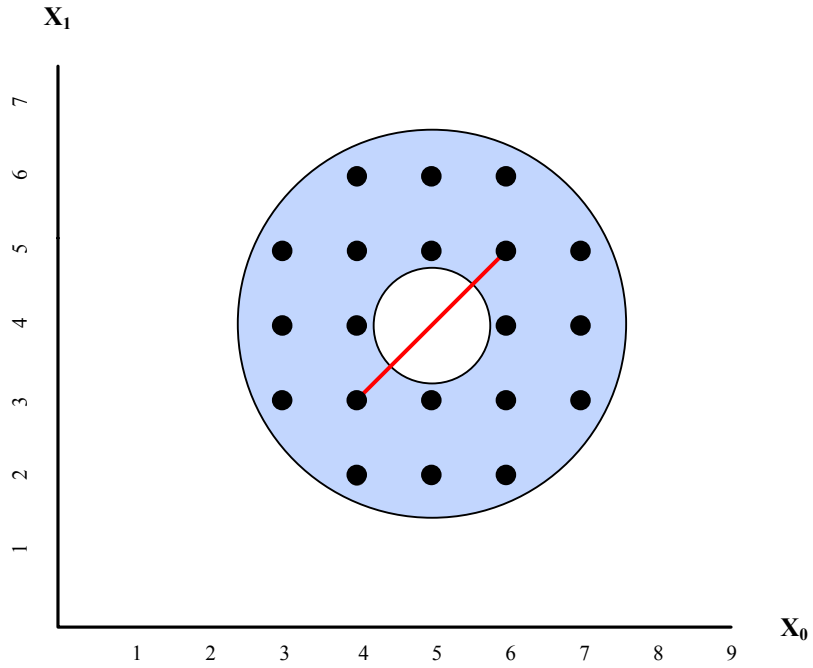
Figure 2.2 shows the two types of spaces for a 2-dimensional solution space: Figure 2.2(a) shows a convex space and Figure 2.2(b) shows a non-convex space. Note that in each figure there is a red line representing one of the cases that determines if the space is convex or non-convex.

2.2 Topological and Metric Spaces

In this section we introduce the concept of *topology* and *metric space*. They provide a formal framework allowing to define proximity relations between the elements, which



(a) Convex space.



(b) Non convex space.

Figure 2.2: 2-dimensional convex and non-convex spaces.

it is used in Chapter 3, Chapter 6 and Chapter 7. This is viable due to the possibility of expressing the CSP model as a topology. More information about topologies can be pursued in the book (William, 2006). The following definition of topology can be found in this book as well.

Definition 2.2.1. *Given a set of elements S, a topology on S is a collection Y of subsets of S, called the open sets (a subset of S is said to be open if it is in Y), such that:*

1. *Any union of open sets is open.*
2. *Any finite intersection of open sets is open.*
3. *Both S and the empty set $\emptyset$ are open.*

The set S with the topology Y is called the topological space and it is denoted as (S, Y). The spaces associated with the CSP models can be represented as topological spaces. In such case, the set S of elements could be the set of candidates solutions (search space), the set of solutions itself (solution space) or a reduced set of assignments (for instance the set of assignments that are feasible according to certain criterion), etc.

The topological space is the most general mathematical representation of a 'space'. However, the metric spaces are specializations of topological spaces that incorporate extra data structures and/or constraints. As the name suggests, the metric space is composed by a metric, which is a function that fixes the rules that describe what it means for elements of some space to be 'close to' or 'far away from' each other. There exist many distance functions that can be defined over each pair of elements x and y. The distance function, denoted as $d(x, y) : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}^+$, is a numerical description of how far away objects are between them according to a certain metric. Among all the distance functions, here we introduce the definitions of two of them: The Euclidean distance and the Chebyshev distance.

The Euclidean distance measures the square root of the squared differences along any coordinate dimension of two vectors (see Equation 2.4). The main characteristic of this distance metric is that distinguishes hyperspheres in n-dimensional spaces, which are analogous to circles for $n = 2$ and spheres for $n = 3$.

$$d(x, y)_{Euclidean} = \sqrt{\sum_{i=1}^{n} (\mathit{diff}(x_i, y_i))^2} \tag{2.4}$$

The Chebyshev distance measures the maximum absolute differences along any coordinate dimension of two vectors (see Equation 2.5). The main characteristic of this distance metric is that distinguishes hypercubes in n-dimensional spaces, which are analogous to squares for $n = 2$ and cubes for $n = 3$. In particular, the corners of a cube are at the same distance from the central point as the edges are, a feature not obtained with Euclidean distance metric.

$$d(x, y)_{Chebyshev} = \max_i (|x_i - y_i|) \tag{2.5}$$

The requirement for the applicability of the distance functions is that the elements have to be in the set of real ($\mathbb{R}$) numbers. A set of real numbers (and by extension this includes the natural ($\mathbb{N}$), integer ($\mathbb{Z}$) and rational numbers ($\mathbb{Q}$)) is totally ordered by definition.

Definition 2.2.2. *A set of elements S is totally ordered under $\leq$ iff given three elements of the set $\{a, b, c\} \in S$ they satisfy all the following properties:*

1. Antisymmetry*: If $a \leq b$ and $b \leq a$ then $a = b$.*
2. Transitivity*: If $a \leq b$ and $b \leq c$ then $a \leq c$.*
3. Totality*: $a \leq b$ or $b \leq a$.*

We recommend the book (Schröder, 2003) among others for more definitions and demonstrations. When we deal with other types of domain elements that are not in $\mathbb{R}$, as for instance, CSPs with symbolic domains, the distance functions cannot be directly applied, unless there exists an ordering relationship between their domain values. In this case, a monotonic mapping function $f(x) : \mathcal{D} \rightarrow \mathbb{R}$ has to be applied in order to map the elements of the CSP domain by preserving their order. Following, we explain a simple example.

Example 2.2.1. *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 $f(x) : \mathcal{D} \rightarrow \mathbb{N}$ that assigns greater values to the bigger clothing sizes can be defined. For example, $f(extra\ small) = 1$, $f(small) = 2$, $f(medium) = 3$, $f(large) = 4$, $f(extra\ large) = 5$, etc.*

As we have already pointed out before, a CSP with ordered domains, both numeric and symbolic (for the latter it is necessary to define the monotonic mapping function), can have a metric function defined over their set of elements S. Therefore, the spaces associated with this type of CSPs are metric spaces and the following definitions, as well as the distance definitions, can be applied to them.

Definition 2.2.3. *The neighbourhood N ($N \subseteq T$) of a valid tuple t is an open set containing all the valid tuples close to t.*

Definition 2.2.4. *A closed ball of a valid tuple $t \in T$ at distance ϵ, is the neighbourhood N of t composed of $\{y \in T : d(t, y) \leq \epsilon\}$, where d can be any distance function.*

2.3 Dynamic Constraint Satisfaction Problems

In this dissertation, we deal with problems that come from uncertain and dynamic environments and therefore they might evolve over time. The original and static model of a CSP only captures the state of a problem in a certain time instant, but it is not able to capture the evolution of the problem over time. For this reason, we use a variant of the static CSP model called Dynamic Constraint Satisfaction Problem, which is more appropriate for handling dynamic real-world problems.

Definition 2.3.1. *A Dynamic Constraint Satisfaction Problem (DynCSP) is a sequence of static CSPs $\langle CSP_{(0)}, CSP_{(1)}, \ldots, CSP_{(l)} \rangle$, each $CSP_{(i)}$ resulting from a change in $CSP_{(i-1)}$ and representing new facts about the dynamic environment being modeled (Dechter and Dechter, 1988).*

As a result of such an incremental change, the set of solutions of each $CSP_{(i)}$ can potentially decrease or increase. If the set of solutions of a static $CSP_{(i)}$ is smaller than the previous one, it is considered a restriction. If the set of solutions of a static $CSP_{(i)}$ is greater than the previous one, it is considered a relaxation.

In this dissertation, we focus our attention on DynCSPs that undergo restrictive changes, i.e., the original set of solutions decreases. We do not analyze DynCSPs in which there only exist relaxations of the CSPs since these changes cannot invalidate a previously found solution. As mentioned in Chapter 1, our technique is applied before the changes occur. Thus, it is applied to the original CSP ($CSP_{(0)}$) of the DynCSP. The main objective of our approaches and the rest of proactive strategies, is that the solution found for CSP_0 remain a solution for future CSP_i, where $i > 0$.

2.4 Constraint Satisfaction and Optimization Problems

Finding solutions with special characteristics (e.g., robustness and stability) consists in applying an optimization criterion. In the original CSP model, the main objective is to find a complete assignment of domain values that satisfies the set of constraints. However, among all the solutions that satisfy the constraints, we could be interested in finding some solutions that maximize/minimize certain criteria. In this case, the main objective is beyond satisfiability, since optimality is also important. Once again, the original CSP model is not able to capture the criterion to be optimized. Hence, we introduce the following definition of the Constraint Satisfaction and Optimization Problem (see the books (Ghédira and Dubuisson, 2013; Tsang, 1993)). We recall that $\mathcal{S}(\text{CSP})$ is the complete set of solutions of the CSP.

Definition 2.4.1. *The Constraint Satisfaction and Optimization Problem (CSOP) is an augmented model of the original CSP that introduces some objective functions. The objective is to maximize/minimize the set of functions $f(s)$ for $s \in \mathcal{S}$(CSP).*

Subsequently and throughout this dissertation, we use well known concepts of complexity theory (specifically, Non-determinstic Polynomial (NP) decision problems) that can be found, among others, in (Garey and Johnson, 1979). Searching for optimality tremendously increases the cost of solving a CSOP, being generally NP-hard. However, this can be alleviated by using Branch & Bound techniques (see Section 2.6.4) combined with Arc Consistency (AC) and heuristics. For instance, in (Larrosa et al., 1999) a $O(ed^3)$ AC algorithm for soft constraints framework is proposed (e is the number of constraints and d is the largest domain size).

2.4.1 Flexible CSPs and Valued CSPs

A type of CSOP for which there exists a single optimization function is called Flexible Constraint Satisfaction Problem (Flexible CSP). This model was developed because the original CSP model is unable to represent correctly some problems. In the original CSP model the solutions have to satisfy all the constraints (imperative) and the constraints have to be completely satisfied (inflexibility), which is the main characteristic of hard constraints. In contrast, for the Flexible CSPs these assumptions do not hold. This fact allows the existence of constraints that can be violated with certain weight/penalization, which are called soft constraints. The main objective is to maximize/minimize a function defined over this set of costs. The Flexible CSPs can model many real life problems for which there exists an optimization criterion or problems that are over-constrained (because the model allows constraint relaxation). In fact, the over-constrained problems can be seen as a problem with an optimization criterion of

finding a solution that maximizes the satisfiability of the CSP. In (Miguel, 2004) can be seen the integration of the DynCSP and Flexible CSP models.

In this dissertation we use a subset of the Flexible CSPs called Valued Constraint Satisfaction Problems (Valued CSP) (Schiex et al., 1995). The main differences of this model with respect to the original CSP model is that the constraints are local cost functions that express the level of satisfaction/unsatisfaction of the associated tuples. These costs are combined by an associative and commutative operator, which depends on the type of Valued CSP. The main objective is to find the solution that has the best combined cost. Following, the formal definition is introduced.

Definition 2.4.2. *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 $\langle \mathcal{X}, \mathcal{D}, \mathcal{C}, S, \varphi \rangle$, where $\varphi(C_i)$ is called the valuation of C_i (Schiex et al., 1995).*

2.4.2 Weighted CSPs

Among all the types of Valued CSPs (Probabilistic CSP, Possibilistic CSP, Fuzzy CSP, Weighted CSP, etc.), we use the latter formalism for modeling robustness of solutions for the enumeration-based techniques developed in this thesis. The main characteristic of the Weighted CSP (WCSP) is that associates weights (or costs) to the tuples of each constraint. Following, we present the formal definition.

Definition 2.4.3. *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 $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C}, S(U) \rangle$, where:*

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

- *$\mathcal{C}$ is the set of constraints expressed as cost functions ($C_i : \prod_{x_j \in var(C_i)} D_j \rightarrow \{0, 1, \ldots, U\}$).*

Assigning the maximum cost U to a tuple *t*, means that *t* is an invalid tuple for C_i. Otherwise (the cost assigned is lower than U) *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:

$$\mathcal{V}(t) = \bigoplus_{C_i \in \mathcal{C}, var(C_i) \subseteq X_t} C_i(t \downarrow_{var(C_i)}) \tag{2.6}$$

The tuple t is $consistent$ if $\mathcal{V}(t) < U$. The main objective of a WCSP is to find a complete assignment with the minimum cost. Note that the constraints of this model must be extensionally represented (see Section 2.1.3) because they assign a cost to each associated tuple.

Example 2.4.1. *We consider an example of WCSP named $E_2 = \langle \{x, y\}, \{\{ v_1, v_2\}, \{v_1, v_2\}\}, S(5), \{C_1, C_2\}\rangle$, which is represented in Figure 2.3. It can be observed that the set of costs is $\{0, \ldots, 5\}$. 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 2.3 (a) and the constraint C_2 is represented in Figure 2.3 (b).*

Table 2.1 shows the set of tuples of $E2$ with their corresponding costs assigned by the constraints and their $\mathcal{V}(t)$ values. In addition, it shows which tuples are solutions for E_2. The tuples $(x = v_1, y = v_2)$ and $(x = v_2, y = v_2)$ are not solutions of E_2 because 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 E_2. The best solution for E_2 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.

Furthermore, Appendix A.1 describes in detail the WCSP file format used in this thesis.

2.5 Inference

Inference is one of the most important matters in Constraint Programming. In this constraint-based framework, inferring conclusions from the constraints of the problem

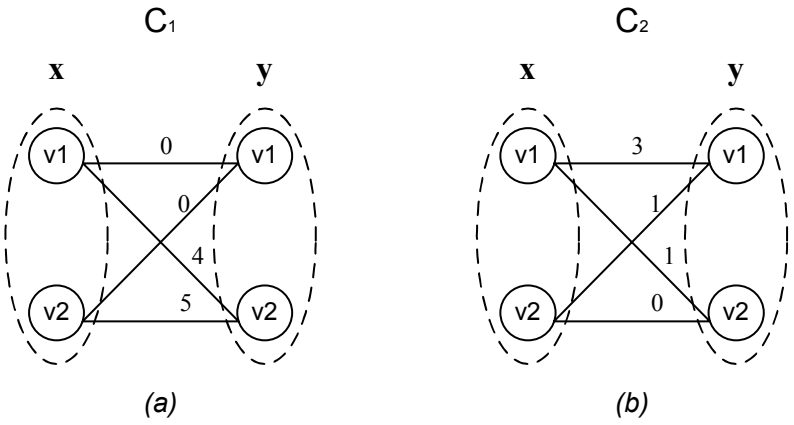


Figure 2.3: WCSP of the CSP of Example 2.4.1.

Table 2.1: Set of tuples of the WCSP of Example 2.4.1 and their 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

plays a fundamental role in the solving process. In this section we only give a general overview about this important and wide topic in the literature. Deeper explanations can be found in the books: (Rossi et al., 2006), (Lecoutre, 2013) and (Dechter, 2003).

The inferential processes for CSPs narrow the search space of possible partial solutions by deducing new constraints derived from those that are explicitly known about the problem. The result is a new CSP, equivalent to the original, which makes explicit the new restrictions contained implicitly in the original CSP. Enforcing-consistency techniques delete inconsistent values from the CSP domains by means of the inferred constraints, reducing (if possible) the set of possible solutions. In some cases, this procedure allows the detection of unfeasibility of a CSP. In case that the CSP has not been demonstrated to be unsatisfiable, search techniques can be applied with more efficiency after the consistency techniques.

2.5.1 Levels of Consistency

Consistency-enforcing processes are related to the level of consistency. A *globally* inferential process would deduce completely all the information of the CSP. These methods are called k-*consistency* algorithms, where k is equal to the number of variables of the CSP. However, in general, the cost of these complete inference process is too high. Lower k-consistency methods are used more often. Generally, these algorithms ensure that partial feasible assignments composed by $k-1$ variables are consistent with any other variable of the CSP. According to the level of consistency, these inference techniques are also called: *node-consistency*, *arc-consistency*, *path-consistency* and *k-consistency*. Subsequently we explain the two levels of consistency used in this work: node-consistency and arc-consistency.

Definition 2.5.1. *Node-consistency is the most simple consistency process and it ensures that all the domain values of a variable satisfy all unary constraints related to such variable. A CSP is called node-consistent iff all its variables are node-consistent:*

$$\forall x_i \in \mathcal{X}, \forall C_i \in \mathcal{C}, \exists a \in D(x_i) : a \textit{ satisfies } C_i.$$

Definition 2.5.2. *Arc-consistency (Mackworth, 1977a) is one of the most known and used consistency procedures. A CSP is arc-consistent iff for any pair of constrained variables x_i and x_j, for each a value in D_i there exists at least one value b in D_j such that the partial assignment $(x_i = a, x_j = b)$ satisfies all the constraints related to both x_i and x_j. Any value in the domain of a variable which is not arc-consistent*

can be eliminated as they can not be part of any solution. The domain of a variable is arc-consistent iff all values are arc-consistent. Thus, a problem is arc-consistent iff all its arcs are arc-consistent:

$$\forall\ C_{ij} \in \mathcal{C}, \forall a \in D(x_i), \exists b \in D(x_j)\text{: } a \text{ and } b \text{ satisfy } C_{ij}.$$

The inference processes are also applied when there exists a partial assignment. In this case, consistency checking of values in the domains is only developed to those values that are consistent with the existent partial feasible assignment. In this dissertation, we denote $D_s(x) \subseteq D(x)$ to the subset of domain values of the variable x that are consistent with s, where s is a feasible assignment. Following, we present an example of the concepts and notations explained above.

Example 2.5.1. *According to the CSP E_1 presented in Example 2.1.1, we consider the partial feasible assignment $s_1 = (x_1 = 2, x_2 = 1)$. Note that the missing variable in this assignment is x_0 and its original domain is $D(x_0) = \{1, 2, 3, 4, 5\}$. After checking the consistency of these values with the partial assignment s_1, the values $(1, 2)$ can be removed because they do not satisfy the constraint $C_0 = \{x_0 - x_1 > 0\}$. Thus, $D_{s_1}(x_0) = \{3, 4, 5\}$.*

2.5.2 Generalized Arc Consistency

Initially, the inference processes were defined over binary constraints. As progresses in the CP field increase, inference has been extended to non-binary constraints. Subsequently, it is defined this consistency term, called Generalized Arc Consistency (GAC) (Mohr and Henderson, 1986; Bessiere, 2006).

Definition 2.5.3. *A CSP is GAC iff all its constraints are GAC. A constraint C_i is GAC iff each variable in its scope $(var(C_i))$ is GAC with respect to C_i. A variable x_i is GAC with respect to a constraint C_i iff for every value $a \in D(x_i)$ there exists a tuple that involves a and satisfies C_i.*

Following, we introduce the enforcing GAC3 algorithm, which is an extension of the well-known AC3 consistency algorithm (Mackworth, 1977b). GAC3 is used in Chapter 6 and Chapter 7.

Algorithm 2.1: GAC3: Global Arc Consistency

Data: $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$
Result: Is P GAC consistent?

```
1 Q ← {(x, c), ∀c ∈ C, ∀x ∈ var(c)} // var(c) is the scope of c
2 while Q ≠ ∅ do
3     (x, c) ← takeElement(Q);
4     seekS ← seekSupport(x, c); // Found support for all D(x) for c?
5     if D(x) = ∅ then
6         return false;
7     if seekS = false then
8         Q ← Q ∪ {(y, c'), ∀c' ∈ C ∧ c' ≠ c ∧ ∀x, y ∈ var(c') ∧ x ≠ y}
9 return true;
```

First, Algorithm 2.1, incorporates a queue of the combinations of all the constraints and their scopes. Recall that $var(c)$ is the scope of $c \in \mathcal{C}$. The `seekSupport` function of GAC3 searches for a support of each domain value. The `seekSupport` function searches for a tuple that satisfies the constraint c by checking each value in the domain of the variable x (this tuple is also called support). If it does not find any support, it deletes the analyzed value from the domain of the variable ($D(x)$) because it can not be part of a solution of the CSP. If any value is deleted because there does not exist any consistent support with respect to the partial assignment, `seekSupport` function returns `false`. In this case, Algorithm 2.1 adds to the queue all the variables related to the analyzed variable and their corresponding constraints. Finally, Algorithm 2.1 returns `false` if the domain of any of the variables analyzed is empty, since this fact means that there does not exist any value in the domain of such value that is GAC and therefore it is not possible to enforce GAC in this CSP. This means also that this CSP is unsatisfiable. On the contrary, if there are no empty domains, it returns `true`.

Depending on the parameters of the CSPs, computing GAC might be very costly. However, in some cases it is possible to reduce this cost by avoiding the consistency checking of each domain value. This weaker consistency notion that is called *bounds consistency* arises from the necessity of reducing the consistency process cost for problems whose domains are large and convex. The main idea is to be sure that there exist an upper and lower bound values of a domain that are consistent. In case that an extreme value is not consistent, it is deleted from the domain and its neighbour is checked until bounds consistency is achieved or such domain is empty. This weaker consistency notion implies a highly cost reduction, specially for CSPs with large domains.

2.6 Hybrid Methods and Efficiency Improvements

As previously stated, by reasoning about the problem with the use of inference approaches, we can narrow the search space and therefore facilitate the resolution of the problem. However, after obtaining a reduced search space, it is necessary to search for the possible assignments until finding a solution or until we are able to demonstrate that there does not exist any solution. Thus, search strategies are as important (or even more) than inference approaches in Constraint Programming. Moreover, the combination of both, search strategies and inference has resulted in hybrid methods. They are some of the most used and efficient algorithms in this field.

There exists a vast variety and types of search strategies, which can be classified in local search and systematic search strategies. Local search algorithms apply local changes in a certain way to the current partial assignment, until a solution is found or a time/repetitions cutoff is reached. That is why they are also classified as 'anytime' algorithms, since they run until the fixed time is over or a number of repetitions is reached. Local search algorithms are *incomplete algorithms* because they do not guarantee that they will eventually either find a solution or state that the problem is unsatisfable.

Nevertheless, systematic search strategies are *complete algorithms* because they systematically explore the search space until finding a solution or declaring that the problem is unsatisfiable. Generally, the cost of solving a CSP is NP-complete. In this type of strategies, the search space is explored as a tree. Each branch of the tree represents different combinations of value assignments and is composed of several nodes. Each node represents a subproblem composed by a partial assignment. The height of the tree is fixed by the number of variables of the CSP. When all the variables have been assigned with the corresponding consistency check, the current state in the tree is a leaf-node and this assignment is complete and feasible, that is to say, it is a solution of the CSP.

Due to search algorithms are complete strategies, we have used a technique of this type for some of the approaches developed in this thesis. We selected the general search strategy called Maintaining Arc Consistency Algorithm (MAC), which is explained below. Furthermore, in this section, we also explain several methods that improve the efficiency of search strategies. Specifically, we will use and explain restarting techniques and variable ordering heuristics. Finally, in this section, we will discuss how to cope with not only satisfiability but also with optimality by means of the Branch and Bound ($B\&B$) technique.

2.6.1 Maintaining Arc Consistency

The MAC algorithm (Sabin and Freuder, 1994) has been vastly used in the literature. In fact, most of the current CSP solvers use or are based on MAC. MAC algorithm is categorized into the *look-Ahead* algorithms, which are characterized by assigning

tentative (provisional) values from the variable domains to a current partial assignment and checking the consistency in each step. When a tentative value is inconsistent, it is deleted and a new value from the current domain is assigned and checked as a tentative value. If the current assignment is not consistent and there are no more values in the domain of the tentative variable to check, these look-Ahead algorithms backtrack. This consists in removing the last inconsistent assignments done. In addition, it is necessary to save the domain of the variables (line 7) in order to restore them (line 15) when a backtrack occurs. Note that constraint propagations can delete values of unassigned variables, so it is mandatory to restore them after a backtrack. One of the most basic algorithms in this category has the same name of the latter process mentioned: Backtracking algorithm.

The MAC algorithm follows the same general pattern as the Backtracking algorithm. However it incorporates the innovation of combining search with inference, because it checks the consistency by applying arc consistency process after each tentative value of the current assignment is selected. Furthermore, MAC algorithm is also extended to non-binary constraints by using the global arc consistency. Following, we explain the Maintaining GAC3 (MGAC3) algorithm (see Algorithm 2.2), which assigns to each variable $x \in \mathcal{X}$ a new value $v \in D(x)$, until the value selected is GAC3 with respect to s, where s is the current partial assignment. Furthermore, Algorithm 2.2 is also responsible for updating the set of assigned variables $\mathcal{X}_s$ and the partial assignment s, which is initially (first call of the MGAC3 algorithm) equal to the empty set. Furthermore, it stores the domains and set of neighbours of all the variables before making an assignment. Note that after a variable x is assigned, $D(x)$ contains a single value that is the value assigned to x. If Algorithm 2.1 (GAC3) returns `false`, then Algorithm 2.2 (MGAC3) carries out the backtracking process and also restores the domains and set of neighbours of all the variables. If all the variables of the CSP have been satisfactorily assigned, the algorithm returns `true`, and the solution found is stored in s.

2.6.2 Variable Ordering Heuristics

In general, heuristics use the data that can be extracted from previous search states to better guide the following search. The variable ordering changes might have an extreme impact over the search space, and as consequence, in the computational time required in the search. There exist two types of variable ordering: static and dynamic. The static variable ordering is fixed before the search algorithm is applied. However, the dynamic variable ordering is fixed while the search algorithm is running. Generally, the dynamic variable ordering produces more effective results. One of the most used dynamic heuristics is known as 'fail-first' and its main objective is to select the next unassigned variable with the highest probability of restricting the most of the remaining search space.

A very successful 'fail-first' heuristic is named $dom/wdeg$ heuristic variable se-

Algorithm 2.2: MGAC3: Maintaining Global Arc Consistency

Data: $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle, \mathcal{X}_s, s$
Result: Is there a solution for P?

```
1  if s = ∅ then
2      X_s ← ∅; // Set of variables assigned
3  else if X_s = X then
4      return true
5  select x ∈ X\X_s;
6  X_s ← X_s ∪ x;
7  save D;
8  while D(x) ≠ ∅ do
9      select v ∈ D(x);
10     s ← s ∪ {x = v};
11     D(x) ← v;
12     if GAC3(P = ⟨X, D, C⟩) then
13         if MGAC3(P⟨X, D, C⟩, X_s, s) then
14             return true
15     restore D\D(x);
16     s ← s\{x = v};
17 X_s ← X_s\x;
18 return false
```

lection (Boussemart et al., 2004). This heuristic combines the information about the domain with the weighted degree heuristic ($wdeg$). The weighted degree of a variable is calculated as the sum of the weights of the constraints associated with this variable. The weight of a constraint is increased in one unit when the consistency checking produces a dead-end. A desirable objective in the 'fail-first' approach is to select a variable with a small domain because it maximizes the changes of a future fail (there are less possible values to try). The other objective is to find a variable that maximizes its weighted degree because if in past assignments it was failing frequently, the likelihood that it will fail in the future is also high. The combination of these two objectives resulted in a heuristic that selects the variable that minimizes the ratio of domain size to weighted degree ($dom/wdeg$), which is a very powerful heuristic.

When there does not exist an established variable ordering heuristic, it is common to use the lexicographical order, which is the alphabetical/numerical order of their component letters/numbers. The same happens for the value ordering heuristics (order of value selection in the search tree), unless there is one specified, by default the lexicographical order is used.

2.6.3 Restarting Techniques

As we have mentioned throughout this chapter, the computational time of solving CSPs of a significant size is an important issue to take into account. One of the reasons is the *thrashing* effect, that is produced when some unfavourable branches of the search tree are systematically checked. In order to solve this problem, approaches exist that restart the search back to the root node, thus potentially avoiding getting stuck in these unpromising regions of the tree search.

The $dom/wdeg$ heuristic (see Section 2.6.2) is also appropriate for avoiding the thrashing because it selects variables that are likely to produce inconsistencies in the first positions of the search tree. Nevertheless, this heuristic is very little informed in the initial variable selections (in fact, it does not have any information at the root level), while these first choices are crucial to avoiding thrashing. Hence, restarting approaches represent such an important advantage, because they are able to re-make choices over the early areas of the search tree and this fact can ward off the direction of the search in unfavourable branches of the search tree. Thus, a restarting approach can be combined with a weighted variable ordering heuristic (for instance, $dom/wdeg$) in the way that after a restart occurs, the weights associated with the constraints are kept. The availability of such information at the beginning of the search tree, can allow a reduction of the repetition of fails in the search.

These approaches restart the search from scratch each time that the number of failures (nbF) reaches a predetermined cutoff value (C). Among the restarting approaches existent in the literature, we have used the geometric restarting approach presented in (Walsh, 1999). This approach has been proved to be a very efficient restarting approach for a wide range of problems (Grimes, 2012). Particularly, the

geometric restart approach increases the number of fails cutoff geometrically according to a multiplicative factor (m), generating the sequence: $(1, m, m^2, m^3, \dots)$. In (Walsh, 1999), the authors determined that the best results for the geometric restart approach are for the values of $m \in (1, 2)$. Subsequently, in (Wu and Van Beek, 2007), a scale factor ($scale$) is introduced for multiplying the sequence generated by the geometric restart approach.

Algorithm 2.3 incorporates this last modification of the geometric restarting approach. Thus, it calls repeatedly to a search algorithm by providing the corresponding cutoff (C) of this iteration as a parameter. The systematic search algorithm described in Algorithm 2.4 is generic, so it could be replaced by any other search algorithm (e.g., Algorithm 2.2). Nevertheless, some modifications have to be included: (i) a variable for counting the number of failures (nbF) and (ii) a stop criterion when the number of failures is equal to the cutoff. Thus, Algorithm 2.4 updates the number of failures that have been produced from the consistency check. When the cutoff is reached, Algorithm 2.4 returns `false`. Note that it returns the same value if all the search tree has been explored and there does not exist any solution. Algorithm 2.3 is responsible of increasing the cutoff until the CSP has been proved to be unsatisfiable or until a solution is found.

Algorithm 2.3: Geometric Restarting

Data: $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle, scale, m$
Result: $s, \mathcal{N}_k, lb$

```
1  s ← ∅
2  i ← 1;
3  repeat
4  |  if nbF < C then
5  |  |  return false
6  |  C ← scale * m^i; //number of fails cutoff
7  |  i ← i + 1;
8  until not Search Algorithm (P = ⟨X, D, C⟩, s, nbF, C);
9  return true
```

2.6.4 Branch & Bound

In Section 2.4, we introduced the constraint satisfaction and optimization framework and the CSOP model. As mentioned, this model extends the original CSP model by adding at least one objective function. The main objective is to find a solution that is the optimal one according to the stabilized criterion. The main difference that optimization algorithms present over the purely satisfaction algorithms is that finding one solution is not enough since the optimal function value is unknown. Thus, it

Algorithm 2.4: Generic Search Algorithm

Data: $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle, \mathcal{X}_s, s, nbF, C$
Result: Is there a solution for P? $\vee$ cutoff reached

```
if s = ∅ then
    X_s ← ∅; // Set of variables assigned
else if X_s = X then
    return true
select x ∈ X\X_s;
X_s ← X_s ∪ x;
save D;
while D(x) ≠ ∅ ∧ nbF < C do
    select v ∈ D(x); s ← s ∪ {x = v};
    D(x) ← v;
    if Consistency (P = ⟨X, D, C⟩) then
        if Search Algorithm (P⟨X, D, C⟩, X_s, s, nbF, C) then
            return true
    nbF ← nbF + 1; // number of failures
    restore D\D(x);
    s ← s\{x = v};
X_s ← X_s\x;
return false
```

could occur that there exist other solutions that are better than the current solution obtained. This fact increases tremendously the cost of solving a CSOP (see Section 2.4). Nevertheless, the Branch & Bound is an 'anytime' algorithm, so we can limit the search time by fixing a time cutoff.

The Branch & Bound algorithm (see Algorithm 2.5) is a well known algorithm for solving optimization problems. The algorithm uses an inference process and prunes the branches whose objective function value is lower or equal to the maximum function value obtained, denoted as lb (lower bound). The consistency procedure can be any of the inference processes explained in Section 2.5, for instance GAC3 (see Algorithm 2.1) or other inference processes from the literature. If the tentative value analyzed is not consistent with the current partial assignment or the function value of the partial assignment is lower than the lower bound, the value is deleted from the domain and this branch of the search tree is rejected, reducing then the search space. Note that the Branch & Bound algorithm bounds the search tree in the way that branches of the tree with lower objective function values, are rejected from the search (as soon as this information is available).

Algorithm 2.5 is initially called with an empty assignment s. In the first execution of B&B some necessary structures are initialized, such as the set of variables assigned $\mathcal{X}_s$ and the lower bound lb. In followings executions of B&B these parameters, as well as the assignment s are updated. In the same manner than MAC algorithm saves the domain of the variables in order to restore them in case that a backtrack occurs, the same storage is done in B&B algorithm. Algorithm 2.5 stops when all the branches have been explored or pruned, providing the solution s (if there exists) with the maximum $f(s)$, where $f(s)$ represents the objective function. We can also limit the search time and therefore the quality of the best solution found by fixing a time cutoff. However, the more time the Branch & Bound algorithm spends searching, the better function values we might obtain.

Algorithm 2.5: B&B: Branch & Bound

Data: $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, $\mathcal{X}_s$, s, f, lb, *time_cutoff* (optional)
Result: lb (maximum $f(s)$), s_{best} (best solution)

```
if s = ∅ then
    X_s ← ∅; // Set of variables assigned
    lb ← −1; // Maximum current function value
    s_best = ∅ ; // Best current solution
select x ∈ X\X_s;
X_s ← X_s ∪ x;
save D;
while D(x) ≠ ∅∧ time_cutoff not reached do
    select v ∈ D(x); s ← s ∪ {x = v};
    D(x) ← v;
    if Consistency (P = ⟨X, D, C⟩) ∧ f(s) > lb then
        if X_s = X then // New solution found
            lb ← f(s);
            s_best = s;
        else
            B&B(P = ⟨X, D, C⟩, X_s, s, f, lb, time_cutoff);
    restore D\D(x);
    s ← s\{x = v};
X_s ← X_s\x;
return lb
```

Chapter 3

New Dynamism Framework for CSPs with Ordered Domains

In this chapter we focus on the dynamism properties of CSPs that model problems whose elements are related by a certain order relationship. Thus, according to the characteristics of this type of problems, we can extract some assumptions about the future possible changes that they can undergo. This is specially useful when we deal with difficult situations for which there does not exist additional detailed information about the dynamism of the problem.

Firstly, we explain the main characteristics associated with the type of problems that we analyze and the assumptions about a certain type of dynamism inherent to them, and the motivation that has driven us to make these assumptions. Furthermore, we compare other dynamism assumptions made in the literature. In addition, we specialize the two features associated with the solutions found by proactive approaches (robustness and stability) to this dynamism environment. Subsequently, we explain how we tackle these type of problems in order to find solutions that can offer a resistance to this dynamism. Specifically, we present two theoretical approaches: one of them is only addressed to CSPs with convex solution spaces whilst the other one does not have this limitation. Finally, we make a detailed comparison between the two types of technical approaches presented in this thesis for dealing with this dynamism: enumeration-based techniques and a search algorithm. For both of them, we analyze their weakness and strengths, as well as their applicability area.

3.1 Dynamism in CSPs with Ordered Domains

As stated in Section 1.5, the level of knowledge available about the uncertain and dynamic environment where the problem comes from plays a fundamental role in the search of robust and/or stable solutions. Real life problems for which there does not exist detailed information about their dynamism, represent difficult situations to deal

with. Many proactive approaches are useless in such problematic situations in which this information is missing or scarce. This is the main motivation of this dissertation, to develop approaches that can provide with certain level of robustness to the solutions, when there does not exist extra data about the future possible changes. That is, when the only information available is the one required for modeling the CSP.

Nevertheless, there exist some types of real life problems that due to their particular structure can provide us more specific information about their possible dynamism. For CSPs that model problems for which the order over its elements is significant, limited (and intuitively reasonable) assumptions about their possible future changes can be made. These assumptions are based on a type of change that these kind of problems can undergo, which takes the form of restrictions/relaxations at the borders of a domain or constraint. These modifications are equivalent to a restriction/modification of certain parts of the bounds of the solution space, because the domains and constraints of the CSP determine the solution space of the CSP. This is illustrated by the following example.

Example 3.1.1. *Figure 3.1 shows a solution space of a CSP (represented with the blue and continuous line), which is composed of two variables* x_0 *and* x_1*. It can be observed that it has* 29 *solutions (black points).*

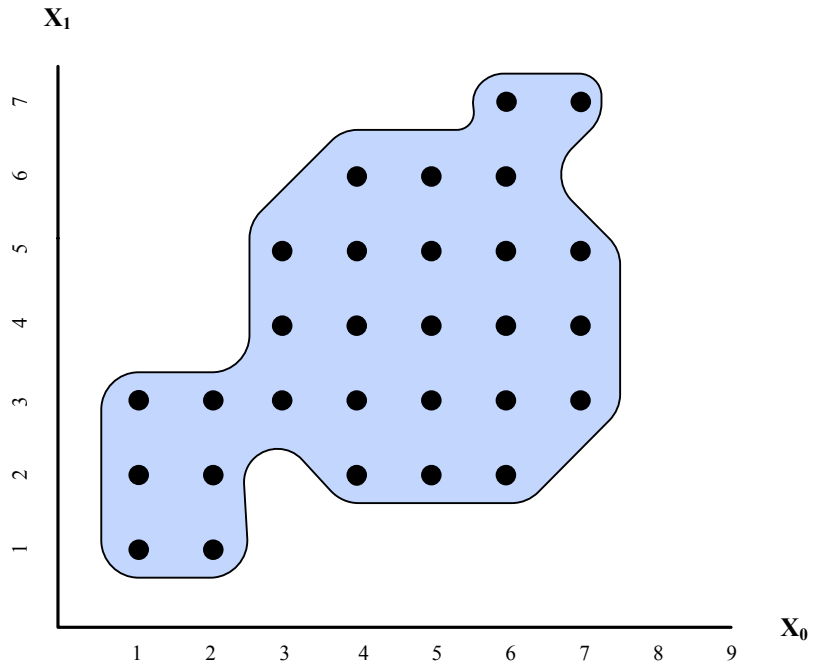


Figure 3.1: Solution space of Example 3.1.1.

If no specific information is given about the dynamics of Example 3.1.1, i.e., the kinds and likelihoods of changes in the problem, it is not easy to decide which solution is considered the most robust. In Figure 3.2, a smaller solution space coloured with a darker blue hue and discontinuous lines is represented. This reduced space is the result of restrictive modifications over the original constraints of the CSP (original solution space that is coloured with light blue).

In this situation we assume that the original bounds of the solution space (delimited by constraints and domains) undergo modifications in the form of range reductions or expansions (this is motivated below with real life problems examples). Note that the possibility of solution loss only exists when changes are restrictive. For this reason, we only focus on restrictive modifications, since our main concern is to find solutions that maximize the resistance to these future changes.

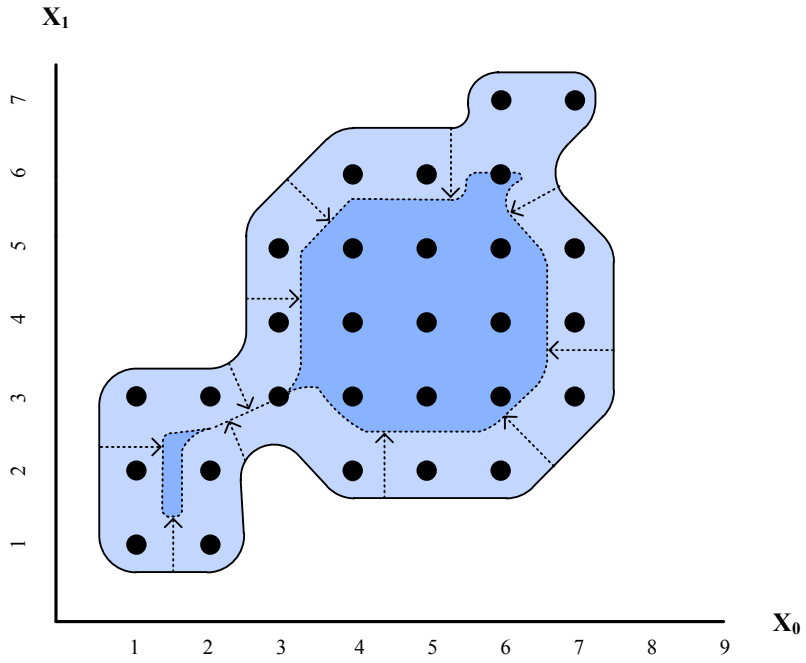


Figure 3.2: Restrictions over the bounds of the solution space of Example 3.1.1.

Situations similar to those depicted in Figure 3.1 often occur in a wide variety of applications (see (Climent et al., 2013b)). One of the most obvious example is the scheduling problem, where the most common form of change is time delays. For instance, we recall the scheduling Example 1.1.1, in which the first task of the scheduling problem has an end time of 3. All the subsequent tasks in the same job must have a start time in the interval $[3, max]$ (see Figure 1.1). In this case, if there is a delay of 1 time unit in the aforesaid task (see Figure 1.2), then the new domains of the subsequent tasks must be reduced to be $[4, max]$ (see Figure 1.4).

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 CSP constraints.

Temporal reasoning-based problems (Dechter et al., 1991) also support this assumption. These real life problems consist on the reasoning of relationships between time and actions/events. This occurs in many contexts: natural language understanding, simulation, diagnosis, scheduling, planning, etc. In temporal problems of this sort, dynamism and uncertainty are almost inherent. Here, modifications are most likely to occur at the bounds of the solution space given that the relationships involve time.

The aforementioned assumptions regarding the character of changes are also supported by spatial and geometric reasoning problems. In such real life problems, measurement errors can result in a partially incorrect representation of the problem, where bounds must be subsequently adjusted. These problems also often include distance constraints, and these are generally modified by relaxing or restricting the constraints in boundaries.

The condition mentioned above also occurs in 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, where constraints represent the properties that these elements must satisfy. For these problems, resources and conditions of the environment may undergo changes. For instance, an example introduced in (Sam, 1995) involves a design problem of damping floor vibrations by means of beams and conduits traversing them. Consider a variable representing the number of conduits to introduce in the beams, with a maximum of 10 conduits. In this case, it is more likely that this maximum value will increase or decrease (modifications of the domain border) than a certain intermediate domain value, let us say 5 conduits, will become invalid.

The dynamism associated with all the real life examples with ordered domains mentioned above, motivate the interest and analysis carried out in this dissertation for dealing with such ordered problems and dynamism framework. Therefore, the rest of the presented work is based on the following assumption.

Assumption *The original bounds of the solution space of CSPs that model problems for which the order over its elements is significant and that do not have extra detailed data about future changes, may undergo modifications in the form of range reductions in the future.*

3.2 Related Dynamism Assumptions in the Literature

In some real life problems that come from uncertain and dynamic environments, sometimes (generally rarely) there exists extra detailed information associated with the problems that estimates the kind of changes and how significant they are. In Section 1.4 we explained several techniques that assume different additional knowledge about the uncertain and dynamic environment. Some of them focus on the changes over the variables and/or their values and also there is a model that focus on the constraints: the Probabilistic CSP (Fargier and Lang, 1993). This model considers that each constraint is associated with a probability of existence. Thus, the most robust solution is the solution that maximizes the probability of satisfying all the constraints.

The main differences in the above new dynamism assumption proposed in this thesis with respect to the Probabilistic CSP, is that we consider variations over the original constraints while the Probabilistic CSP only considers the existence of the constraints and that we also consider changes over the domain constraints of the CSP while the Probabilistic CSP does not. Regarding the extra information that is known about the future changes, with the Probabilistic CSP, as well as another probabilistic approaches, the specific probabilities of the future dynamism are required. However, only one of the enumeration-based techniques developed in this thesis requires such information. The other techniques introduced in this thesis are non-probabilistic approaches and therefore, they can be applied when the probabilities of the dynamism are unknown.

There exists another non-probabilistic proactive approach that does not require extra data about the dynamism: super-solutions (Hebrard, 2006). In this work, the author assumes that the possible future changes are the loss of a values in the solution. Since this type of change implies the loss of the original solution, the feature of robustness does not hold in this type of change. However, the stability feature holds because it measures the ability of a solution of being repaired when its feasibility is broken after the changes invalidate it. Even if the assumption that any value of the solution could be lost with equal probability is reasonable for CSPs with non-ordered domains, it is not always true for CSPs with ordered domains. Section 3.1 motivates a type of dynamism that takes the form of bounds restrictions. In such a framework, it is possible to search for robust solutions, since values that are far from the bounds have a higher likelihood of remaining valid (see Section 3.4.2.1). Thus, it is possible to distinguish different level of robustness between them. Nevertheless, in the type of change assumed in (Hebrard, 2006), there is a uniform likelihood of becoming invalid distribution over all the values. For this reason, it is not possible to distinguish different levels of robustness between the values of these non-ordered CSPs and it is only possible to establish stability levels.

Since we strongly value solution loss prevention in uncertain and dynamic environments, our main goal is the search of robust solutions in the framework of CSPs that model problems for whose elements have an order relationship, even if we also

value their stability. Even if both features, robustness and stability are different, we found that they are related in this dynamism framework associated with CSPs with ordered domains (see Section 3.4.2). For this reason, and due to the fact that the approaches developed in (Hebrard, 2006) do not consider extra detailed information, we have evaluated our non-probabilistic approaches with these approaches in Chapter 9.

3.3 Specializing Robustness and Stability Concepts

In this section we specialize the original definitions of solution robustness (Definition 1.4.3) and solution stability (Definition 1.4.4) (Climent et al., 2014) to consider the assumption made in Section 3.1.

3.3.1 Robustness

Larger restrictions always include (some) smaller ones and therefore, a larger restriction involving a tuple t, must also include any tuple closer to the given border than t. Thus, we can also assume that values affected by larger restrictions are, in general, less likely to be removed. The latter is shown in Figure 3.3, where the different blue hues spaces show restrictions of different magnitude of change over the original solution space of Example 3.1.1 (represented in Figure 3.1). Thus, the solution in the smallest and darkest blue area, which only can be removed by the largest restrictions (see the longest arrows in Figure 3.3), is more likely to remain a solution. The solutions in the intermediate blue hue space are more likely to be removed because restrictions of lower magnitude can also invalidate them (see the shortest arrows in Figure 3.3). However, they are less likely to be removed than the solutions located on the borders of the solution space (lightest blue area) because they can become invalid when faced with restrictions on the border of any magnitude of change (short, intermediate, large, etc.).

In Figure 3.3 we assumed that all the bounds of the solution space are dynamic (general case). However, there are occasions in which due to the nature of the problems, it is known that some of the bounds are static. This type of bound never will undergo any restrictive modification. Throughout this thesis, we have used static bounds mainly for introducing simpler and clearer examples.

Given the assumptions made in this chapter and that there does not exist extra data about the dynamism, we specialize the Definition 1.4.3 for this framework as follows.

Definition 3.3.1. *The most robust solution of a CSP with ordered domains without extra detailed dynamism data is the solution that maximizes the distance from all the dynamic bounds of the solution space.*

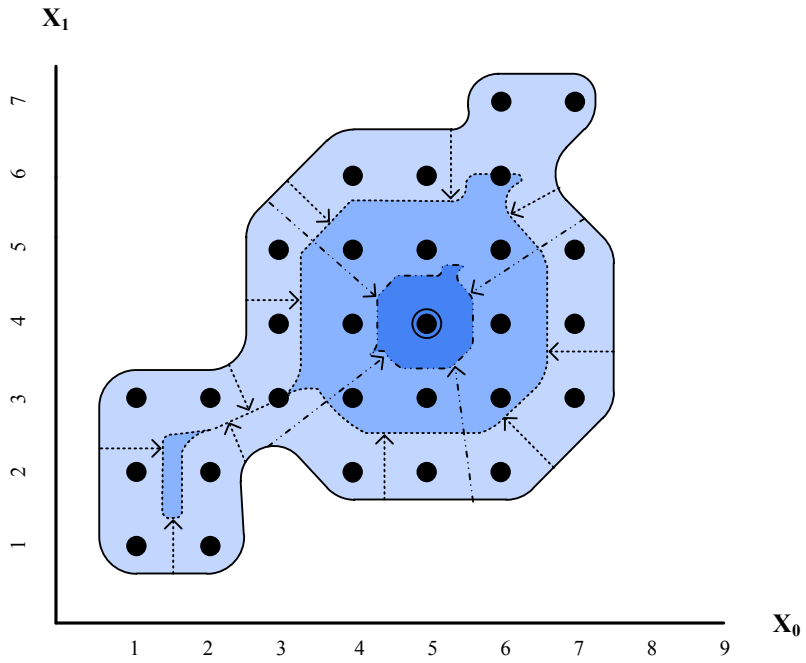


Figure 3.3: Most robust solution of the CSP of Example 3.1.1.

Consequently, our main goal is to search for these solutions in both convex and non-convex spaces. For the Example 3.1.1, which is a 2-dimensional non-convex solution space, the solution $(x_0 = 5, x_1 = 4)$ is the most robust (see the solution highlighted in Figure 3.3). 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 solution spaces.

Many real life problems involve non-linear constraints, for instance, design problems (Sam, 1995). For this reason, in this dissertation, we have not only developed a specific approach for Linear CSPs (we recall that this type of CSP always produces convex solution spaces), but also other approaches that can be applied to both convex and non-convex spaces.

Furthermore, even if the main focus of this thesis is the search of robust solutions that have a high likelihood of remaining valid when faced with changes in any border of the solution space, we have also developed an approach that considers differences in such changes. In the latter case, Definition 3.3.1 does not hold since there are some bounds that can have a higher likelihood of undergoing restrictive modifications, or some of the bounds can undergo bigger/smaller restrictions. In such case, the desirable objective is to find a solution that maximizes the criterion of being far from bounds with higher likelihood of change, according to their magnitude of change. In the rest of this chapter we will assume that dynamism data is unknown (since this situation is more difficult), however, more information about informed dynamism situations can

be found in Chapter 4.

3.3.2 Stability

The definition of stable solution (see Definition 1.4.4) can be more accurate for CSPs with ordered domains. This is due to the fact that it is possible to define a more specific notion of 'closeness' between two solutions due to the given order over the domain values. In (Hebrard, 2006) the level of dissimilarity between two solutions is measured by counting the number of variables that take different values in both solutions (see Section 1.4.3). This is calculated with the Hamming distance.

$$d(s1, s2)_{Hamming} = \sum_{i=1}^{n} (s1_i \neq s2_i) \tag{3.1}$$

For this equation the dissimilarity condition $(s1_i \neq s2_i) \in \{0, 1\}$. In a following work developed by (Hebrard et al., 2007), the authors deal with a different issue, however they consider another similarity measure between solutions of CSPs: the Manhattan distance. This distance measures the sum of the absolute difference of values associated with each variable of two solutions. Note that unlike Hamming distance, Manhattan distance requires an order relation defined over the elements. Otherwise, it would not be possible to calculate the absolute difference of the values.

$$d(s1, s2)_{Manhattan} = \sum_{i=1}^{n} |s1_i - s2_i| \tag{3.2}$$

In this section we extend the stability concept associated with solutions of CSPs that model problems for which there exists a significant order relation between their elements. Due to this characteristic, it is possible to go beyond the Hamming distance and to apply the Manhattan distance to the notion of stable solutions for CSPs of this type. Subsequently, we define the stability of the solutions in such framework.

Definition 3.3.2. *Given an order relationship over the values of a set of solutions, a solution $s1$ is more stable than another solution $s2$ iff, in the event of a change that invalidates them, there exists an alternative solution to $s1$ with lower Manhattan distance than the Manhattan distance of any alternative solution to $s2$.*

Furthermore, we present an extension of Definition 1.4.5 for CSPs with ordered domains by fixing a maximum Manhattan distance between the original solution and the repaired solution, which is called c.

Definition 3.3.3. *A solution is an (a, b, c)-super-solution if the loss of values of a variables at most, can be repaired by assigning other values whose Manhattan distance with respect to the original values is lower or equal to c, and this involves changing the values of b variables at most.*

The above definition also holds for (1, 0, c)-super-solutions and maximizing the (1, 0, c)-repairability, which are the main focus of the stability analysis in this work, as mentioned in Section 1.4.3.

By extending the stability concept, we also extend the repairability value concept for CSPs with ordered domains. In the work developed in (Hebrard, 2006), the 'closeness' between the original values broken (values that are not longer valid due to the changes in the problem) and the repair values was not considered. However, this characteristic is important for finding stable solutions whose associated repair solutions are as similar as possible to the original one, according to the order between elements. This idea is the **second** statement of the abstract, obtained from (Verfaillie and Jussien, 2005).

Second *"Limit as much as possible changes in the produced solution"*.

Furthermore, the extension of the stability concept that we define for the analyzed framework is also consistent with the similarity definition between solutions proposed in (Hebrard et al., 2007).

3.4 New Theoretical Approaches for this Environment

In this section we explain the theoretical approaches that we have developed for CSPs with ordered domains that come from a dynamic and uncertain environments. Hence, these problems can undergo the dynamism assumption made in Section 3.1, our main objective is the search of the solutions that are located as far away as possible from the bounds of the solution space.

Since the solution spaces of CSPs with ordered domains have been proved to be topological and metric spaces, we make use of concepts and definitions associated with these spaces (see Section 2.2). We recall that domains that are in $\mathbb{R}$ have an order relationship by definition. In contrast, when we deal with other types of domain elements (e.g., symbolic domains), a monotonic mapping function $f(x) : \mathcal{D} \rightarrow \mathbb{R}$ has to be applied in order to map the elements of the CSP domain by preserving their order (see Section 2.2). In such case, the definitions and concepts of metric spaces can also be used. Following, we explain an approach that is specific for convex spaces and another method for general purpose.

3.4.1 The Centroid of the Solution Spaces of CSPs

Subsequently, we explain a theoretical approach addressed to convex spaces that is the basis of the technical approach presented in Chapter 5 (Climent et al., 2011). Here, we assume that all the bounds of the solution space are dynamic. As previously mentioned, the closer the solutions are located to the bounds of the solution space of a CSP with ordered domains, the less robust the solutions are. On the contrary, the solutions that are located as far as possible from the bounds of the solution space have the highest likelihood of remaining valid after these kind of changes, so they are considered to be the most robust solutions (Definition 3.3.1).

Taking into account the Euclidean geometry definitions, we can calculate the centroid or gravity center of a geometric object. Following, we introduce the definition of the centroid of *2-dimensional* spaces and we introduce an example of such type of space.

Definition 3.4.1. *The centroid of a 2-dimensional solution space is the intersection of all straight lines that divide the solution space into two parts of equal moment. An illustration of the latter definition is given with the following toy example.*

Example 3.4.1. *We consider a CSP that is composed of two variables* x_0 *and* x_1 *with domains* $D_0 : \{1..8\}$ *and* $D_1 : \{1..7\}$ *respectively, and constraints:*

- $C_1 : x_1 - 6 \leq 0$
- $C_2 : x_0 - 7 \leq 0$
- $C_3 : x_1 - 2 \geq 0$
- $C_4 : x_0 - 3 \geq 0$

The solution space of Example 3.4.1 is a *2-dimensional* convex solution space. Figure 3.4 shows the geometric figure of this solution space (coloured in blue), which is a square whose centroid is the solution $(x_0 = 5, x_1 = 4)$, which it is highlighted with a circle in the figure. Note that this point is the intersection of the two straight lines that divide the square in two equal areas.

It must be taken into account that for convex spaces, the centroid of a figure is the point located as far as possible from all the bounds of the figure. This does not occur for non-convex spaces, for which the centroid of the solution space can be located outside the figure itself (see Figure 3.5).

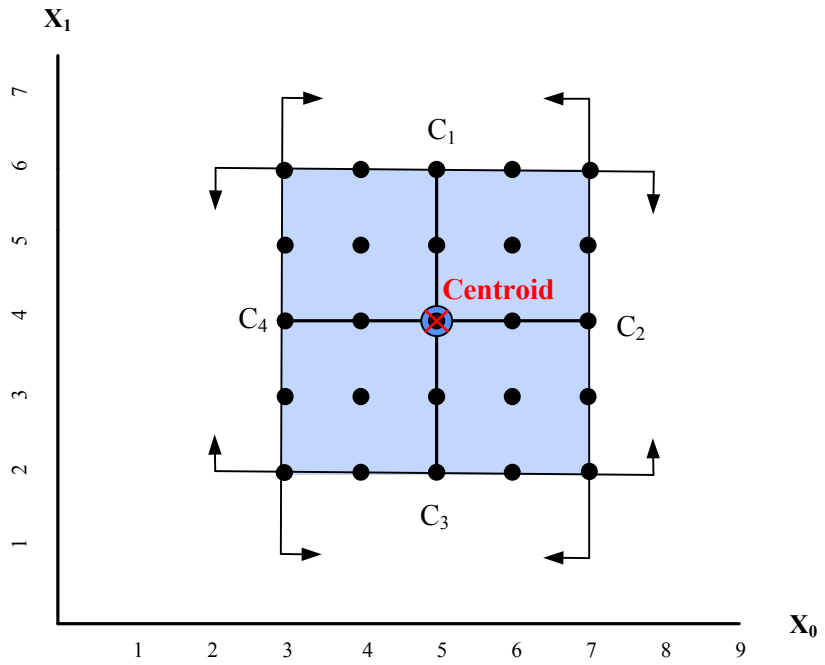


Figure 3.4: Centroid of the convex solution space of the CSP of Example 3.4.1.

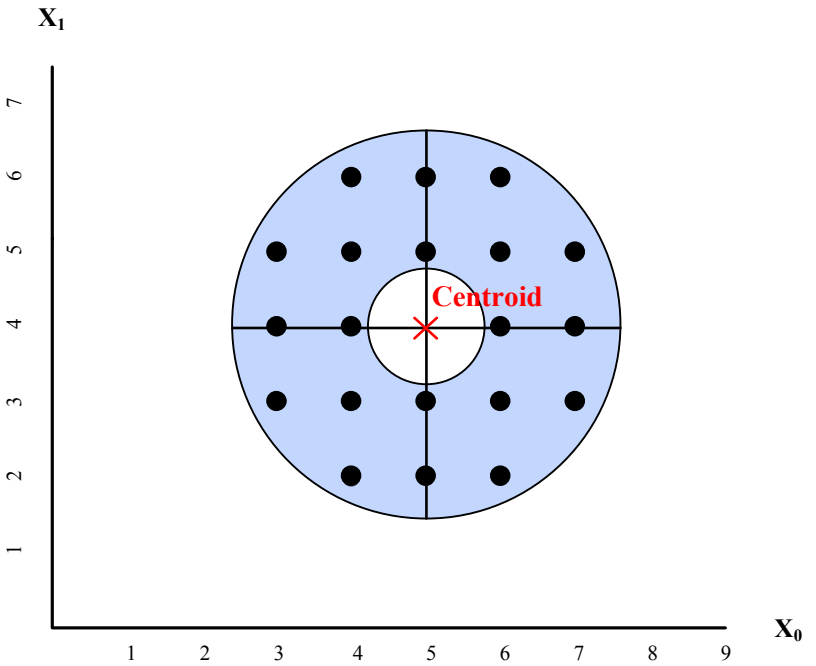


Figure 3.5: Centroid of a non-convex space.

Nevertheless, the centroid of a convex geometric figure always lies in the geometric figure, so we can take advantage of this characteristic of convex spaces for locating the most robust point of the convex geometric figure, since it has the greatest likelihood of remaining *inside* the geometric figure after changes in its bounds.

Definition 3.4.2 extends the previous centroid definition to a *n-dimensional* solution space.

Definition 3.4.2. *The centroid of a n-dimensional solution space is the intersection of all hyperplanes that divide the solution space into two parts of equal moment (Gao and Adviser-Revesz, 2006).*

Note that for CSPs with discrete domains, the centroid of a solution space may not be a solution, so that *the most robust solution is the nearest solution to the centroid*, according to the dynamism assumptions made for CSPs with ordered domains.

3.4.2 Neighbourhood and 'Onion topology'

In the previous section, we show how the centroid of a solution space is a good indicator of the solution that maximizes the distance from all the bounds of the convex solution spaces. Unfortunately, this method can not be used for the same purpose for non-convex spaces since the centroid can be located outside the solution space itself (see Figure 3.5). For this reason, we have found an alternative way for determining the most robust solution for non-convex spaces. The idea that we present in this section is based on the relation between the neighbour solutions of a solution s and the distance to s from the bounds of the solution space. This theoretical approach uses concepts associated with the topology of CSPs with ordered domains, which were introduced in Section 2.2. Part of the following explanation can be found in (Climent et al., 2013b).

This idea is first shown with a very simple example in Figure 3.6. In this figure, it can be observed a selection of two solutions of a CSP. The solution analyzed is highlighted and it has a very close neighbour solution located on its right side. We can observe in solid blue a small space, which is the minimum space that can be demonstrated to be solution space. There also exists a dashed blue area that represents the rest of the search space for which we can not ensure the feasibility due to the fact that we do not know the rest of the neighbours of the analyzed solution. The fact that there exists a valid assignment on the right side of the analyzed tuple ensures that it is not located next to a solution space bound on this side. Therefore, by checking the valid neighbours of the analyzed tuple we can ensure minimum distances to the bounds of the solution space, which gives us an approximation to the point located the furthest away from all the bounds of the solution space.

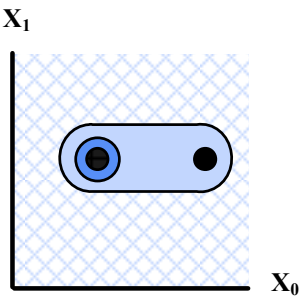


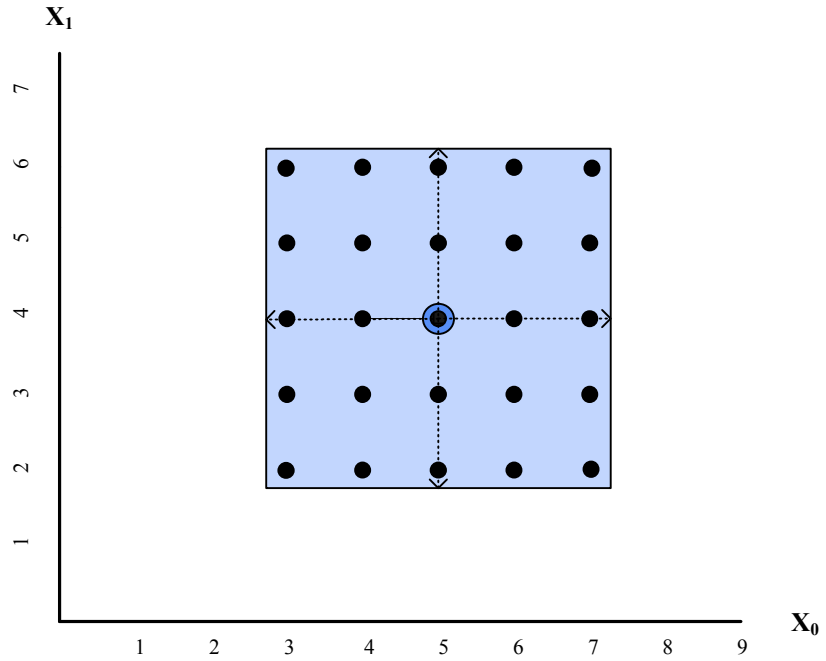
Figure 3.6: Neighbour solutions.

Previously, we have explained a very simple neighbourhood analysis, composed of only two neighbours. Following, we explain two cases for which we have analyzed all the solutions of the CSP, both are shown in Figures 3.7. In the Figure 3.7(a), the analyzed solution, which is highlighted, is surrounded by a great amount of neighbours. This solution is located to a minimum distance of 2 units from the bounds, since it has two contiguous neighbour solutions in all the directions (see the arrows in Figure 3.7(a)). However, in Figure 3.7(b) we can observe that the analyzed solution has a high amount of close neighbours but we can not ensure that the minimum distance to the bounds of the solution space is 2. This is due to the fact that the closest neighbour placed below the analyzed solution is not feasible (see the red cross) and therefore this solution is located above a bound, being very sensitive to restrictive changes over this bound. From the latter example, we can infer the following lemma:

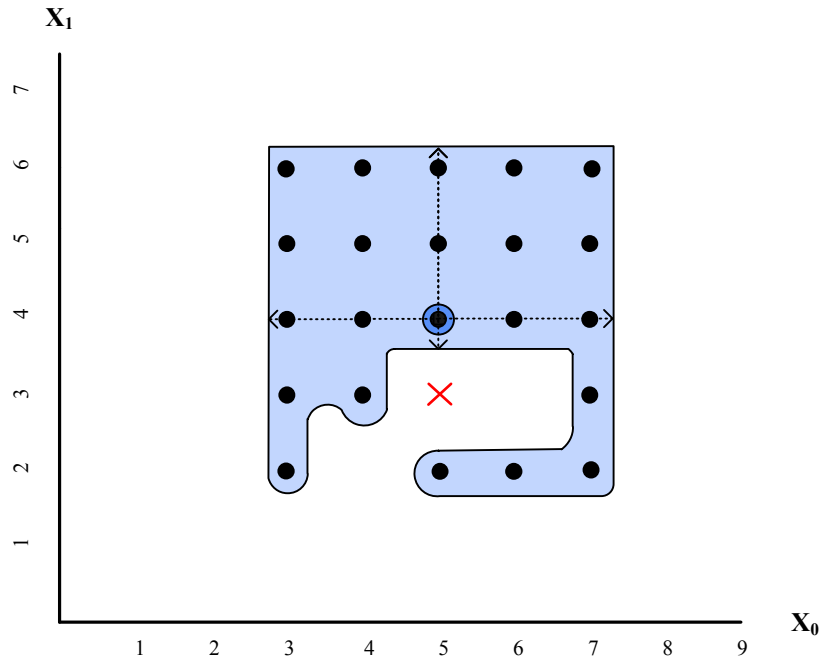
Lemma 3.4.1. *It only can be ensured that a solution s is located to a distance d from a bound in a certain direction of the n-dimensional space if all the tuples at distance lower or equal to d from s in such direction are feasible.*

This concept is crucial for the approaches described in Chapter 6 and Chapter 7 (both based on this theoretical approach) and it is also referred as the contiguity between the feasible neighbours. For Figure 3.7(b), we can not ensure a distance of 2 units from the bounds in the below direction of the tuple analyzed because there is one tuple that is not feasible and therefore it is not inside the solution space (see the point in the space marked with a red cross).

We would like to point out that since this theoretical approach analyzes the feasible neighbours, the domains of the CSPs have to be discrete. By means of checking the feasible neighbours of an analyzed tuple, we can estimate its robustness, since this information provides data about the distances to the bounds. Furthermore, there is evidence that certain forms of network robustness also fit this model of robustness. It consists in the application of the 'onion structure' to dynamic networks. In (Herrmann et al., 2011) the authors analyze how the structure of a network can affect



(a) Solution located at distance 2 from the bounds of the solution space.



(b) Solution located on a bound of the solution space.

Figure 3.7: Neighbours of different analyzed solutions.

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". In this thesis, we have applied the 'onion structure' concept to CSPs, which is a novel idea.

For CSPs, we roughly define a layer of this 'onion structure' as a convex hull of valid neighbour tuples that surround a specific tuple. Thus, the layer measures the protection of the tuple against perturbations. A valid tuple with more layers is presumed to have a higher probability of remaining valid than a tuple with fewer layers, and therefore, it is more robust. As mentioned above, there exists a direct relation between the 'onion layers' of a tuple and its distance from the bounds of the solution space: the more layers it has, the greater is its distance to the bounds. Due to the fact that the approaches presented in this dissertation search for solutions that have a high number of layers, our solutions were introduced in Section 1.5 as 'covered solutions' (see Figure 1.5).

Here we explain the relationship between this theoretical approach and the theoretical approach for convex solution spaces, which was introduced in Section 3.4.1 and how the theoretical approach introduced in this section behaves with such solution spaces. Figure 3.7(a) represents the same solution space as Figure 3.4. In Figure, 3.4 the closest solution to the centroid, which is the most robust, is highlighted (it coincides with the centroid for this example). In Figure 3.7(a), the solution with more complete 'onion layers', which is the most robust, is highlighted. Notice that both theoretical approaches consider as the most robust solution this one that maximizes the distance from all the bounds of the solution space. In convex solution spaces, due their very definition, there can not exist situations in which a tuple has a complete layer of feasible tuples surrounding it and that it has in the same time, a closer incomplete layer (as it happens in Figure 3.7(b)). From this characteristic, we can derive that for convex solutions spaces, the solutions with the greatest number of layers are the closest ones to the centroid of the solution space.

There exists a relationship between this robustness criterion based on the feasibility of neighbours and the stability concept. The main objective of finding solutions with close feasible neighbours, it also confers stability to the solutions. This is shown in Figure 3.6, where the highlighted solution has a neighbour solution on its right side. This feasible neighbour has the same value for the variable x_1 than the analyzed solution. This means that the variable x_0 is repairable because if its assigned value is lost, it can easily be repaired by assigning the neighbour value (since this value is consistent with the rest of the values of the assignment). Thus, all the neighbours located in the direction of the axis of a variable, represent repairable solutions for the breakage (value loss) of this variable. For instance, in Figure 3.7(a) the marked solution has four repairable solutions for each breakage of a variable, whilst the highlighted solution in Figure 3.7(b) has only two repairable solutions if the value assigned to x_1 (value 4) is lost.

The feasibility of neighbour assignments confers more stability to the solution according to Definition 1.4.4. In addition, if the feasible neighbours are *close* to the solution, it confers also stability to it according to the new extended definition introduced for CSPs with ordered domains (see Definition 3.3.2). The latter is a more specific concept of stability for solutions that are associated with such framework. Note that the difference between the lost value and the repairable value of one of the closest neighbour solution is minuscule, since they are immediate neighbours. In fact, their value difference is one, which is the minimum possible. Thus, by means of the search of robust solutions proposed in this chapter, we can also increase the stability of the solutions found. We recall the **third** statement made in the abstract of this thesis, which comes from the survey about dynamic and uncertain environments (Verfaillie and Jussien, 2005).

Third *"The production of solutions that are at the same time robust and flexible, that have every chance to resist changes and can be easily adapted when they did not resist, is obviously a desirable objective."*.

3.4.2.1 Formalization

In this thesis, we consider situations in which only a limited (and intuitively reasonable) assumption need be made about possible changes that can occur in CSPs with ordered domains: namely that changes always take the form of restrictions at the borders of the solution space. This means that for a restriction in a given direction of the solution space, a tuple further from the given border is less likely to be lost than one closer. 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 longer be 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 restrictions possible.) 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, then, the border value a is the most robust value, although a only has one feasible neighbour, while b has two feasible neighbours.

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 example just described, if the probability of losing a is 1/2 while the probability of losing c is 1/10, and the probability of losing b is 1/10 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 feasible neighbours 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 3.5, where we define a type of weighted CSP whose solution gives us the most robust solution under our assumptions.

3.5 New Techniques for this Environment

When we search for robust and/or stable solutions of a CSP, on one hand, we aim at finding a solution that satisfies all the constraints, while optimizing some criteria. Thus, the problems become in CSOPs (see Definition 2.4.1) whose objective function is the maximization of the robustness and/or stability. Dealing with CSOPs increases the computational time in comparison with CSPs. Generally, the cost of solving a CSP is NP-complete, meanwhile, solving a CSOP is NP-hard (see Section 2.6). Hence, the computational costs of the algorithms developed are of vital importance. Furthermore, there exists the additional difficulty that the optimality criterion of the theoretical approach of Section 3.4.2 depends on the feasibility of the neighbour solutions, which is an unknown information when the search starts. This has been one of the most difficult challenge of the approaches developed in this thesis. In this section we provide an overview of these approaches, which can be classified as:

- Enumeration-based techniques: They model robustness and stability in CSPs as WCSPs.
- Search algorithm: It searches for robust and stable solutions by maximizing a certain objective function.

In this thesis we present three enumeration-based techniques for finding robust solutions (Climent et al., 2012a, 2011, 2013b). Due to the enumeration characteristic of them, they only can be applied to discrete domains. One of the enumeration-based approaches is a probabilistic approach meanwhile the others are non-probabilistic. This classification depends on the information that is known about the dynamic and uncertain environment. We differentiate the level of knowledge about the possible future changes over the CSPs according to if it is only inherent to the structure of the CSPs or if it is additional external information about the future possible changes. Note that in the second case the availability of knowledge is much higher than in the first case. Specifically, the technique that uses additional dynamism data, is a probabilistic approach because part of this information is gathered as probabilities of change. This informed approach is presented in Chapter 4.

The two non-probabilistic enumeration-based approaches developed only assume changes that are inherent to the structure of the CSPs. The first non-probabilistic enumeration-based approach presented in Chapter 5 is specifically addressed to finite linear CSPs (see Definition 2.1.12), which is composed exclusively of linear constraints (see Definition 2.1.11). This specific feature allows to reduce significantly the temporal complexity of the algorithm that searches for robust and stable solutions. On the contrary, the second non-probabilistic enumeration-based approach presented in Chapter 6 is for general purpose and as consequence it has a higher temporal complexity than the specific approach for finite linear CSPs. Due to the solution spaces associated with linear CSPs are always convex, we have used the theoretical approach described in Section 3.4.1. However, we used the theoretical approach described in Section 3.4.2 for the non-probabilistic enumeration-based approach for general CSPs.

For all the enumeration-based techniques presented in this thesis, we use an extension of the original CSP model named WCSP model (see Definition 2.4.3). The aim of modeling CSPs with ordered domains as WCSPs is for obtaining robust and stable solutions according to Definition 3.3.1 and Definition 3.3.2. This is achieved with the costs that the constraints assign to each involved tuple. This additional expressiveness offered by the WCSP model provides a mechanism for penalizing the tuples associated with a constraint. This mechanism is used to express the dynamism of the constraints of the original CSP. However it has as disadvantage that all the constraints must be extensionally expressed, which entails a high spatial cost. Finally, the modeled WCSPs are solved by a generic WCSP solver from the literature. The best solution of a WCSP is the solution with the minimum global cost, which it is computed as the aggregation of all applicable costs.

The presented enumeration-based approaches do not, in and of themselves, restrict us to the WCSP, since there are a variety of valued CSPs and other models that could also be compatible. However, many of these models either involve assumptions which are questionable (e.g., probabilistic CSP, where valuations based on 'onion layers' would be multiplied) or are insufficiently discriminating (e.g., fuzzy CSP). In addition, the WCSP model adequately incorporates the enumeration aspect of number of neighbours or number of 'onion layers' of an assignment, unlike most other valued CSP models.

The penalizations assigned to the valid tuples of each constraint are based on certain information computed for the specific enumeration-based technique. This information is different for each of the three enumeration-based approaches developed. A disadvantage of these approaches is that the computation of the information that is used for the penalization process is computed for each bound. Thus, if there exists another bound that is tightening the analyzed bound, some of the valid tuples of the first bound can be invalidated by the second bound. Hence, the enumeration-based approaches give us approximations to the real distances of the solutions to the bounds of the solution space. Ideally, determining the complete solution space would provide the solution located exactly the furthest away from the bounds of a CSP. However,

this is not viable generally because it would be extremely time consuming due to the high combinatoriality (NP-hard generally). This has motivated the development of the enumeration-based approaches, which allow to find approximations to the desired solution. This is, itself, a disadvantage of this type of approach, however, by using inference techniques (see Section 2.5), it is possible to narrow the search space in order to make this approximation more accurate. For all the approaches developed in this thesis we have used the Algorithm 2.1 (GAC3), although any other preprocessing inference techniques can also be applied.

The latter disadvantage of the enumeration-based approaches has motivated the development of the search algorithm in Chapter 7, which searches for solutions that maximize an objective function that measures the feasibility of their neighbours (Climent et al., 2012c, 2014). This search approach is addressed to CSPs with discrete domains because it checks the feasibility of the finite neighbours of an assignment. One of the disadvantages of this approach is its high computational cost (which can be alleviated with several techniques, as we will explain in Chapter 7). In addition, the level of information available at the beginning of the search tree is low. However, as previously mentioned, the strength of this approach is that the objective function values associated with the solutions obtained correspond with the exact value of those solutions in the solution space. In this way, this approach avoids the disadvantage explained above for the enumeration-based approaches. Table 3.1 summarizes the weak (represented by the '-' symbol) and strong (represented by the '+' symbol) points of both types of techniques.

Table 3.1: Characteristics of the types of the presented techniques.

Enumeration-based techniques	Search Algorithm
Independent Solver	Dependent Solver
(+) High information initially	(-) Low information initially
(-) Optimality criterion approx.	(+) Exact objective function value
(-) High spatial cost	(+) Very low spatial cost

3.6 Summary and Limitations

In this chapter we address the difficulties entailed by solving problems under uncertain and dynamic environments without extra detailed data about the future changes. The focus is on a common type of change associated with CSPs with ordered domains: restrictions over the bounds of the solution space. This assumption is compared with related dynamism assumptions from the literature. In addition, the concepts of robustness and stability are extended to this dynamic environment. We propose two

theoretical approaches for finding solutions with both features that are based on: the centroid of convex solution spaces and the feasibility of the neighbour solutions (including non-convex spaces). Finally, we do an overview of the technical approaches presented in this thesis: enumeration-based techniques and a search algorithm.

As a conclusion, this chapter answers the main question that has been the focus of this thesis: Is it possible to find robust solutions for CSPs that do not have extra information about the future changes that they can undergo? The answer is 'yes' but the elements that compose the CSP must have a significant order between all of them. However, this is not a strong limitation due to the high fraction of real life problems that have order relationships between their elements.

Chapter 4

Probabilistic Enumeration-based Technique

This chapter introduces a new probabilistic enumeration-based technique that models CSPs as WCSPs (Climent et al., 2012a). Its main objective is to find robust solutions according to extra information about the future changes over the bounds of the CSPs. Figure 4.1 shows a diagram that represents the basic steps that this technique carries out. Initially, the problem is modeled as a CSP (*P*). Then, by using additional information (see Section 4.1) about the dynamism of the problem, a WCSP is generated (*modP*). Finally, the modeled WCSP (*modP*) is solved by a generic WCSP solver. The solution space of *modP* is the same as the solution space of *P*. Furthermore, the best solution of *modP* is considered to be one of the most robust solution for *P*, according to the given extra dynamism data.

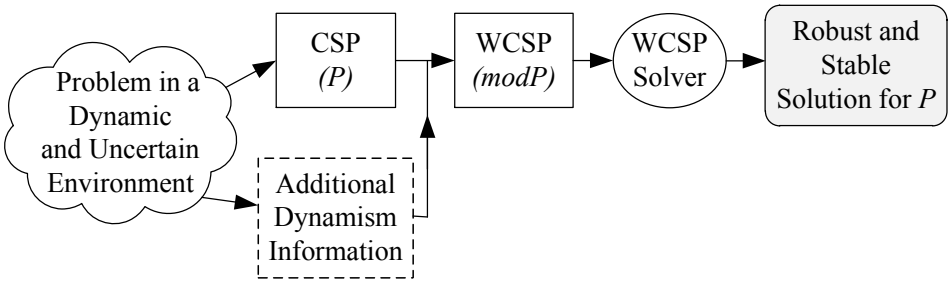


Figure 4.1: Modeling an informed CSP as a WCSP.

In order to represent the dynamism over the original CSP, the modeled WCSP is composed of the original constraints and domains of *P* and sets of new generated constraints for each dynamic bound of the original CSP. These constraints represent possible future restrictive modifications of the constraints and domains of the original CSP. The sets of new constraints are treated as soft constraints, since they can be

unsatisfied because they represent changes that have a certain likelihood to occur. The tuples associated with them have assigned costs, which represent penalizations. The more new generated constraints a solution satisfies, the more robust it is according to the extra information about the dynamism that this technique assumes. This solution has a high likelihood of remaining solution when faced with these possible restrictions over the original bounds.

4.1 Additional Information about the Dynamism

In this section we explain in further details the additional information about the dynamism that is required for the technique introduced in this chapter. As mentioned in Section 3.2, there exists a model called Probabilistic CSP that considers the constraint deletion as a possible future change. Thus, the information known about the dynamism is a function that indicates the probability of existence of the constraints. On the contrary, our probabilistic enumeration-based technique considers that the constraints and/or domains of the original CSP may undergo restrictive modifications in the future.

For representing this kind of possible future changes, it is not enough descriptive to use only one dynamism function as the Probabilistic CSP model does. This is due to the fact that restrictive modifications of the constraints can be of different magnitude. Note that slight restrictive modifications in constraints affect in lesser extent to the solution space of the CSP than abrupt restrictive variations. Thus, we consider an additional function to the probability of undergoing a restrictive modification: the magnitude of the modification. We define two functions that contain information about these possible future restrictive modifications over the constraints and domains of the CSPs:

- $p(C_i)$: Each original constraint $C_i \in (\mathcal{C} \cup \mathcal{DC})$ has a probability $p(C_i) \in [0, 1[$ that measures the probability that C_i undergoes a restrictive modification. It is called dynamism likelihood function. The minimum value ($p(C_i) = 0$) means that C_i will never undergo restrictive modifications in the future. On the contrary, when $p(C_i) > 0$, C_i is dynamic. The maximum value ($p(C_i) \approx 1$) means that C_i is very dynamic so the probability that it undergoes a restrictive modification is very high.

- $d(C_i)$: This function measures the magnitude of the restrictive modification that could undergo a dynamic original constraint $C_i \in (\mathcal{C} \cup \mathcal{DC})$. Thus, $d(C_i) \in]0, 1[$ measures the percentage of valid tuples that could become invalid if C_i finally undergoes this type of change in the future. A value of $d(C_i) \approx 0$ means that the modification over C_i would be slight and therefore almost all the valid tuples will remain valid. Meanwhile $d(C_i) \approx 1$ means that almost all the valid

tuples could be invalid if the future restrictive modification occurs. This function is not defined for constraints that cannot undergo restrictive modifications (constraints whose $p(C_i)$=0).

There are many criteria for selecting the set of valid tuples that become invalid for the new generated constraints. However, since the CSPs with ordered domains are the main center of our dissertation, without loss of generality, we use a criterion that is inherent to their structure: the nearest tuples from the bounds of the solution space have a higher probability of becoming invalid when faced with restrictive modifications over these bounds (see Section 3.4.2.1 for a formal rationale). Therefore, the nearest valid tuples from the dynamic bounds are invalidated for the new constraints. Note that if there exists specific information about which valid tuples are more likely to become invalid, this other criterion could also be used. In such a case, this technique is not restricted to CSPs with ordered domains.

In the following section it is explained how our probabilistic approach uses the function that measures the magnitude of change $d(C_i)$ for the generation of new modified constraints whereas the function that measures the probability of change $p(C_i)$ is used in the cost assignment to the tuples.

4.2 New Constraints Generation

The main objective of the new constraint generation is to represent future possible restrictive modifications over the bounds of the original CSP based on the two functions that describe their dynamism. The new constraints are generated for each dynamic constraint $C_i \in (\mathcal{C} \cup \mathcal{DC})$. As mentioned, this condition is determined by $p(C_i)$, so if $p(C_i) > 0$, C_i is considered as a dynamic constraint. Otherwise ($p(C_i) = 0$) the constraint is static and therefore it can not undergo changes.

Due to dynamic constraints have certain probability of undergoing restrictive modifications in the future, for each of them a set of new modified constraints ordered by their tightness $\{C_{i1}, C_{i2}, \ldots, C_{iw}\}$ is generated, where w is the number of constraints additions for each C_i of the original CSP. The parameter w is fixed by the user according to the desired granularity. Thus, the greater the parameter w is, the slighter the difference between the new generated constraints is. Each new constraint generated is a tighter version of the previous one and the level of its tightness depends on the parameter w and the $d(C_i)$ function. The last constraint generated (C_{iw}) represents the most restrictive modification possible that can undergo C_i in the future.

In order to clarify the new constraints generation, a toy CSP with one dynamic constraint is presented.

Example 4.2.1. *Consider a CSP composed of two variables x_0 and x_1 with domains*

$D_0 : \{3 \dots 7\}$ and $D_1 : \{2 \dots 6\}$ respectively (discontinuous lines), and four constraints (continuous lines):

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

Figure 4.2 shows the CSP of Example 4.2.1 and the set of new modified generated constraints for the dynamic constraint C_1 for $w = 2$. The set of new constraints is composed by C_{11} and C_{12} (see Figure 4.2). These constraints are considered as soft constraints, since they can be unsatisfied. However, the solutions that satisfy C_{11} and C_{12} (darkest blue area) have a greater probability of remaining valid when faced with restrictive modifications of C_1. Note that the solutions in the darkest blue area are more robust than the solutions in the intermediate blue area because the first ones satisfy C_{12}, while others do not. In addition, all the solutions located in both areas (darkest and intermediate blue areas) are more robust than the solutions located in the lightest blue area because the last ones do not satisfy C_{11} nor C_{12}. They are non-robust solutions for this CSP because their probability of remaining valid after restrictive modifications of the dynamic constraint C_1, is very low.

4.2.1 Properties of the New Generated Constraints

Each new generated constraint C_{ij} is a more restricted version of an original dynamic constraint. Therefore, the new constraints are composed of a subset of the valid tuples of the original constraint. Each new constraint C_{ij} is a more restricted version of $C_{i(j-1)}$ or C_i (when the new constraint is C_{i1}). Following, we formalize the properties previously mentioned of the new constraints. We recall that $T(C_i)$ is the set of valid tuples for C_i (see Definition 2.1.9).

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

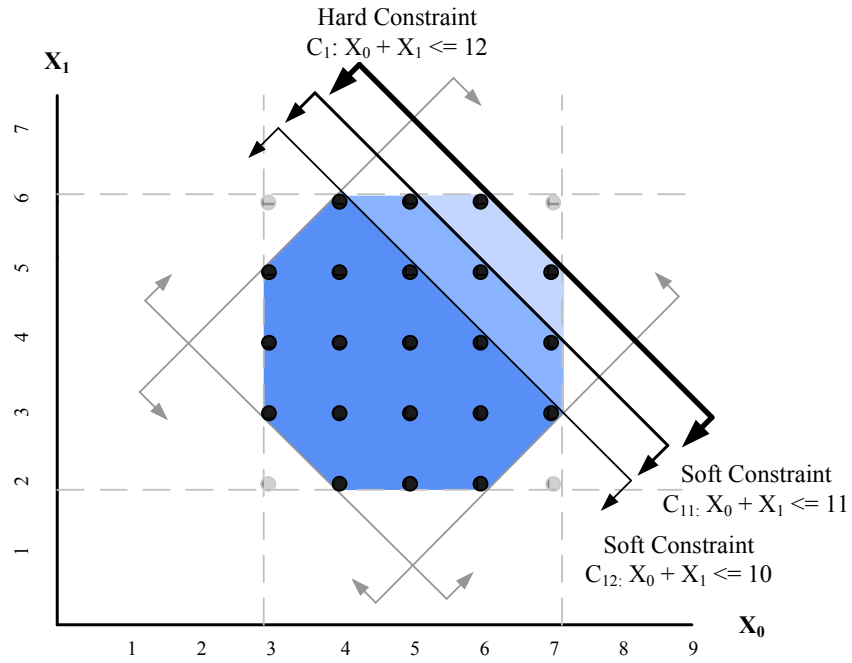


Figure 4.2: New generated constraints for the constraint C_1 of Example 4.2.1.

The number of valid tuples for the new set of constraints depends on the dynamism functions, which contain information about the possible future restrictive modifications over the dynamic constraints. Specifically, the function that measures the magnitude of the future restrictive modifications, called $d(C_i)$, is the responsible for determining the number of valid tuples of the new constraints. If a constraint has a high value of $d(C_i)$ and it undergoes a change, this change will be highly pronounced. For this reason, the new constraints generated for such constraint have a significantly reduced number of valid tuples associated with them. The granularity parameter w fixes the number of new constraints that are generated for each dynamic constraint. Therefore, for high granularity values, the differences between the number of valid tuples associated with the new constraints set is lower. Thus, for all the sets of new constraints, the number of tuples that become invalid for each C_{ij} regarding to $C_{i(j-1)}$ is $\lfloor (d(C_i) * |T(C_i)|)/w \rfloor$. In other words, the number of valid tuples of a new generated constraint is:

$$|T(C_{ij})| = \lfloor ((1 - d(C_i)) * |T(C_i)|) * (w + 1 - j) \rfloor \tag{4.1}$$

Example 4.2.2. *Here, we consider an example of a dynamic original constraint* C_1 *that has* 8 *valid tuples and* $d(C_1) = 0.75$*, which means that there exists an estimation*

that up to 75% *of their tuples could become invalid in the future, or said in another way, only* 25% *of the original valid tuples will remain valid for the most restrictive of the new constraints (*C_{iw}*).*

For a fixed value of $w = 3$, the new constraints generated for Example 4.2.2 are: C_{11}, C_{12} and C_{13}. In this intuitive example, the reader could guess the percentages of valid tuples that each new constraint will have with respect to C_1: 75% for C_{11}, 50% for C_{12} and 25% for C_{13}. This is equivalent to say that the number of valid tuples are : $|T(C_{11})| = 6$, $|T(C_{12})| = 4$ and $|T(C_{13})| = 2$. These results are obtained by applying Equation 4.1 to each constraint:

- $|T(C_{11})| = \lfloor((1 - 0.75) * 8) * (3 + 1 - 1)\rfloor = \lfloor(0.25 * 8) * 3\rfloor = \lfloor 2 * 3\rfloor = 6$
- $|T(C_{12})| = \lfloor((1 - 0.75) * 8) * (3 + 1 - 2)\rfloor = \lfloor(0.25 * 8) * 2\rfloor = \lfloor 2 * 2\rfloor = 4$
- $|T(C_{13})| = \lfloor((1 - 0.75) * 8) * (3 + 1 - 3)\rfloor = \lfloor(0.25 * 8) * 1\rfloor = \lfloor 2 * 1\rfloor = 2$

Unlike original constraints, the set of new constraints $\{C_{i1}, C_{i2}, \ldots, C_{iw}\}$ are considered as soft constraints. Note that if they were considered as hard constraints, the solution space of the CSP analyzed would be reduced, which is not the purpose of the enumeration-based technique presented here. By considering the new set of constraints as soft constraints, we can confer some levels of priority to the tuples that satisfy them, achieving in this sense several level of robustness in our solutions according to the additional extra information associated with the constraints.

4.3 Cost Assignment

One of the main characteristics that introduces the WCSP model with respect to the classical CSP model, is the cost assignment to the tuples associated with the constraints of the WCSP. This feature provides the opportunity of penalizing the tuples that do not satisfy the new generated constraints. The costs are assigned according to the dynamism functions, which contain information about the possible future restrictive modifications over the dynamic original constraints. Thus, the tuples that are invalid for the new constraints have a lower likelihood to be selected as part of the solution of the WCSP. Recall that the best solutions obtained for the WCSP modeled by the probabilistic enumeration-based technique, are the solutions with the minimum global cost. Therefore, the least penalized solutions are the best solutions for the modeled WCSP and they have also a certain level of robustness for the original CSP according to the dynamism functions associated with the dynamic constraints.

The cost associated with the tuples of the constraints is determined by the cost functions. We define two different types of cost functions, depending on whether the

constraint is hard (C_i) or soft (C_{ij}). The main utility of the cost function applied to an original constraint $C_i \in \mathcal{C}$ is to forbid the tuples that do not satisfy it. Thus, $C_i(t \downarrow_{var(C_i)})$ assigns a cost U to the tuple t if it does not satisfy C_i. The value U is a large number symbolizing the infinite cost. In the WCSP model, as mentioned in Definition 2.4.3, assigning the maximum cost U to a tuple t, means that t is an invalid tuple for C_i. On the contrary, if a tuple t satisfies the constraint C_i ($t \in T(C_i)$), the tuple does not have to be penalized. Therefore, the cost assigned to t for C_i is zero. Equation 4.2 shows the formalization of the cost function of the original constrains of the CSP, where $t \downarrow_{var(C_i)}$ is the projection of the tuple t over the variables that compose C_i (see Definition 2.1.5).

$$C_i(t \downarrow_{var(C_i)}) = \begin{cases} 0 & \text{if } t \in T(C_i) \\ U, (U \approx \infty) & \text{if } t \notin T(C_i) \end{cases} \tag{4.2}$$

The cost function applied to the new constraints $\{C_{i1}, C_{i2}, \ldots, C_{iw}\}$ allows to prioritize among all the original dynamic constraints $C_i \in (\mathcal{C} \cup \mathcal{DC})$ based on their likelihood of change $p(C_i)$. Since the dynamic constraints with a high value of $p(C_i)$ have a high likelihood of undergoing restrictive modifications, the tuples that do not satisfy new constraints generated from an original constraint whose $p(C_i)$ is high, are strongly penalized. Thus, all the new constraints generated by the same original constraint have the same cost function. In the same way that the cost function of the original constraints of the CSP does not penalize the tuples that satisfy them, the same happens with the cost function of the new generated constraints. The valid tuples of a new constraint do not have to be penalized because if in the future the original constraint becomes into this restricted new constraint, these tuples will still satisfy it. Hence, the cost assigned to a valid tuple $t \in T(C_{ij})$ is zero.

On the contrary, if a tuple t does not satisfy a new restricted constraint, it is penalized with a normalized value of $p(C_i)$. By definition, the possible values for the set of costs of a WCSP are positive natural integers, bounded by U (see Definition 2.4.3). For this reason, the value of $p(C_i)$, which is a real number, has to be normalized. This is achieved by multiplying $p(C_i)$ by 100 and the result is rounded by the floor function. The floor function returns the largest integer that is less than or equal to the argument. Hence, the cost function $C_{ij}(t \downarrow_{var(C_{ij})})$ assigns a cost of $\lfloor p(C_i) * 100 \rfloor$ to the invalid tuples, that is to say $t \notin T(C_{ij})$. Therefore, the set of costs that the invalid tuples can have associated is $\{0, \ldots, 99\}$. Note that this set of costs is negligible in comparison with U (U is a large number symbolizing the infinite cost), allowing then, that invalid tuples of the new constraints are valid partial assignments for the WCSP modeled and the original CSP. For this reason, we stated that the new constraints are actually soft constraints because the solutions can dissatisfy them. Equation 4.3 shows the formalization of the cost function of the new constrains generated for the WCSP model.

$$C_{ij}(t \downarrow_{var(C_{ij})}) = \begin{cases} 0 & \text{if } t \in T(C_{ij}) \\ \lfloor p(C_i) * 100 \rfloor & \text{if } t \notin T(C_{ij}) \end{cases} \quad (4.3)$$

4.4 Algorithm

In this section we present an algorithm that models CSPs into WCSPs. These CSPs must have associated knowledge of the future changes, represented by the dynamism functions associated with their dynamic constraints. Moreover, the computation complexity of this algorithm is explained.

4.4.1 Algorithm Description

The objective of this WCSP modeling is to find robust solutions for the original CSP according to the extra information about the future changes that it may undergo. Given an original CSP $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ and the dynamism functions p and d, Algorithm 4.1 generates a WCSP $modP = \langle \mathcal{X}, \mathcal{D}, S(U), \mathcal{C}' \rangle$ which is composed of the original variables and domains, and the set of constraints $\mathcal{C}'$. $\mathcal{C}'$ is defined as the union of the original constraints plus the new constraints ($\mathcal{C}' = \{C_i \in (\mathcal{C} \cup \mathcal{DC}) \cup \{C_{i1}, C_{i2}, \ldots, C_{iw}\}\}$). Algorithm 4.1 also assigns the costs to the tuples associated with each constraint according to the cost functions explained in Section 4.3.

The aim of generating a WCSP from the original CSP is finding solutions that satisfy the maximum number of new constraints generated for the WCSP, considering their priorities (which are based on the dynamism functions). Thus, these solutions have the highest probability of remaining valid after restrictive modifications over the dynamic bounds of the original CSP. The priority of each set of new constraints is expressed in terms of the cost assigned to their invalid tuples (the cost depends on the dynamism function p). A high cost means a high penalization for the tuples that do not satisfy such constraint. As previously mentioned, the original CSP and its corresponding derived WCSP model both share the same solution space. In addition, the solutions of the WCSP have an associated cost that depends on the penalizations assigned by the soft constraints. In Definition 2.4.3 it was explained that the best solution of a WCSP is the solution s with the minimum $\mathcal{V}(s)$ cost. Since we model the WCSP by assigning costs that represent penalizations, the best solution s for the WCSP that we model is the solution which minimizes the global associated penalization.

Following, the algorithm for modeling WCSPs from CSPs with extra information about the dynamism of their constraints and domains, is described in detail. Algorithm 4.1 has as input arguments the original CSP (called *P*), the granularity parameter w, which fixes the number of constraints additions for each $C_i \in (\mathcal{C} \cup \mathcal{DC})$ of the original

CSP *P* and the dynamism functions $p(C_i)$ and $d(C_i)$ for each C_i. In addition, a time cutoff can be optionally provided by the user. First, Algorithm 4.1 analyzes each C_i by applying the cost function for hard constraints (see lines 3-8) and generating the set of new restricted constraints ordered by their tightness $\{C_{i1}, C_{i2}, \ldots, C_{iw}\}$ for each C_i (see lines 9-16), as well as assigning them costs. The sets of new constraints represent possible restrictive modifications over each original constraint/domain C_i, based on their dynamism functions $p(C_i)$ and $d(C_i)$. Each new constraint C_{ij} is composed of a subset of $T(C_i)$. In this way, C_{ij} is a more restricted version of C_i.

Finally, Algorithm 4.1 solves the WCSP modeled ($modP$) with a usual WCSP solver (line 18), and it returns the best solution and its global cost (if such a solution exists)) for *modP* (line 19). The best solution is the one that has the lowest associated cost, which is considered to be one of the most robust solutions for the original CSP, according to the dynamism functions associated with the dynamic constraints and domains. Typically, the WCSP solver allows to configure the search by fixing a time cutoff ('anytime' algorithms). In such case, only the best solution found until reaching the time cutoff is returned.

4.4.2 Computational Complexity

The complexity of the modeling phase of Algorithm 4.1 (lines 1-17) is mainly related to the number of dynamic constraints and domains of the CSP, their number of associated tuples and the granularity parameter w. A set of w new modified constraints is generated over each $C_i \in (\mathcal{C} \cup \mathcal{DC})$. (Here, we consider the worst case in which all the bounds are dynamic). For each constraint of $\mathcal{C}'$ (this set includes original and new constraints), a cost is assigned to each tuple. Nevertheless, the WCSP file format (see Appendix A.1) that we use for the WCSP modeling, allows the assignment of a default cost to the invalid tuples of a constraint. Thus, it is only necessary to analyze and assign a cost to the valid tuples. Since we are analyzing the worst complexity case, we consider the maximum number of valid tuples of the constraints a CSP, which are denoted as $T(C)_{Max}$ (see Definition 2.1.10 and Equation 2.2).

After the previous complexity analysis, we can state that the modeling phase of Algorithm 4.1 (lines 1-17) is $O(T(C)_{Max} * |\mathcal{C}'|)$, which is equivalent (when all the constraints and domains of the CSP are dynamic) to $O((e+n)*w*T(C)_{Max})$, where $e = |\mathcal{C}|$ and $n = |\mathcal{DC}| = |\mathcal{X}|$. Recall that the cardinality of a set is denoted with a vertical bar on each side of the set name. Therefore, the term $|\mathcal{C}'|$ denotes the number of constraints of $\mathcal{C}'$ and $|(\mathcal{C} \cup \mathcal{DC})|$ is the number of constraints and domains of the CSP. We would like to point out that the complexity of solving the modeled WCSP is not discussed due to it depends on the WCSP solver used.

Algorithm 4.1: Algorithm for modeling an informed CSP as a WCSP.

Data: A CSP $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, $p(C_i)$ and $d(C_i)$ functions, parameter w and *time_cutoff* (optional).
Result: Best solution s and its associated global cost $\mathcal{V}(s)$.

begin
  foreach $C_i \in (\mathcal{C} \cup \mathcal{DC})$ **do**
    // Hard constraints.
    foreach $t \in C_i$ **do**
      if $t \in T(C_i)$ **then**
        $C_i(t \downarrow_{var(C_i)}) = 0$;
      else
        $C_i(t \downarrow_{var(C_i)}) = U$;
    // Soft constraints.
    foreach $j \in \{1, \ldots, w\}$ **do**
      Generate $\{C_{ij}\}$ based on $d(C_i)$;
      foreach $t \in C_{ij}$ **do**
        if $t \in T(C_{ij})$ **then**
          $C_{ij}(t \downarrow_{var(C_{ij})}) = 0$;
        else
          $C_{ij}(t \downarrow_{var(C_{ij})}) = \lfloor p(C_i) * 100 \rfloor$;
  Generate the WCSP $modP = \langle \mathcal{X}, \mathcal{D}, S(\mathrm{U}), \mathcal{C}' \rangle$ where, $\mathcal{C}' = \{C_i \cup \{C_{i1}, C_{i2}, \ldots, C_{iw}\}\}, i \in \{1, \ldots, (e+n)\}$;
  $(s, \mathcal{V}(s)) \leftarrow$ Solve $modP$ in *time_cutoff*;
  return $(s, \mathcal{V}(s))$;

4.5 Example

In this section, we introduce an example of a CSP with extra knowledge about the dynamism of its constraints. Each constraint is labeled with two real numbers. The first number between the parenthesis represents the dynamism likelihood $p(C_i)$ and the second the magnitude of change $d(C_i)$. The CSP is modeled into a WCSP and solved by a WCSP solver. The result obtained is the set of solutions for the original CSP with an associated cost. The solutions with the lowest costs associated are the most robust solutions of the CSP according to its extra information about the dynamism of its constraints and domains.

Example 4.5.1. *We consider a CSP with two variables x_0 and x_1 with domains $D_0 : \{3 \dots 7\}$ and $D_1 : \{2 \dots 6\}$, respectively. All its domains are static and all its constraints are dynamic. The corresponding dynamism functions of the constraints are:*

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

Figure 4.3 is the representation of the CSP of Example 4.5.1 and Table 4.1 shows the extra dynamism data. We fix the granularity parameter $w = 1$. The parameter U has to be fixed to a huge number because the costs associated with valid tuples must be negligible with respect to U.

Table 4.1: Probabilities and magnitude of change of Example 4.5.1.

C_i	$p(C_i)$	$d(C_i)$
C_1	0.2	0.2
C_2	0.8	0.4
C_3	0.4	0.3
C_4	0.2	0.4

Following the steps presented in Figure 4.1 and Algorithm 4.1, this CSP is modeled as a WCSP ($modP$). Furthermore, in Appendix A.2 it is described the Weighed

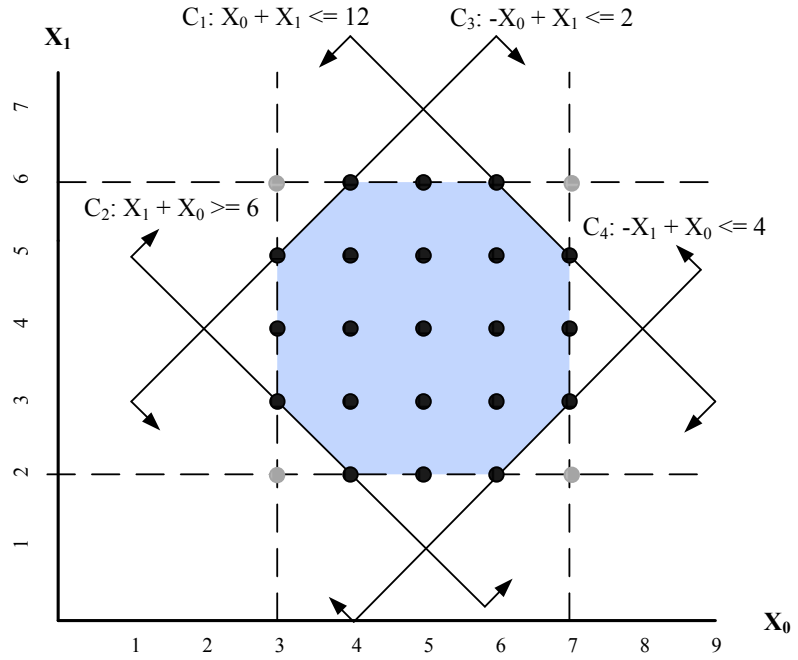


Figure 4.3: Solution space of the informed CSP of Example 4.5.1.

CSP file format specification of $modP$. This modeled WCSP is composed of the same variables and domains than the original CSP. However, it is composed of 8 constraints, of which 4 are the original dynamic constraints that have 24 valid tuples each one and 4 are the new generated constraints, which are restricted versions of the original constraints with a lower number of valid tuples. Thus, $modP$ is composed of two variables x_0 and x_1 with domains $D_0 : \{3 \dots 7\}$ and $D_1 : \{2 \dots 6\}$ and the following constraints:

- $C_1 : x_0 + x_1 \leq 12$ (24 tuples).
- C_{11} : restricted version of C_1 with only 19 tuples.
- $C_2 : x_1 + x_0 \geq 6$ (24 tuples).
- C_{21} : restricted version of C_2 with only 14 tuples.
- $C_3 : x_1 - x_0 \leq 2$ (24 tuples).
- C_{31} : restricted version of C_3 with only 16 tuples.
- $C_4 : x_0 - x_1 \leq 4$ (24 tuples).
- C_{41} : restricted version of C_4 with only 14 tuples.

As it has been explained in Section 4.3, all the valid tuples of the constraints (both original and new constraints) have an associated cost of zero. The invalid tuples have different associated cost depending on if the constraint is original or it is a new one. As previously mentioned, the cost associated with the invalid tuples of the original constraints is U. The costs associated with the invalid tuples of the new constraints depends on the probability of change ($p(C_i)$) of each original constraint that they come from (see Equation 4.3). Thus, the cost associated with the invalid tuples of C_{11} is 20 ($p(C_1) = 0.2$), for C_{21} is 80 ($p(C_2) = 0.8$), for C_{31} is 40 ($p(C_3) = 0.4$) and for C_{41} is 20 ($p(C_4) = 0.2$). The number of tuples that become invalid for the new constraints is calculated according to Equation 4.1, which involves the parameter that measures the magnitude of change of each original constraint. Following, we introduce their computation:

- $|T(C_{11})| = \lfloor((1 - 0.2) * |T(C_1)|) * 1\rfloor = \lfloor(0.8 * 24)\rfloor = 19$
- $|T(C_{21})| = \lfloor((1 - 0.4) * |T(C_2)|) * 1\rfloor = \lfloor(0.6 * 24))\rfloor = 14$
- $|T(C_{31})| = \lfloor((1 - 0.3) * |T(C_3)|) * 1\rfloor = \lfloor(0.7 * 24)\rfloor = 16$
- $|T(C_{41})| = \lfloor((1 - 0.4) * |T(C_4)|) * 1\rfloor = \lfloor(0.6 * 24)\rfloor = 14$

The tuples that become invalid for the new constraints are those tuples that are located closer to the corresponding constraint (see these specific tuples in Appendix A.2). As we have pointed out, the WCSP solver finds as best solution, the solution s with the minimum value of $\mathcal{V}(s)$ associated. The best solution for $modP$ is $(x_0 = 5, x_1 = 4)$, because $\mathcal{V}(x_0 = 5, x_1 = 4) = 0$. This is due to the fact that this solution satisfies all the set of new modified constraints (C_{11}, C_{21}, C_{31} and C_{41}) and therefore it is never penalized. For this reason, it is one of the most robust solutions for P.

The solution $(x_0 = 3, x_1 = 3)$ that is the solution provided by an ordinary CSP solver with a lexicographical value ordering, only satisfies two of the four new modified constraints of $modP$. It does not satisfy the constraints C_{21} and C_{31}. As mentioned, the cost assigned to a tuple that does not satisfy C_{21} is 80 and the cost assigned to a tuple that does not satisfy C_{31} is 40. That is why, $\mathcal{V}(x_0 = 3, x_1 = 3) = 120$. This solution is an unrobust solution for P since it has a low probability of remaining valid when faced with changes in the constraints C_2 and C_3.

4.6 Summary and Limitations

In this chapter we deal with CSPs that model problems whose elements share an order relationship and, in addition, they have certain associated information about the future possible changes that they might undergo. Specifically, these changes can take the form of restrictive modifications of the bounds of the solution space, which is composed by the constraints and domains. The dynamism data associated with

the problems is gathered in two parameters: the first one measures the probability of change of the constraints and the second one measures their magnitude of change (that it is to say, how much a change could narrow the solution space).

The approach introduced in this chapter for dealing with such situations is a probabilistic enumeration-based technique that models the original CSP as a WCSP. In the WCSP model new constraints that represent restrictive modifications of the original constraints, are generated according to the desired granularity parameter. These new constraints are treated as soft constraints and the cost associated with their tuples and the quantity of valid tuples that they have depends on the dynamism parameters associated with the original constraint that they were induced from. The main objective is to find solutions that minimize the sum of costs associated with the new restricted constraints, where these costs represent penalizations to the tuples that do not satisfy them. The penalizations depend on the probability of change of the constraints that they come from. In this way, if a solution only does not satisfy a new constraint that is 90% likely to occur and another solution only does not satisfy a new constraint that is 10% likely to occur, the latter solution is more robust than the first solution.

This approach contributes to deal with an informed type of restrictive change over the bounds. This is a novelty brought by our work since, as far as we know, it has not been analyzed in the literature before. The most similar type of change that was previously analyzed was the constraint deletion, measured by a probability of existence (Probabilistic CSP). However, in this chapter we introduce the parameters that measure the possible restrictive modifications over the bounds (both, constraints and domains). Moreover, we propose a probabilistic technique that it is able to find robust solutions according to the initial known information about this type of changes. Both, the probabilistic approach introduced and the parameters that are able to gather this type of changes, represent an innovation in the literature for dealing with this type of change, which is common among the problems with ordered domains (see Section 3.1).

The main limitation of this approach is the same that many other proactive approaches that search for robust solutions: they require detailed dynamism information. This approach is a probabilistic approach that gathers detailed information in two parameters associated with each constraint. This fact represents a strong difficulty when this information is unknown. However, if the information availability is partial, that is to say that only the dynamism data of certain constraints is known, it is possible to model their dynamism and to find robust solutions according to this known data. Unfortunately, the solutions found can not be guaranteed to be robust for restrictive modifications over the constraints for which we do not possess dynamism information. The latter fact has motivated that the rest of approaches presented in this dissertation do not require extra detailed data about the dynamism.

Chapter 5

Enumeration-based Technique for Finite Linear CSPs

This chapter introduces an enumeration-based technique that models robustness and stability in finite linear CSPs as WCSPs (Climent et al., 2011). Unlike the approach introduced in Chapter 4, this approach is non-probabilistic. The probabilistic enumeration-based technique is very effective for finding robust solutions based on the information of the dynamism associated with the CSP. However, this dynamism information dependency is problematic when there is a lack of such data (see Section 1.5). Hence, in this thesis we have focused on developing approaches for which only the information available for modeling the original CSP is necessary. Our approaches are based on assumptions that can be made for common changes associated with CSPs with ordered domains (see the motivation in Section 3.1). Thus, we have developed two enumeration-based techniques for CSPs with ordered domains where the only assumptions made about changes are those inherent in the structure of these problems. In this chapter we explain the enumeration-based technique developed specifically for finite linear CSPs. In Chapter 6 another enumeration-based approach for general purpose is presented.

The approach presented in this chapter is based on the theoretical approach of the centroid of convex solution spaces (Section 3.4.1). Figure 5.1 shows the diagram that represents the basic steps that this technique carries out. Initially, before applying the technique, the real problem is modeled as a CSP (P). Then, by using distance measures from the bounds of the solution space, a WCSP is generated ($modP$). Finally, the modeled WCSP ($modP$) is solved by a generic WCSP solver. The solution space of $modP$ is the same as the solution space of P. Furthermore, the best solution of $modP$ is considered to be one of the most robust solution for P, because this solution has a high likelihood of remaining valid when faced with restrictive modifications of the linear constraints of P.

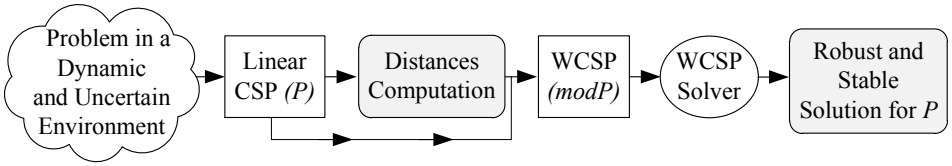


Figure 5.1: Modeling an uniformed finite linear CSP as a WCSP.

5.1 Dynamism over Linear CSP with Ordered Domains

In this chapter, we deal with problems that come from uncertain and dynamic environments that can be modeled as finite linear CSPs (see Definition 2.1.12). We recall that this type of CSP is characterized by its linear constraints and domains that can be expressed as unary linear constraints. Due to this fact, the domains of a linear CSP are numeric and therefore, they are ordered. In this context, it is reasonable to assume that the original linear constraints may undergo modifications in the form of range reductions, even if no specific data of these future changes exists (see Section 3.1). The solution space of CSPs composed of linear constraints (see Definition 2.1.11) is a convex and *n-dimensional* space, where $n = |\mathcal{X}|$ (number of variables of the CSP). In order to clarify the explanation of the dynamism over the bounds of the solution space, we show an example in a convex and *2-dimensional* space.

Example 5.1.1. *We consider a finite linear CSP composed of two variables* x_0 *and* x_1 *with static domains* $D_0 : \{3..7\}$ *and* $D_1 : \{2..6\}$ *and four dynamic constraints:*

- $C_1 : x_0 + x_1 - 12 \leq 0$
- $C_2 : x_1 - x_0 - 2 \leq 0$
- $C_3 : x_1 + x_0 - 6 \geq 0$
- $C_4 : x_0 - x_1 - 4 \leq 0$

Figure 5.2 shows the search and solution space of the CSP of Example 5.1.1. The search space is composed by the domain bounds, which are represented with discontinuous lines. The solution space is delimited by the domain bounds and the constraints (blue area). The Cartesian product of the variable domains involve 25 complete assignments (candidate solutions), of which only 21 are actual solutions. The solutions are represented by black points while the other candidates are represented by grey points. Since the constraints are considered dynamic but no detailed

data about their dynamism is known, then, it is difficult to determine which is the most robust solution. In Figure 5.2, four possible modifications, one for each original dynamic constraint, are represented (C'_1, C'_2, C'_3 and C'_4). If the original constraints of the problem change restrictively in this way, the solution space will be reduced and the tuples located in the lightest blue area, which are: $(x_0 = 3, x_1 = 3)$, $(x_0 = 3, x_1 = 5)$, $(x_0 = 4, x_1 = 2)$, $(x_0 = 4, x_1 = 6)$, $(x_0 = 6, x_1 = 2)$, $(x_0 = 6, x_1 = 6)$, $(x_0 = 7, x_1 = 3)$ and $(x_0 = 7, x_1 = 5)$ will not satisfy the new constraints C'_1, C'_2, C'_3 and C'_4 and therefore will not remain solutions of the problem.

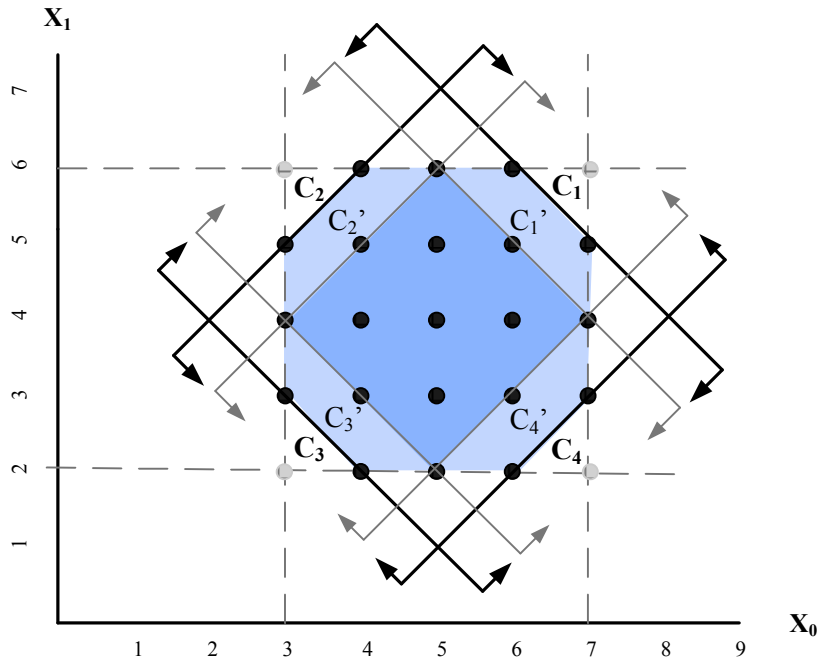


Figure 5.2: Restrictions over the solution space of the CSP of Example 5.1.1.

The solutions located close to the bounds of the original dynamic constraints have a low probability of remaining valid after changes in the problem. For this reason, the main idea of our technique is based on searching for solutions which are located as far as possible from the dynamic bounds of the solution space. In this example, it can be observed that the most robust solution is the central solution $(x_0 = 5, x_1 = 4)$.

Example 5.1.1 is a toy example in which the most robust solution can be easily observed. Nevertheless, in problems with a *n-dimensional* solution space, it is more complicated to obtain this solution. In the following section we explain the main objective for achieving the most robust solutions for *n-dimensional* finite linear CSPs.

5.2 Distances Computation and Cost Assignment

As motivated in Section 3.1, the closer the solutions are located to the bounds of the solution space, the less robust the solutions are for the CSP with ordered domains. This is due to the fact that these solutions have a low likelihood of remaining valid when faced with restrictive modifications over the linear constraints. However a solution that is located as far as possible from the bounds of the solution space has a high likelihood of remaining valid after changes, so these solutions are considered to be more robust solutions. In Section 3.4.1, it was explained the theoretical approach for meeting this criterion for convex solution spaces: the search of the closest solution to the centroid.

The aim of modeling a CSP as a WCSP is to obtain an approximation to the centroid of its solution space. Searching for the centroid of a CSP requires determining the solution space of the CSP, which is NP-hard. However, our aim is not to calculate all the solutions of the CSP, but to find a solution close to the centroid of the solution space. To this end, we have developed an enumeration-based technique based on the assignment of costs to the tuples of each dynamic $C_i \in (\mathcal{C} \cup \mathcal{DC})$, where $\mathcal{C}$ is the set of constraints and $\mathcal{DC}$ is the set of unary constraints associated with $\mathcal{D}$. The cost associated with each tuple represents its penalization, which depends on the distance of the tuples to each dynamic constraint.

In the following, the process of calculating distances and subsequently the cost assignment to the valid tuples is explained. The cost assigned is indirectly related to this distance, since a greater penalization has to be assigned to the tuples that are close to a bound of the solution space. The distances from the constraints are calculated only for the valid tuples, since the invalid tuples can not be part of a solution. Recall that, as mentioned in Chapter 2, the set of valid tuples (or partial assignments) of C_i is denoted as $T(C_i)$. For calculating these distances, we consider that this constraint is represented as a hyperplane in the search space and a complete assignment is represented as a point of the search space. A tuple t of C_i is also represented as a point in the hyperplane space associated with C_i. Thus, in order to calculate the distance of t from C_i we use the Euclidean distance from a point to an hyperplane $D(t, C_i)_{Euclidean}$. Particularly, if t is located on the C_i boundary then $D(t, C_i)_{Euclidean} = 0$. Equation 5.1 express the Euclidean distance between a point and a hyperplane.

$$D(t, C_i)_{Euclidean} = \frac{|a_1 x_1 + ... + a_n x_n + l|}{\sqrt{a_1^2 + ... + a_n^2}} \tag{5.1}$$

To assign penalizations to the valid tuples of the constraints, it is necessary to know the maximum distance from any valid tuple $t \in T(C_i)$ to C_i, because a tuple t located to a maximum distance from C_i is the most robust tuple for C_i. This is due to its likelihood to remain valid when faced with restrictive modifications over C_i is the highest. Thus, a tuple that is located at the maximal possible distance is not penalized.

The maximum distance of any $t \in T(C_i)$ from C_i is denoted as $maxD(T(C_i), C_i)$.

$$maxD(T(C_i), C_i) = max\{D(t, C_i)_{Euclidean}\} \forall t \in T(C_i) \tag{5.2}$$

The main utility of the cost functions is to prioritize among the valid tuples of each dynamic constraint. In addition, they are also the responsible for forbidding the invalid tuples. The cost function for the tuples of C_i is denoted as $C_i(t \downarrow_{var(C_i)})$, where $var(C_i)$ represents the scope of C_i . In the WCSP model, as mentioned in Definition 2.4.3, assigning the maximum cost U to a tuple t, means that t is an invalid tuple for C_i. Hence, $C_i(t \downarrow_{var(C_i)})$ assigns a cost of U to a tuple *t* that does not satisfy the constraint C_i, since it can not be part of a solution of the original CSP. On the contrary, if a tuple t satisfies C_i ($t \in T(C_i)$), then $C_i(t \downarrow_{var(C_i)}) \in [0, \lfloor maxD(T(C_i), C_i)^2 \rfloor]$.

$$C_i(t \downarrow_{var(C_i)}) = \begin{cases} \lfloor (D(t, C_i)_{Euclidean} - maxD(T(C_i), C_i))^2 \rfloor & \text{if } t \in T(C_i) \\ U, (U \approx \infty) & \text{if } t \notin T(C_i) \end{cases} \tag{5.3}$$

A penalty of zero is assigned to a tuple $t \in T(C_i)$ whose distance from C_i is the maximum possible. This is due to the fact that this tuple t is considered the most robust tuple for C_i since t is the valid tuple with the highest probability of remaining valid when faced with restrictive modifications over C_i. However, we assign the maximum penalization, which is $\lfloor maxD(T(C_i), C_i)^2 \rfloor$ to a tuple $t \in T(C_i)$ that is located on the C_i boundary, that is $D(t, C_i)_{Euclidean} = 0$. Note that t is the least robust tuple for C_i because it has the lowest probability of remaining valid when faced with restrictive modifications over C_i.

5.3 Algorithm

In this section we present an algorithm that models finite linear CSPs into WCSPs and its computational complexity.

5.3.1 Algorithm Description

The objective of this WCSP modeling is to find robust solutions for the original finite linear CSP because they are located close to the centroid of the solution space, and therefore, it has a high likelihood to resist changes over the dynamic bounds of the solution space. Given an original finite linear CSP called $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, Algorithm 5.1 generates a WCSP that we name *modP* $= \langle \mathcal{X}, \mathcal{D}, S(U), \mathcal{C}' \rangle$ which is composed of

the original CSP and the sets of costs associated with all the tuples of each dynamic constraint. When all the bounds are dynamic, $\mathcal{C}' = (\mathcal{C} \cup \mathcal{DC})$.

The associated costs are assigned according to the cost functions of Equation 5.3. The solutions of the WCSP modeled are also solutions of the original CSP and they have an associated global cost. This is due to the fact that the only associated cost that is great enough for invalidating the tuples is the cost U and this cost is only assigned by C_i to the tuples that do not satisfy it. Since the set of constraints of $\mathcal{C}'$ must be satisfied, they are hard constraints. However, among all the set of valid tuples that satisfy $C_i \in \mathcal{C}'$ a cost is assigned to each one. Thus, the global cost associated with a solution s, denoted as $\mathcal{V}(s)$, is the sum of all its associated costs for each C_i (Definition 2.4.3).

Algorithm 5.1 shows the reformulation of the finite linear CSP as a WCSP and the solving process. The assignment of the cost to each associated tuple of each constraint of the CSP is carried out in lines 2-8. First, for each C_i Algorithm 5.1 calculates the maximum possible distance of any $t \in T(C_i)$ from it (see Equation 5.2). Subsequently, a cost is assigned to each associated tuple according to the cost function described in Equation 5.3. Once this process is finished, Algorithm 5.1 shows the generation of the modeled WCSP, called $modP$ (line 9). The WCSP is solved by a general WCSP solver in line 10 by fixing a time cutoff if it has been provided. The best solution found by the WCSP solver as well as its global associated cost is returned by Algorithm 5.1 in line 11. As previously explained, the best solution of the WCSP is the solution with the minimum sum of costs, which fits with the philosophy of our enumeration-based technique that consists in assigning penalizations to the tuples. Thus, the solution with the minimum global cost is also the one that has the minimum total sum of penalizations assigned. For this reason, it is considered to be one of the most robust solutions for the original CSP, according to our dynamism assumptions.

5.3.2 Computational Complexity

The complexity of the modeling phase of Algorithm 5.1 (lines 2-8) is mainly related to the number of dynamic constraints and domains of the CSP and their number of associated tuples. Here, we consider the worst case in which all the bounds are dynamic ($C' = (\mathcal{C} \cup \mathcal{DC})$). For each constraint of $\mathcal{C}'$, a cost is assigned to each tuple. Nevertheless, as mentioned in the analysis of the computational cost in Chapter 4, the WCSP file format used in this dissertation (see Section A.1), allows the assignment of a default cost to the invalid tuples of a constraint. Thus, it is only necessary to analyze and assign a cost to the valid tuples. Since we are analyzing the worst complexity case, we consider the maximum number of valid tuples of the constraints of a CSP, which are denoted as $T(C)_{Max}$ (see Definition 2.1.10 and Equation 2.2).

The set of valid tuples of each C_i is analyzed twice by Algorithm 5.1. The first time is for calculating the maximum possible distance and the second time for assigning the cost to each valid tuple. Hence, the modeling phase of Algorithm 5.1 is

Algorithm 5.1: Algorithm for modeling an uninformed finite linear CSP as a WCSP.

Data: A CSP $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, *time_cutoff* (optional).
Result: Best solution s and its associated global cost $\mathcal{V}(s)$.

begin
  foreach $C_i \in (\mathcal{C} \cup \mathcal{DC})$ **do**
    Calculate $maxD(T(C_i), C_i))$ according to Equation 5.2
    foreach $t \in C_i$ **do**
      if $t \in T(C_i)$ **then**
        $C_i(t \downarrow_{var(C_i)}) = \lfloor (D(t, C_i)_{Euclidean} - maxD(T(C_i), C_i))^2 \rfloor$;
      else
        $C_i(t \downarrow_{var(C_i)}) = U$;
  Generate a WCSP $modP = \langle \mathcal{X}, \mathcal{D}, S(U), \mathcal{C}' \rangle$, where, $\mathcal{C}' = C_i, i \in \{1, .., (e+n)\}$;
  $(s, \mathcal{V}(s)) \leftarrow$ Solve $modP$ in *time_cutoff*;
  return $(s, \mathcal{V}(s))$;

$O(T(C)_{Max} * |\mathcal{C}'|)$. This is equivalent to $O((e+n) * T(C)_{Max})$ when all the bounds are dynamic, where $e = |\mathcal{C}|$ and $n = |\mathcal{DC}| = |\mathcal{X}|$.

5.4 Example

In this section the WCSP modeling process of the CSP of Example 5.1.1 is explained. The obtained solutions are shown in decreasing robustness order according to their corresponding global cost $\mathcal{V}(s)$ (see Table 5.1 (right)). The solutions with the lowest associated costs are considered to be the most robust solutions for this linear CSP.

Following, the cost assignment of the constraint C_1 to its valid tuples is explained in detail in order to show to the reader the process of the cost assignment. In Table 5.1 (left), the set $T(C_1)$ can be observed for the modeled WCSP and its $C_1(t \downarrow_{var(C_1)})$. The application of Equation 5.2 to C_1 results on the maximum possible distance of any tuple from a constraint C_1:

$$maxD(T(C_1), C_1) = max\{\frac{|x_0 + x_1 - 12|}{\sqrt{1^2 + 1^2}}\} \forall t \in T(C_1). \tag{5.4}$$

The maximum distance value is obtained for the tuple $t_1 = (x_0 = 3, x_1 = 2)$:

$$maxD(T(C_1), C_1) = \frac{|3 + 2 - 12|}{\sqrt{1^2 + 1^2}} = \frac{7}{\sqrt{2}} = 4.949 \tag{5.5}$$

Applying $maxD(T(C_1), C_1)$ to Equation 5.3, it is obtained the cost function of C_1:

$$C_1(t \downarrow_{var(C_1)}) = \begin{cases} \lfloor (D(t, C_1)_{Euclidean} - 4.949)^2 \rfloor & \text{if } t \in T(C_1) \\ U, (U \approx \infty) & \text{if } t \notin T(C_i) \end{cases} \tag{5.6}$$

The interval of costs that C_1 assigns to the valid tuples is calculated by substituting $D(t, C_1)_{Euclidean}$ in Equation 5.6 for the maximum distance possible, which is 4.949, and for the minimum distance possible, which is zero. Therefore, $C_1(t \downarrow_{var(C_1)}) \in [0, 24]$ for $t \in T(C_1)$.

Here, we describe the cost assignment of the solution $s_{21} = (x_0 = 7, x_1 = 5)$. The cost assigned by the constraint C_1 to this tuple is the maximum possible, which is 24. This is due to the solution s_{21} being located on the bound of the constraint C_1, therefore the Euclidean distance of s_{21} from C_1 is zero. Thus, applying the Equation 5.3, $C_1(s_{21}) = \lfloor (0 - maxD(T(C_1), C_1))^2 \rfloor = \lfloor -4.949^2 \rfloor = 24$. For this example, there is the coincidence that $maxD(t_i, C_i)$ are equal for all the constraints. Thus, the distances of this solution to the other constraints are:

- $D(s_{21}, C_2)_{Euclidean} = \frac{|5-7-2|}{\sqrt{2}} = 2.828$
- $D(s_{21}, C_3)_{Euclidean} = \frac{|5+7-6|}{\sqrt{2}} = 4.242$
- $D(s_{21}, C_4)_{Euclidean} = \frac{|7-5-4|}{\sqrt{2}} = 1.414$

Furthermore, the costs assigned by these constraints are:

- $C_2(s_{21}) = \lfloor (2.828 - 4.949)^2 \rfloor = 4$
- $C_3(s_{21}) = \lfloor (4.242 - 4.949)^2 \rfloor = 0$
- $C_4(s_{21}) = \lfloor (1.414 - 4.949)^2 \rfloor = 12$

After the computation of all the costs, the global cost assigned to the solution is the sum of all of them, so $\mathcal{V}(s_{21}) = C_1(s_{21}) + C_2(s_{21}) + C_3(s_{21}) + C_4(s_{21}) = 24 + 4 + 0 + 12 = 40$, which is the maximum global associated cost.

Table 5.1 (right) shows the solutions of the CSP of Example 5.1.1 and their global costs. They have been obtained by solving the modeled WCSP with a WCSP solver. It can be observed that among the set of solutions there are three different global costs: 32, 36 and 40. Since they represent penalizations due to their distance to each bound of the solution space, the lower the global cost of a solution is, the more robust

Table 5.1: Constraint C_1 of the WCSP (left) and solutions of the CSP(right).

$T(C_1)$		$C_1(t \downarrow_{var(C_1)})$	Solutions		$\mathcal{V}(s)$
x_0	x_1		x_0	x_1	
3	2	0	**5**	**4**	**32**
3	3	0	**5**	**3**	**32**
3	4	2	**5**	**5**	**32**
3	5	4	**4**	**4**	**32**
3	6	8	**6**	**4**	**32**
4	2	0	6	3	36
4	3	2	4	5	36
4	4	4	4	3	36
4	5	8	6	5	36
4	6	12	3	3	40
5	2	2	3	4	40
5	3	4	4	2	40
5	4	8	5	6	40
5	5	12	6	2	40
5	6	18	4	6	40
6	2	4	5	2	40
6	3	8	3	5	40
6	4	12	6	6	40
6	5	18	7	3	40
6	6	24	7	4	40
7	2	8	7	5	40
7	3	12			
7	4	18			
7	5	24			

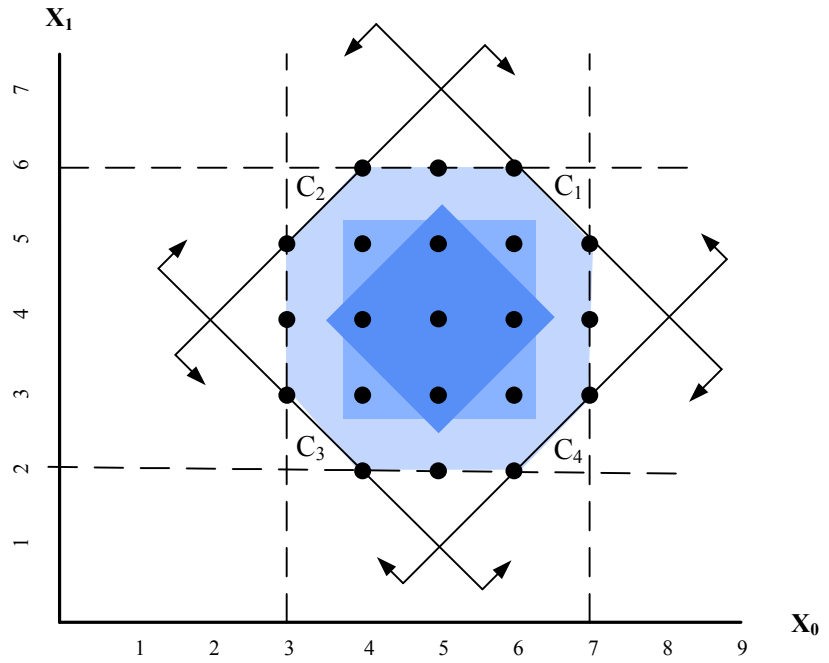


Figure 5.3: Solutions of the CSP of Example 5.1.1 and their robustness.

the solution is. Figure 5.3 shows the 3 robustness areas obtained. As mentioned, the darker the blue area is, the more robust the solutions located in this area are. It can be observed that the approximation of the robustness is correct, because solutions located on the dynamic bounds of the solution space ($(x_0 = 3, x_1 = 3)$, $(x_0 = 3, x_1 = 5)$, $(x_0 = 4, x_1 = 2)$, $(x_0 = 4, x_1 = 6)$, $(x_0 = 6, x_1 = 2)$, $(x_0 = 6, x_1 = 6)$, $(x_0 = 7, x_1 = 3)$ and $(x_0 = 7, x_1 = 5)$) have been classified as the least robust solutions, in contrast to the central solutions. The solutions located on a dynamic bound have the maximum global cost of 40 (see Table 5.1, right). It means that the sum of their penalizations is the maximum possible for this CSP.

One of the best solutions found for the modeled WCSP (the solution s that has the minimum $\mathcal{V}(s)$), corresponds to the solution located in the centroid of the polygon that represents the solution space of P: $(x_0 = 5, x_1 = 4)$. This is shown in figure 5.4. Note that this solution has the minimum penalization associated, which is $(\mathcal{V}((x_0 = 5, x_1 = 4)) = 32)$. Thus, the enumeration-based technique presented in this chapter, has been able to classify the solution located in the centroid of the solution space as one of the most robust solutions.

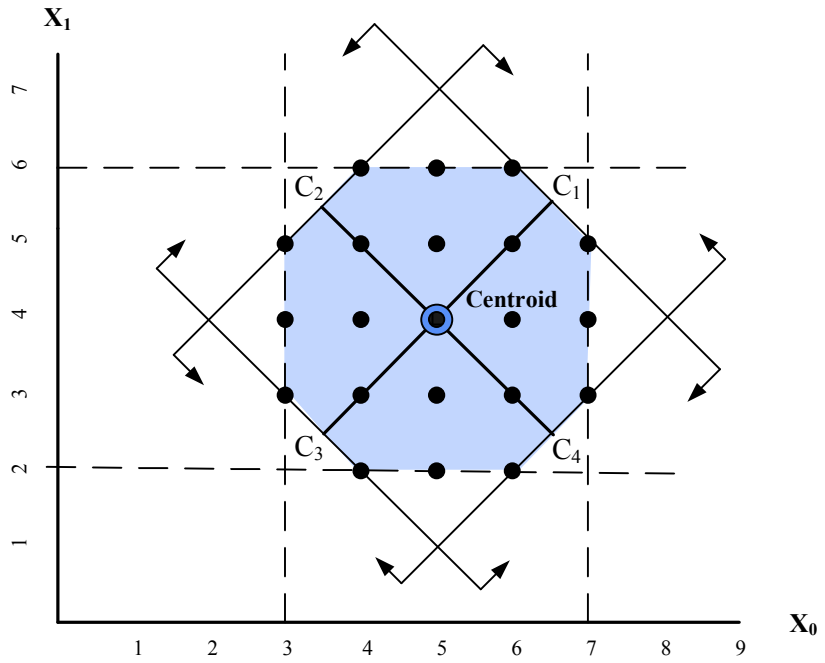


Figure 5.4: Example of the CSP *P* and its centroid.

5.5 Summary and Limitations

In this chapter we cope with linear CSPs that model dynamic problems whose elements are ordered according to a certain relationship. These problems do not have detailed information about the future changes that they can undergo. However, as motivated in Section 3.1, there exists a common type of change associated with problems with ordered domains: it takes the form of restrictive modifications over the bounds of the solution space (which is composed by the constraints and domains). The peculiarity of the linear CSPs is that all their constraints and domains can be modeled as intensional linear constraints and as a consequence, the solution spaces are always convex. This fact makes possible the use of the theoretical approach explained in 3.4.1, which is to search for the solution located as close as possible to the centroid of the solution space. In this chapter we introduce an enumeration-based approach that models the original linear CSP as a WCSP in which the valid tuples of each constraint have an associated cost that depends on the distance of the tuple from the linear constraint. This is calculated by using the equation of the Euclidean distance of a point from a hyperplane.

The main advantage of this specific enumeration-based approach over the approach introduced in Chapter 6, which can be applied for any type of constraints, is the lower computational time. The computation of the distance equation is quicker than the neighbours feasibility checking (this theoretical approach is explained in

Section 3.4.2) used in the further approaches introduced in this thesis. Some experiments about the latter statements are shown in Chapter 9. The disadvantage of the enumeration-based technique for linear CSP introduced in this chapter is the specifiability of this approach itself. The fact that it only can be applied in the area of linear CSPs makes it useless when faced with other types of CSPs, such as CSPs with non-convex solution spaces and CSPs with any constraint extensionally represented. Even so, it provides a quick answer when the problems can be modeled as finite linear CSPs.

Chapter 6

Enumeration-based Technique by Coverings

This chapter presents an enumeration-based technique by coverings that models CSPs with ordered domains as WCSPs (Climent et al., 2012b, 2013c,b), with the objective of finding robust and stable solutions. Contrarily to the approach introduced in Chapter 5, the methodology presented in this chapter addresses both CSPs with convex and non-convex solution spaces (see Section 2.1.5). Both enumeration-based techniques share that they are non-probabilistic, and do not require extra detailed information about the dynamism and they both consider the dynamism assumptions explained in Section 3.1. The approach presented in this chapter deals with non-convex solution spaces and therefore the theoretical approach of the centroid of the solution space (used in Chapter 5) does not hold. Instead, we use the theoretical approach based on the 'onion model' (see Section 3.4.2). This approach is based on checking the feasibility of the neighbourhood of the solutions, since they provide information about the distances to the bounds of the solution space.

Figure 6.1 shows the diagram that represents the basic steps that this technique carries out. The modeling process is quite similar to the one explained for linear CSPs (see Figure 5.1). The major difference lies in the computation of the costs associated with the valid tuples. After modeling the real problem as a CSP (P), we apply our approach for checking the feasibility of the neighbours of the valid tuples, which is also referred to as *covering* computation. Thus, a WCSP is generated ($modP$) by associating costs to the tuples based on their coverings. Finally, the modeled WCSP ($modP$) is solved by a generic WCSP solver. The solution space of $modP$ is the same as the solution space of P. Furthermore, the best solution of $modP$ is considered to be one of the most robust solution for P, because this solution has a high likelihood of remaining valid given the dynamism assumptions associated with CSPs with ordered domains.

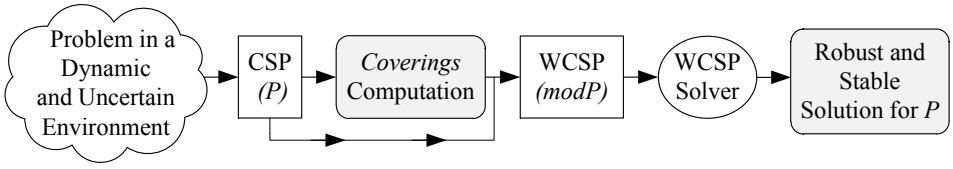


Figure 6.1: Modeling an uniformed CSP a WCSP by using Coverings.

6.1 The Concept of Coverings

According to the assumptions in Section 3.1, the closer the solutions are located to the bounds of the solution space, the less robust the solutions are for the CSP with ordered domains. In Section 3.4.2 we explained the theoretical approach for finding robust solutions for such environment and for both convex and non-convex spaces. It is based on the 'onion topology' that has been proved to provide robustness to dynamic networks in (Herrmann et al., 2011). We defined the layer of this 'onion structure' as a convex hull of valid neighbour tuples that surround a specific tuple. Therefore, the core of an 'onion structure' is the most robust part of the structure. Since surrounding layers protect it against perturbations, and the core is located at the furthest point from the outer layer. The same is true for the solutions of the CSPs with ordered domains that our model is concerned with: 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').

In the following, we present a formal framework using some of the definitions and notations related to the topological and metric spaces associated with CSPs that model problems for which there exists a significant order over their elements (see Section 2.2). As mentioned, in metric spaces (this includes search and solution spaces of CSP with ordered domains) 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 use the Chebyshev distance, which measures the maximum absolute differences along any coordinate dimension of two vectors (see Equation 2.5). The main reason for selecting this distance metric is that it distinguishes between hypercubes in n-dimensional spaces. In particular, the corners of a cube are at the same distance from the central point as the edges, a feature not obtained with Euclidean distance metric. By checking areas of satisfiability inside these hypercubes, we can ensure minimum distances to the bounds, which is used for the robustness computation. Subsequently, we formally define the concept of covering.

Definition 6.1.1. *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 6.1.1 the following property can be deduced: k-covering(t)$\supseteq$ (k-1)-

covering(t).

To compute the coverings of CSPs with ordered non-integer domains, a monotonic function has to be applied in order to map the elements (it must also be an order preserving function). Example 2.2.1 shows an example of mapping function for symbolic domains. Below we give an example for rational domains.

Example 6.1.1. *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 6.1.2. $maxTup(k, |t|)$ *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 6.1.1. *The maximum possible number of tuples inside a k-covering(t) is:*

$$maxTup(k, |t|) = (2k + 1)^{|t|} - 1 \tag{6.1}$$

Proof. The exclusion of the central tuple t from the n-dimensional generalization of the Moore neighbourhood (an adaptation from (Goles and Martínez, 1990)) is equivalent to maxTup$(k, |t|)$. The number of elements in a Moore neighbourhood in a 2-dimensional space is $(2k + 1)^2$. For a generalization of this formula, we 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 a 2-dimensional space, Moore neighbourhood is $(2 \cdot 1 + 1)^2 = 9$. It is straightforward that for t dimensions $\mathbb{R} \times ...^{t} ... \times \mathbb{R}$: Moore neighbourhood is $(2k + 1)^{|t|}$. Thus, if we remove the tuple $(p_1, p_2, ..., p_t)$, the number of neighbours of this tuple is: maxTup$(1, |t|) = (2k + 1)^{|t|} - 1$. When $k > 1$, the same argument holds, since we are just adding tuples on either side of the central tuple in each dimension. So, we can conclude that maxTup$(k, |t|) = (2k + 1)^{|t|} - 1$. □

From Proposition 6.1.1, it is obvious that |k-covering(t)| ≤maxTup$(k, |t|)$, where |k-covering(t)| is the covering cardinality.

Definition 6.1.3. *A k-covering(t) is complete if |k-covering(t)| = maxTup*$(k, |t|)$.

If k-covering(t) is complete, it means that t is located at a Chebyshev distance of at least k from the bounds, because inside the k-covering(t) all the tuples are valid. On the contrary, if at least one invalid tuple is inside 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 neighbours of t is invalid, this 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.

Example 6.1.2. *Here we explain an example of the coverings concepts and definitions presented in this section. Figure 6.2 represents the solution space of Example 3.1.1). We recall that this 2-dimensional solution space is composed by* 29 *solutions (black points). It can be observed in this figure the 1-covering(t) and 2-covering(t) for the highlighted solution* $t = (x_0 = 5, x_1 = 4)$. *We can see 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)*$| = 6$ *and |1-covering(w)*$| = 2$. *This information is equivalent to the closeness of both tuples to the bounds of the solution space. Note that the tuple* v *only has bounds on its left and left-up. On the contrary, the tuple* w *has bounds on its right,*

right-up, right-down, left-up, up and down sides.

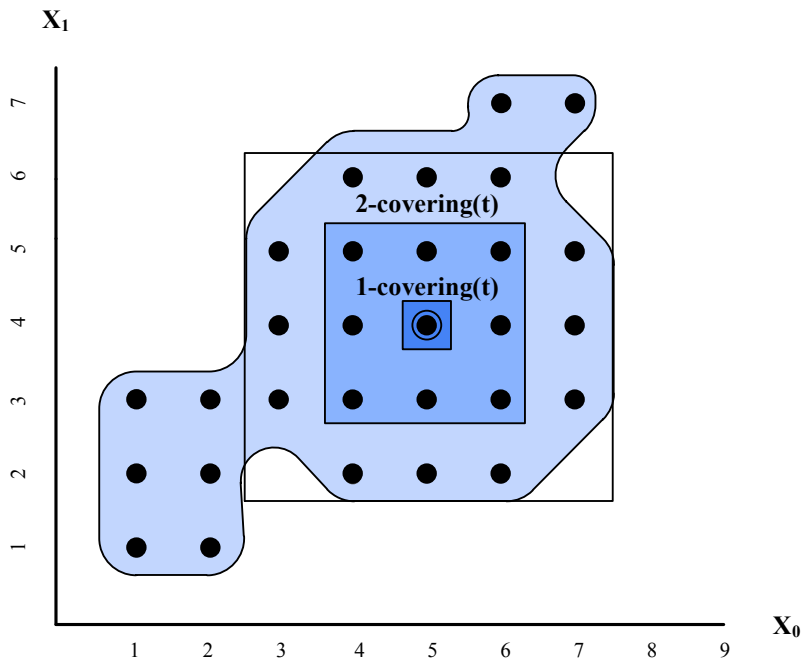


Figure 6.2: Coverings of a solution of the CSP of Example 3.1.1.

6.2 Algorithm for Calculating Coverings

Searching for a solution that is completely surrounded by 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 CSP solutions is not generally viable because it would be extremely time consuming due to the high combinatoriality of CSPs. Consequently, we have developed an approach that is based on modeling the coverings of each valid tuple of constraints of the CSP as a WCSP. The algorithm that performs this procedure and its computational complexity are described in this section. We would like to remark that solving WCSPs is also NP-hard in general (since it is an optimization problem). However, this can be alleviated using Branch & Bound techniques combined with Arc Consistency (AC) and heuristics. In addition, the Branch & Bound is an 'anytime' algorithm, so we can limit the search time by fixing a time cutoff.

6.2.1 Algorithm Description

Here we present Algorithm 6.1, which calculates the coverings of the valid tuples, after first carrying out a global arc-consistency (GAC) 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 and effective GAC3 (Mackworth, 1977b) (Algorithm 2.1). However, other consistency techniques could also be applied (see Section 2.5 for some examples).

For calculating coverings, Algorithm 6.1 begins with $k = 1$ (1-covering(t)), increasing this value by one unit in each iteration until reaching the K upper bound. In each iteration, $\forall t \in T(C_i)$, $\forall C_i \in (\mathcal{C} \cup \mathcal{DC})$, k-covering($t$) is computed, iff $k = 1$ or (k-1)-covering(t) is complete (see Definition 6.1.3).

Definition 6.2.1. *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 the last-covering(t) term is equal to $min(K, (k + 1))$, if k-covering(t) is the highest complete covering of t.*

Algorithm 6.1 returns the size of last-covering(t) computed for each valid tuple t of each constraint and domain of the CSP, which as defined above, is a measure of the robustness of each tuple of each constraint. 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 kth 'onion layer'). This is because the neighbours of the lower coverings have already been calculated in previous iterations and stored in last-covering(t). If a tuple t has an incomplete covering, which is indicated by the boolean variable $t.incomplete$ (lines 13 and 15), the algorithm does not compute its further coverings.

Algorithm 6.1 first initializes some necessary structures (lines $2 - 7$). 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 $T(C_i)$; however, with the ordering that we have selected we can reduce computation time by exploiting the symmetry of the neighbours relation. In this way, only a subset of possible neighbours of each valid tuple $t \in T(C_i)$ need to be checked. This reduced set is 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. More formally, for a given t, the reduced set is composed of $y \in T(C_i) : y < t \wedge d(t_1, y_1)_{Chebyshev} \leq k$

Algorithm 6.1: Algorithm for computing the coverings of a CSP

Data: A CSP $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$ and K
Result: |last-covering(t)| $\forall t \in T(C_i) \forall C_i \in (\mathcal{C} \cup \mathcal{DC})$

```
1  GAC3(P);
2  k ← 1;
3  foreach C_i ∈ (C ∪ DC) do
4      T(C_i) ← Ordered list of valid tuples of C_i;
5      foreach t ∈ T(C_i) do
6          |last-covering(t)| ← 0;
7          t.incomplete ← false;
8  repeat
9      foreach C_i ∈ (C ∪ DC) do
10         foreach t ∈ T(C_i) do
11             foreach {y ∈ T(C_i): y < t ∧ d(t_1, y_1)_Chebyshev ≤ k } do
12                 if isNewNeighbour(k,t,y) then
13                     if t.incomplete = false then
14                         |last-covering(t)| ← |last-covering(t)| + 1;
15                     if y.incomplete = false then
16                         |last-covering(y)| ← |last-covering(y)| + 1;
17                     if ∀i ∈ [1, |y|], d(t_i, y_i)_Chebyshev = k∧ not
                       isComplete(k,|last-covering(y)|,|y|) then
18                         y.incomplete ← true;
19     k ← k + 1;
20 until k > K;
21 return |last-coverings|
```

(line 11). We would like to point out that other orderings are also possible, as for example the full lexicographical order. Here, we select the partial order because it is quite discriminative and only requires the time to compute the difference of two values for each tuple ordering, while the full ordering requires a higher computing time.

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 (lines 3 and 5). If procedure `isNewNeighbour` returns `true`, Algorithm 6.1 increments the value of the |last-covering| for each of the two tuples involved, but only if the tuple does not have any incomplete covering. This last-mentioned condition is checked in the procedure `isComplete`, which, first calculates the maximum number of tuples of k-covering(t): maxTup($k, |t|$) (see Equation 6.1) and then checks if |last-covering(t)| is equal to this value.

Procedure isNewNeighbour(k,t,y) : Boolean

```
1 equalDist ← false;
2 for i ← 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 return |last-covering(t)| ≥ maxTup(k, |t|)
```

Algorithm 6.1 fixes the value of $y.incomplete$ to `true` in line 18 if its analyzed covering is incomplete and y has had all the possible neighbours already analyzed. This condition is checked in line 17 by checking if all the variable values differences with respect to t are equal to k (this means that t is the greatest neighbour of y according to the value ordering). For the remaining tuples the completeness is not checked because there may be other neighbours that will be analyzed later due to their order.

6.2.2 Computational Complexity

The complexity of Algorithm 6.1 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 on the computational cost. In the following, we analyze the impact of these factors on the steps that the algorithm carries out.

The first step of the algorithm is a GAC3 process, 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 6.1 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 cardinality of the largest possible set of valid tuples that we have used in previous computational analysis is $T(C)_{Max}$ (see Definition 2.1.10 and Equation 2.2). However, for this computational cost, we separate its individual terms in order to group similar terms in the final cost equation. Thus, we consider worst-case upper bound of valid tuples as $d^r * (1 - t_M)$, 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 constraints and domains is $O((e + n) * d^r * (1 - t_M) * log(d^r * (1 - t_M)))$, where $n = |\mathcal{DC}| = |\mathcal{X}|$.

Finally, the k-coverings are computed for each valid tuple t of each $C_i \in (\mathcal{C} \cup \mathcal{DC})$. As noted above, our 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, and in the worst case is equal to $((k+1) * d^{r-1} * (1 - t_M))$. Note that the reason for subtracting one unit from r in d^{r-1} is because one variable's value is fixed to the value difference of k (the first variable), while the values of the remaining variables can have any combination of domain values. 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 K parameter is equal to $\lfloor \frac{d}{2} \rfloor$ (all tuples are valid for all 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 total number of constraints and domains ($e + n$) and the worst case of K. The result is: $O(d^{2r} * (1 - t_M)^2 * k * r * (e + n) * \frac{d}{2})$.

For calculating the total computational cost of Algorithm 6.1, we sum the three steps: GAC3 process, ordering of the reduced set of tuples and calculating the coverings, and we group common terms, which results in: $O((er^3d^{r+1}) + (e+n) * d^r * (1 - t_M) * ((d^r * (1-t_M) * k * r * \frac{d}{2}) + log(d^r * (1-t_M))))$. Note that the number of valid tuples of the constraints is crucial in the computational cost of Algorithm 6.1. For this reason, the GAC3 process (line 1), which restricts the search space by deleting inconsistent values, has a large impact on the reduction of the computational cost.

6.3 Modeling CSPs as WCSPs by Coverings

In this section we introduce how to model robustness and stability in a CSP as a WCSP by using the information about the |last-coverings| of the valid tuples that Algorithm 6.1 provides. Thus, our enumeration-based technique models a CSP as a WCSP based on the |last-coverings| of the valid tuples for each constraint. As argued earlier and in Chapter 3, under many conditions |last-covering($s \downarrow_{var(C_i)}$)| for a solution s is a reasonable measure of its robustness for C_i, and it is moreover reasonable to use the sum of |last-covering($s \downarrow_{var(C_i)}$)| for each $C_i \in (\mathcal{C} \cup \mathcal{DC})$ as an approximation of the robustness of s for the CSP.

We recall that 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. Although there are other valued CSPs that could conceivably be used to model robustness, the WCSP adequately incorporates the enumeration aspect of coverings and other advantages, unlike the other valued CSP models (see Section 3.5 for a more detailed explanation).

6.3.1 Cost Assignment

The modeling process begins by assigning a cost to each valid tuple t involved in each constraint, which represents its penalty as a function of its |last-covering(t)|. Tuples with the highest last-covering for a $C_i \in (\mathcal{C} \cup \mathcal{DC})$ have the lowest associated cost, because this value indicates the minimum distance of t from the bounds of C_i. Hence, the maximum last-covering of a constraint is subsequently defined.

Definition 6.3.1. *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 6.2). The penalty is based on the size of the last-covering of this tuple in comparison with the maximum size of last-covering for the analyzed constraint.

$$p_i(t) = \text{max-|last-covering}(C_i)| - |\text{last-covering}(t)| \tag{6.2}$$

However, this penalty is not the final cost assigned to the valid tuples. Since the maximum possible size of a covering increases with the arity of the tuples (see Equation 6.1), with unnormalized penalties constraints with higher arities would have greater 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 (see Section 3.1 in page 45). By using a normalization process, we can achieve the same cost range for all the constraints.

To obtain normalized scores, we use the maximum penalty assigned to the tuples of each constraint and the maximum penalty assigned to tuples across all the constraints. The cost function of the constraints $C_i \in (\mathcal{C} \cup \mathcal{DC})$ assign the normalized cost to each tuple t, which is following defined.

$$C_i(t \downarrow_{var(C_i)}) = \begin{cases} 0 & \text{if } t \in T(C_i) \wedge p_i(t) = 0 \\ \left\lfloor \frac{p_i(t) * max\{p_j(x), \forall j \in [1...e] \forall x \in T(C_j)\}}{max\{p_i(y), \forall y \in T(C_i)\}} \right\rfloor & \text{if } t \in T(C_i) \wedge p_i(t) \neq 0 \\ U, (U \approx \infty) & \text{if } t \notin T(C_i) \end{cases} \tag{6.3}$$

In line with the version of WCSP used (see Definition 2.4.3), $C_i(t \downarrow_{var(C_i)})$ assigns a cost of U to each tuple t that does not satisfy the constraint C_i because it is not a partial solution. Note that for valid tuples $C_i(t \downarrow_{var(C_i)}) \in [0, max\{p_j(x), \forall j \in [1 \dots e] \forall x \in T(C_j)\}]$. The cost associated with a valid tuple t whose |last-covering(t)| = max-|last-covering(C_i)| $(p_i(t) = 0)$ is 0. This tuple is not penalized because it has the highest likelihood of remaining valid when faced with future changes in C_i. However, 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 because it is located in its bound), t receives the maximum possible cost: $max\{p_j(x), \forall j \in [1 \dots e] \forall x \in T(C_j)\}$.

6.3.2 Modeling Algorithm

As previously mentioned, the cost assignment of the constraints to their corresponding tuples is intended for penalizing the tuples according to their location with respect to the bounds of each constraint/domain. By means of the cost assignment, we model the original CSP as a WCSP, which is solved using a WCSP solver. As mentioned previous chapters, the solution space of the modeled WCSP is the same solution space as the original CSP. The solutions of the WCSP have an associated cost that depends on the penalizations assigned by the soft constraints. Furthermore, the best solution s with the minimum global cost, denoted as $\mathcal{V}(s)$, (see Definition 2.4.3) is taken to be one of the most robust solutions for the original CSP according to the cost functions specified, which depend on our dynamism assumptions.

Following, we introduce an algorithm for modeling robustness in CSPs as WCSPs, which is a generalization of Algorithm 5.1 in Chapter 5. Thus, Algorithm 5.1 assigns a cost to each tuple associated with the dynamic constraints and domains of the original CSP. Depending on the cost functions ($C_i(t \downarrow_{var(C_i)})$) applied in Algorithm 5.1, we are modeling the original CSPs as WCSPs according to different criteria. When we apply Equation 6.3, we are applying the enumeration-based technique introduced in

this chapter, which is based on the coverings of the tuples. However, if we apply Equation 5.3, we are applying the enumeration-based technique based on euclidean distances for linear constraints.

Algorithm 6.2: General algorithm for modeling an uninformed CSP as a WCSP.

Data: A CSP $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle$, *time_cutoff* (optional).
Result: Best solution s and its associated global cost $\mathcal{V}(s)$.
begin
  foreach $C_i \in (\mathcal{C} \cup \mathcal{DC})$ **do**
    foreach $t \in C_i$ **do**
      Assign a cost $C_i(t \downarrow_{var(C_i)})$ to t;
  Generate the WCSP $modP = \langle \mathcal{X}, \mathcal{D}, S(U), \mathcal{C}' \rangle$, where, $\mathcal{C}' = C_i, i \in \{1, .., (m+e)\}$;
  $(s, \mathcal{V}(s)) \leftarrow$ Solve $modP$ in *time_cutoff*;
  return $(s, \mathcal{V}(s))$;

6.4 Examples

In order to clarify the concepts, definitions and algorithms introduced in this chapter, we explain in detail two different examples. Firstly, we present a simple 2-dimensional example and later we introduce a more complicated 3-dimensional example. Both of them are non-convex spaces. For every example we show the costs assigned to the solutions of the problems for every dynamic constraint and domain of the modeled WCSP of the enumeration-based technique that computes the coverings.

6.4.1 2-dimensional Example

Example 6.4.1. *We consider a CSP composed of two variables x_0 and x_1 with domains $D_0 : \{2..6\}$ and $D_1 : \{1..5\}$, respectively. Figures 6.3 show the CSP representation. The domains associated with x_0 and x_1 are represented in Figure 6.3(a) and Figure 6.3(b), respectively. There exist three extensional constraints C_0, C_1 and C_2 (Figure 6.3(c), Figure 6.3(d) and Figure 6.3(e), respectively). The valid tuples of the constraints and domains are represented with black points; the invalid tuples are*

represented with grey points.

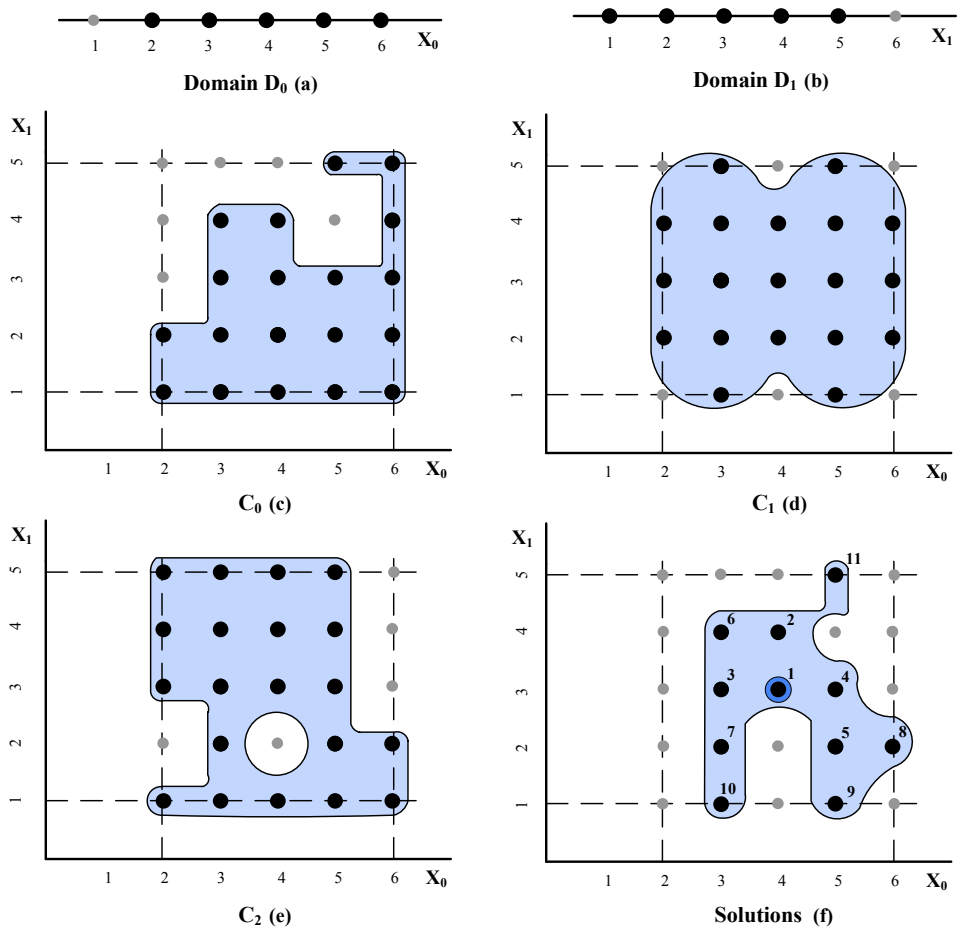


Figure 6.3: Representation of the 2-dimensional CSP of Example 6.4.1.

The solution space of the CSP described in Example 6.4.1 is represented in Figure 6.3(f). This figure shows all the solutions ranked by the level of robustness, as assessed by our enumeration-based technique by coverings. Table 6.1 also shows the complete set of solutions of the CSP of Example 6.4.1 ($\mathcal{S}(CSP)$) 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 $|\text{last-covering}(s)|$ for the complete solution space of the CSP of Example 6.4.1. Note that in this case, it is possible to compute the complete solution space because we are dealing with a toy problem. As expected, the robustness results obtained 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 6.1 and note that |last-coverings| is shorted as |cov|).

Table 6.1 also shows the costs assigned by the constraints (C_i) and domains (D_i), whose sum is $\mathcal{V}(s)$ (see Equation 2.6). To clarify the process of cost assignment, we explain it in detail for the specific case of the constraint C_0 to the solution $s_3 = (x_0 = 3, x_1 = 3)$. We compute $K = 2$ The penalty associated with the solution s_3 (before the normalization process) is $p_0(s3) = 16 - 6 = 10$ (see Equation 6.2), 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 (following normalization) is: $C_0(s_3) = \lfloor \frac{10*15}{14} \rfloor = 10$ (see Equation 6.3), 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 this CSP) 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).

Table 6.1: Solutions of the CSP of Example 6.4.1 ordered by their robustness.

s_i	$\mathcal{S}(CSP)$	|cov(s)|	$\mathcal{V}(s)$	$D_0(s)$	$D_1(s)$	$C_0(s)$	$C_1(s)$	$C_2(s)$
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

The best solution found is $s1 = (x_0 = 4, x_1 = 3)$ and its |last-covering($s1$)| in the solution space equals 6, which is the highest for the CSP of Example 6.4.1. This fact can be observed in Figure 6.3(f), in which the solution with the maximum number of neighbours solutions, that is to say 6 neighbours, is the solution s_1 (highlighted in the figure). As previously mentioned, the solution s with the highest |last-covering(s)| in the solution space is the solution located furthest away from the solution space bounds. As a result, its likelihood of remaining valid in the face of future restrictive modifications over the bounds of the solution space, is higher than for any other solution. Therefore, it is the most robust solution for the original CSP according to our dynamism assumptions for CSPs with ordered domains. 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| $= 1$, means that each of these solutions only has 1 neighbour solution in the solution space (see Figure 6.3(f)). And this is the lowest size of last-covering for this CSP. Thus, it is very probable that $s10$ and $s11$ will become invalid after restrictive modifications over the original constraints and domains of the CSP.

6.4.2 3-dimensional Example

Example 6.4.2. *We consider a CSP composed of three variables x_0, x_1 and x_2 with domain D_i : $\{0, 1, 2\}$. Figures 6.4 show the CSP representation. There are three extensional constraints: C_0 (3-ary), C_1 and C_2 (binary constraints) (Figures 6.4(a), 6.4(b) and 6.4(c), respectively). The valid tuples of the constraints and domains are represented with points; the invalid tuples are represented with crosses. We have used a different way of representing invalid tuples in this figure due to the difficulty of plotting a 3-ary constraint.*

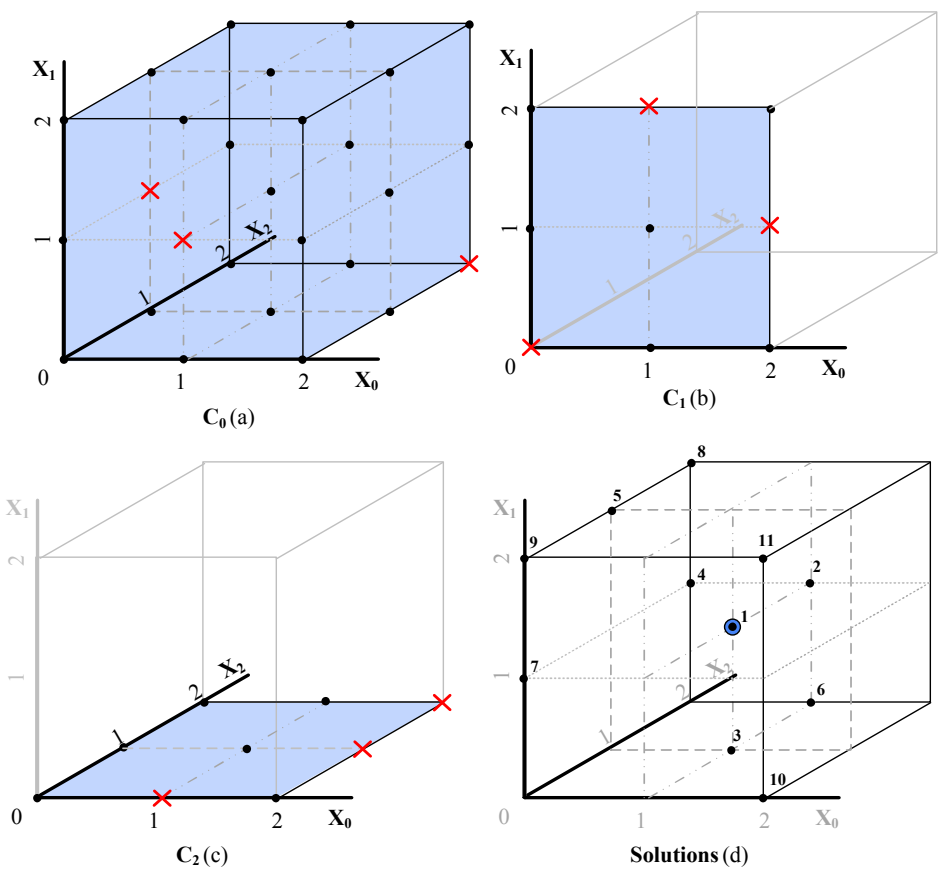


Figure 6.4: Representation of the 3-dimensional CSP of Example 6.4.2.

The solution space of the CSP described in Example 6.4.2 is represented in Figure

Table 6.2: Solutions of the CSP of Example 6.4.2 ordered by their robustness.

s_i	$\mathcal{S}(CSP)$	|cov(s)|	$\mathcal{V}(s)$	$D_0(s)$	$D_1(s)$	$D_2(s)$	$C_0(s)$	$C_1(s)$	$C_2(s)$
1	(1,1,1)	10	0	0	0	0	0	0	0
2	(1,1,2)	6	35	0	0	18	8	0	9
3	(1,0,1)	6	36	0	18	0	9	9	0
4	(0,1,2)	6	67	18	0	18	13	9	0
5	(0,2,1)	6	67	18	18	0	14	13	4
6	(1,0,2)	4	68	0	18	18	14	9	9
7	(0,1,0)	4	72	18	0	18	14	9	13
8	(0,2,2)	4	93	18	18	18	17	13	9
9	(0,2,0)	3	98	18	18	18	18	13	13
10	(2,0,0)	2	102	18	18	18	17	13	18
11	(2,2,0)	1	107	18	18	18	17	18	18

6.4(d). In the same way than Example 6.4.1, the order of the solutions obtained by our enumeration-based technique by coverings is represented in the figure as well as in the table (Table 6.2 in this case). As expected, the robustness results obtained match with the |last-coverings| order 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 6.2).

Following, we present an example of cost assignment in this 3-dimensional example. In particular, we explain the cost assignment of the constraint C_0 to the solution $s_2 = (x_0 = 1, x_1 = 1, x_2 = 2)$, in detail. Considering $K = 1$, the penalty associated with the solution s_2 (before the normalization process) is $p_0(s_2) = 23 - 15 = 8$ (see Equation 6.2), since max-|last-covering(C_0)| $= 23$ (|last-covering($(1, 1, 1)$)| $= 23$ for C_0, which is the maximum for C_0) and |last-covering(s_2)| $= 15$ for C_0. The associated cost (considering the normalization) is: $C_0(s_2) = \lfloor \frac{8*18}{18} \rfloor = 8$ (see Equation 6.3). Because $max\{p_0(y), \forall y \in T(C_0)\} = 18$ ($p_0((0, 2, 0)) = 23-5 = 18$) and this cost is also the maximum penalization for *R*, since $max\{p_j(x), \forall j \in [1 \dots e] \forall x \in T(C_j)\} = 18$.

The best solution found is the solution $s1 = (x_0 = 1, x_1 = 1, x_2 = 1)$ and its $|$*last-covering*$(s1)| = 10$ in the solution space, which is the highest for this analyzed CSP (see Figure 6.4(d)). As previously mentioned, the solution s with the highest |last-covering(s)| in the solution space, has the highest likelihood of remaining valid when faced with future possible restrictive modifications over the bounds of the solution space. Therefore, it is the most robust solution for the original CSP according to our dynamism assumptions for CSPs with ordered domains. In contrast, the solution $s_{11} = (x_0 = 2, x_1 = 2, x_2 = 0)$ is classified by our technique as the least robust solution. Note that s_{11} only has one neighbour in the solution space (see Figure 6.4(d)). Thus, it is very probable that they will become invalid after restrictive modifications over the original constraints and domains of the CSP.

6.5 Summary and Limitations

In this chapter we have presented an enumeration-based approach that models CSPs as WCSPs by calculating coverings, with the aim of finding robust solutions for the original CSPs. The main advantage (and also the motivation) of this covering approach over the approach introduced in Chapter 5 is that it can be applied to all the set of CSPs that model problems for which the order is significant. On the contrary, the previous approach is only addressed for finite linear CSPs.

Both enumeration-based approaches share the same dynamism assumptions. Dynamism takes the form of restrictive modifications over the bounds of the solution space of CSPs with ordered domains. They also share the same general WCSP modeling algorithm (see Algorithm 6.2). The differences between them lie in the way of penalizing the tuples by means of the cost functions. The penalizations to the tuples are computed based on the information that the approaches compute in order to figure out the distance of each tuple to a bound. While the approach designed for linear CSPs computes Euclidean distances to the bounds, the approach presented in this chapter computes coverings. A covering is a convex hull of feasible assignments surrounding the tuple, which ensures minimum distances to the bounds. The coverings computation allows to extend the applicability of the approach to non-convex search spaces. For this reason, it is not only restricted to finite linear CSPs.

Extending the applicability of the enumeration-based approach to CSPs with non-convex search spaces is not costless. The computation cost of the algorithm that calculates the coverings (Algorithm 6.1) is quite higher than the Algorithm 5.1 because it requires the checking of feasible neighbour tuples for each valid tuple analyzed, whilst Algorithm 5.1 only computes a distance equation for each valid tuple analyzed. For this reason, the number of valid tuples of the constraints is crucial in the computational cost of Algorithm 6.1. The smaller the search space is, the quicker the algorithm is. As previously mentioned, the GAC3 process used before the coverings computation can help to restrict the search space by deleting inconsistent values, in such case, the computation time is reduced.

A weakness of the enumeration-based approaches that have been already introduced in Chapter 5 and Chapter 6 is that the computation of the information that is used for the penalization process is computed for each bound (constraint/domain). Thus, if there exists another bound that tights the analyzed bound, the information is not completely certain. For this reason, the enumeration-based approaches give us approximations to the real distances of the solutions to the bounds of the solution space. This issue was introduced in Section 3.5 and it would be deeply treated in the following Chapter 7.

Chapter 7

Search Algorithm for Finding Robust and Stable Solutions

The previous approaches presented in this dissertation are enumeration-based approaches that model robustness and stability in CSPs as WCSPs. In contrast, in this chapter we introduce a search algorithm that aims for finding the best solution according to an objective function that captures robustness and stability (Climent et al., 2012c, 2014). Along with the approach introduced in Chapter 6, in this chapter we propose a non-probabilistic technique which does not require extra detailed information about the dynamism. It is based on the theoretical notion about neighbourhood explained in Section 3.4.2.

Figure 7.1 shows a diagram that represents the basic steps that this approach carries out. In the first inspection of the diagram, it can be observed the simplicity of its steps in comparison with the diagrams of the enumeration-based approaches. Instead of modeling CSPs as WCSPs, the presented search algorithm solves directly the CSPs with ordered domains, according to the little dynamism assumptions made for such framework. For this purpose, we define an objective function that is based on the feasibility of contiguous surrounding neighbour solutions for each assignment of a variable.

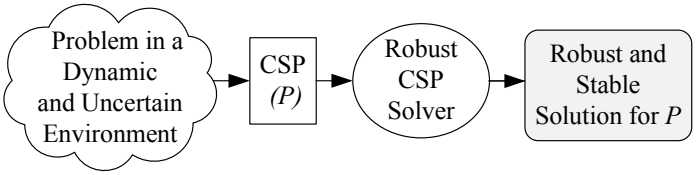


Figure 7.1: Robust and stable solutions search by our uninformed CSP solver.

The enumeration-based approach by coverings discussed previously has the disadvantage that the coverings information that they extract from the valid tuples of

each constraint is only based on themselves and the preprocessing inference process. Thus, some controversial situations might arise, in which some neighbour tuples are feasible for one constraint but are not for another one. In this chapter we introduce a search algorithm that checks the feasibility of the neighbours in the solution space instead of for each bound of the CSP. In this way, the search algorithm copes with this weakness. Furthermore, it introduces some advantages, such as the versatility of its objective function and the computational space saved due to avoiding the WCSP modeling step.

7.1 Neighbourhood in the Solution Space

In this section we apply the neighbourhood notion described in Section 3.4.2 to the solution space of the CSPs. We recall the Lemma 3.4.1 that we concluded in this section:

It only can be ensured that a solution s is located at least at a distance d from a bound in a certain direction of the n-dimensional space if all the tuples at distance lower or equal to d from s in such direction are feasible.

Thus, the number of feasible contiguous surrounding neighbours of a solution is a measure of its robustness. In addition, as mentioned, having feasible neighbours close to a solution provides stability to it. This is because if the value assigned to a variable has at least one of these feasible neighbour values, then this variable is repairable. That is, if its assigned value is lost, it can be easily repaired by assigning the neighbour value (since this value is consistent with the rest of the values of the assignment).

Computing all the solutions of a CSP is generally NP-Hard. This is due to the high combinatoriality of all the values in the domain for all the variables of the CSP. Thus, computing all the solutions surrounding another solution is not viable. However, computing the closest neighbour solutions in the greater and lower order directions for a fixed axis is reasonable. These neighbour assignments are equivalent to the analyzed assignment when it is modified the value assigned to a variable in increasing or decreasing order. In Chapter 6, we called covering to the set of surrounding neighbours tuples. However, in this proposal, we store feasible surrounding values on both sides for each variable. We use the term $\mathcal{N}$ for denoting this set of neighbours. Furthermore, we use the same maximum distance parameter k that the coverings technique uses ($k \in \mathbb{N}$).

For the coverings approach we used the Chebyshev distance, which measures the maximum absolute differences along any coordinate dimension of two tuples (see Equation 2.5). We also introduced the Euclidean distance in Equation 2.4. In this chapter, the distance is applied to tuples of only one dimension, that is to say, it is applied to values. In such cases, the Chebyshev and Euclidean, both distances are equivalent to the absolute difference between two values, which is used for the

distance measurement. These concepts are following illustrated with an example.

Example 7.1.1. *Here we take as an example again the solution space of the CSP described in Example 3.1.1. We recall that this 2-dimensional solution space is composed by* 29 *solutions (black points). Figure 7.2 shows the surrounding contiguous neighbour solutions for* $k = 1$ *and* $k = 2$ *of the highlighted solution* $s = (x_0 = 5, x_1 = 4)$*. We can observe that all the possible neighbours for both directions for each variable are feasible for both distances* $k = 1$ *and* $k = 2$*. Note that the contiguity property (explained in detail in Section 3.4.2) is satisfied because all the closest neighbours (all the neighbours located at distance* $k = 1$ *from the highlighted solution), are feasible. Therefore, we can ensure that the highlighted solution is located at least to a distance two from the bounds in each direction of the two axis.*

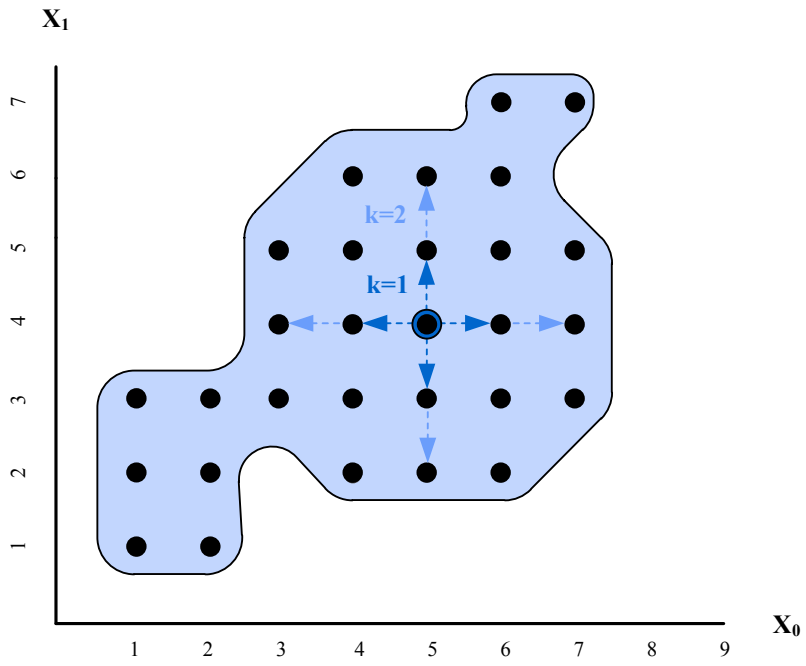


Figure 7.2: Surrounding contiguous neighbours for a solution of the CSP of Example 3.1.1 for $k = 1$ and $k = 2$.

Note that the diagonal axis solutions are not computed (due to the impossibility

of computing such combinatoriality). This does not represent a problem for convex solution spaces since their main characteristic is that there can not exist holes (unfeasibility areas) in such spaces. However, there can be controversial situations for $k > 1$ for non-convex solution spaces because some of the closest neighbour located on a diagonal can be unfeasible even if the contiguous surrounding neighbours for $k > 1$ are all feasible. Nevertheless, for non-convex solution spaces, due to the fact that there can exist holes in the space, the likelihood that a solution does not have an unfeasible neighbour close to it, is lower than in convex spaces. For this reason, usually for non-convex solution spaces, the adequate value of the k parameter is lower, which make more unlikely these controversial situations.

The set of feasible contiguous neighbour values of the value v that have differences not greater than k with respect to v in increasing, or decreasing, or both directions with respect to the order relationship is denoted as $\mathcal{N}_k(x, v, s, \oplus)$. Value v is a feasible value for variable x in the feasible partial/complete assignment s. Here, when we say that other values are feasible, we mean that they are also feasible with respect to s. We recall that, as mentioned in Chapter 2, $D_s(x) \subseteq D(x)$ denotes the subset of domain values that are consistent with the feasible partial assignment s. The list of operators $\oplus$ is composed of a set of paired elements, or operator pairs. Each operator pair is denoted as $\oplus_i \in \{\{>, +\}, \{<, -\}\}$. The operator pairs fix the order directions to analyze. Thus, the set $\{>, +\}$ refers to values greater than v (increasing direction) and the set $\{<, -\}$ refers to values lower than v (decreasing direction). For each operator pair, the operator in position j is referenced as $\oplus_{ij}$. For instance, if the list of operators is $\oplus = \{\{>, +\}, \{<, -\}\}$, the operator pair $\oplus_1$ references $\{>, +\}$ and the operator $\oplus_{12}$ references the operator $+$. Given this notation, we define $\mathcal{N}_k(x, v, s, \oplus)$ as:

$$\begin{aligned}\mathcal{N}_k(x, v, s, \oplus) = \{w \in D_s(x) : \exists \oplus_i, w \oplus_{i1} v \wedge |v - w| \leq k \wedge \\ \forall \oplus_z \forall j \in [1 \ldots (|v - w| - 1)], (v \oplus_{z2} j) \in D_s(x)\}\end{aligned} \tag{7.1}$$

The first condition of Equation 7.1 ensures that the value w is greater or lower than v according to the operator $\oplus_{i1} \in \{>, <\}$ and the distance between these values is less or equal to k. The second condition ensures that all values that are closer to v than w are also feasible values for s. If at least one of them is not feasible, the value w cannot belong to $\mathcal{N}_k(x, v, s, \oplus)$. As mentioned previously, the set of feasible neighbours of a value has to be contiguous. Otherwise, there is an unfeasible space between this value and another feasible value.

Due to the specific nature of some types of real life problems, in some occasions it is possible to extract more assumptions about their dynamism. For this reason the list of operator pairs does not have to be necessary the same for all the problems.

For instance, in Chapter 8 will be analyzed a type of real life problem for which the list of set operators only contains one element. However, for the general case of CSPs with ordered domains in which we assume that all the bounds are dynamic, the desirable objective is to find contiguous surrounding feasible neighbours on both sides. Therefore, $\oplus = \{\{>, +\}, \{<, -\}\}$. For this list of operator pairs, the last condition of Equation 7.1 checks that all the values in both directions that are closer to v than w, are also feasible values for s. Subsequently, we present an example of a lack of contiguity in the feasibility.

Example 7.1.2. *We consider a small two dimensional non-convex solution space composed only of* 10 *solutions. In Figure 7.3, it can be observed the surrounding contiguous neighbour solutions for any* k *for the highlighted solution* $s = (x_0 = 2, x_1 = 2)$. *There are two surrounding neighbours for each variable assignment. If we consider* $k > 2$, *the value* 5 *of* x_1 *does not belong to* $\mathcal{N}_k(x_1, 2, \{x_0 = 2\}, \oplus)$ *because the value* 4 *(this value is located at distance two) is not a feasible value and therefore it is outside the bounds of the solution space. Thus,* $\mathcal{N}_k(x_0, 2, \{x_1 = 2\}, \{\{>, +\}, \{<, -\}\}) = \{1, 3\}$ *and* $\mathcal{N}_k(x_1, 2, \{x_0 = 2\}, \{\{>, +\}, \{<, -\}\}) = \{1, 3\}$ *for any* k *value. Note that these neighbours are on both sides of the value* 2 *with respect to the* x_0 *axis and the value* 2 *with respect to the* x_1. *Therefore, we can only ensure that the highlighted solution is located at least to a distance one from the bounds in each direction of the two axis.*

To apply Equation 7.1 to domains that are not ordered in $\mathbb{Z}$, a monotonic and order-preserving function has to be applied in order to map the elements. Following, we describe a short example of such application.

Example 7.1.3. *Consider a symbolic domain* $\mathcal{D} = \{freezing, cold, mild, warm, hot, boiling\}$, *a monotonic function that assigns greater values to values with higher temperatures could be defined. For example,* $f(freezing) = 1, f(cold) = 2, f(mild) = 3, f(warm) = 4, f(hot) = 5$ *and* $f(boiling) = 6$.

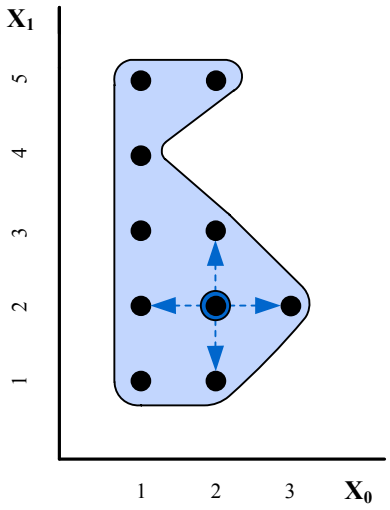


Figure 7.3: Surrounding contiguous neighbours for a solution of the CSP of Example 7.1.2 for any k.

7.2 Objective Function

In Section 7.1 we stated that the main desirable objective for a selected value is to have as many contiguous feasible neighbours in a certain direction, because they determine the minimum distance of this value from the bound in such direction. For approximating the distance of several assigned values (partial or complete assignment), we compute the sum of the number of neighbours of each value. Therefore, we define as an objective function of our search algorithm the sum of the size of $\mathcal{N}_k(x, v, s, \oplus)$ (denoted $|\mathcal{N}_k(x, v, s, \oplus)|$) for each variable $x \in \mathcal{X}$. If s is an incomplete assignment, we calculate the maximum $|\mathcal{N}_k(x, v, s, \oplus)|$ for each $v \in D_s(x)$ of each unassigned variable $x \in \mathcal{X} \backslash \mathcal{X}_s$ (upper bound).

The maximum size of the set of neighbour values for each variable is $| \oplus | * k$, where $| \oplus |$ is the number of pair operators. Thus, the maximum size of the set of neighbour values is $2k$ if $\oplus$ is composed of two operator pairs or k if $\oplus$ is composed of only one operator pair. Note that it is not necessary to check all the values of $D_s(x)$ if, for at least one of them, the size of the set is the maximum possible. In the following equation, we formalize the objective function that is used by our search algorithm.

$$f(s, k, \oplus) = \{ \sum_{x \in \mathcal{X} \backslash \mathcal{X}_s} max\{|\mathcal{N}_k(x, v, s, \oplus)|, \forall v \in D_s(x)\} + \sum_{y \in \mathcal{X}_s} |\mathcal{N}_k(y, s(y), s, \oplus)| \} \tag{7.2}$$

Example 7.2.1. *Considering again the solution space represented in Figure 7.3, it can be observed that the objective function for the highlighted solution* $s = (x_0 = 2, x_1 = 2)$ *is* $f(s, k, \{\{>, +\}, \{<, -\}\}) = 4$*, for any* k *value, since every value assigned to each solution has two contiguous neighbours on both sides.*

7.2.1 Formalization

Next, we give a formal rationale for using the total number of neighbours of the solution (sum of feasible surrounding neighbours of each value of the solution) as a measure of robustness.

For $k = 1$, in a convex solution space, each value has either zero, one or two feasible neighbours. Here we can discount the case of zero neighbours because if an assignment has zero feasible neighbours, then it must be part of a singleton domain, and it will be part of all solutions. So we need only to consider values with one or two feasible neighbours.

In this case, a solution with a greater sum is one whose assignments have more feasible neighbour pairs. This can be easily seen if we consider the difference between a solution all of whose values have only one feasible neighbour and any other solution; this difference will be equal to the number of feasible neighbour pairs associated with the latter's assignments. Hence, the following proposition can be trivially deduced.

Proposition 7.2.1. *If we assume that having two feasible neighbours confers greater robustness than having one and that the probabilities of single changes are independent, then a solution with a greater feasible neighbour-sum than another will also be more robust, and vice versa.*

In the non-convex case, it is unfortunately possible for one assignment to have zero feasible neighbours, while other assignments to the same variable have one or two. In this case, we cannot assume Proposition 7.2.1. However, as the number of variables in the problem increases, it becomes increasingly unlikely that a variable with an assignment having zero feasible neighbours will be associated with the largest neighbour-sum for the remaining variables.

Regarding the stability, a solution that maximizes the (1, 0, k)-repairability (see Definition 3.3.3) also maximizes the number of variables that can be repaired by a neighbour value at a distance less or equal to k (without modifying any other variable). However, to obtain robust solutions we maximize the sum of neighbour values

of each value of the solution. Note that even if both maximization criteria are not identical, as mentioned, when the number of variables in the problem increases, it becomes increasingly unlikely that a non-repairable variable will be associated with the largest neighbour-sum for the remaining variables. So in this work we will use the same technique for finding robust and stable solutions for CSPs with ordered domains. Nevertheless, the basic units of measure for both criteria are different. The robustness is measured as the total sum of contiguous neighbours at distance lower or equal to k; and the stability is measured as the number of variables whose values have at least one feasible neighbour at distance lower or equal to k.

7.3 Search Algorithm

In this section we present an algorithm for finding robust and stable solutions according to the main objective described in Section 7.2. For this purpose, we have incorporated this optimization criterion into a Branch & Bound algorithm. For a detailed explanation of the Branch & Bound algorithm (see Section 2.6.4). We have developed a search algorithm denoted as B&B-$\mathcal{N}_k$ that maximizes the objective function $f(s, k, \oplus)$ (see Equation 7.2). As mentioned, this function sums $|\mathcal{N}_k|$ of each assigned variable and the maximum possible $|\mathcal{N}_k|$ of each unassigned variable. Note that this computation is an upper bound of the final total number of feasible contiguous neighbours of the solution.

Algorithm 7.1 (B&B-$\mathcal{N}_k$) is an 'anytime' algorithm that uses an inference process and prunes the branches whose objective function value is lower or equal to the current maximum function value obtained, referred to as lb (lower bound). The process stops when all the branches have been explored or pruned, providing the solution s with the maximum $f(s, k, \oplus)$. We can also limit the search time and therefore the quality of the best solution found by fixing a time cutoff. Of course, the more time Algorithm 7.1 spends for searching, the more robust and stable the solution provided can be. In addition, we compute the maximum possible objective function value, which is the maximum number of neighbours for each variable multiplied by the number of variables of the CSP, denoted as ub (upper bound). Thus, if the objective function value of a new solution found is equal to ub, the algorithm stops, since this solution is optimal.

We have implemented the Branch & Bound algorithm using a Geometric restart strategy (Walsh, 1999) in order to reduce the repetition of fails in the search due to very early wrong assignments (*thrashing*). The detailed explanation of the restarting techniques can be found in 2.6.3. In addition, the pseudo code for the geometric restart algorithm was described in Algorithm 2.3. We recall that for this restarting algorithm, each time that the number of failures (referenced as nbF) reaches the number-of-fails cutoff value condition (C) that is checked in Algorithm 7.3, the algorithm restarts the search from scratch, except for some specific information that is stored. The value

Algorithm 7.1: B&B-$\mathcal{N}_k$: Branch & Bound anytime algorithm

Data: $P = \langle \mathcal{X}, \mathcal{D}, \mathcal{C} \rangle, \oplus, k, scale, m$, *time_cutoff* (optional).
Result: Best solution s and its associated $f(s, k, \oplus)$.

```
1  s ← ∅; // Partial assignment
2  X_s ← ∅; // Set of variables assigned
3  N_k ← ∅; // Set of contiguous surrounding neighbours
4  lb ← −1; // Maximum f(s, k, ⊕) for the solutions
5  ub ← | ⊕ | * k * |X|;
6  i ← 1;
7  GAC3-N_k(P, s, X_s, N_k, ⊕, k, lb);
8  repeat
9      if restarting-scratch ∧ new solution found then
10         i ← 1;
11     C ← scale * m^i; //number of fails cutoff
12     i ← i + 1;
13 until time_cutoff ∨ not MGAC3-N_k(P, s, X_s, N_k, ⊕, k, lb, 0, C, ub);
```

of the number of fails cutoff is increased geometrically in Algorithm 7.1 according to the $scale$ factor and the multiplicative factor (referred as m). The information that we store after each restart is the constraint weights computed by the $dom/wdeg$ heuristic variable selection (Boussemart et al., 2004). This variable heuristic was explained in Section 2.6.2 and we select it due to its successful results, especially combined with restarting strategies.

We have implemented two different options to carry out after a solution is found. In the first, called *restarting-completion*, when the first solution is found, the algorithm continues the search until completion (this is done by assigning a huge number representing infinite to the number of fails cutoff). In the second option, called *restarting-scratch*, after each solution found, the algorithm restarts the search from scratch and also restarts the number of fails cutoff computation (the constraint weighs remain the same). For instances with very large domain sizes, this restarting option can be effective because it avoids spending a large amount of time in a specific branch. The latter happens when Algorithm 7.1 checks many domain values of variables located at low levels of the search tree, because the objective function of the partial assignment is better than the current maximum (lb). In this case, if there exists a time cutoff, Algorithm 7.1 could not to have time for analyzing other branches of the tree that may contain solutions of better quality.

The inference process is carried out by Algorithm 7.2 (GAC3-$\mathcal{N}_k$), which is an extension of the GAC3 consistency algorithm (see Algorithm 2.1). We recall that $var(c)$ is the scope of $c \in \mathcal{C}$. The original `seekSupport` function of GAC3 searches for a support for each domain value. We have modified this function slightly by

providing the set of values to be analyzed as a parameter of the function. Thus, if any of these values is deleted because there does not exist any consistent support with respect to the partial assignment, `seekSupport` returns `false`. This function is first called with the values of the domain of the variables (for checking if the partial assignment s is GAC3) and later with $\mathcal{N}_k$ just for assigned variables (for checking if each $\mathcal{N}_k(x, s(x), s, \oplus)$ is GAC3 with respect to s). In order to ensure the contiguity of the values in $\mathcal{N}_k$, Algorithm 7.2 checks the consistency of subsets of $\mathcal{N}_i \subseteq \mathcal{N}_k$, where i is equal to one initially, and it is increased by one unit until at least one of the values of $\mathcal{N}_i$ is inconsistent or i reaches the value of k. The complexity of updating $\mathcal{N}_i$ can be reduced to $|\oplus| * i$ if the domains are ordered. Note that in the case where both greater and lower values are candidates to be in the set, the updating cost is $2 * i$. After composing the set of contiguous neighbour values that are GAC3 with respect to s, Algorithm 7.2 analyzes whether the objective function $f(s, k, \oplus)$ is greater than lb. If not, or s is not GAC3, it returns `false`.

Algorithm 7.2: GAC3-$\mathcal{N}_k$: Global Arc Consistency algorithm

Data: $P, s, \mathcal{X}_s, \mathcal{N}_k, \oplus, k, lb, nbF$
Result: $\mathcal{D}, \mathcal{N}_k, nbF$

```
1  Q ← {(x, c), ∀c ∈ C, ∀x ∈ var(c)} // var(c) is the scope of c
2  while Q ≠ ∅ do
3      (x, c) ← takeElement(Q);
4      seekD ← seekSupport(x, D(x), c); // Found support for all D(x)?
5      if D(x) = ∅ then
6          nbF ← nbF + 1; // number of failures
7          return false
8      if not seekD then
9          Q ← Q ∪ {(y, c'), ∀c' ∈ C ∧ c' ≠ c ∧ ∀x, y ∈ var(c') ∧ x ≠ y}
10     if x ∈ X_s then
11         i ← 1;
12         repeat
13             update N_i(x, s(x), s, ⊕) applying Equation 7.1;
               seekN ← seekSupport(x, N_i(x, s(x), s, ⊕), c);
14             i ← i + 1;
15         until seekN = false ∨ i > k;
16         N_k(x, s(x), s, ⊕) ← N_i(x, s(x), s, ⊕)
17 return f(s, k, ⊕) > lb // See Equation 7.2
```

As it was explained in Section 2.6.1, the maintaining arc consistency algorithm assigns tentative domain values to each variable by checking their consistency in each step. Here, we use the maintaining GAC3 algorithm, whose pseudo code can be found in Algorithm 2.2. Algorithm 7.3 (MGAC3-$\mathcal{N}_k$) is based on the latter algorithm, even

if it incorporates several novelties. The maintaining consistency process is done by assigning to each variable $x \in \mathcal{X}$ a new value $v \in D(x)$, until the value selected is GAC3-$\mathcal{N}_k$ with respect to s. We have implemented two value selection heuristics: lexicographical order and selection of the value that maximizes $|\mathcal{N}_k(x, v, s, \oplus)|$, starting from intermediate values. There are some real life problems for which the lexicographical selection order is effective in finding feasible solutions quickly. An example is scheduling problems, whose domain values represent time units; hence the importance of selecting low values in order not to exceed the maximum fixed makespan. However, if it is not important to select low values, the other heuristic may offer better results because it is selecting values that maximize the objective function at the current node of the search tree. Furthermore, since search starts with intermediate values, the likelihood of selecting values located far from the domain bounds is higher.

Algorithm 7.3: MGAC3-$\mathcal{N}_k$: Maintaining Global Arc Consistency

Data: $P, s, \mathcal{X}_s, \mathcal{N}_k, \oplus, k, lb, nbF, C, ub$
Result: $s, \mathcal{N}_k, lb$
select $x \in \mathcal{X} \backslash \mathcal{X}_s$; // $dom/wdeg$ heuristic
$\mathcal{X}_s \leftarrow \mathcal{X}_s \cup x$;
save $\mathcal{D}$ and $\mathcal{N}_k$;
while $D(x) \neq \emptyset \wedge nbF < C$ **do**
    select $min(v) \in D(x)$; // Heuristic 1: lexicographical value order
    select $v \in D(x), max\{|\mathcal{N}_k(x, v, s, \oplus)|\}$ starting by intermediate values; // Heuristic 2
    $s \leftarrow s \cup \{x = v\}$
    $D(x) \leftarrow v$;
    if *GAC3-*$\mathcal{N}_k(P, s, \mathcal{X}_s, \mathcal{N}_k, k, lb, nbF)$ **then**
        if $\mathcal{X}_s = \mathcal{X}$ **then** // New solution found
            $lb \leftarrow f(s, k, \oplus)$;
            if $lb = ub$ **then**
                return `true` // Best possible sum achieved
            $C \leftarrow \infty$; // *restarting-completion*
            return `false` // *restarting-scratch*
        else if *MGAC3-*$\mathcal{N}_k(P, s, \mathcal{X}_s, \mathcal{N}_k, k, lb, nbF, C, uB)$ **then**
            return `true`
    restore $\mathcal{D} \backslash D(x)$ and $\mathcal{N}_k$;
    $s \leftarrow s \backslash \{x = v\}$;
$\mathcal{X}_s \leftarrow \mathcal{X}_s \backslash x$;
return `false`

Algorithm 7.3 is also responsible for updating the set of assigned variables $\mathcal{X}_s$, the partial assignment s and the maximum objective function value lb (for each solution found). Furthermore, it stores the domains and the set of neighbours of all the variables before making an assignment. Note that after a variable x is assigned, $D(x)$ contains a single value that is the value assigned to x. If Algorithm 7.2 (GAC3-$\mathcal{N}_k$) returns `false`, then Algorithm 7.3 (MGAC3-$\mathcal{N}_k$) carries out the backtracking process and also restores the domains and set of neighbours of all the variables.

Bounds Arc Consistency for discrete CSPs (Lhomme, 1993) (see Section 2.5.2) allows a computational time reduction in CSPs with convex domains. Hence, it has been implement and included in the search algorithm. The main feature of this consistency technique is that the arc consistency is restricted with respect to the bounds of each convex domain. Thus, including it in the search algorithm only affects to the `seekSupport` function, which instead of seeking for a support for all the set of values, just checks the minimum and maximum bounds. Note that this implementation is not necessary for the search of robust and stable solutions; however it allows a significant reduction of the search time. We only apply bounds consistency to the tentative values of the assignment but not to their set of neighbours, since they require a complete consistency check. Otherwise there could exist unfeasible gaps, which would break the contiguity requirement that ensures minimum distances to the bounds.

7.4 Summary and Limitations

In this chapter we have introduced a search algorithm that seeks for solutions that maximize an objective function that sums the set of feasible contiguous surrounding neighbours of each value that compose an assignment. Thus, minimum distances to the bounds in the increasing and decreasing directions of each axis of each variable can be ensured. In this way, the algorithm searches for robust solutions according to the dynamism assumptions made for CSPs with ordered domains. Furthermore, these feasible surrounding neighbours provide stability to the solution, since a value loss can be repaired by means of these close feasible neighbours and therefore, the difference between the original value and the new one is extremely low.

In order to check for the feasibility of the analyzed assignment as well as its neighbours, we have implemented several algorithms from the Constraint Programming literature: a branch and bound algorithm, an inference algorithm, a heuristic variable selection and a bounds consistency technique. We have thus developed a search algorithm that is able solve the CSP providing robust and stable solutions.

The approach presented in this chapter differs significantly from the previous approaches introduced, which are enumeration-based approaches. The main advantage of the search algorithm is that the checking of the neighbourhood feasibility is made in the solution space. This fact avoids certain controversial situations that the enumeration-based approaches have when a neighbour assignment is feasible for a

bound of the CSP but is not for another. However, this precise search has a higher temporal cost, due to the difficulty in bounding branches of the search tree in early steps. This difficulty is due to the lack of information about the feasibility of the unassigned parts of the tentative solution. Another issue of the search algorithm is that diagonal axis of the solutions are not computed (due to the impossibility of computing such combinatoriality). As it has been shown in this chapter, this fact is not problematic for convex solution spaces but it can be for some non-convex spaces for $k > 1$.

The strength of the search algorithm presented in this chapter comes from its versatility. The objective function can be easily adapted for a specific problem and it can be converted into a multi-objective function by adding other optimality criteria, then, it could handle Constraint Satisfaction and Optimization Problems (see Section 2.4). Hence, our model could be extended by including other kinds of optimization, as for instance, minimizing the time, maximizing the profit, etc. Furthermore, as contrary as the previous introduced modeling techniques, we can add as many techniques from the literature as we consider to this search algorithm.

Chapter 8

A Case Study: Scheduling Problems

In this chapter we explore a highly well-known type of real life problems: scheduling problems. They represent one of the most crucial optimization problems of high-level synthesis. In addition, they are one of the most complex real life problems and therefore they have been extensively explored in the related literature. Scheduling problems come from uncertain and dynamic environments, hence the importance of the analysis and application of proactive approaches. All the reasons previously mentioned have motivated the selection of such problems as case study, as well as for the evaluation of the approaches presented in this dissertation. For more extended information about scheduling, we recommend the book (Pinedo, 2012).

As previously mentioned, scheduling problems are optimality problems. A common main optimization criterion is to minimize the makespan. However, typically scheduling problems can be converted into satisfiability problems by fixing a maximum makespan and they can be modeled as CSPs.

In this thesis we focus on dynamic problems for which the only limited dynamism assumptions are extracted from the nature of the problems for which the order is significant. Scheduling problems are also problems with a straightforward order function associated with the domain elements: they represent time units. Due to their particular structure, they can provide us more specific information about their dynamism. The latter would be explained in detail throughout this chapter, as well as some specific robustness measures for scheduling problems.

In this chapter we have explained in detail the adaptation of our approaches to scheduling problems in order to increase their performance for such type of problems. We also comment the adaptation of another CSP proactive approach to this type of problems. Finally, all the adapted approaches have been tested in a benchmark from the literature, with the purpose of graphically show the robustness of the obtained schedules.

8.1 Modeling Scheduling Problems as CSPs

Before explaining the dynamism in scheduling, we consider the importance of explaining the process of modeling scheduling problems as CSPs. It is well known that a scheduling problem is composed by a finite set of tasks that use particular resources. The main objective consists in assigning the tasks to the resources over time, avoiding resources conflicts. A solution of this type of problem is called schedule. Typically, in scheduling problems, the main objective is to minimize the *makespan*, although there also exist other criteria to optimize.

Definition 8.1.1. *The makespan of a schedule is the latest finishing time of all the tasks of that schedule.*

Scheduling problems can be converted into satisfiability problems by fixing a maximum desired makespan. In this case, the main objective is to find any schedule configuration that does not exceed the maximum desired makespan and that satisfies all the constraints. The CSP model of such problems usually consists in associating to each variable a start or end time of a task (in this thesis we use the start time). In addition, the domains associated with each variable represent the possible time units and they fix the maximum allowed makespan. The latter is achieved by the upper domain bound of each variable, which is fixed to the maximum allowed makespan minus the duration of the task associated with the variable. The variable associated with a task represents its starting time, the end of the task is therefore the variable value plus the duration of the task. Finally, the duration of the tasks and their order can be fixed by means of the CSP constraints. These constraints forbid the variables representing the posterior tasks from taking values lower than the previous task variable (also considering their duration).

In Chapter 1 a toy scheduling example (Example 1.1.1), was roughly modeled as a CSP. After formally defining the CSP model and also explaining the scheduling modeling process, a formal CSP model can be defined. This scheduling problem is composed of 2 tasks of 3 time units each, and both tasks share the same resource. As previously mentioned, we associate a CSP variable to each task, representing its starting time. Thus, $\mathcal{X} = \{x_0, x_1\}$, where x_0 refers to T_0 and x_1 refers to T_1. The domains of the variables are in the integer interval of the start time and the maximum fixed makespan (called max for this instance) minus the duration of the related tasks. Therefore, $\mathcal{D} = \{\{0, 1, \ldots, (max - 3)\}, \{0, 1, \ldots, (max - 3)\}\}$. Regarding the task order, there is one constraint fixing that task T_0 must be executed before than task T_1. There are several constraint representations that can model the task order. One of them is: $\{x_0 + 3 \leq x_1\}$. This constraint forbids than x_1 takes lower values than x_0 plus 3. This is because task T_0 has 3 time units duration and task T_1 must start after task T_0 finishes (start time plus duration). The composition of these 3 sets ($\mathcal{X}, \mathcal{D}$ and $\mathcal{C}$) form the CSP representation (we name it E_2) of this scheduling problem. Thus,

$E_2 = \langle \{x_0, x_1\}, \{\{0, 1, \ldots, (max-3)\}, \{0, 1, \ldots, (max-3)\}\}, \{\{x_0+3 \leq x_1\}\}\rangle$.

8.2 Dynamism in Scheduling

As mentioned in Chapter 1, there are two main approaches for dealing with problems that come from uncertain and dynamic environments: *reactive* approaches and *proactive* approaches. For dynamic scheduling the classification is more extended: *reactive* scheduling, *stochastic* scheduling, scheduling under *fuzziness*, *proactive* scheduling and *sensitivity* analysis. A more detailed explanation of these different types of approaches can be seen in the survey of scheduling under uncertainty (Herroelen and Leus, 2005).

It has already been commented that re-solving the problem after the loss of a solution (reactive approaches) consumes computational time. In addition, for scheduling problems, another disadvantage appears if the new schedule is delivered late, since it could cause the shutdown of the production system, the breakage of machines, the loss of the material/object in production, etc., which can also entail an economic loss. Due to all these negative effects derived by the solution loss, authors who deal with dynamic and uncertain scheduling problems strongly value the scheduling loss prevention in dynamic environments. However, stochastic scheduling and scheduling under fuzziness require detailed extra dynamism information (for instance probabilities distributions of the dynamism), which is unknown or scarce in many real life problems. For the reasons mentioned above, the use of proactive approaches that do not require extra detailed information about the changes that the schedule may undergo is highly valued in the literature, as well as by the authors of the approaches presented in this dissertation.

As previously mentioned, scheduling is a rich context where uncertainties and changes can not be easily avoided (Verfaillie and Jussien, 2005). One of the most common form of changes in this type of problems is time delays. In (Fu et al., 2012), the authors stated that unexpected external events such as manpower availability, weather changes, etc. lead to delays or advances in completion of activities in scheduling problems. 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. The fact that a task finishes earlier than expected (relaxation) cannot invalidate the original schedule obtained. Nevertheless, the schedule unfeasibility can be produced when there are delays in the schedule (restriction). Furthermore, a delay of a task due to a disruption at some point of the schedule, may produce a propagation of the delay through the entire schedule, since future related tasks may be not able to start on time due to this delay. For all these reasons, delays in scheduling problems are an important issue to take into account.

As explained in Chapter 1, a buffer time located after a task in a schedule can

absorb a delay in such task as long as the buffer itself. The term buffer (also the term slack) refers to the spare time between related tasks. For instance, in Figure 1.4, which is a valid schedule for Example 1.1.1, we can observe how a delay of one time unit in a task is absorbed by a buffer of one time unit. Therefore, the schedule remains valid after such little delay, which is our main objective. If no specific information is given about the dynamics of the scheduling problem, it is reasonable to assume that any task of the schedule may undergo a delay due the expected duration of an activity being exceeded, because a required resource has became unexpectedly unavailable or other external situations. Consequently, the presented work for scheduling problems is based on the following assumption.

Assumption *Any task of a schedule may undergo a delay in the future.*

In (Climent et al., 2008a,b), an analysis of the above dynamism mentioned concepts are presented for a real life scheduling problem: the railway timetabling problem. In this work, the authors empirically demonstrate how the buffers (and other factors that indirectly generate them) contribute not only to increase the robustness but also to decrease the total delay propagation when a re-scheduling is necessary because the original schedule has become invalid after a disruption.

8.3 Robustness Measurement in Scheduling

In this section, we introduce several criteria for measuring the robustness in scheduling. There are two main factors that increase the capability of the schedule to absorb unexpected delays in its activities: the number of buffers and their duration. Since longer delays in a task include smaller delays in the aforesaid task, we assume that shorter delays are more likely to occur over all the tasks. For this reason, in this work and in the literature it is strongly valued that the slack are uniformly distributed across the whole schedule. The simplest way of determining this feature is calculating the number of buffers.

Ideally, according to the robustness criterion, a buffer time should be as long as possible because the longer it is, the longer are the delays that is able to absorb. Hence, another straight-forward robustness measurement was proposed in (Leon et al., 1994) as the slack average in the schedule. The combination of the duration of the buffers and their distribution across the schedule provides a more accurate robustness measure denoted as R^s_{slack} (see Equation 8.1). It is a slight variant of a measure introduced in (Surico et al., 2008) that consists in maximizing the slack average (shorted as avg) and minimizing their standard deviation (shorted as std) for a schedule s. For regulating the importance of the standard deviation term, the authors use a parameter called α, which can take any value in the interval [0.2,0.25], according to the authors considerations.

$$R^s_{slack} = avg(slack) - \alpha\ std(slack) \tag{8.1}$$

Another measure unit defined in (Kitano, 2007) for measuring the robustness of a system, is related to its resistance when faced with perturbations with certain probability to occur. This measure was used in (Escamilla et al., 2012) for scheduling problems in which the probabilities of task delays are unknown. Therefore, the authors assumed an equally probability disruption distribution over all the scheduling tasks. The authors of (Escamilla et al., 2012) adapted the Kitano's robustness measure to these scheduling problems, denoting it as $R^s_{F,Z}$ (see Equation 8.2), where:

- s is a schedule.
- Z is the discrete set of unexpected independent incidences that represents delays in the duration of tasks.
- F measures whether s is still feasible after the disruption.
  - $F(z) = 1$ iff the affected task is modified by z. Therefore, there is a buffer time after the aforesaid task, and this buffer is able to absorb the incidence.
  - $F(z) = 0$, iff more tasks are modified by z. Therefore, there does not exist a buffer time after the aforesaid task or it exists but it is not long enough for absorbing the incidence. Thus, the incidence is propagated to the rest of the schedule.
- $p(z) = \frac{1}{|z|}, \forall z \in Z$. It is the probability for an incidence $z \in Z$. All the tasks have the same probability of undergoing a delay due to there does not exist detailed information about the future possible incidences.

$$R^s_{F,Z} = \sum_{z \in Z} p(z) * F(z) \tag{8.2}$$

8.4 Neighbourhood in Scheduling

As stated before, ideally a robust schedule should exhibit well distributed slack, so that the probability that it can absorb delays and still remain valid is high. To meet this objective, we extend our approaches for CSPs with ordered domains to scheduling problems.

For the general CSP robust search in both convex and non-convex solution spaces, we stated that robust solutions are surrounded by feasible contiguous neighbours.

However, there are some types of real life problems whose structure can provide us with specific information about their dynamism. In CSP models of scheduling problems, the fact that domain values represent time units has implications involved with the measures of robustness and stability. For these problems, when a value of the solution is lost, lower values cannot be used for replacing this unfeasible value because they represent time units that have already taken place. Thus, if there is an incident and the time point t is not available, neither are the values that are lower than t. Therefore, having lower feasible neighbour solutions does not improve the robustness nor the stability of a solution of a CSP that models a scheduling problem (since they cannot absorb delays nor be used as repairable values).

Given these characteristics, the main desirable objective is to search for neighbours solutions greater than the value assigned. Furthermore, in the two following sections we describe in detail the scheduling adaptation for the search algorithm and the enumeration-based approach by coverings. This adaptation allows to improve the robustness of the schedules obtained by these approaches. Note that since both approaches are based on the feasibility of neighbour solutions, they are versatile for their adaptation to problems with specific characteristics.

Example 8.4.1. *We consider a scheduling problem with two tasks: T_0 and T_1. Both have a duration of two time units and they must be executed in the order listed. The maximum makespan allowed is six time units. In Figure 8.1 we can see the associated CSP model and its solution space. The variables X_0 and X_1 represent the start times of tasks T_0 and T_1, respectively. The domain of both variables (represented by discontinuous lines) is $[0 \dots 4]$, which preserves the maximum makespan of six time units (the maximum start time of a task is the maximum makespan minus the duration of the aforesaid task). There is one constraint controlling the execution order of the tasks (T_0 must start before T_1), which is $C_0 : X_1 \geq X_0 + 2$. The solution space is represented by a blue area, where there are six solutions (black dots).*

If no specific information is given about the dynamic environment, which schedule is the most robust? As stated in Section 8.3, the greater number of buffer times and the greater their duration is, the more robust the schedule is. But how can we determine which solution of the modeled CSP meets these characteristics? The answer is obtained by calculating the distance to the bounds of the solution space that restrict greater values. As stated in Section 3.4.2, minimum distances are related to contiguous feasible neighbours. Thus, if we want to ensure a minimum distance of k from a solution to a bound located on a greater position in a certain direction (accord-

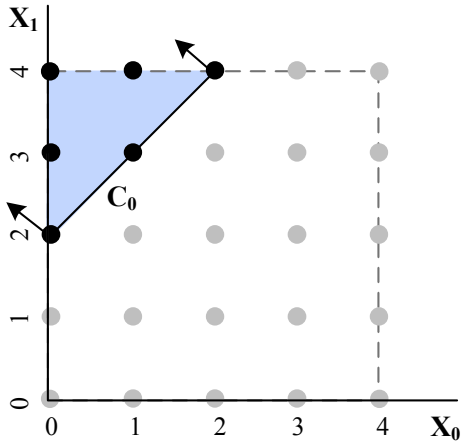


Figure 8.1: CSP model associated with Example 8.4.1 and its solution space.

ing to the order relationship over the domains), all the greater neighbours located at a distance lower or equal to k from a solution on this direction must be feasible.

The number of greater feasible neighbours associated with each variable, corresponds to the duration of the slack that is located after the task that is represented by this variable. This is explained in detail in Section 8.5. Thus, the slack is able to absorb a delay in the previous task as long as itself, without modifying the other tasks of the obtained schedule (robustness feature). Furthermore, if the slack associated with a task is not long enough to absorb a delay, the start of the following task can be delayed if there is a long enough buffer associated with this following task (stability feature).

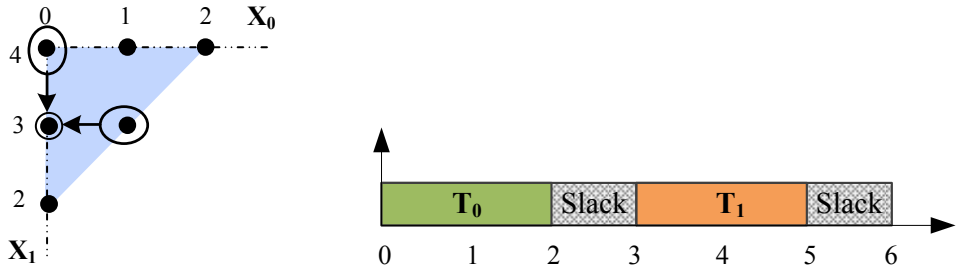
8.5 Adaptation of the Search Algorithm to Scheduling

Firstly, we describe the application of the search algorithm introduced in Chapter 7 to the above example. Afterwards, we introduce an adaptation of the neighbourhood set concept and therefore the objective function of the search algorithm for scheduling problems (Climent et al., 2013a). As previously mentioned, due to the nature of scheduling problems, only greater feasible neighbours provide robustness to the schedule. For this reason, when we compute the set of neighbour values $\mathcal{N}_k$ (see Equation 7.1) we fix the list of operators to only one operator set: $\oplus = \{\{>, +\}\}$. We recall that the operator pairs fix the order directions of the neighbours feasibility checking. Thus, the set $\{>, +\}$ analyzes the greater values than v (increasing direction) and the set $\{<, -\}$ analyzes the lower values than v (decreasing direction). Since we are not interested in the feasibility of lower neighbours, the operator $\{<, -\}$ is not used for scheduling problems.

For the Example 8.4.1, which is a 2-dimensional CSP representing a two tasks

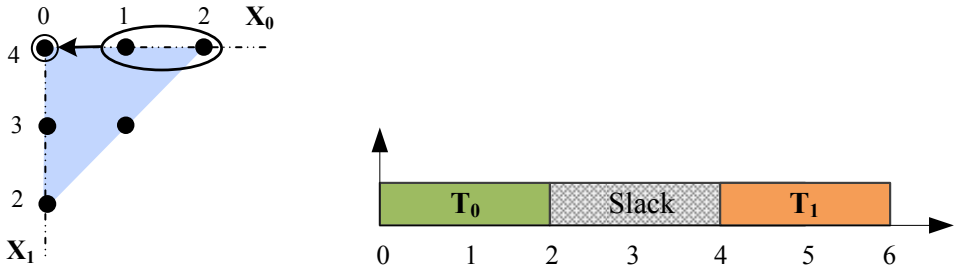
scheduling problem, there exist three more robust schedules according to the robustness criteria stated in Section 8.3. If we maximize the sum of greater neighbours values that are located at a distance one ($k = 1$) from each value of the assignment, the solution obtained is shown in Figure 8.2(a), whose sum is $f(s_0, k = 1, \{>, +\}) = 1+1$ (see Equation 7.2). The first number of the sum is $\mathcal{N}_k(x_0, v_0, s, \{>, +\})$ and second is $\mathcal{N}_k(x_1, v_1, s, \{>, +\})$, where v_0 and v_1 are the values assigned to the variables x_0 and x_1 respectively. Note that the sums of greater neighbours located at a distance one of the solutions represented in Figures 8.3(a) and 8.4(a) are $f(s_1, k = 1, \{>, +\}) = 1 + 0$ and $f(s_2, k = 1, \{>, +\}) = 0 + 1$, respectively. In the following (a) figures, the greater neighbours are surrounded by a circle and connected to the marked solution. In the following (b) figures, the schedules equivalent to the solutions marked in (a) figures are shown. Note that the greater neighbours surrounded in (a) figures correspond to the slack in figures (b). For instance, in Figure 8.2(b) each task have an associated slack of duration one time unit due to the existence of one greater neighbour for each value assignment in Figure 8.2(a).

If we maximize the sum of greater neighbours values for $k > 1$, the three solutions represented in Figures 8.2(a), 8.3(a) and 8.4(a) are classified as best solutions according to our objective function. The computation of the sum of neighbours located at a distance lower or equal to k, for $k > 1$ is: $f(s_0, k > 1, \{>, +\}) = 1 + 1$ (Figure 8.2(a)), $f(s_1, k > 1, \{>, +\}) = 2 + 0$ (Figure 8.3(a)) and $f(s_2, k > 1, \{>, +\}) = 0 + 2$ (Figure 8.4(a)). Note that the schedules in Figures 8.3(b) and 8.4(b) only have one buffer time each one, but its duration is two time units, unlike the schedule represented in Figure 8.2(b) that has two buffer times of one time unit each one. We would like to point out that by fixing $k = 1$ we are prioritizing the seek of a high number of buffer times. In contrast, for greater k values, the duration of the buffers is prioritized, even if in this case their distribution may not be optimal. The number of feasible greater contiguous neighbours than a variable assignment has, it is equivalent to the duration of the buffer time after the corresponding task in the schedule. Thus, if there does not exist any feasible greater contiguous neighbour for a variable, its corresponding task does not have a buffer time.



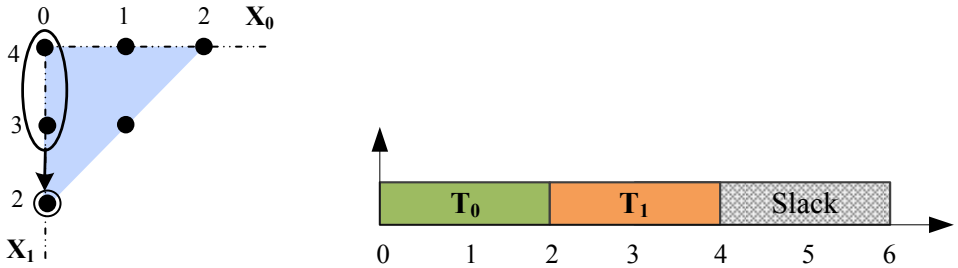
(a) Solution space. (b) Schedule of the marked solution.

Figure 8.2: Robust schedule $s_0 = (x_0 = 0, x_1 = 3)$ for Example 8.4.1 and its greater neighbours for $k \geq 1$.



(a) Solution space. (b) Schedule of the marked solution.

Figure 8.3: Robust schedule $s_1 = (x_0 = 0, x_1 = 4)$ for Example 8.4.1 and its greater neighbours for $k > 1$.



(a) Solution space. (b) Schedule of the marked solution.

Figure 8.4: Robust schedule $s_2 = (x_0 = 0, x_1 = 2)$ for Example 8.4.1 and its greater neighbours for $k > 1$.

At this stage, we wonder again which feature is preferable: greater number of little buffer times or lower number of long buffer times? (if both features, high number of buffers of long duration, cannot coexist). The answer is obviously related to the type of dynamism associated with the scheduling problem. Lower k values are performing better for frequent short delays that can occur in any task. In contrast, the greater k values can obtain schedules that may face greater delays in the tasks, but these schedules probably will become invalid when faced with high frequency delays (because of the lower number of tasks that have slack associated). For Example 8.4.1, if in the future a task undergoes only one delay of 2 time units, the schedules in Figures 8.3(b) and 8.4(b) have a 50% likelihood of remaining valid. However the schedule in Figure 8.2(b) has 0% chances of remaining valid. However, if the two tasks undergo a short delay of 1 time unit each one, the schedule in Figure 8.2(b) will remain valid (100% likelihood) unlike the schedules in Figures 8.3(b) and 8.4(b), which will not resist the two changes (0% likelihood). In Chapter 9 we evaluate our approaches analyzing several ranges of k values and we also compute several robustness measures (each one favouring several different types of future changes).

The above previous analysis is measuring the robustness of the solutions. How-

ever, if we consider the stability of the solutions, small modifications in the solutions are always preferred. In this case, if it is not possible that a task starts at the scheduled time, by reassigning its start time to a closer greater neighbour, we are composing another schedule that is very similar to the original one. Therefore, the search of the feasible greater neighbours (which introduces buffers into the tasks of the schedule) are improving both, the robustness and stability of the obtained schedules.

As previously motivated, $\oplus = \{\{>, +\}\}$ for scheduling problems (search of greater feasible neighbours). Thus, the general equation of the set of neighbours for any operator list ($\oplus$), introduced in Equation 7.1 can be simplified for this specific case. Equation 8.3 shows this simplification. The first condition of the equation checks that the feasible neighbour values are greater than the value v and their distance from v is lower or equal to k, where $k \in \mathbb{N}$. The second condition ensures that all the greater values that are closer to v than w are also feasible values (contiguity condition). If at least one of them is not, the value w does not belong to $\mathcal{N}_k(x, v, s, \{>, +\})$. The set of feasible greater neighbours has to be contiguous because otherwise, it means that there exists at least a task that is making these intermediate values unfeasible. Therefore, non-feasible contiguous values cannot compute as a measure of robustness because they do not represent slack in the schedule.

$$\begin{aligned}\mathcal{N}_k(x, v, s, \{>, +\}) = \{&w \in D_s(x) : w > v \wedge w - v \leq k \wedge \\ &\forall j \in [1 \ldots (w - v - 1)](v + j) \in D_s(x)\}\end{aligned} \tag{8.3}$$

Furthermore, when we apply our search algorithm (Algorithm 7.1 (MGAC3-$\mathcal{N}_k$)) to scheduling problems, we use the lexicographical value selection order. This heuristic allows us to find feasible solutions for scheduling problems quicker, since their domain values represent time units and hence the importance of selecting low values in order to not exceed the maximum makespan. In addition, in order to reduce the computational time, we have used Bounds Arc Consistency for discrete CSPs (Lhomme, 1993) when dealing with convex domains. This allows a significant reduction of the search time, which is very important in complex scheduling problems due to their elevated combinatoriality.

8.6 Adaptation of the Coverigns Technique to Scheduling

In this section we describe the adaptation of the enumeration-based approach by coverings introduced in Chapter 6. The main idea for such adaptation is the same than the explained in Section 8.5: checking only feasibilities of greater neighbours. Thus, instead of analyzing a whole covering (see Definition 6.1.1), we analyze a part of

it, which is called informally *greater* covering (it is also differentiated by the $>$ operator). In such area, all the values of the neighbour assignments must be equal or greater than the values of the analyzed assignment. According to the hyper-cube representation of coverings, only the right top hyper-cube (space between the analyzed assigment and the right top corner) satisfies such condition. This is illustrated below in Figure 8.5, which represents the solution space of Example 3.1.1. The part of the covering that includes the neighbours that are greater to the highlighted solution is represented in the red area. Thus, only these red areas represent the greater coverings of this solution.

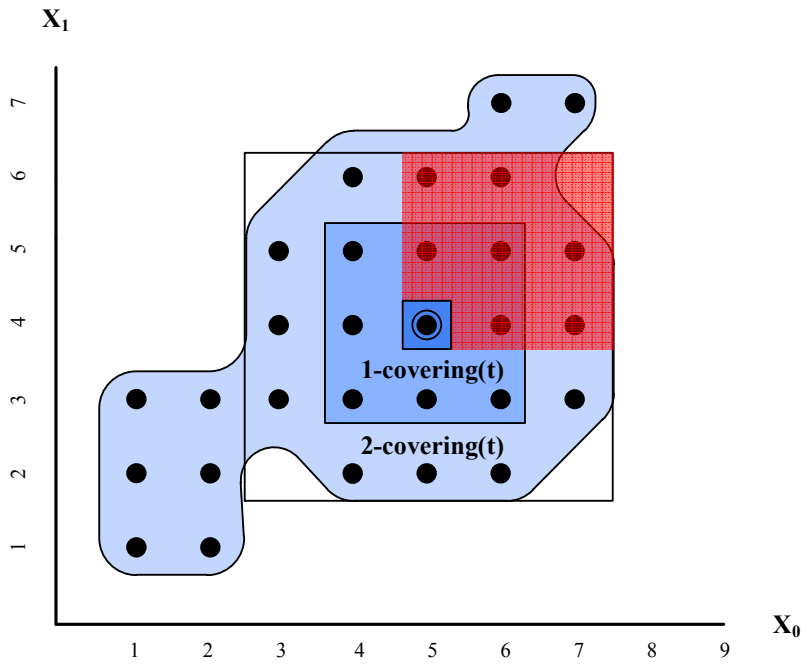


Figure 8.5: Greater coverings of a solution of the CSP of Example 3.1.1.

Bellow we formally define the k-covering$(t, >)$, which is composed only by assignments with all the values greater or equal to the analyzed assignment t.

Definition 8.6.1. *We define the k-covering$(t, >)$ of a valid tuple $t \in T$ as its neighbourhood $\{y \in T : y \neq t \wedge \{\forall i \in |y| : y_i \geq t_i \wedge y_i - t_i \leq k\}\}$, where $k \in \mathbb{N}$.*

Since Definition 8.6.1 is a more restrictive concept of covering than the one introduced in Definition 6.1.1, the maximum number of tuples of k-covering$(t, >)$ is lower. In Equation 8.4 we present a little modification of Equation 6.1 that describes the maximum size of a greater covering. The only difference with respect to Equation 6.1 is that k is not multiplied by two. This was representing the two directions of the

analysis (greater and lower values), however for a greater covering only one direction is analyzed.

$$maxTup(k, |t|, >) = (k+1)^{|t|} - 1 \tag{8.4}$$

By changing the above definition and equation, the rest of the concepts explained in Chapter 6 hold for the greater covering introduced in this section. For instance, the completeness property (see Definition 6.1.3) holds also for Definition 8.6.1 and Equation 8.4. Thus, we can ensure minimum distances to bounds that restrict greater values by checking the completeness of the coverings of an assignment. In the same way, the algorithm that computes the coverings (Algorithm 6.1) holds according to the adaptation presented.

8.7 Alternative Proactive Approaches in Scheduling

In this section we analyze another CSP proactive approach for dynamic scheduling problems that does not either require extra detailed dynamism information. In addition, it is addressed also to general dynamic CSPs, as well as our approaches. Furthermore, we also explain a model reformulation technique with the same purpose of obtaining slack.

A proactive CSP approach based on super-solutions (Hebrard, 2006) was deeply explained in Section 1.4.3. Here, we only explain a little modification that the authors made for adapting the approach to scheduling problems. For general CSPs the authors considered that a *breakage* was the lost of the value assigned to a variable of an assignment. Thus, any other value that is feasible with the rest of the assignment could be used as a repairable value. Afterwards, the authors extended this concept for scheduling problems in the way that a breakage is considered a delay of duration d in a task. Therefore, only values that are greater than the value assigned in d time units are considered as repairable values.

Thus, this adaptation has been incorporated to the (1, 0)-repairability approach for the evaluation of scheduling problems in Chapter 9, as well as for the analysis of the toy scheduling benchmark in this chapter (see Section 8.8). For a proper comparison of this approach with our approaches, we used the same values for k and d parameters. In the following, in order to avoid term repetition, we assume that $d = k$. Thus, the approach that maximizes the (1, 0)-repairability for a breakage of k in scheduling problems, searches for schedules that have the highest number of buffers of k time units. However, our approaches applied to scheduling problems (see Section 8.4) include in their robustness criterion the buffers that are smaller than k units (even if they try to find buffers of k time units). The latter condition is advantageous for the robustness of the schedule in certain cases, as it will be shown in the next section.

The search of schedules with buffers that are up to k time units can also be

achieved with model reformulation techniques by adding two variables to each original variable (the variables that represent the start time of the tasks). One variable represents the slack that follows the task and the other variable represents the sum of this slack and the original starting time. For instance, let p_i be the starting time of the task x_i. Thus, we would add the constraint $p'_i = p_i + s_i$, where s_i represents the slack associated with task x_i. In addition, depending on the maximum desired duration of the buffers, another constraint may be added, such as $s_i \leq k$. In this case, the delay is up to k time units. In addition, an objective function that express the goal of maximizing the total slack ($max \sum_{i=1}^{n} s_i$) must be defined.

An advantage of our adapted approaches over the above model reformulation technique is that they are generic for CSPs with ordered domains whilst the other approach can only be applied to scheduling problems. The other main advantage when we deal with scheduling problems is that our approach can be applied when all the slack-values require a consistency check. This requirement is necessary in scheduling problems where intermediate non-valid slack values are possible. Examples of this type of problem are scheduling problems with limited machine availability (see for instance (Schmidt, 2000)). In these cases, some machines are unavailable in certain time intervals; for this reason, tasks that require these resources cannot be executed in such time units. The same occurs with some scheduling with operators, where the workers have some breaks during the day, which produces unfeasible time gaps in the schedules.

8.8 Example

In this section an example of a scheduling benchmark is described to graphically show the robustness of schedules obtained by our approaches. Moreover, this benchmark is also used for showing the trade-off between robustness and makespan. Specifically, we analyze an instance derived from Taillard optimization problems (Taillard, 1993): "os-taillard-4-105-0", which is an instance of the well-known open-shop problem and was used in the CSP solver competition[1]. This problem was modeled as a satisfaction problem by fixing the maximum makespan allowed (latest finishing time). The best makespan known for such problem is 193 time units. However, for the analyzed instance the maximum makespan is set to 105% of the best one. Therefore, the obtained schedules for this problem may be, and usually will be, non-optimal. In this way, it is possible to have schedules with instances of greater slack. Similar descriptions of this example can be found in (Climent et al., 2013a,b).

Since the open-shop benchmark is composed of four machines, four jobs and four tasks per job, the resultant CSP model contains 16 variables and 48 constraints; the latter prevents two tasks from using the same machine at the same time as well as from ensure that two tasks of the same job do not overlap. Here, in this example,

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

we converted the open shop to job shop by fixing the order of the tasks to the same order obtained by an ordinary CSP solver. We have done this modification in order to increase the clarity of all the graphs presented in this section since the order of the tasks of the schedules obtained by each technique is the same, and therefore, it is easier to compare the slack between different schedules. The machine used for solving this benchmark was a Intel Core i5-650 Processor (3.20 Ghz) and we have fixed a time cutoff of $30s$. All the approaches use the lexicographical value order selection.

Figure 8.6 shows a non-robust schedule obtained by an ordinary CSP solver. The jobs are represented on the vertical axis and time is represented on the horizontal axis. Tasks are shown with the corresponding task number and with different colours according to the machine that they use. Thus, all the tasks of the same colour use the same machine. The striped slack represents the natural slack produced because the earlier starting next task related to a task, waits to the release of a machine or to the end of another task, or by the gap between the last task of a job and the makespan. It can be observed that this schedule has only seven natural buffers after the tasks, so a delay at any other place in the schedule will invalidate the obtained solution. Furthermore, three of these natural buffers are after the last task of the jobs that do not fix the makespan, which are inherent to all the schedules.

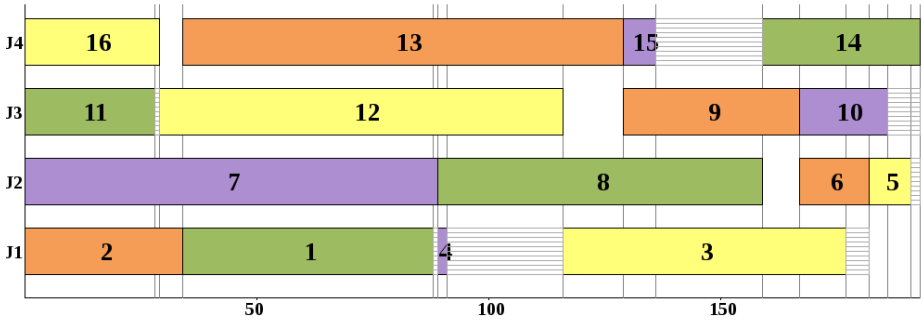


Figure 8.6: Non-robust schedule (makespan=193).

8.8.1 Obtaining Robust Schedules

Following, we introduce schedules obtained by our approaches and for the approach that maximizes the (1, 0)-repairability, all of them adapted for scheduling problems as indicated in the previous sections. The restarting parameters setting have been fixed as follows: the scale factor to 10 and the multiplicative factor to 1.5. The schedules obtained by our WCSP modeling approach presented in Chapter 6 are referred as "WCSP-mod" schedules. The schedules obtained by our search algorithm presented in Chapter 7 for *restarting-scratch* are referred as "neigh-search(R)". For calculating schedules that maximize the number of repairable values for (1, 0)-super-solutions

(referred as "super-sol" schedules), we modified Algorithm 7.1 (B&B-$\mathcal{N}_k$) by changing MGAC3-$\mathcal{N}_k$ and GAC3-$\mathcal{N}_k$ algorithms for MAC+ and GAC+ (Hebrard, 2006), respectively.

Figure 8.7 shows the schedule obtained for the three techniques mentioned above for $k = 1$. It can be observed that it has additional instances of slack (in dark gray). These buffer times are not produced because the earlier starting next task related to a task is not able to start before. Hence, they are differentiated from the natural slack by denoting them as robust. Because of them, this schedule is more robust and stable than the previous one. It is more robust because more tasks maintain slack to handle short delays. Moreover, it is more stable because in case some tasks can not start on time, their start can be delayed (reassigned) as much as their associated buffer time. Note that this schedule has 13 robust-stable buffer times, each of one time unit. Therefore, if any task has a delay of one time unit the schedule will still be valid. The makespan of this schedule is only two units longer than the makespan of the schedule represented in Figure 8.6 (trade-off between robustness and makespan).

All the approaches obtained the best schedule for $k = 1$ according to their respective objective functions since they finished the search before reaching the time cutoff. This fact, and also the fact that there was the possibility of introducing a little buffer time of one time unit after all the tasks has entailed that the schedule obtained was the same for the three approaches.

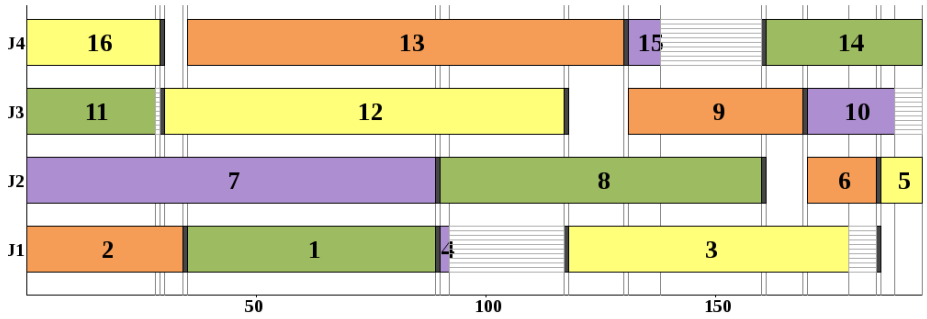


Figure 8.7: Schedule for $k = 1$ for WCSP-mod, neigh-search(R) and super-sol (nbRobustBuffers=13, robustBuffersSum=13, makespan=195).

As mentioned, the duration of each robust-stable buffer time obtained in the previous schedule is one time unit. However, when we increase the value of k for the three approaches analyzed, the slack in the obtained schedules vary. Figure 8.8, Figure 8.9 and Figure 8.10 show the schedules obtained by neigh-search(R), WCSP-mod and super-sol, respectively. In Figures 8.8 and Figures 8.9, it can be observed that our approaches find schedules with a makespan that is three time units greater than the schedule of Figure 8.7. However, the total slack of the schedule neigh-search(R) (see Figure 8.8) is 30 time units and WCSP-mod obtains even five time units more of total slack (see Figure 8.9).

Finding a schedule that has a buffer of five time units after each task is not possible for such instance. However, WCSP-mod finds a schedule with eight of its nine robust-stable buffers of four time units (see Figure 8.9). Neigh-search(R) has found one less buffer time, all of them in the interval of $[1, 4]$ time units (see Figure 8.8). Both approaches find very robust schedules, fact that is more obvious when we compare the results with the super-sol approach (see Figure 8.10), which only obtains two buffer times of five time units each. This fact is mainly due to two factors. The approach that searches for super-solutions only searches for buffers of k time units (it does not consider lower duration buffer times), thus it performs poorly generally for high k values because it is difficult to find many buffers of such great size for a certain fixed makespan. In addition, this technique maintains as a repairable values set all the feasible values in the domain of each variable. This requires more time when checking the satisfiability of such domain than our reduced set of feasible neighbours.

From all the approaches analyzed for $k = 5$, only the enumeration-based approach (WCSP-mod) obtains the best schedule according to its objective function criterion. The approach that maximizes the (1, 0)-repairability and our search algorithm (for both restarting options) provide the best schedule found until reaching the cut-off time, so that they are not able to analyze all the search tree. For the cut-off time allowed for this analysis and for $k > 1$, the enumeration-based approach performs better because the neighbour feasibility is computed before the solving process, also, there are almost no conflicts in such computed feasibility between the constraints.

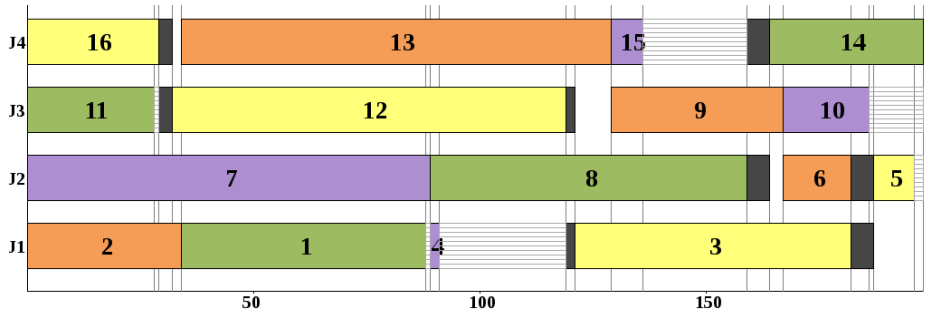


Figure 8.8: Schedule for $k = 5$ for neigh-search(R) (nbRobustBuffers=8, robustBuffersSum=30, makespan=198).

Note that even if the sum of robust slack is higher, all the obtained schedules for $k = 5$ have a smaller number of buffer times than the schedule obtained for $k = 1$. Since it is more difficult to find buffers with up to five time units, it may happen than our algorithm sacrifices some shorter buffers in order to find one buffer time of five time units. Thus, the obtained schedules with lower k values tend to maximize the number of buffers even if their size is small. However, the computation of higher k values tends to give priority to the duration of the buffers and as a consequence, the number of buffers obtained can be lower. Therefore, depending on the dynamic

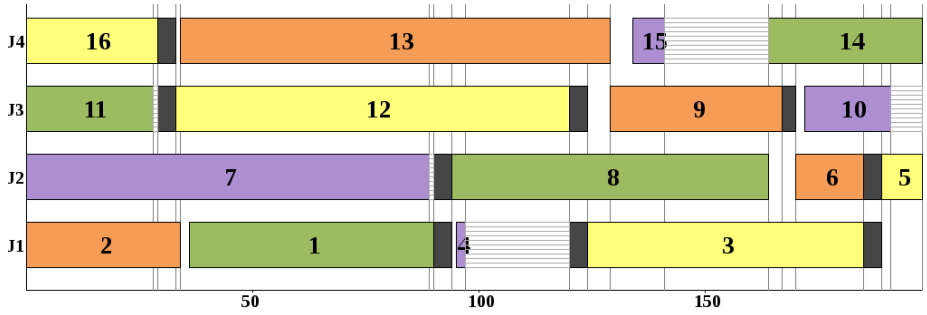


Figure 8.9: Schedule for $k = 5$ for WCSP-mod (nbRobustBuffers=9, robustBuffersSum=35, makespan=198).

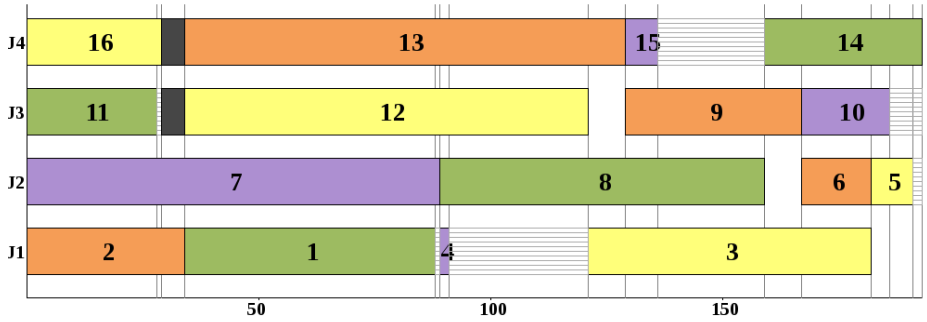


Figure 8.10: Schedule for $k = 5$ for super-sol (nbRobustBuffers=2, robustBuffersSum=10, makespan=193).

nature of the problem, it would be desirable to prioritize between a higher number of short buffers or a lower number of long buffers (in case it is not possible to maximize both features). The greater the number and duration of robust buffers, the higher the probability that the rest of the schedule will remain satisfiable after changes and as a consequence, the more robust the schedule will be according to the assumptions that any task can undergo a future delay.

8.8.2 Trade-off between Robustness and Makespan

Here we want to analyze the well-known trade-off between robustness and makespan in scheduling. For such purpose, we analyze the "os-taillard-4-100-0" problem more extensively by fixing the makespan over a range of different percentages out of the best known makespan (which is 193 time units). The solutions were obtained by our enumeration-based approach by coverings for $k = 7$. Figure 8.11 shows the makespan of the schedules on the horizontal axis, the number of robust-stable buffers on the left vertical axis and the sum of the robust-stable buffers on the right vertical axis. The time required to calculate our schedules was around between $[4, 10]$ seconds more than that required to obtain a simple schedule (greater time for the greater makespans).

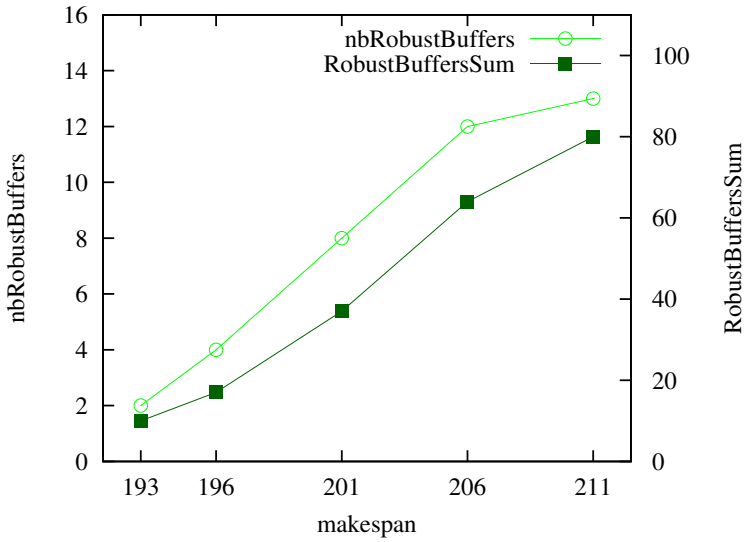


Figure 8.11: Robustness-makespan analysis for $k = 7$ for WCSP-mod for "os-taillard-4-100-0".

As expected, the robustness of the solutions obtained is positively correlated to the makespan. It can be observed that the number of buffer times obtained increases proportionally until reaching the point of the makespan 211. This schedule has only one buffer time more than the previous schedule (makespan = 206), even if there are

five time units of difference between their makespans. The reason is that by this point almost all the tasks have an associated robust-stable buffer (or they are the last task in their job), so finding further buffers becomes more difficult. However, the sum of buffer times increases in a more uniform fashion, which shows that at some point it is more difficult to find new buffers than it is to increase the size of a buffer.

Figure 8.12 shows the schedule with the highest makespan of the analysis done previously, which is 211 time units. Here we can appreciate how much the robustness of the schedule can increase when we exceed highly the best known makespan. Note that for all the tasks in which it is possible to include a robust-stable buffer there is a robust-stable buffer of five, six and often seven time units, with the unique exception of one buffer of two time units.

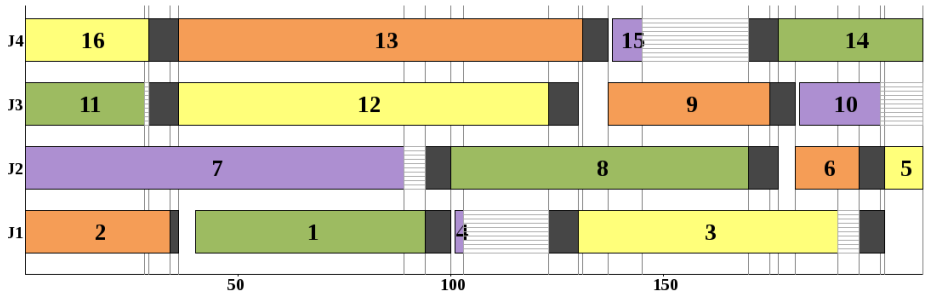


Figure 8.12: Schedule for $k = 7$ for WCSP-mod (nbRobustBuffers=13, robustBuffersSum=80, makespan=211).

8.9 Summary and Limitations

In this chapter we have described the relation of the new dynamism framework associated with ordered domains (see Chapter 3) with scheduling problems. Further, we explain in detail the adaptation of our approaches to such particular environment with the purpose of increasing the robustness of the obtained schedules. Dynamism in scheduling usually takes the form of delays over the tasks, which can produce a propagation of the delay through the entire schedule. That is why handling delays by keeping the schedule feasible is an issue of extreme importance. This is achieved by introducing slack in the schedule. In this chapter we also explain several specific robustness measurement units for the scheduling problems. These measures take into account the duration of the slack and also its distribution over the schedule.

After introducing the dynamism in scheduling and the main characteristics of a robust schedule, we have incorporated them into the theoretical approach for searching robust solutions for CSPs with convex and non-convex solution spaces. We would like to remind the reader that the theoretical base of conferring robustness and stability to

the solutions of such CSPs is to select a solution with a high number of feasible contiguous surrounding solutions. However, the latter criterion can be adapted to scheduling problems due to their particular characteristic: their domain values represent time units. Thus, when there is an incident at some point, lower values represent time that has already happened and they can not be used for dealing with the incidence. For this reason, it was stated that lower feasible neighbour solutions do not improve the robustness nor the stability of a solution of a CSP that models a scheduling problem. Thus, the adaptation of our approaches to scheduling problems consists in restricting the feasibility checking only to the neighbour assignments that are greater than the analyzed one.

The scheduling adaptations have been formalized and applied to a benchmark from the literature in order to show graphically the slack of the schedules obtained by our approaches. Furthermore, the scheduling adaptation for the approach that searches for super-solutions has also been analyzed in this chapter. The first obtained results show that in situations in which it is not possible to add a slack of k time units after each task, our approaches find schedules with greater number of buffers and also they are longer in average. The enumeration-based technique obtained slightly better results than the search algorithm for $k > 1$, since the latter has the limitation of not finding the optimal solution because the cut-off time is reached (as the technique that maximizes the (1, 0)-repairability). However, this approach has the limitation of the spatial cost, which is significantly higher that what is required by the searching algorithm.

Although the trade-off between the makespan and the robustness has been analyzed in the literature, we have also analyzed it with this scheduling problem in order to corroborate the behaviour of the adaptation of our theoretical approach of selecting solutions with greater contiguous neighbours. As expected, the greater the makespan, the more robust we found the obtained schedules to be.

Chapter 9

Evaluation

The main purpose of this chapter is to evaluate the quality of the solutions obtained by the approaches presented in this dissertation. In Chapter 8, a preliminary analysis was done for a case study of a toy scheduling problem. In this chapter, experiments are carried out on both structured and random CSPs. Among the structured CSPs there are scheduling benchmarks, since these problems are complex and often come from a dynamic and uncertain environment. The purpose of evaluating random CSPs is to analyze the performing of the approaches based on a wide range of CSP parameter setting, as well as the analysis of non-convex solution spaces.

The evaluation performed in this chapter is based on the two main features associated with solutions obtained by proactive approaches: stability and robustness (see Section 1.4.1). In addition to assess the approaches presented in this thesis, we also evaluate another proactive method that does not require specific additional information about the dynamism: super-solutions (Hebrard, 2006). This approach was explained in detail in Section 1.4.3. In addition, we compute and analyze solutions obtained by an ordinary CSP solver.

First, we describe in this chapter the problems setting, providing specific technical information that it is necessary for the correct performing of the experiments. Subsequently, we explain the dynamism simulation, which is used for measuring the robustness of the solutions obtained. At this stage, we explain several experiments performed to the approaches presented in this dissertation, which are presented as it follows. First, the probabilistic enumeration-based approach is described, followed by the enumeration-based approach for linear CSPs; and subsequently the general methods of the enumeration-based approach by coverings and the search algorithm are evaluated. Finally, all the conclusions and limitations of the approaches extracted from the experimental results are summarized.

9.1 Problems Setting

In this section it is explained in detail the necessary information for performing all the experiments presented in this thesis. This description is divided in three parts: description of the structured and random CSPs used for the evaluation, the explanation of the comparison of our techniques with other approaches and the description of the solvers used and the parameters setting.

9.1.1 Structured and Random CSPs

For the experiments performed in this chapter, we used benchmarks from two different sources. Some of the benchmarks are WCSPs that were acquired from Givry's web page [1]. The format of these benchmarks is the WCSP file format (see Appendix A.1 for an explanation of such format). In this format, the constraints are extensionally represented. The original soft constraints were removed from the problems due to our technique has not been developed for constraint satisfaction and optimization problems (CSOPs). The other set of benchmarks was acquired from Christophe Lecoutre's web page [2]. The format of these benchmarks is in XCSP (the detailed description of the format can also be found in this web page). Among all the benchmarks analyzed, there are scheduling problems, due to they are complex real life problems that usually come from dynamic and uncertain environments.

For generating non-convex random CSPs, we used the RBGenerator 2.0, which can also be found in Christophe Lecoutre's web page. These random CSPs are composed of non-convex constraints (therefore, they are extensionally represented). Due to this non-convexity property, the bounds consistency technique (see Section 2.5.2) cannot be used. The CSP parameters: arity, number of variables, domain size, number of constraints and tightness are configurable. The domain values of these random CSPs are integer values in the interval $[0, |D| - 1]$, where $|D|$ is the domain size.

Furthermore, for some experiments we used finite linear CSPs. Since it is not possible to generate this type of CSPs with the Lecoutre's random generator, we implemented a random linear CSP generator. As well as RBGenerator 2.0, the domains are integer values in the interval $[0, |D| - 1]$. The random linear constraints are generated according to Equation 2.3. In addition, all of them must satisfy the correct number of valid tuples according to their tightness (see Equation 2.1). However, there exists the difficulty of balancing the random linear constraints with the selected tightness because we must check that the number of tuples invalidated for the linear constraint coincides with the tightness value. Hence, we allowed a variability of 10% of the fixed tightness for the generation of random constraints. Moreover, all constraints must be non-redundant, that is, a new random constraint must invalidate at least one different

[1] *http://www7.inra.fr/mia/T/degivry/*

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

tuple more than the rest of the generated constraints. In order to generate n-ary random linear constraints, n random variables are selected. Each variable is preceded by a plus or negative sign and a coefficient of 1 or 2 multiplies it, all of these options have the same likelihood of being randomly selected. Finally, the lower/greater or equal sign is randomly selected as well, and based on it, the domain sizes and the tightness, the constant term is fixed.

In the following, an idea that will be further explained is introduced. It is not possible to distinguish robustness according to the dynamism assumptions made in this dissertation when there only exists one or few solutions scattered over the solution space. This is due to the fact that in these cases, all the solutions are located in a bound. For this reason, benchmarks and random CSPs that have such characteristics are not considered for the robustness evaluations.

9.1.2 Uninformed Approaches Comparison

Throughout this chapter, we compared the solutions obtained by our the approaches with solutions obtained without specifying any robustness/stability criterion (referred to as "simple solutions"). This analysis has not been included as an alternative to our approaches obviously, but in order to detect whether there are cases in which all solutions have similar robustness/stability.

In the experimental results performed, we do not include proactive approaches from the literature that use specific dynamism information. This is due to the fact that the quality of the solutions depends on such known information and therefore solutions obtained by approaches that use different dynamism data can not be properly compared. Nevertheless, we incorporated in the evaluation a proactive approach that does not require extra dynamism data: super-solutions (Hebrard, 2006). We would like to emphasize that this technique searches for stable solutions, but not for robust solutions. The main focus is in (1, 0)-repairability (see Section 1.4.3). To implement this technique, we modified Algorithm 7.3 (B&B-$\mathcal{N}_k$) by exchanging MGAC3-$\mathcal{N}_k$ and GAC3-$\mathcal{N}_k$ algorithms for MAC+ and GAC+, respectively. The solutions obtained by this technique are referred to as "super-solutions". Furthermore, for scheduling problems, the extension explained in Section 8.7 has been included for the approach that searches for super-solutions. For such problems, this approach considers a k parameter, as well as our approaches introduced in Chapter 6 and Chapter 7. For non-scheduling problems, both the ordinary CSP solver and the super-solutions solver do not consider the k parameter for solving the CSPs.

Regarding the approaches presented in this dissertation, the abbreviations for referring to the solutions obtained by them are as follows: "WCSP-mod solutions" for enumeration-based techniques and "neighbours solutions" for the search algorithm (*restarting-completion*). For *restarting-scratch*, the solutions are differentiated with a "(R)".

9.1.3 Solvers and Parameters Setting

Subsequently, we explain the solvers used for carrying out the experiments. We use the WCSP solver ToulBar2 (see Givry's web page) for the enumeration-based approaches. One of the CSP formats accepted by the solver is the WCSP format (see Appendix A.1). However, the solver does not support the XCSP format. Hence, we implemented a parser from the XCSP to the WCSP format. Furthermore, we also implemented our own solver for incorporating the search algorithm introduced in Chapter 7. For coding our solver, we used the classes and structures that Christophe Lecoutre provides in package Tools2008, which can be downloaded from Lecoutre's web page. Thus, our solver supports the XCSP format. For implementing the approach that maximizes the (1, 0)-repairability, as well as for obtaining a simple solution, we used our solver (in order to make a fair measurement of the computing times). This solver includes the geometric restart (Section 2.6.3) and bounds consistency (Section 2.5.2). The restarting parameters setting were fixed as follows: the scale factor to 10 and the multiplicative factor to 1.5 (common parameters for such technique according to the related literature). Unfortunately, it was necessary to use a different solver for the enumeration-based approaches because they generate WCSP models. Throughout all the evaluation, we included the computational times of both solvers, however, the fact that the solvers are different should be taken into account.

All the approaches evaluated with the exception of the ordinary CSP solver, are also 'anytime' techniques, so we fix a maximum time cutoff, which is different according to the difficulty of the experiments. Experiments were run on an Intel Core i5-650 Processor (3.20 Ghz). In order to evaluate the quality of the solutions obtained, we measure stability and robustness. We use the robustness and stability measures described in Section 3.3. Specifically, the stability is measured by the (1, 0, c)-repairability, which incorporates the distance c parameter to the repairable values; and the robustness is measured by the remoteness of the solution to the bounds of the solution space. For scheduling problems there exist specific robustness measures (see Section 8.3) that were computed as well. For all the tables presented in this chapter, the best robustness/stability results obtained are highlighted in bold.

9.2 Dynamism Simulation in Uninformed Environments

In this section we explain the dynamism simulation performed for the robustness evaluation of the solutions obtained by the approaches that do not require extra detailed dynamism data. To the best of our knowledge, there are no benchmarks of DynCSPs (see Definition 2.3.1) in the literature, so authors working in this field simulate changes in CSPs in order to generate DynCSPs. In this thesis we assume that there does not exist detailed knowledge about the future possible changes in the problems. Hence, the difficulty of simulating such dynamism. According to the little

dynamism assumptions made about CSPs with ordered domains (see Section 3.1), we present two different types of dynamism simulation: over the search space and over the solution space.

9.2.1 Dynamism in the Search Space

A straight-forward idea for the dynamism simulation is to restrict the original constraints and domains. Since we assume that there is not information about which bounds are more likely to undergo modifications, the dynamism of the bounds is randomly performed with an equal likelihood distribution. Thus, we 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 created two types of DynCSPs: with dependent/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 of the solution space of $CSP_{(i-1)}$. However, if the changes are independent, each $CSP_{(i)}$ includes a restrictive modification with respect to some bound of the solution space of $CSP_{(0)}$. Thus, in the first case changes are cumulative but not in the second case. For dependent changes, we 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 to carry out the same proportion of restrictions for instances with different number of variables and/or constraints.

For simulating dynamism in linear CSPs, dependent restrictive changes over the bounds of the original random linear CSPs are made. Specifically, we modify the constant term of their constraints (see Equation 2.3) in i units with regard to $CSP_{(0)}$ (each time the constant term is restricted in one unit for all the constraints of the problem). In this way, we generate as many static CSPs as possible, with the condition that they must maintain at least one solution. More detailed examples of simulations in linear constraints can be found in (Climent et al., 2009a,c).

The simulation of restrictive changes over non-convex constraints was done by randomly selecting an invalid tuple located next to any bound of the CSP and invalidating all the tuples that surround it to a distance of d. This idea was inspired from (Wallace and Grimes, 2010), where the 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). Inspired by this simulation, we added the idea that tuples located on the bounds of the solution space are more likely to become invalid after changes in the original problem. For DynCSPs whose changes are dependent we fixed $d = 1$. (The changes are cumulative; therefore, a bigger d than 1 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 selected magnitudes at random from the interval $d \in [1, \ldots, max_d]$ for each $CSP_{(i)}$; because these changes were non-cumulative, they could be of larger magnitude without unduly restricting the original CSP. We fixed different values of max_d commensurate with the domain sizes of the problems analyzed. Note that if $d > 1$ valid tuples that are not located on a bound can also became invalid (but they have to be at distance lower or equal to d from some bound).

For each DynCSP, new restrictive CSPs were generated until the obtained solution became invalid. Thus, l indicates the number of restrictive changes that a solution is able to resist for a DynCSP. For each experiment performed, we generate a certain number of random different DynCSPs for each problem and compute the average number of changes satisfied, which can be considered a measure of the robustness of the solutions according to Definition 3.3.1. The higher the number of changes satisfied l (or its percentage) is, the more robust the solution is, due to the solution is able to remain valid after a high number of changes in the constraints of the problem. Furthermore, in order to perform uniform changes simulation over the constraints when the experiments involves random CSPs, we used an option of the RBGenerator 2.0 that merges the constraints of similar scope. We would like to point out that only for the probabilistic-based approach, we added random dynamism functions to the bounds of the CSPs evaluated. This is deeply explained in Section 9.3.

9.2.2 Dynamism in the Solution Space

In addition to the above dynamism simulation described, we also developed a dynamism simulation over the solution space. A reason for performing this type of dynamism is that a restriction in a constraint/domain does not necessarily implies a restriction over the solution space. The latter happens in cases in which there also exist other constraints/domains that are partially/totally more restrictive than the aforementioned simulation. In order to perform restrictions over all the solution space, it would be necessary to compute the complete set of solutions of a CSP. However, as mentioned in Section 2.4, this option is not feasible due to it is usually NP-hard. Instead, we sample the closest surrounding neighbours (at some distance k). For performing this sampling, we make a certain number of random modifications of magnitude k over the values assigned to the variables of the solutions. Thus, we fix a number of variable assignments to be modified (denoted as $nbVarMod$). Thus, if a closest surrounding neighbour obtained by such modifications is not a solution of the CSP, it means that the analyzed solution could become unfeasible after a change of magnitude k or greater to the original bound/s that invalidate such neighbour. On the contrary, if the neighbour is a solution of the CSP, this means that this restrictive modification would not invalidate the analyzed solution. Therefore, satisfiability checking of a ran-

dom sample of the neighbours of the solutions provides an estimation of the likelihood that the solutions will remain valid, which represents an estimation of their robustness. A first proposal of dynamism simulation in the solution space was presented in (Climent et al., 2009b; Salido et al., 2009) and was further extended in (Climent et al., 2014).

We believe that both types of dynamism simulations (in the search space and in the solution space) are interesting for the analysis of the robustness. Commonly, both simulations coincide when the level of interaction between constraints is low. That is to say, when the scope of the constraints differs significantly. However, we consider that it is also interesting to include the dynamism simulation over the solution space because it provides a good view of the interaction of the changes over all the bounds of the solution space. For instance, in scheduling problems, as mentioned in Section 8.2, one of the most common type of dynamism is delays over the tasks of the schedule, which represent a change over the solution space. We also would like to mention that according to the dynamism assumptions in scheduling problems, for such type of problems the robustness was evaluated with robustness measures from the literature (see Section 8.3).

9.3 Probabilistic Technique Evaluation

In this section, the probabilistic enumeration-based approach introduced in Chapter 4 is evaluated. As its name indicates, this approach requires probability distributions. Specifically, the dynamism information that the approach considers is composed of two functions associated with the constraints and domains of the CSPs: their probability of undergoing a restrictive modification ($p(C_i)$) and the magnitude of the restrictive modification ($d(C_i)$). As we mentioned in Chapter 1, the information dependency of a proactive approach determines its usability. As previously mentioned, there do not exist dynamic benchmarks in the literature, either benchmarks with such dynamism information. Hence, we generated random dynamism functions associated with the constraints and domains of CSP benchmarks from the literature: Academics and Planning (acquired from Givry's web page). For each benchmark, we generated 10 random instances with different random dynamism functions. For all the instances, we fixed a cutoff time of $600s$. The experimental results presented in this section can be classified in two different parts: firstly, depending on the number of restrictive constraints added to the modeled WCSPs and secondly, depending on the dynamism functions.

The dynamism information associated with the CSPs determines the robustness measurement of the solutions obtained. In Section 9.2 we explained several ways for measuring the robustness when the dynamism data is unknown. In contrast, for this probabilistic-enumeration based approach, each bound of the CSP has a likelihood and a magnitude of change. For this reason, this information should be considered

in the robustness measurement. In Section 4.2 we explained the new constraints generation for the modeled WCSPs. Due to these new generated constraints are based on the dynamism information associated with the bounds of the original CSPs, they represent future possible modifications that the bounds may undergo. Therefore, the number of new generated constraints that a solution satisfies is a robustness indicator considering this dynamism information. In (Climent et al., 2010) there are examples of the simulation of restrictive modifications of informed constraints. Moreover, the evaluation of the probabilistic enumeration-based approach can be found in (Climent et al., 2012a).

In the tables and figures presented in this section, the number of new modified constraints generated for a problem is denoted as m-$\mathcal{C}$ and from this set, the ones that are satisfied by a solution, are denoted as s-$\mathcal{C}$. Note that each addition of a modified constraint to the previous static CSP_i represent a dependent change and the union of all the static CSPs composes a DynCSP.

9.3.1 Experiments based on the Number of Constraints Additions

In this first developed analysis our aim was to determine the effectiveness of the probabilistic enumeration-based approach presented in this dissertation and to analyze the influence of the parameter w in the robustness of the solution obtained. We recall that the parameter w is the number of new generated constraints (which are added in the modeled WCSP) for each bound of the original CSPs. This parameter is fixed by the user according to the desired granularity. Thus, the greater the w parameter is, the slighter the difference between the new constraints is.

Table 9.1 shows the number of original constraints ($\mathcal{C}$) of the problems analyzed. In addition, it can be observed how many new modified constraints were generated (m-$\mathcal{C}$). This number is directly related to the parameter w. However, there is a saturation point for the number of new modified restrictive constraints that it is possible to create. This happens when the granularity is so high that it is not possible to restrict more the previous generated constraint considering the dynamism parameters. When this happen for certain w value, we call it the saturation point. This is further explained in Section 9.3.1. For each analyzed problem, it is shown the number of modified constraints satisfied (s-$\mathcal{C}$) by the solutions computed by the probabilistic enumeration-based approach (referenced as "WCSP-m") and by an ordinary CSP solver (referenced as "simple"). Furthermore, the difference in the number of satisfied modified constraints between both approaches is represented in table as D, that is to say, that D is the improvement of our approach with respect to the ordinary CSP solver.

Table 9.1: Robustness analysis based on the w parameter.

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

The robustness results obtained for the analysis show that for all the problems, the number of new modified constraints satisfied (s-$\mathcal{C}$) for the solutions found by the probabilistic enumeration-based approach is always greater than by an ordinary CSP solver. Therefore, the solutions obtained by our technique are more robust. The highest improvement of the probabilistic enumeration-based approach is achieved for the problem *slangford_3_11* because the obtained solutions have the highest difference in the number of modified constraints satisfied (D) with respect to an ordinary solver.

Figure 9.1 shows a graphic with four of the most representative problems of the Table 9.1 and the difference in the number of modified constraints satisfied. In both, this figure and Table 9.1, it can be observed the evolution of the robustness based on the w parameter. From them, it can be concluded that the number of modified constraints satisfied (m-$\mathcal{C}$) by our approach and the improvement with respect to the ordinary solver (D) are directly related to w: the greater w is, the greater these robustness measures commonly are. For example, the best robustness improvement is achieved for $w = 32$ in the *slangford_3_11* benchmark, since the solutions obtained by our technique satisfy 720 new modified constraints more than the solutions obtained by an ordinary CSP solver.

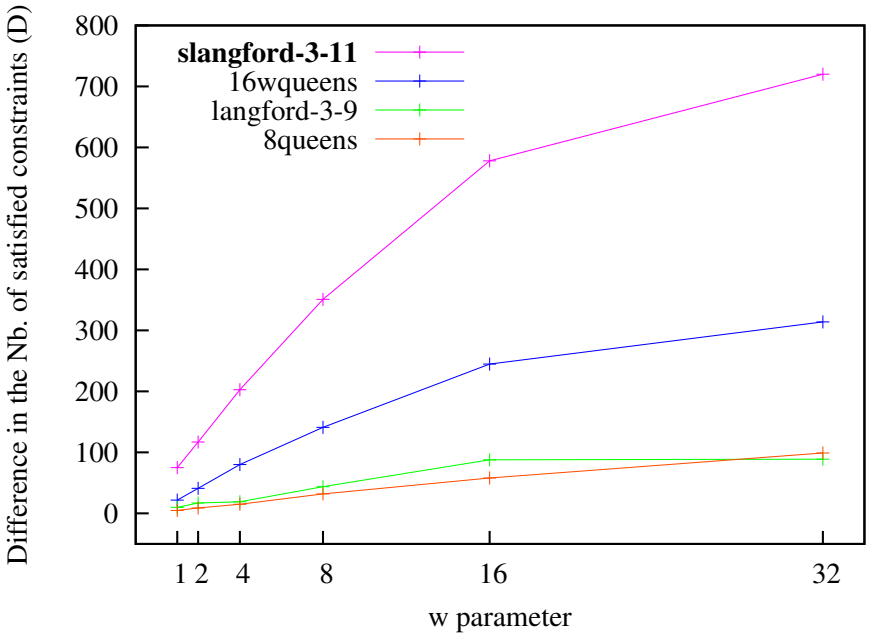


Figure 9.1: Robustness analysis of the benchmarks based on w.

Nevertheless, as previously explained, when w takes the value of the saturation point, it is not possible to increase the number of new constraints (m-$\mathcal{C}$). Thus, in this saturation point, D does not increase. Each problem has a different saturation point, depending on the number of valid tuples for the constraints of the problem.

For example, in Figure 9.1, it can be observed that the problem *langford_3_9* has a saturation point of $w = 16$, because D stops its increase for $w > 16$. Nevertheless, Figure 9.1 shows that the other three problems do not have a saturation point lower than or equal to $w = 32$, because in this point, D has not stopped its increase.

9.3.2 Experiments based on the Dynamism Functions

In this second evaluation performed, the main objective is to analyze the effect of the dynamism functions in the robustness of the solutions obtained. For this purpose, we selected two benchmarks: *driverlog01cc* and *slangford_3_11*, and we fixed $w = 1$.

Table 9.2 shows the number of new modified constraints (m-$\mathcal{C}$), the number of modified constraints satisfied (s-$\mathcal{C}$) and the difference between these values (D). The dynamism functions $p(C_i)$ and $d(C_i)$ are shorted as p and d. It can be observed that the dynamism function $p(C_i)$ does not have a significant effect in the robustness improvement (D). The function $p(C_i)$ is used in the cost assignment to the invalid tuples of the new modified constraints. Thus, the function $p(C_i)$ affects to the global cost of the solutions obtained ($\mathcal{V}(s)$) by increasing its magnitude, but it does not affect significantly to the number of modified constraints satisfied (s-$\mathcal{C}$).

Nevertheless, we can observe in Table 9.2 that the dynamism function $d(C_i)$ has a significant relation with the number of constraints satisfied (s-$\mathcal{C}$) and also with the robustness improvement (D) of the probabilistic enumeration-based approach over the ordinary CSP solver. The latter is shown graphically in Figure 9.2 for the two problems *driverlog01cc* (Figure 9.2(a)) and *slangford_3_11* (Figure 9.2(b)) with a fixed $p(C_i) = 0.2$.

The robustness improvement of our approach over the ordinary CSP solver (D) increases when $d(C_i)$ increases because the percentage of invalid tuples for a new modified constraint also increases. For this reason, it is more likely that a non-robust solution obtained by an ordinary solver satisfies less modified constraints (s-$\mathcal{C}$). However, there is a saturation point which is $d(C_i) = 0.8$ for both problems. At this point the number of new satisfied constraints decreases due to their high tightness, and therefore, it is unlikely that the solutions are able to satisfy a high number of such magnitude of restrictive modifications. For both problems the maximum level of robustness improvement (D) is achieved for $d(C_i) = 0.6$.

9.4 Technique for Finite Linear CSPs Evaluation

In this section we evaluate the enumeration-based approach for finite linear CSPs. To this end, we used our own random linear CSP generator (explained in Section 9.1). We recall that Lecoutre's generator only produces CSPs with non-convex constraints, which are therefore extensionally represented. Furthermore, for these experiments, we apply the dynamism simulation over the search space for linear constraints (see

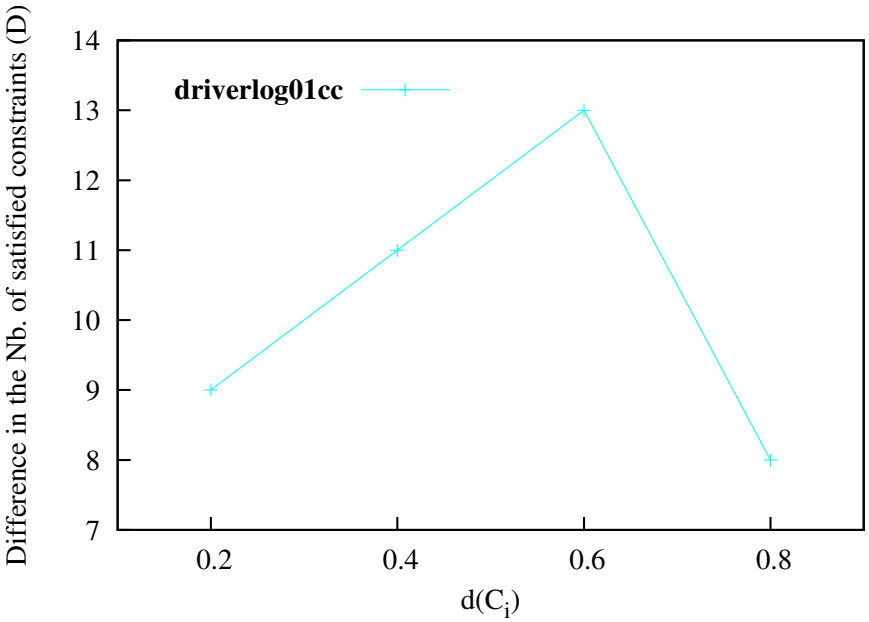
Table 9.2: Robustness analysis based on $p(C_i)$ and $d(C_i)$.

(a) driverlog01cc benchmark ($\mathcal{C} = 472$).

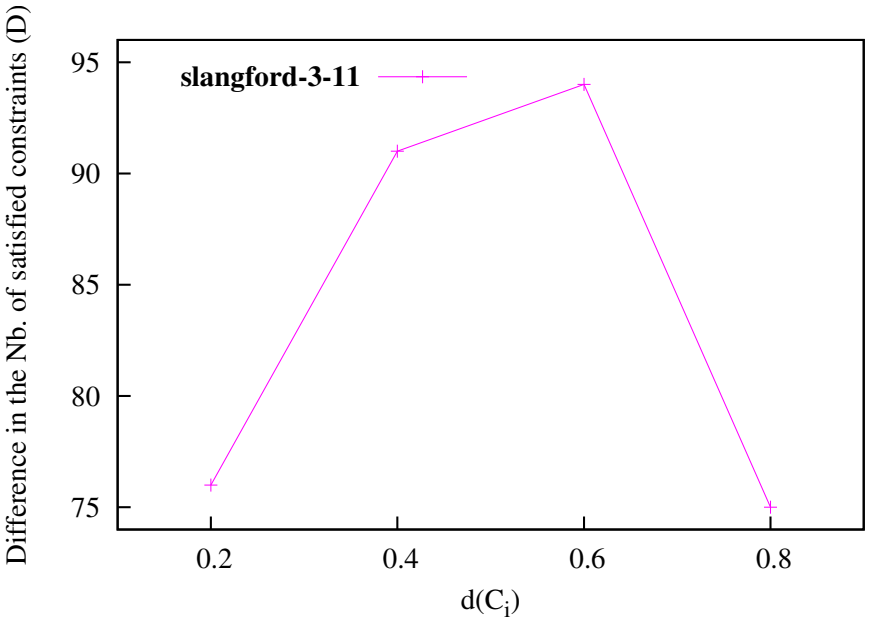
			p=0.2		p=0.4		p=0.6		p=0.8	
d	Approach	m-$\mathcal{C}$	s-$\mathcal{C}$	D	s-$\mathcal{C}$	D	s-$\mathcal{C}$	D	s-$\mathcal{C}$	D
0.2	simple	26	16	9	18	6	17	7	18	7
	WCSP-m		25		24		24		25	
0.4	simple	43	19	11	20	11	22	8	19	12
	WCSP-m		30		31		30		31	
0.6	simple	49	11	13	14	11	14	10	13	11
	WCSP-m		24		25		24		24	
0.8	simple	49	3	8	4	7	3	8	3	8
	WCSP-m		11		11		11		11	

(b) slangford_3_11 benchmark ($\mathcal{C} = 550$).

			p=0.2		p=0.4		p=0.6		p=0.8	
d	Approach	m-$\mathcal{C}$	s-$\mathcal{C}$	D	s-$\mathcal{C}$	D	s-$\mathcal{C}$	D	s-$\mathcal{C}$	D
0.2	simple	550	437	76	437	75	440	72	435	78
	WCSP-m		513		512		512		513	
0.4	simple	550	330	91	331	93	331	89	333	91
	WCSP-m		421		427		420		424	
0.6	simple	550	215	94	217	95	219	91	224	86
	WCSP-m		309		312		310		310	
0.8	simple	550	107	75	108	73	112	65	178	71
	WCSP-m		182		181		117		107	



(a) driverlog01cc



(b) slangford_3_11

Figure 9.2: Robustness analysis based on $d(C_i)$ with a fixed $p(C_i) = 0.2$.

Section 9.2). The following evaluation of the enumeration-based technique for finite linear CSPs is also shown in (Climent et al., 2011).

Table 9.3 shows several binary CSP topologies and the average results obtained for the 10 random instances generated for each topology. The classification is shown according to the CSPs parameters: number of variables: $|\mathcal{X}|$, domain size: $|D|$, number of constraints: $|\mathcal{C}|$ and tightness. For each instance type, we measure the average number of static CSPs (Mod) that could be generated from CSP_0 (until there does not exist a solution for the instance). All the static CSPs of an instance compose the DynCSP. The percentage of static CSPs satisfied by the solutions found by our approach is presented in $\%Sat$. The average runtime (in seconds) of the WCSP modeling phase is denoted as $timeM$. Here, it is encompassed the time for calculating the distances to the bounds and also the time for modeling the WCSP according to the cost functions. In addition, the average runtime for solving the WCSP by the ToulBar2 solver is indicated as $timeS$.

Table 9.3: Percentage of static CSPs satisfied and computing times (s).

$\|\mathcal{X}\|$	$\|\mathcal{D}\|$	$\|\mathcal{C}\|$	$tightness$	Mod	$\%Sat$	$timeM$	$timeS$
50	100	50	0.1	78.2	50.9%	0.22	1.34
100	100	50	0.1	90.8	68.5%	0.23	1.11
150	100	50	0.1	99.3	82.1%	0.23	0.94
100	80	60	0.2	60.8	53%	0.16	0.7
100	120	60	0.2	87.4	56.8%	0.3	2.37
100	200	60	0.2	158	62.9%	0.9	13.72
100	50	30	0.2	46.6	78.8%	0.27	0.2
100	50	60	0.2	36.1	50.4%	0.48	0.38
100	50	80	0.2	10.6	31.5%	0.64	0.44
150	200	60	0.1	184	72.1%	1.16	6969
150	200	60	0.25	143	54.9%	0.81	8405
150	200	60	0.4	95.6	50.9%	0.82	6452

The results show that the enumeration-based approach for finite linear CSPs finds solutions that remain valid for a high number of restrictions over the constant term of the bounds of the CSPs. For example, for instances with $|\mathcal{X}| = 150, |\mathcal{D}| = 100, |\mathcal{C}| = 50$ and $tightness = 0.1$, an average of 99.3 static and feasible CSPs could be generated and the 82.1% of them were satisfied by the solutions obtained by our enumeration-based approach, which indicates the high robustness of the solutions when faced with this type of restrictive modifications. The modeling phase was carried out (in average) in $0.23s$ and the solutions for the modeled WCSPs are found by ToulBar2 solver in $0.94s$ in average. We would like to point out that the modeling time required by our approach for all the instances analyzed, is low in comparison with the solving time. We recall that the little computational time required by this

evaluated approach (see Section 5.3.2) is one of its advantages.

For these experimental results, we presented a wide range of CSP parameters in order to analyze their influence in the robustness of the solutions. The highest percentages are achieved when the number of variables and the domain sizes are high and the tightness and the number of constraints are low. Note that for the rest of the parameters fixed, when the number of variables or the domain sizes increase, the solution space increases as well. In contrast, when the number of constraints or the tightness increase, the solution space decreases. Hence, it can be concluded that the higher robustness performance for these experiments was obtained for the low restricted instances, which are instances with greater solution spaces.

9.5 Coverings Evaluation based on the CSP Parameters

In this section, we make an exhaustive analysis of a wide interval of values for the CSP parameters and their influence in the robustness of the solutions obtained for non-convex instances. For this purpose, we selected the enumeration-based approach by coverings and we used RBGenerator 2.0 for generating 500 random non-convex CSPs instances for each value of each parameter analyzed. Since in this analysis we deal with the general case of CSPs with ordered domains, we use the general covering definition (see Definition 6.1.1). For these experiments, instead of using a fixed k parameter for all the CSP typologies, k is determined according to the characteristics of the topology. We achieved a commensurate k value by fixing it to the smallest value for which there is no tuple whose k-covering is complete for some $C_i \in (\mathcal{C} \cup \mathcal{DC})$. Furthermore, for this evaluation, we also analyze the approach that maximizes the (1, 0)-repairability because it deals with non-convex spaces and it does not require extra additional dynamism data. For all the approaches analyzed, the time cutoff was fixed to $200s$. The solutions obtained by an ordinary solver are also included in the experiments. We performed a dynamism simulation over the search space, applying both types of changes: dependent and independent. For the latter type of change, we fixed $max_d = 5$, commensurate with the domain sizes of these problems. Part of the evaluation presented in this section can be found in (Climent et al., 2013b).

The robustness analysis developed for the CSP parameters show a general pattern according to the level of constrainedness of the CSPs. Firstly, we explain these general behaviours and subsequently more specific information about the evaluation of each CSP parameter is provided. For all the CSPs that are not extremely highly restricted the mean number of changes satisfied by the solutions obtained by the enumeration-based approach is higher than the changes satisfied by the solutions that maximize the (1, 0)-repairability and the simple solutions. However, for CSPs that are very highly restricted, the number of changes satisfied by the three solutions is similar, because in these cases the CSPs have very few solutions and consequently the distances of the solutions from the bounds is very low. Note that for most of these instances, the

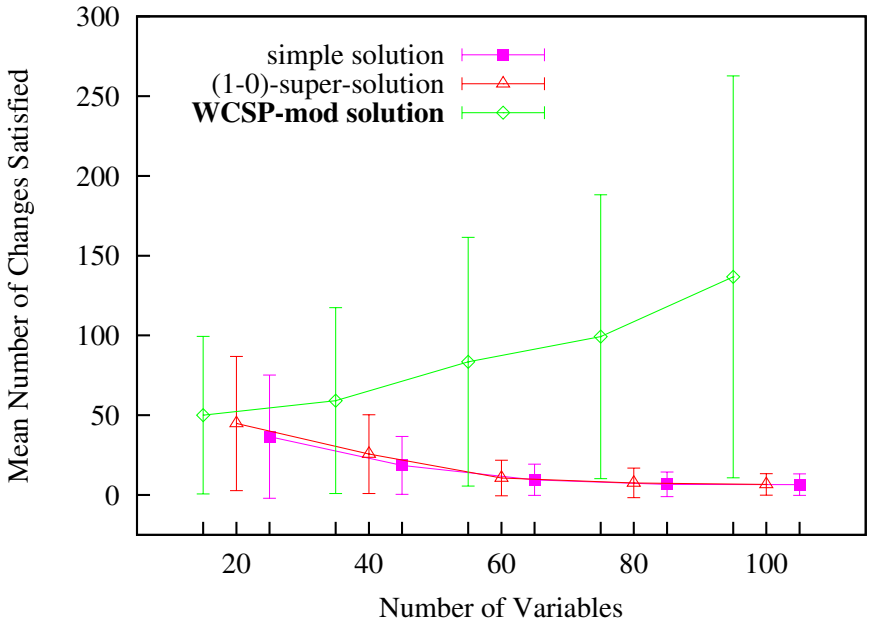
number of solutions is as low that the solutions are scattered, so the likelihood that the solutions are located on the bounds of the solution space is very high. It could even be the case that there do not exist solutions with at least one feasible neighbour solution inside its *1-covering*. In such case, all the solutions are equally non-robust according to the limited assumptions made for CSPs with discrete and ordered domains. In this section we present the experimental results obtained for the following parameters of the CSP: number of variables, domain size and tightness. Similar results were obtained for other parameters such as domain size and constraint graph density, which are not included (for avoiding data repetition).

For all the following figures, the parameters of the analyzed CSPs 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 6.1 for calculating the coverings and also the time necessary for modeling the CSP as a WCSP. Data for dependent and independent changes are shown separately.

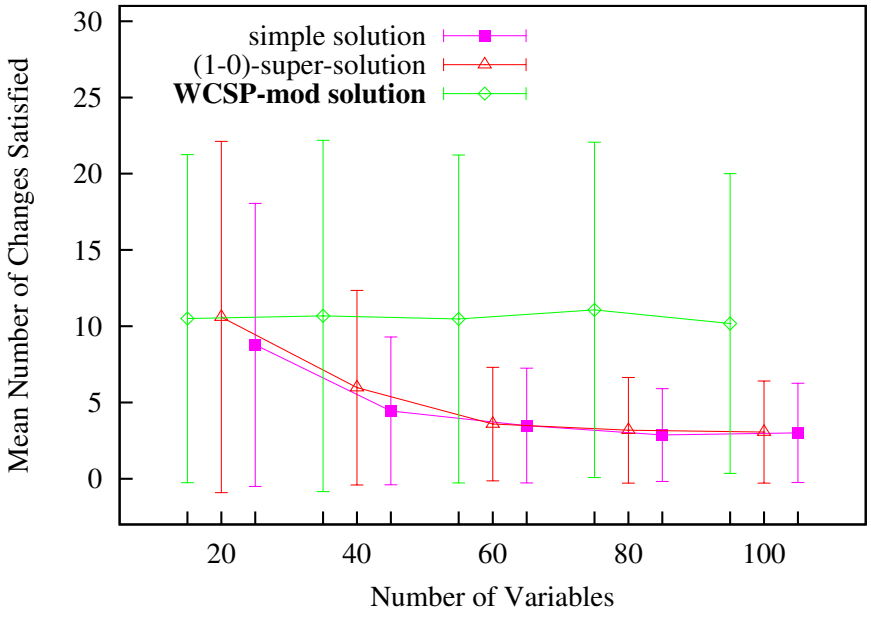
Figure 9.3 shows robustness results when the number of variables of the CSP is varied. In Figure 9.3(a) we see that the WCSP-mod solutions satisfy a higher mean number of dependent changes for higher numbers of variables. (This is because the larger CSPs are less constrained and therefore the number of solutions is higher.) The maximum number of cumulative changes required to invalidate the WCSP-mod solutions is 136.72, which was found for the maximum number of variables evaluated (100 variables). However, for the same condition, the simple solutions and the solutions that maximize the $(1, 0)$-repairability 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 improvement of the enumeration-based approach over the other two approaches is much lower because this problem is highly constrained. The robustness obtained by our approach for DynCSPs with independent changes (see Figure 9.3(b)) is more 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 computing the coverings and subsequently penalizing the valid tuples of the modeled WCSP.

Figure 9.4 shows the WCSP modeling time (including the covering computation) required by the enumeration-based approach 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 Vari-



(a) Dependent Changes



(b) Independent Changes

Figure 9.3: Robustness-variables analysis (<2, $|\mathcal{X}|$, 18, 200, 0.22 >).

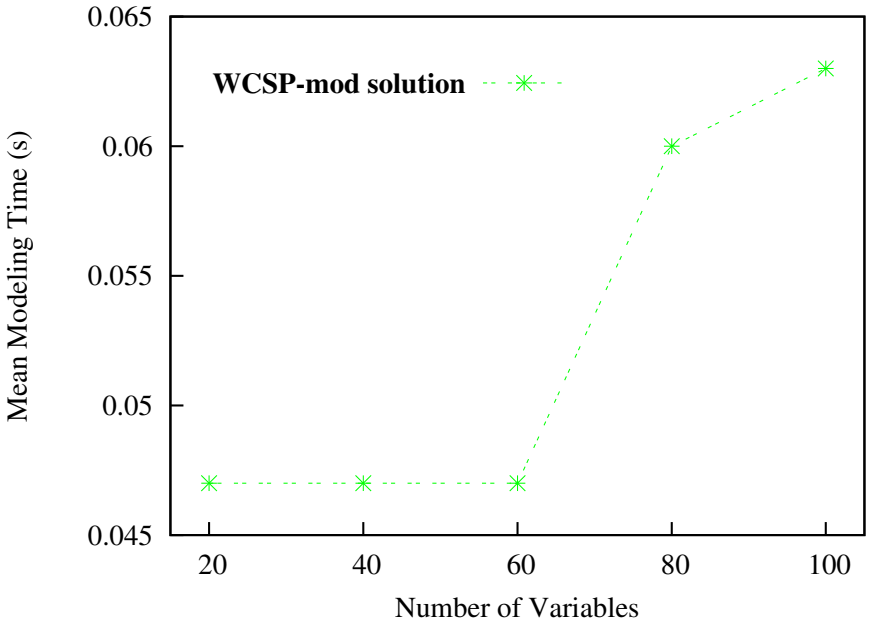


Figure 9.4: Modeling Time (s) of WCSP-mod technique.

ance (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 9.5 shows the robustness analysis for different range of domain sizes. In Figures 9.5(a) and 9.5(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 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 the WCSP-mod solutions were obtained for the maximum domain size analyzed, which is 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.

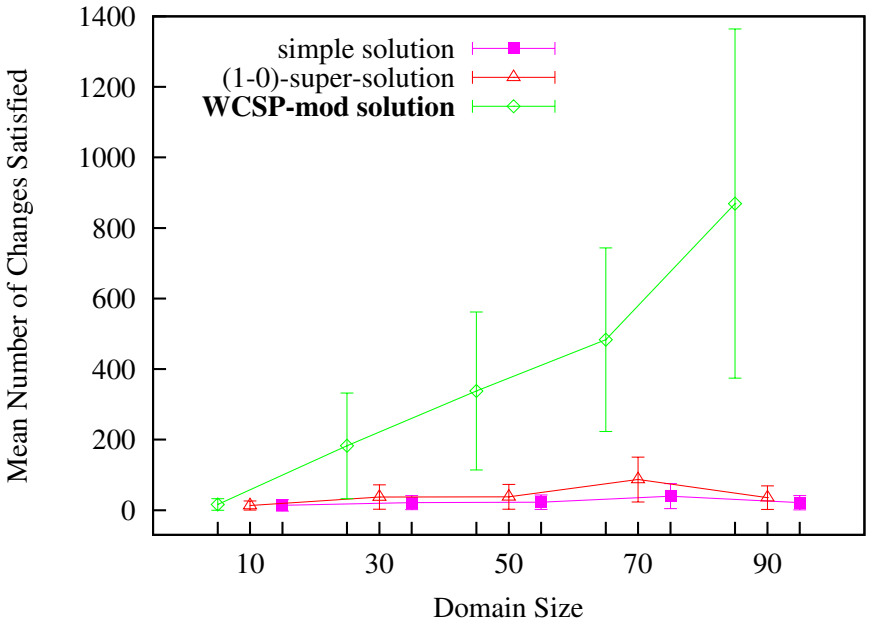
As shown in Figure 9.6, the maximum difference in modeling time required for the different types of instances is $0.78s$. The main reason for the increase with increasing domain size is that this parameter is directly related to the number of valid tuples of the constraints, which has a high impact on the computational time of Algorithm 6.1.

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 $p = 0.01$.

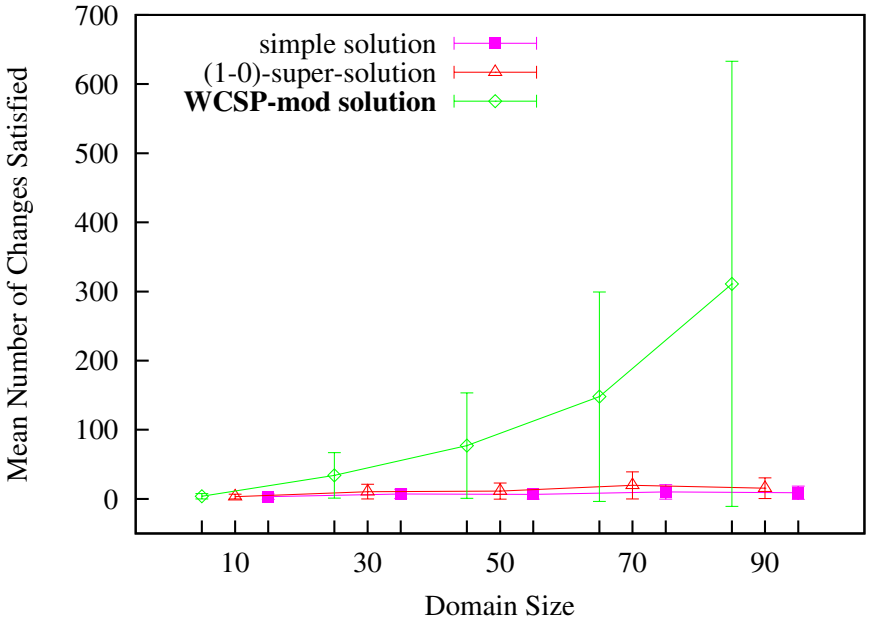
In Figure 9.7(a) the robustness analysis according to several tightness values is shown. It can be observed that the mean number of cumulative changes before solution breakage decreases as tightness increases and the problems become more constrained overall. The maximum mean number of cumulative changes allowed by the WCSP-mod solutions is 289.83, which was found for the lowest tightness value (0.1). For this instance, the simple solutions and the solutions that maximize the $(1, 0)$-repairability 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 enumeration-based approach by coverings 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).

From these experiments we conclude that for CSPs that are not highly restricted, the mean number of changes before solution breakage for solutions obtained by the enumeration-based approach is much greater than the number of changes for solutions obtained by either of the other techniques. 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, 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. For problems of low constrainedness, the solution space is greater and therefore the likelihood that there are solutions surrounded by neighbour solutions is also higher. However, this feature presents a disadvantage for techniques that search for (1, 0)-super-solutions because the likelihood that there is a large percentage of solutions with a high number of repairable values is greater also. In this case, among all equally stable solutions this technique finds a solution according to the lexicographical value ordering.

With respect to the two types of DynCSPs generated (dependent/independent changes), we found that the mean number of dependent changes satisfied is higher (and less variable) than the mean number of independent changes satisfied. This is largely due to the magnitude of the changes simulated. Figure 9.7(b) shows that the mean number of independent changes allowed by solutions found by the enumeration-



(a) Dependent Changes



(b) Independent Changes

Figure 9.5: Robustness-domain analysis ($< 2, 25, |D|, 75, 0.45 >$).

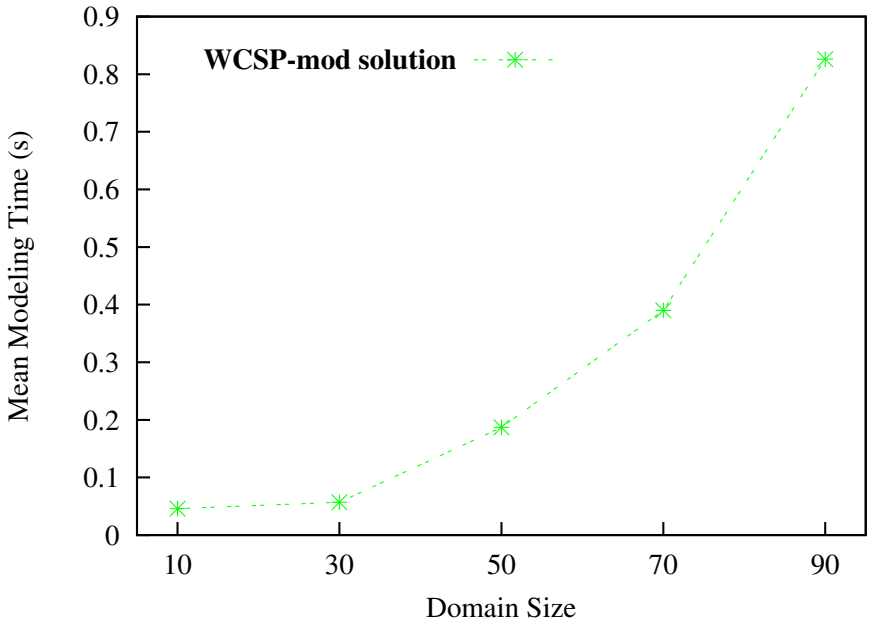


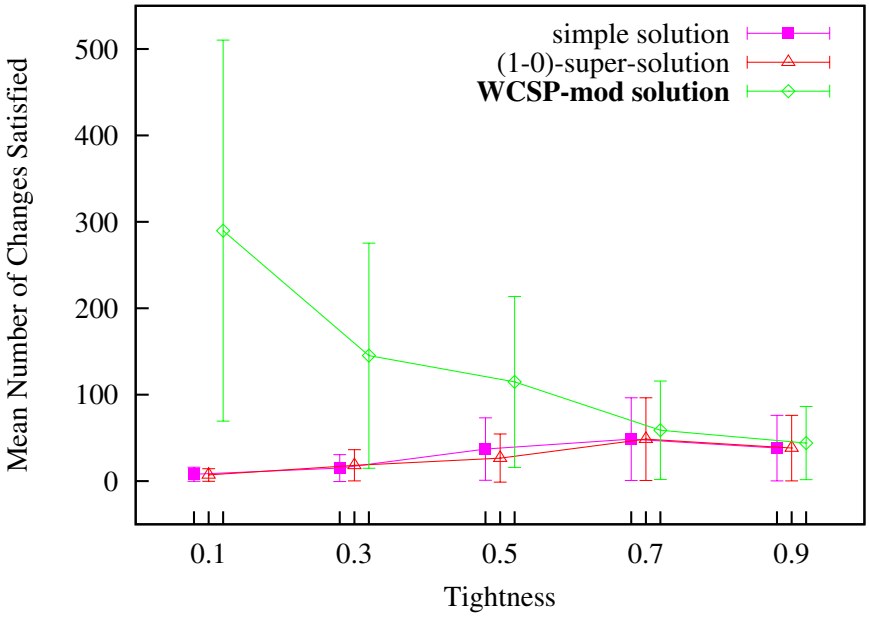
Figure 9.6: Modeling Time (s) of WCSP-mod technique.

based approach 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 9.8 shows the modeling time required by our technique for each instance. The maximum difference between instances was $0.155s$. As mentioned, Algorithm 6.1 spends more time finding coverings for low tightness values (because the number of valid tuples is also low).

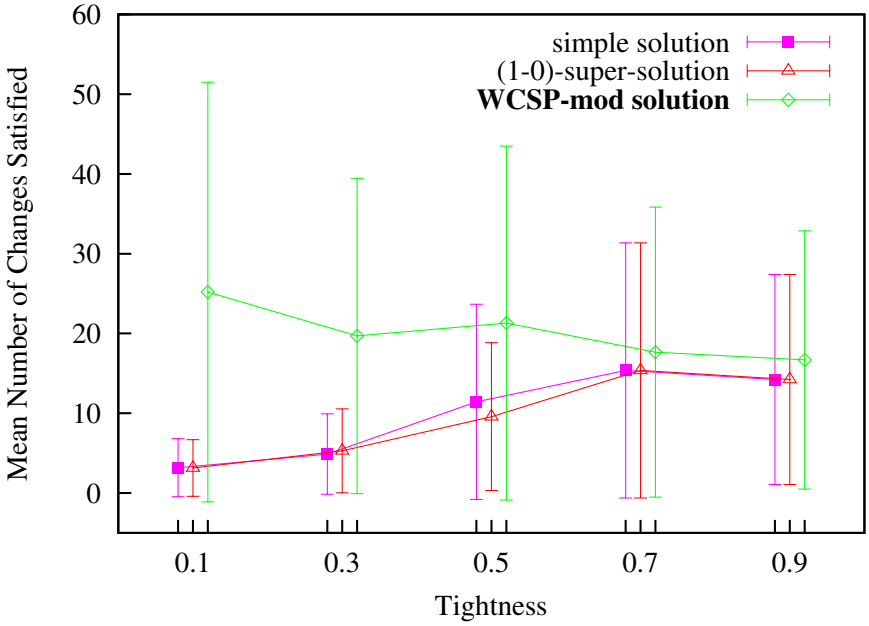
These results were evaluated statistically with a two-factor Analysis of Variance (ANOVA) followed by the Tukey HSD test for differences between pairs of individual means ((Hays, 1973; Winer, 1971)). For both tightness experiments, the ANOVA gave F values that were highly significant statistically. Because of the large number of tests made under each condition, the Tukey HSD statistic was very small (about 0.1 and 0.08), so almost all differences between individual means were statistically significant for $p = 0.01$.

9.6 Non-convex Random CSPs Evaluation

In this section, we evaluate the robustness and stability of the solutions obtained by the two approaches that deal with general non-convex CSPs with ordered domains: the enumeration-based approach by coverings and the search algorithm (Climent et al., 2014). Unlike the enumeration-based approach for Linear CSPs, these approaches



(a) Dependent Changes



(b) Independent Changes

Figure 9.7: Robustness-tightness analysis ($< 2, 40, 25, 120, \textit{tightness} >$).

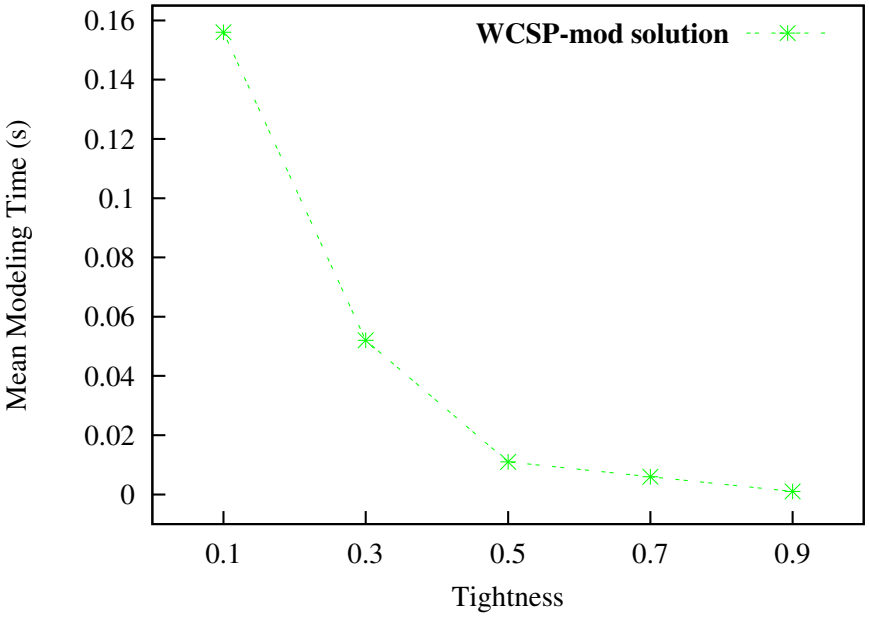


Figure 9.8: Modeling Time (s) of WCSP-mod technique.

are able to deal with non-convex CSPs. For the evaluation of such type of CSPs, non-convex random CSPs were generated by the RBGenerator 2.0. The generated CSPs have 25 variables with domain size 30 and 200 binary extensional constraints. Domain values are integer values in the interval $[0, 29]$. In this section, a range of tightness values are analyzed in order to extract conclusions about their effect in the quality of the solutions obtained. Specifically, we analyze the tightness values: 0.1, 0.2, and 0.3. (Note: 0.34 is the critical value of the tightness of this CSP typology). For each tightness we generated 10 random instances. Since in this analysis we deal with the general case of CSPs with ordered domains, we use the general covering definition (see Definition 6.1.1) for the enumeration-based approach and we fix the set of operators for our search algorithm (see Algorithm 7.1) to $\oplus = \{\{>, +\}, \{<, -\}\}$ and its value selection heuristic is the second, which maximizes $|\mathcal{N}_k(x, v, s, \oplus)|$ starting from intermediate values. The time cutoff was fixed to $100s$.

For the evaluation of these both approaches, we pretend to analyze the effect in the approaches performance of situations where a neighbour assignment is feasible for a bound of the CSP but is not for another. In Section 9.2 we explained that changes made in the solution space instead of in the search space consider such situations. Therefore, the dynamism simulation selected for this evaluation was the dynamism over the solution space. For such purpose, we sample the feasibility of the neighbourhood of the solutions by making certain number of random modifications of magnitude $k = 1$ over the values assigned to the variables of the solutions. The number

of values assigned to the variables of the solutions that are modified, is denoted as $nbVarMod \in [1 \dots 10]$. For each value of $nbVarMod$, we sampled 500 neighbours over the solution analyzed and checked their feasibility. The average number of feasible neighbours for each type of solution are shown in Table 9.4.

It can be observed that our search algorithm with either restarting options, dramatically outperformed the ordinary CSP solver and the technique that maximizes the (1, 0)-repairability. It also outperformed the WCSP modeling approach for tightness 0.2 and 0.3. The weakness of the modeling approach is when there exist situations in which there is a high relationship between constraints, because it computes feasible neighbours for each constraint boundary. Thus, the higher the tightness, the higher the likelihood of the existence of neighbour tuples that are feasible for one constraint/domain but not for another one. These conflicting situations are less frequent in very unconstrained instances. Hence, for tightness 0.1, the performance of the modeling approach is better for a high number of variables modified in the sampling ($nbVarMod$). In regard to our search algorithm (see Algorithm 7.3), the restarting-completion option provides better results than restarting-scratch (differentiated with "R") for very unconstrained instances, while they preform similarly for higher tightness values. In Figure 9.9(b) we selected the $nbVarMod = 2$ to emphasize trends in robustness and stability as a function of varying tightness.

Table 9.4: Robustness-tightness analysis ($< 2, 25, 30, 200,$ *tightness* $>$).

tightness	0.1					0.2					0.3		
$nbVarMod$	2	4	6	8	10	2	4	6	8	10	2	4	6
Approach	Average Number of feasible neighbours in the sample												
simple	7.2	0.6	0	0	0	0.2	0	0	0	0	0	0	0
super	8.8	0.6	0	0	0	1	0	0	0	0	0.4	0	0
WCSP-m	152.4	60.2	24.5	**12.8**	**5.8**	5.7	0.1	0	0	0	0.4	**0.1**	0
neigh	**206.8**	**75**	**27.4**	10.8	3.8	**36.2**	**2.5**	0	0	0	**2.8**	**0.1**	0
neigh(R)	191.6	74.4	24.4	7.9	2.9	33.9	2	**0.5**	**0.1**	0	2.5	0	0

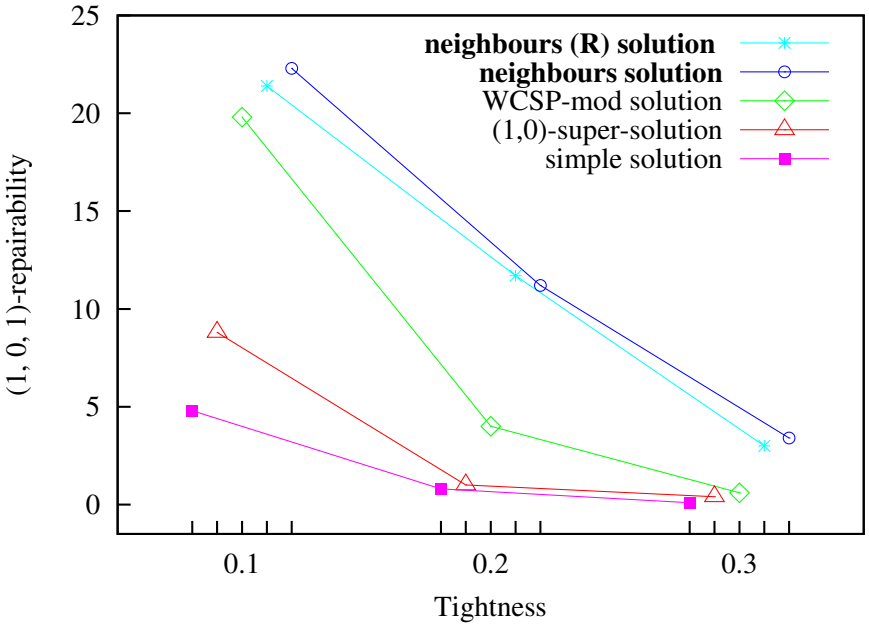
For the stability measurement, the (1, 0, 1)-repairability is used (see Definition 3.3.3), which measures the number of variables that can be replaced by a value located at a distance one from the value assigned without modifying the rest of values in the solution. Stability results are shown in Figure 9.9(a). As mentioned, if a solution value is lost, the objective is to find the closest repairable values. For this reason, our algorithm does not consider feasible values that are k units greater or smaller than the value assigned, since this could result in future solutions where the Manhattan distance between the new solution and the original one would be exaggeratedly great (see Section 3.3.1 for a detailed explanation). In contrast, the technique that maximizes the (1, 0)-repairability considers any value as a repairable value. This fact represents a disadvantage when searching for close repairable values in ordered domains. This can be observed in Figure 9.9(a), where we can see the poor performance of the super-solutions solver for the (1, 0, 1)-repairability.

Note that for CSPs that are very highly restricted, the stability and robustness of the solutions obtained by all the evaluated methods are very similar. As previously mentioned, this is due to the fact that in these cases the CSPs have very few solutions and consequently the distances of *all* solutions from the bounds are 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. For the same reason, the likelihood that a variable has a feasible repairable value that is near-by is very low. It can even be the case that none of the solutions has an assignment with feasible neighbours located at distance k. In this case, all the solutions are equally robust and stable for this k value.

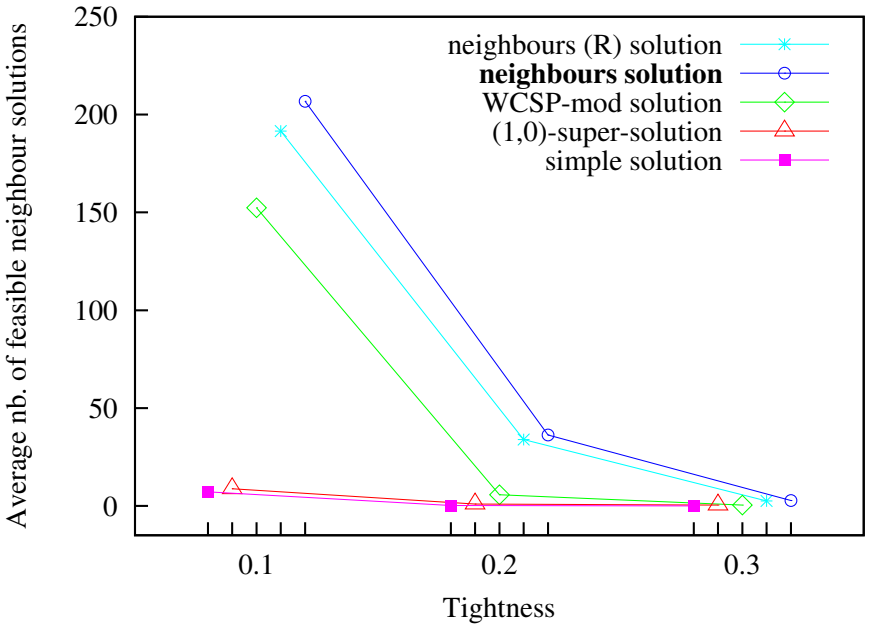
9.7 Scheduling Benchmarks Evaluation

In this section, we evaluate scheduling benchmarks from the literature with the two approaches that deal with general CSPs with ordered domains: the enumeration-based approach by coverings and the search algorithm. In addition, a wide range of k values is analyzed in order to evaluate the robustness/stability performance according to such parameter. Part of the following evaluation can be found in (Climent et al., 2014). The scheduling benchmarks analyzed are part of five sets of 10 job-shop CSP instances, studied in (Sadeh and Fox, 1996). Each instance is composed of 10 jobs of five tasks each and there are five resources. Each job has a random linear sequence of resources to visit, with the exception of the *bottleneck* resources, which are visited after a fixed number of operations (in order to further increase resource contention). The equivalent CSP models that can be found in Christophe Lecoutre's web page are composed of 50 variables of domain sizes not greater than 150 elements and 265 constraints.

Since in these experiments we are dealing with scheduling problems, we included the scheduling extensions explained in Chapter 8 to the approaches evaluated. Thus, for the search algorithm approach, the set of operators of Algorithm 7.3 were fixed



(a) Stability analysis



(b) Robustness analysis for $nbVarMod = 2$

Figure 9.9: Robustness-stability-tightness analysis ($< 2, 25, 30, 200,$ *tight.* $>$).

to $\oplus = \{\{>, +\}\}$ and the value selection is done with heuristic 1, in which values are selected in lexicographical order. The enumeration-based approach by coverings was also adapted by using the concept of greater covering (see Definition 8.6.1). As mentioned, it was incorporated to the super-solutions solver the scheduling adaptation explained in Section 8.7 that consists in considering only values greater or equal to k as repairable values. For a proper comparison of all these approaches, we use the same k range interval, as well as the same maximum cutoff time: $100s$.

As mentioned in Section 9.1, the enumeration-based approach by coverings uses a different solver than the other three approaches evaluated in this section. For this reason, it is not possible to make a completely fear time computation comparison. For the solver that we implemented, we used techniques from the literature that reduce the solving computational time (i.e., restarting technique and bounds-consistency). This effect is more marked in complex problems, such as scheduling problems. However, it is unknown if such techniques are implemented in the ToulBar2 WCSP solver. This WCSP solver is not able to solve all the instances of these benchmarks in the cutoff time fixed. Hence, the enumeration-based approach was not included in all the experiments presented in this thesis. However, it was included in a table that shows only the results for the instances that the WCSP solver is able to solve in the fixed time cutoff.

For measuring the robustness of the obtained schedules, we used the robustness measures introduced in Section 8.3. A first robustness measurement assessment is made by measuring the total slack whose duration does not exceed k, which is denoted as $tS(k)$. In addition, a more accurate measure is also used, $R^s_{slack}(k)$ (see Equation 8.1), which measures the average total slack, minus the standard deviation multiplied by the α parameter. The α parameter was fixed to 0.25, which is inside the interval that the authors consider appropriate for this parameter. Another robustness measure used is based on the resistance of a schedule when faced with perturbations, and is denoted as $R^s_{F,Z}$ (see Equation 8.2), where Z is the set of incidents that consist in delays of durations up to max_d over the tasks. We used 2 different values for max_d: 1 and k. In each case, we independently simulated 500 delays up to max_d units with equal probability over the entire schedule and checked if the schedule remained valid. For the stability measurement, again, (1, 0, 1)-repairability is used (see Definition 3.3.3), which is equivalent to the measurement of the number of buffers of the schedule, denoted as nbB. Note that the desired objective is that in cases where repairs are necessary, the start time of a task is delayed in the shortest time possible.

The following figures and tables show the evaluation for two of the Sadeh problem sets. We show results for the *e0ddr1* and *e0ddr2* benchmarks in order to compare robustness and stability of schedules obtained with different numbers of bottlenecks in the problem (other parameters are fixed). Sadeh stated that the *e0ddr1* benchmark contained just one bottleneck and *e0ddr2* benchmark contained two bottlenecks. Tables 9.5 and 9.6 show the means for the robustness and stability measures for scheduling problems. In addition, other measurements are showed, including the number of

schedules obtained nbS, total number of restarts done by the search algorithm nbR, the total number of nodes explored nbN and the total number of failures nbF. Figure 9.10 shows the stability and robustness measurements (vertical axis): the mean number of buffers and mean $R^s_{slack}(k)$ for the *e0ddr1*. The horizontal axis of the figures represents the value of the ratio of parameter k.

Table 9.5: Evaluation of ‘e0ddr1’ benchmark.

k	Approach	nbS	nbR	nbN	nbF	nbB	$tS(k)$	$R^s_{slack}(k)$	$R^s_{F,Z}(1)$	$R^s_{F,Z}(k)$
1	simple	1	3.1	208	85	16.1	16.1	0.208	0.328	0.338
	super	9.1	3.1	7770.4	3043.7	21.4	21.4	0.308	0.434	0.43
	neigh	12.7	3.1	10171.1	2465.9	27.8	27.8	0.436	0.562	0.555
	neigh(R)	15.5	28.8	2820.4	628.1	**31.3**	**31.3**	**0.509**	**0.628**	**0.618**
3	simple	1	3.1	208	85	16.1	44.3	0.555	0.328	0.311
	super	7.3	3.1	8138.2	2619.2	18.9	52.4	0.702	0.384	0.36
	neigh	15.9	3.1	5880.5	2485.1	20.9	59.4	0.832	0.424	0.409
	neigh(R)	15.5	27.7	2406.5	670.5	**22.3**	**62.1**	**0.886**	**0.448**	**0.413**
5	simple	1	3.1	208	85	16.1	67.8	0.832	0.328	0.288
	super	7.1	3.1	8373.2	2654.2	19.1	82.9	1.101	0.388	0.343
	neigh	19	3.1	3947.7	1674.7	19.5	86.3	1.159	0.396	**0.364**
	neigh(R)	12.9	23.8	2082.3	517.8	**20.2**	**87.3**	**1.182**	**0.406**	0.35
7	simple	1	3.1	208	85	16.1	88.1	1.057	0.328	0.271
	super	6.5	3.1	8319.1	3219.5	18.1	101.8	1.298	0.368	0.303
	neigh	19.9	3.1	3205.9	1032.6	**18.9**	107.8	1.4	**0.384**	**0.331**
	neigh(R)	11.5	21.2	1871.8	489.5	18.7	**108.6**	**1.413**	0.376	0.314
9	simple	1	3.1	208	85	16.1	105.7	1.242	0.328	0.257
	super	5.7	3.1	8715.7	2620.7	17.6	117.6	1.452	0.358	0.277
	neigh	20.8	3.1	2793.6	974.2	**18.6**	**126.5**	**1.602**	**0.378**	**0.303**
	neigh(R)	11.4	19.7	1711.4	462.8	18.2	126	1.588	0.368	0.293
11	simple	1	3.1	208	85	16.1	120.5	1.389	0.328	0.244
	super	6	3.1	8019.9	1775.8	18.2	133.6	1.629	0.37	0.256
	neigh	19	3.1	2518.5	844.4	18.1	**140.2**	**1.72**	0.368	**0.28**
	neigh(R)	7.9	16.9	1593.9	435.5	**18.4**	138.8	1.693	**0.374**	0.277

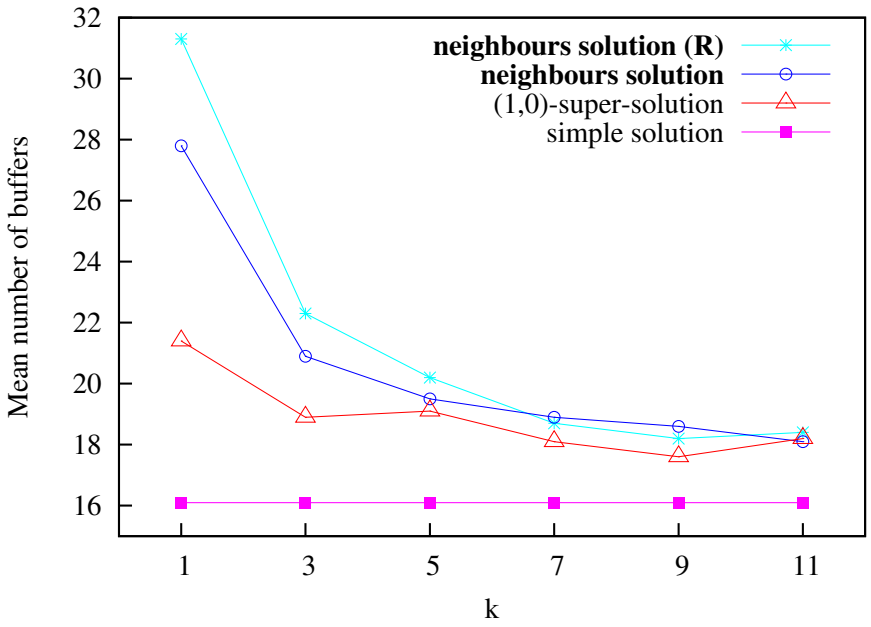
Table 9.6: Evaluation of ‘e0ddr2’ benchmark.

k	Approach	nbS	nbR	nbN	nbF	nbB	$tS(k)$	$R^s_{slack}(k)$	$R^s_{F,Z}(1)$	$R^s_{F,Z}(k)$
1	simple	0.9	4	227	98.89	14.11	14.11	0.17	0.28	0.28
	super	6.5	3.9	8024.33	1975.67	19.89	19.89	0.28	0.4	0.41
	neigh	10.1	4	11264.22	1865.22	24.33	24.33	0.37	0.49	0.49
	neigh(R)	12.4	25.4	2555.67	709.33	**27.44**	**27.44**	**0.43**	**0.55**	**0.55**
3	simple	0.9	4	227	98.89	14.11	37.56	0.44	0.28	0.25
	super	4.9	3.9	8602.67	1198.67	17.33	47.22	0.61	0.35	0.32
	neigh	16.4	3.9	7141.78	1711.33	**20.22**	**56**	**0.77**	**0.4**	**0.37**
	neigh(R)	11.9	22.2	2282.67	503.89	20.11	55.22	0.76	**0.4**	**0.37**
5	simple	0.9	4	227	98.89	14.11	55.11	0.63	0.28	0.22
	super	4.4	3.9	9102.78	743.67	17.11	70.89	0.89	0.34	0.29
	neigh	17.3	3.9	5755.22	1657.33	**18.11**	**76.67**	**0.99**	**0.36**	**0.31**
	neigh(R)	8.6	18.9	2036.11	452.78	17.89	73.89	0.95	**0.36**	0.3
7	simple	0.9	4	227	98.89	14.11	68.22	0.75	0.28	0.2
	super	3.8	3.9	9721.67	914.22	15.78	82.22	0.97	0.32	0.25
	neigh	15.2	3.9	4903	1272.56	16.89	**88.78**	**1.09**	0.34	**0.26**
	neigh(R)	7.7	17.5	1827.44	428.78	**17.22**	88.67	**1.09**	**0.35**	**0.26**
9	simple	0.9	4	227	98.89	14.11	78.67	0.84	0.28	0.18
	super	3.1	3.9	9971	959.56	15.56	92.89	1.06	0.31	0.21
	neigh	15.7	3.9	4344.44	1161.78	**16.78**	**101.11**	**1.2**	**0.34**	**0.24**
	neigh(R)	6.4	16.2	1657.56	449.22	**16.78**	100.44	1.19	**0.34**	0.23
11	simple	0.9	4	227	98.89	14.11	87.44	0.91	0.28	0.16
	super	2.3	3.9	10698.22	1090.44	15.22	98.89	1.08	0.3	0.19
	neigh	14.7	3.9	4251.78	1223.56	**16.22**	**109.67**	**1.25**	**0.32**	**0.21**
	neigh(R)	5.6	14.4	1588.89	390	16.11	107.67	1.22	**0.32**	0.2

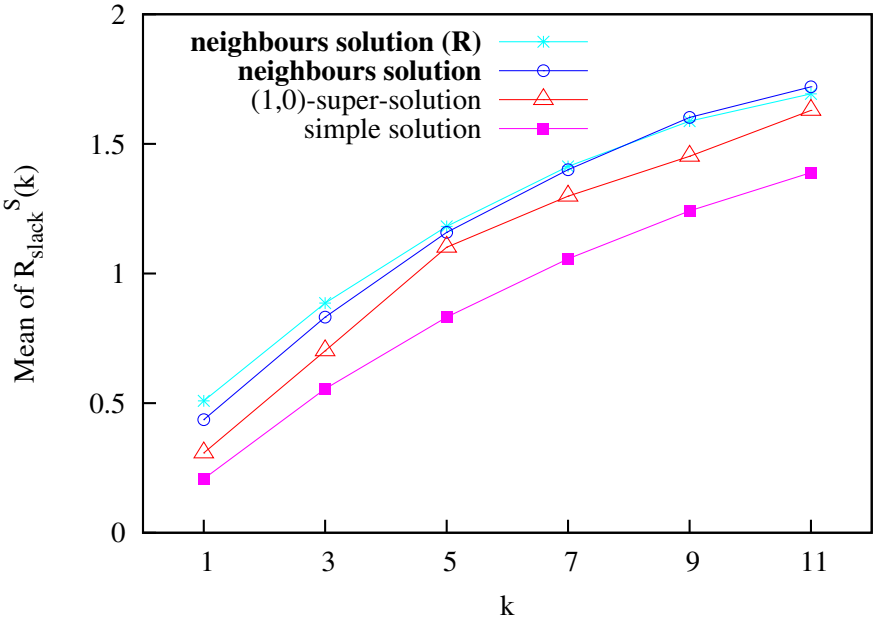
As expected, schedules obtained by all of the approaches for the e0ddr1 benchmark are more robust and stable than those for the e0ddr2 benchmark (see Tables 9.5 and 9.6) From the robustness analysis, we see that our search algorithm for $k = 11$ (for both restarting options) increased the robustness measure $R^s_{F,Z}(k)$ by more than 0.5 units for problems with only one bottleneck. Therefore, as expected, the fewer bottlenecks a scheduling problem has, the more robust the schedule obtained by our algorithm. Detailed results for all robustness measures are found under columns $tS(k)$, $R^s_{slack}(k)$, $R^s_{F,Z}(1)$ and $R^s_{F,Z}(k)$ in the tables. For instance, for the largest k value analyzed ($k = 11$), the total sum of all the buffer times of duration up to k of the schedule obtained by Algorithm 7.3 for restarting-completion is 140.2 time units for the *e0ddr1* benchmark and 109.67 time units for the *e0ddr2* benchmark (more than 30 time units difference). Regarding the stability analysis, our algorithm for $k = 1$ restarting-scratch (differentiated with "R") found schedules with four mean number of buffers (nbB) more for the problems with one bottleneck than for the problems with two bottlenecks for the best case. Therefore, as expected, the fewer bottlenecks a scheduling problem has, the more stable the schedule obtained by our algorithm.

In both tables and both figures, we can see that Algorithm 7.3 with either restarting option outperformed both the ordinary CSP solver and the super-solutions solver. Furthermore, the analysis of the k parameter shows that when these parameters have the lowest values, the number of buffers of the schedules found by our algorithm are markedly greater than these two techniques (see Figures 9.10(a) and 9.11(a)). In contrast, the improvement in robustness for our algorithm with respect to the ordinary solver is a little more marked for greater k values. For the *e0ddr1* benchmark the robustness improvement of our schedules with respect to the (1, 0)-repairability is more uniform over all the values of the k parameter (see Figure 9.10(b)). This fact is due to the *e0ddr1* benchmark is less constrained (it only has 1 bottleneck) and therefore it is more likely that there exist buffers of long durations. Recall that (1, 0)-repairability technique for scheduling problems only considers as repairable values those that are k units greater than the assigned ones. For this reason, for high k values the likelihood of finding those values is lower for more constrained instances.

Regarding the other robustness measures that are not plotted in the figure but are shown in Tables 9.5 and 9.6, we see that there is a correlation between the $R^s_{F,Z}(1)$ measure and the number of buffers. This relation is expected, since the random incidents generated for measuring $R^s_{F,Z}(1)$ were delays of one unit time. Therefore, the more buffers there are (whatever is their duration) the greater the likelihood that a schedule can absorb delays of one time unit. In addition, the $tS(k)$, $R^s_{slack}(k)$ and $R^s_{F,Z}(k)$ measures are correlated. Recall that $tS(k)$ is the total slack whose duration does not exceed k and $R^s_{slack}(k)$ is its average minus the standard deviation multiplied by an α parameter. Therefore, unless the distribution of the slack is very poor, the two values must be proportional. Note that the lower the α parameter for $R^s_{slack}(k)$, the greater the proportionality with respect to the other two robustness measures. The $R^s_{F,Z}(k)$ measure is calculated by generating random delays up to duration k over the

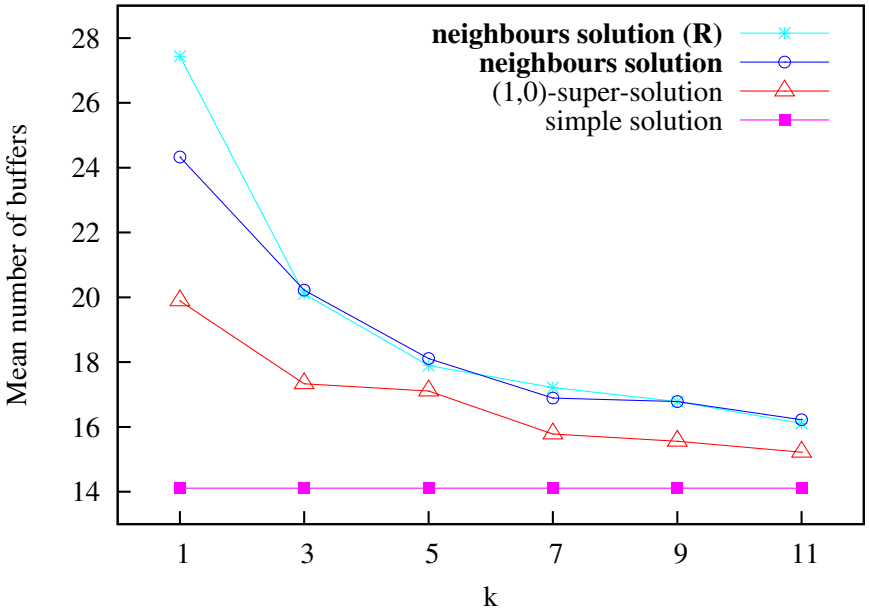


(a) Stability analysis

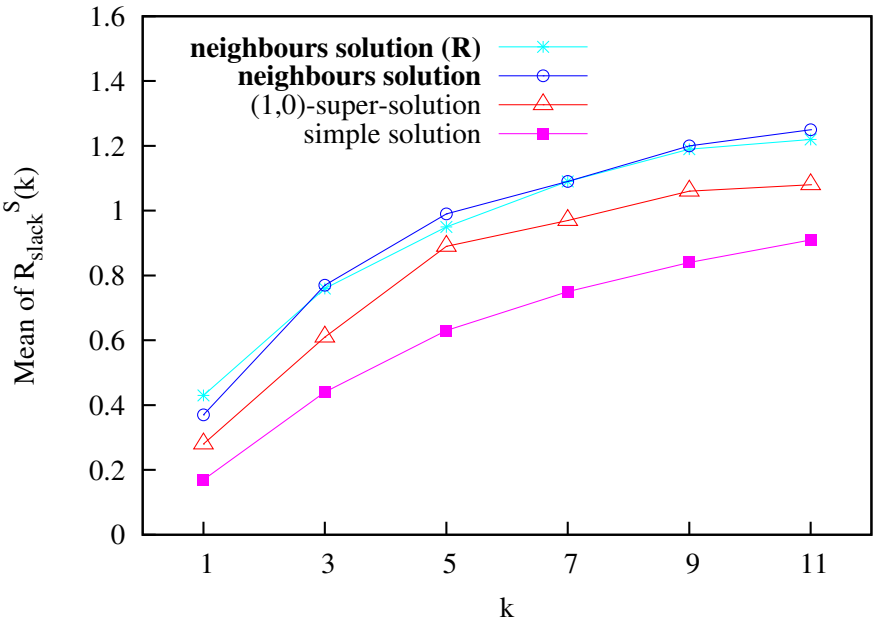


(b) Robustness analysis

Figure 9.10: Robustness-stability-k analysis for the e0ddr1 benchmark.



(a) Stability analysis



(b) Robustness analysis

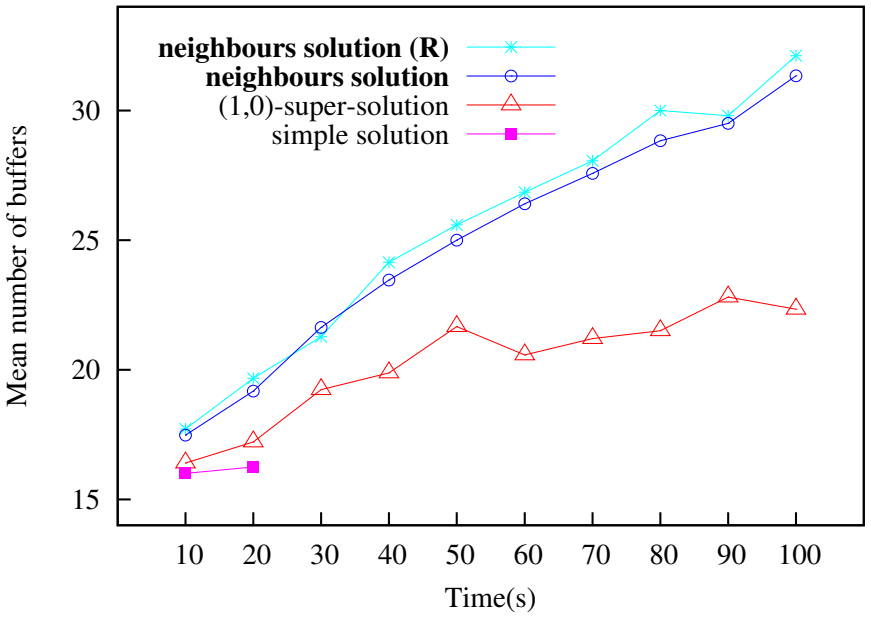
Figure 9.11: Robustness-stability-k analysis for the e0ddr2 benchmark.

schedule. For this reason, this robustness measure is strongly related with the two aforementioned. A example of the relation of all the aforementioned measurement units can be observed in Table 9.5 for $k = 11$, where the schedules obtained with restarting-scratch option (differentiated with "R") have greater numbers of buffers and $R^s_{F,Z}(1)$ values, and schedules obtained with restarting-completion option have greater $tS(k)$, $R^s_{slack}(k)$ and $R^s_{F,Z}(k)$ values. This means that the latter has a greater total slack whose duration does not exceed k, but its distribution is more limited.

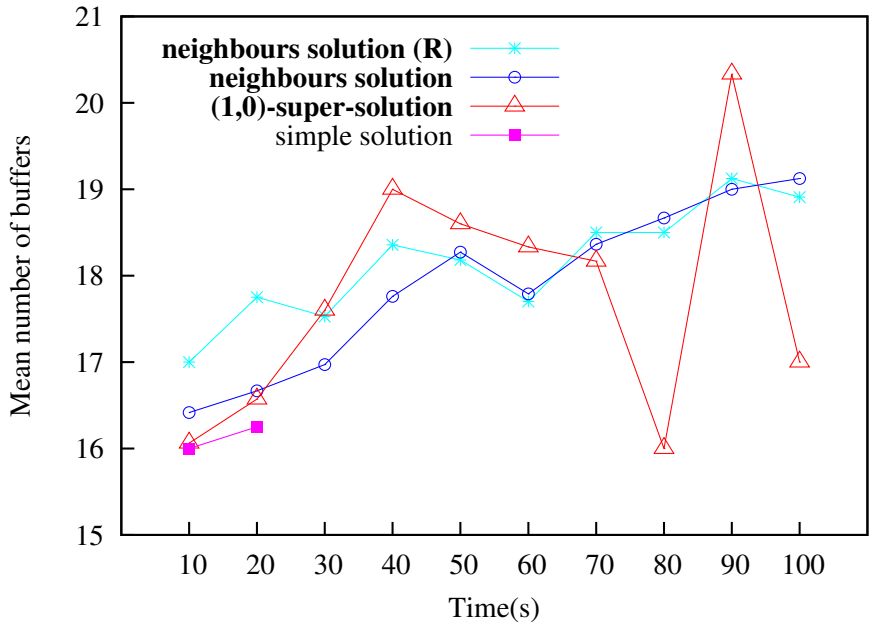
In Tables 9.5 and 9.6 we also observe measurements that are not correlated with robustness or stability, but important information can still be extracted from them. For $k > 1$, the restarting-completion for our algorithm finds the greater mean number of solutions (nbS). Only for $k = 1$ does the restarting-scratch (differentiated with "R") find more solutions. The greater k is, the easier it is to find new solutions whose objective function is better than the maximum one (if the instance is not highly restricted). Hence, the mean number of solutions found is greater for high k values. For both restarting options, the mean number of solutions is considerably higher than for the technique that maximizes the (1, 0)-repairability. This effect is stronger for greater values of k because the condition of a repairable value for the latter technique becomes more restrictive. Moreover, this technique considers all feasible values in the domains as repairable values; as a result, feasibility checking is slower than for techniques that assume only k neighbours (as our technique does). As expected, the mean number of restarts (nbR) is much greater for the restarting-scratch option because the other techniques only restart until finding the first solution. As a consequence, their mean number of nodes explored (nbN) and mean number of failures (nbF) is lower.

The schedules obtained by Algorithm 7.3 for the lowest k value had the highest number of buffers. However, the robustness measures are greater for the greater k values. Depending on the dynamic nature of the problem, it would be desirable to prioritize between a higher number of buffers of short duration and a lower number of buffers of long duration (if the two features cannot both be maximized). Thus, if there exists the knowledge that the possible future delays will have a duration of at least d time units, it does not make sense to compute k values lower than d because the obtained time buffers could not absorb the delay. Nevertheless, if it is known that possible future delays cannot have a duration greater than d time units, then it does not make sense to compute k values greater than d because this may decrease the number of buffers. Hence, the more information about possible future changes we have, the better the robustness results we can obtain. However, even if this information is unknown, we can obtain a schedule with certain level of both robustness and stability by setting k to an intermediate value in Algorithm 7.3.

The above evaluation consists in analyzing the best results obtained for each technique for the fixed cutoff time. However, we also wanted to analyze the change in the degree of robustness and stability of the schedules found over the time. For this evaluation, we used the e0ddr1 benchmark and determined the mean for 50 instances for each interval of time with a discretization of $10s$. Figures 9.12(a) and 9.12(b)



(a) $k = 1$



(b) $k = 7$

Figure 9.12: Number of buffers over time intervals for the e0ddr1 benchmark.

show the mean number of buffers found by each approach for k values equal to one and seven. Other measures are not shown since similar trends were found in these cases. We would like to note that after $20s$ the simple solution technique does not find better schedules because it only searches for one schedule for each instance (which is done in less or equal to $20s$). The most remarkable aspect is that for $k = 1$ Algorithm 7.3 for both restarting options obtains a greater number of buffer times than the super-solutions solver for $k = 1$ for all time intervals (see Figure 9.12(a)).

Figure 9.12(b), which represents $k = 7$, shows more unstable results. Since it is difficult to find buffers with up to seven time units, it may happen than our algorithm sacrifices some shorter buffers in order to find one buffer of seven time units. Thus, even if the overall tendency is for the measure to increase over the time, it is not entirely uniform. However, the upward shape of the trend for the super-solutions solver is due to the fact that it considers values as repairable if there is any possible alternative for the start time of a task that follows a task sharing the same resource, which is not equivalent to have a slack associated with this task in the schedule. For this reason, schedules that are considered more stable than other ones by this technique may contain a lower number of buffers. This feature is more marked for greater values of k, since the repairable values have to be at least k unit times greater than the assigned values, and therefore it is more unlikely to find repairable values that are close to the assigned ones.

On the basis of this evaluation, we can conclude that the difference in performance between the two restarting options (restarting-completion and restarting-scratch) is not very significant. Sometimes, the time needed to restart from scratch after each solution makes this option less effective than restarting-completion. In other cases, the restarting-completion option loses time in branches in which there are no better solutions, while restarting-scratch explores other branches. For instance, for the random experiments analyzed in Section 9.6, we concluded that restarting-completion provided slightly better results generally (see Table 9.4 and Figure 9.9(b)), while for the scheduling problems, restarting-scratch obtained schedules that were a bit more robust and stable for lower k values (see $k \in [1, 5]$ in Figure 9.10). For greater k values, both restarting options gave similar results.

Subsequently, we present the experiments obtained for the enumeration-based approach by coverings for the *e0ddr2* benchmark. As previously mentioned, we did not incorporate the results in the other tables presented in this section because the WCSP solver (ToulBar2) can not solve some of the instances. We recall that for developing our solver, the simple solver and the super-solution solver, we included several algorithms that improve the solver performance for scheduling problems, such as restarting option, bounds consistency, etc. We would like to point out that depending on the value of the k parameter, the modeling time required can produce that for some instance the solving time is not enough. For instance, for k parameters that are greater than 5, the modeling time required exceed the $100s$, producing a *time out*. In Table 9.7 we incorporated the averages only for the instances that the enumeration-based ap-

proach could solve (the same procedure was done for the other approaches analyzed). The number of instances solved are as follow: five instances for $k = 1$, four instances for $k = 3$ and six instances for $k = 5$. For greater k parameters we computed the results of all the instances of the benchmark for the rest of the approaches analyzed. We do not include the two restarting options for our search algorithm since, as mentioned, both obtain similar performances. In addition, we incorporated a new column in Table 9.7 that indicates the average of the modeling time (coverings computation and WCSP modeling) required by the enumeration-based approach by coverings. The rest of the time until reaching $100s$ is the time that the WCSP ToulBar2 has for finding the solutions. Even if the enumeration-based approach by coverings using the ToulBar2 as a solver could not solve all the instances, it can be observed in the table that for the solved instances outperformed all the other approaches analyzed (search algorithm, super-solutions solver and ordinary solver). This improvement is more stressed for the lowest k value, that is to say for $k = 1$. For which the enumeration-based approach achieves almost 15 buffers more than the search algorithm and more than 25 than super solutions. However, the disadvantage is that the modeling time increases strongly when the k parameter increases and at some point is ($k \geq 7$) not able to provide a solution.

To conclude, we can state that for these experiments the super-solution approach finds solutions with lower robustness and (1, 0, 1)-repairability than the approaches that we presented in this dissertation for general CSPs with ordered domains. One of the reasons of this behaviour, is that this technique considers any value greater or equal to k than the value selected as repairable value. Thus, repairable values located far from the value assigned are considered as repairable. Another reason is that this technique only assumes that delays are of duration k. Thus, only values greater than this value are considered as repairable values. However, we consider up to k neighbours and therefore, slacks of duration lower than k are also valued by our objective function in contrast to the (1, 0)-repairability objective function.

9.8 Summary and Limitations

In this chapter we performed experiments to the approaches presented in this dissertation in order to evaluate their significance and to determine their behaviour and performance for different CSP topologies. Furthermore, we also analyzed benchmarks from the literature. Among them, there are scheduling problems, which usually are complex problems that come from dynamic and uncertain environments.

Firstly, we evaluated the probabilistic enumeration-based approach, which considers extra dynamism data associated with the problems. As mentioned, this information gathers the probability and magnitude of change of the bounds of the CSPs. By means of the generation of new restricted constraints based on such information, we concluded that the solutions obtained by the probabilistic approach are able to

Table 9.7: Evaluation of 'e0ddr2' benchmark with the coverings technique.

k	Approach	$modT(s)$	nbB	$tS(k)$	$R^s_{slack}(k)$	$R^s_{F,Z}(1)$	$R^s_{F,Z}(k)$
1	simple	-	14	14	0.168	0.28	0.286
	super	-	19	19	0.262	0.38	0.39
	neigh	-	29.8	29.8	0.478	0.596	0.596
	WCSP-m	9.21	**45.8**	**45.8**	**0.848**	**0.916**	**0.908**
3	simple	-	14.25	37.25	0.44	0.285	0.2575
	super	-	16.5	44.25	0.5625	0.33	0.3
	neigh	-	21.25	57	0.7925	0.425	0.39
	WCSP-m	38.03	**31**	**86.25**	**1.3725**	**0.62**	**0.56**
5	simple	-	14.17	52.17	0.58	0.28	0.205
	super	-	23.83	53.79	0.77	0.48	0.44
	neigh	-	18.17	73.5	0.94	0.36	0.295
	WCSP-m	82.91	**23.83**	**97**	**1.38**	**0.48**	**0.37**
7	simple	-	14.11	68.22	0.75	0.28	0.19
	super	-	15.78	82.22	0.97	0.32	0.23
	neigh	-	**17.22**	**88.67**	**1.09**	**0.34**	**0.25**
	WCSP-m	-	-	-	-	-	-
9	simple	-	14.11	78.67	0.84	0.28	0.17
	super	-	15.56	92.89	1.06	0.31	0.2
	neigh	-	**16.78**	**100.44**	**1.19**	**0.34**	**0.22**
	WCSP-m	-	-	-	-	-	-
11	simple	-	14.11	87.44	0.91	0.28	0.15
	super	-	15.22	98.89	1.08	0.3	0.17
	neigh	-	**16.11**	**107.67**	**1.22**	**0.32**	**0.19**
	WCSP-m	-	-	-	-	-	-

satisfy a high number of these new constraints, specially for high values of granularity (number of new generated constraints for each original bound), until arriving to a saturation point. In addition, we explained how the magnitude of change has a relationship in the robustness of the solutions obtained. Nevertheless, the limitation of this informed approach is precisely the requirement of such dynamism data. Regarding the enumeration-based approach for finite linear CSPs, the solutions obtained for such technique satisfied a high percentage or restricted linear constraints, specially for low constrained problem topologies. However, the limitation of this technique is that it is only addressed to finite linear CSPs.

At this stage, we evaluated the two general approaches presented in this dissertation that are much more versatile because they do not have the above mentioned drawbacks. The first one is also an enumeration-based approach and it is addressed to a type of dynamism restriction over the search space whilst the latter approach proposed, which is a search algorithm, it is addressed to a type of dynamism restriction over the solution space. After several experiments, we can conclude that in the dynamic and ordered environment analyzed in this dissertation, our general approaches dramatically outperform both the ordinary CSP solver and the super-solutions solver (for scheduling problems the improvement is less abrupt for the latter approach) 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 of the problem are not so great that there are only a few valid solutions. Regarding scheduling problems, the improvement over the approach that maximizes the (1, 0)-repairability for high k values is not as highly marked. Even so, our approaches found schedules with a higher number of buffers and with a greater total slack, generally.

Regarding the comparison of our two general approaches, which check the feasibility of the neighbourhood in different spaces (search space and solution space, respectively) we observed that for scheduling problems the enumeration-based approach by coverings highly outperforms the search algorithm for low k values. However, due to the modeling time increases highly when the k parameter increases, for high k values, the technique reached the time cutoff without providing any solution. The highly modeling time required for instances with a high number of tuples for k high values represents a limitation for this technique. In addition, we performed experiments simulating dynamism over the solution space, in order to analyze the other limitation of the enumeration-based technique, which happens when the tightness and the topology of the CSPs produce situations in which some neighbour tuples are feasible for a bound of the CSP but are not for another (when the tightness is not very low). In such cases, the search algorithm obtained better results than the enumeration-based approach.

For the analysis of the search algorithm, we performed experiments with both restarting options: restarting-completion and restarting-scratch. We concluded that the difference in performance between they is not very significant, even if in some

types of problems there exist little differences in their behaviour. For instance, restarting-completion provided slightly better results generally in the random performed experiments, while for the scheduling problems, restarting-scratch obtained schedules that were a bit more robust and stable for lower k values. Furthermore, we show that the search algorithm has a characteristic that represents a limitation in some complex instances such as scheduling problems: the algorithm is not well informed at the beginning of the search (because the feasibility checking is done while the CSP is being solved), therefore, the time spent in some non-optimal branches due to this fact could produce the reach of the time cutoff.

To conclude, in this chapter we empirically demonstrated the high performance of the approaches presented in this dissertation for the ordered environments and correspondingly their assumed dynamism. We also concluded after several experiments with different CSP topologies that the best robustness results were obtained for low constrained instances. We also applied the general approaches to complex scheduling problems, obtaining schedules that are both robust and stable. Furthermore, we analyzed the situations that are favouring/disfavouring each technique.

Chapter 10

Conclusions

In this dissertation, our main challenge has been the search of robust solutions for CSPs in difficult situations in which there is a lack of extra detailed data about the future possible changes that the problems may undergo. This challenge has represented a difficulty itself due to the relationship of the robustness with regard to the information that is known about the dynamism of the environment. In this thesis, we focus on CSPs that model problems with ordered domains and therefore, we analyzed and formalized a dynamism framework related to this field.

Furthermore, we presented theoretical approaches for dealing with such dynamism. Based on them, we designed and developed four technical approaches. First, we presented a probabilistic enumeration-based technique that considers extra detailed information of the dynamism. The rest of the approaches presented in this thesis do not consider extra dynamism information. One of them, which it is also an enumeration-based approach, it is only addressed to finite linear CSPs, with the advantage that the computational time is low. However, the other two approaches complete the drawback of the specifiability of the latter approach. One of them is also an enumeration-based approach that address a type of dynamism restriction over the search space. Whilst the latter approach is a search algorithm addressed to a type of dynamism restriction over the solution space.

Finally, after evaluating the techniques introduced in this dissertation, we state several specific conclusions for each approach introduced. Furthermore, we showed the competitive performing of our approaches for the extended concepts of robustness and stability according to the assumptions made for problems with ordered domains. Therefore, our approaches are effective for the search of robust and stable solutions for CSPs with ordered domains. In this chapter we summarize the evidences that support this statement. Thus, the contributions made in this thesis are explained in detail, subsequently the limitations of the work presented in this dissertation are mentioned and finally we describe some future lines of work and final conclusions. In addition, the list of publications associated with this thesis is shown.

10.1 Contributions

Typically, previous proactive approaches that search for robust solutions consider some data (usually detailed) about future possible changes. However, if such information is uncertain or unknown, the approach itself can not be applied. This drawback has motivated the exploration of this new branch in dynamic and uncertain environments: robustness search without extra data about the future dynamism. We found that it is possible to extract little dynamism assumptions in problems for which there exists an order relationship over their elements. The explanation and formalization of such assumptions and its framework is the first contribution of this thesis. Furthermore, we designed four different algorithms for dealing with this type of dynamism in several scenarios. Finally, we extended the dynamism framework to scheduling problems and adapted the general algorithms to this type of problem. All these contributions mentioned, are summarized as follows.

10.1.1 New Dynamism Framework for Ordered Domains

The first contribution of this dissertation is the analysis and formalization of a type of dynamism that is usually associated with this type of problems: dynamism takes the form of expansions and restrictions over the bounds of the solution space. This was motivated in Chapter 3 with different types of real life problems. In regard to this dynamism, we only consider the restrictive changes since relaxations can not invalidate the original solution found, which is exactly what we try to avoid by searching for solution robustness. We would like to point out that, as far as we know, such dynamism framework is new in the literature. In addition it represents a significant contribution since many real life problems have order relationships and therefore they can undergo this kind of dynamism.

Another important contribution of this dissertation is the extension of the robustness and stability concepts to this analyzed dynamism framework. Thus, according to the little dynamism assumptions made, a solution is robust if it is located far from the bounds of the solution space. This robustness concept has not been previously analyzed in the literature. Moreover, we extend the stability by incorporating the distance concept, since in this context, the order over the elements is significant. Therefore, in cases where a value of the solution is lost, it is important to replace it by a nearby value in order to have a solution as similar as possible to the original one. This closeness feature is not handled by the existent proactive approach that searches for stable solutions called super-solutions approach (Hebrard, 2006).

Furthermore, we presented two theoretical approaches that describe the main ideas for dealing with such possible future restrictive dynamism over the bounds of the CSP. One of them is addressed only to finite linear CSPs (convex solution spaces) and the other theoretical approach is opened to both, convex and non-convex solution spaces that are discrete. We would like to recall that the second theoretical approach, which

introduces obvious advantages, since it is addressed to a large number of different types of CSPs, has been a challenge itself due to the difficulty of calculating distances to the bounds when the only information that the constraints provide is the set of valid/invalid tuples associated with them. For this purpose, we proposed the search of solutions with a great feasible neighbourhood, based in an 'onion-like' topology. Even if this model has been successful in other fields, the application of this model to CSPs is a novel idea. Furthermore, by fulfilling this criterion, we also increase the stability of the solution related to its close repairable values.

Regarding the dynamism associated with this environment analyzed, and considering that in the literature there do not exist DynCSPs benchmarks, we proposed and implemented several ways of simulating dynamism. We simulated random dependent/independent restrictive changes over the search/solution space. This simulation allows the solution robustness measurement in such ordered environments without extra dynamism data.

10.1.2 Proposed Techniques and Designed Algorithms

In this dissertation we presented four approaches for finding robust and stable solutions for CSPs that model problems with ordered domains that come from uncertain and dynamic environments. One of the approaches introduced considers extra detailed data about this type of dynamism. The other approach does not possess such extra information.

Moreover, three of the approaches introduced are enumeration-based approaches that assign certain penalizations to the tuples associated with each bound of a CSP, composing a WCSP. The other approach presented is a search algorithm that searches for a solution that maximizes an objective function. Following, the approaches introduced are listed by their order of presentation:

1. Probabilistic Enumeration-based Technique.
2. Enumeration-based Technique for Linear CSPs.
3. Enumeration-based Technique for CSPs by Coverings.
4. Search Algorithm for CSPs.

As mentioned, the first approach is applicable to an informed dynamism ordered framework. The main reason of the development of this informed approach is that it is more accurate in the robustness search when there is detailed data about the possible future restrictions over the bounds of the solution space. The second approach is restricted to linear bounds. However it includes the advantage of the easiness and quickness of the bounds distance computation. Therefore, its application is recommended in cases in which a quick answer is needed and always that the linear requirements

are satisfied. The third and fourth approaches are opened to any type of restriction, therefore they have an extent field of applicability. The main difference between them is that for the third approach, the robustness is modeled according to the feasibility of the tuples of each bound, whilst for the fourth approach, the feasibility checking is performed considering all the bounds. In addition, in Section 10.2.1 the repercussions of the differences of these approaches are further explained.

For each of the above approaches, we developed an algorithm that fulfills the purpose of its associated approach. For the first approach, we designed an algorithm that considers the data that describes the possible future restrictions over the bounds and it generates a WCSP that incorporates new modified constraints. For the second approach, which it is addressed to finite linear CSPs, we designed an algorithm that calculates the distances to each linear CSP bound. For the third approach, we designed an algorithm for calculating the coverings of an assignment in a certain space. This algorithm represents an important contribution since it is the first of this kind in the literature. Finally, for the last approach, we designed a search algorithm that searches for a solution that maximizes the number of closest contiguous solutions in each axis of each variable of the solution. This idea is also novel in the literature and it has been proved to be effective in the robustness and stability search. Furthermore, a CSP solver that includes this search algorithm was implemented for the correct evaluation of the approach. The design and development of the four approaches presented in this dissertation, as well as their corresponding algorithms and the solver. They represent a contribution to the related literature because they cover the unexplored field of robustness and stability search in the new dynamic framework with ordered domains presented in this thesis.

Furthermore, in this dissertation we adapted the enumeration-based approach by coverings and the search algorithm to scheduling problems. These problems are a highly well-known type of real life problems that usually come from uncertain and dynamic environments, and therefore they are very significant for the related literature. The extension of our approaches and algorithms to scheduling problems consists, basically, in restricting the selection of feasible neighbours only to greater values. This has allowed the achievement of schedules with slack, which it is able to absorb unexpected delays in the tasks.

10.2 Conclusions of our Approaches

In this section we recall and summarize statements made throughout this dissertation about the performance, and the advantages and disadvantages of the approaches presented in this dissertation.

10.2.1 Performance

We evaluated the approaches presented in order to demonstrate their performance. In addition, we analyzed their behaviour when faced with a wide range of CSP topologies and real life problems. We also analyzed the influence of the parameters associated with each approach. Moreover, a proactive approach that does not require dynamism information has been compared with our uninformed approaches.

For analyzing a wide range of CSP topologies, we generated random CSPs of two types: finite linear CSPs and non-convex CSPs. For the generation of the first type of CSPs, we implemented a random linear CSPs generator. In addition, we also used an existing non-convex random CSP generator (RBGenerator 2.0). As previously mentioned, in order to perform the experiments, we implemented a CSP solver that incorporates the search algorithm introduced in Chapter 7 (Algorithm 7.1). In addition, we used a WCSP solver (ToulBar2) for solving the WCSPs modeled by our enumeration-based approaches.

Furthermore, for the evaluation of the approaches presented in this thesis, we had the difficulty that there do not exist benchmarks of DynCSPs. For this reason, we performed a dynamism simulation by randomly restricting the bounds of the search/solution space with dependent/independent modifications with the purpose of estimating the robustness of the solutions obtained. For scheduling problems, we use existing robustness measures from the literature. As previously mentioned, the stability measurement was calculated with the (1, 0, 1)-repairability (see Definition 3.3.3).

After we conducted the experiments, we concluded that in the dynamic and ordered environment analyzed, our general approaches outperformed both ordinary CSP algorithms and algorithms that maximize the (1, 0)-repairability under many conditions where there are real differences in the robustness of solutions that might be obtained. This improvement was more marked in non-scheduling problems, for which the best improvement was obtained for low k values. One of the disadvantages of the super-solutions approach in this ordered type of environment is that this approach considers as repairable value any value in the domain. Thus, the feasible repairable values can be located far from the selected solution. Furthermore, it only searches for an alternative solution for each variable of the CSP, whilst our approaches search for the contiguous surrounding solutions at distance lower or equal to k (which ensures minimum distances to the bounds). Hence, the super solutions technique considers all the feasible values in the domains as repairable values; as a result, the feasibility checking is slower than for techniques that assume only k neighbours (as our approaches do).

Regarding the influence of the k parameter over the solutions obtained, lower k values tend to distribute the distance to the bounds more uniformly over all the variable assignments. However, their total distance to the bounds is not usually high. With higher k values we obtain higher total distances, even if their distribution is worse. Depending on the dynamic nature of the problem, it would be desirable to prioritize between these two criteria (if the two features cannot both be maximized).

Thus, if there is knowledge that the possible future changes are going to be of a low magnitude and highly frequent and distributed, it does not make sense to compute high k values. The opposite would occur with high magnitudes of change but less frequent: a high k value would be the recommended choice. This above mentioned information would make more accurate the robustness and stability of the solutions found, but it is not a requirement, since for any k value, our approaches are able to find certain level of robustness and stability.

10.2.2 Strengths and Limitations

In this section we explain the advantages and weaknesses of the approaches presented in this thesis. Some of them were briefly introduced in Section 10.1.2.

The most remarkable characteristic of the probabilistic enumeration-based technique is its accurateness due to the fact that it is an informed approach. However, this informed property shows also a drawback, since as its name indicates, it requires specific data, among them, the likelihoods of change. Moreover it also uses the information of the magnitude of the changes. In this way, this is a limitation due to the necessity of such dynamism data. Even so, if the available dynamism information is partial, the approach could be applied by modeling only this knowledge in the WCSP. For instance, if there is a lack of information about the dynamism of a bound of the CSP, it would be excluded of the new constraint generation and would be treated as a hard bound (the same would happen if one of the bounds appears not to be dynamic). Thus, the solution found would be robust for the other bounds with the dynamism data available, but not for the uninformed bound.

The main advantage of the enumeration-based technique for linear CSPs is how quick the bounds distance computation is and, consequently, the WCSP robustness modeling. Nevertheless, its specific characteristic of being applicable only to finite linear CSPs represents its main weakness.

Regarding the enumeration-based technique by coverings and the search algorithm presented, indisputably their main advantages are that they do not need extra dynamism data and that they are applicable to any type of constraint. In Table 3.1 we listed the main properties of both approaches. Moreover, we further explain in detail the differences and the positive/negative repercussions of both approaches.

- **Type of Solver:** Since the search algorithm technique is the main piece of a solver itself, the solver depends on this algorithm. This fact has the advantages that it is easy to incorporate other techniques, functions and/or heuristics that improve their performance according to the characteristics of the problems analyzed. The latter is not possible for the enumeration-based approach because it uses a generic solver. This provides versatility to the enumeration-based technique, since the modeling phase is independent of the solving phase, and therefore the user can select any type of WCSP solver.

- **Initial Information:** Since the enumeration-based technique by coverings models the robustness in the WCSP before the solving process, the solver is highly informed at the beginning of the search. However, the feasibility checking of the search technique is performed while it is searching the solution. This property only represents a limitation for the search algorithm in complex problems in where we fix a time cutoff. If there were no time restrictions, the algorithm would find the optimal solution.

- **Optimality criterion:** In one hand, the search algorithm checks the neighbourhood feasibility in the solution space and therefore it deals properly in situations where a neighbour assignment is feasible for a bound of the CSP but it is not for another, which is very useful in problems in which the constraints have a high number of interactions and that have high tightness values. On the contrary, the enumeration-based approach checks the feasibility for each individual bound. Hence, a tuple that is considered a feasible neighbour for a constraint, could be unfeasible for another constraint, and therefore it would be unfeasible in the solution space. The fact that the search algorithm checks the neighbourhood feasibility in the solution space, has the limitation that only neighbours on both directions of each axis are checked. The diagonals are not checked due to the vast combinatoriality of finding as many neighbours solutions (NP-hard).

- **Spatial Cost:** The strength of the search algorithm is that it does not generate another extra data model and therefore its spatial cost is very low. On the contrary, the enumeration-based approach generates an extra WCSP file. Its spatial cost is high for complex problems, since all the constraints must be extensionally represented.

After all the strengths and limitations mentioned above, subsequently we analyze the best situations for using each approach. We endorse the use of the probabilistic enumeration-based technique when there exist detailed dynamism information about the future possible restrictions that the bounds of the CSPs may undergo. The use of the enumeration-based technique for finite linear CSPs is only suitable for such type of problems when a quick answer is required.

The approaches that we highly recommend for their generality and versatility are the enumeration-based technique by coverings and the search algorithm. Among them, we distinguish between two cases according to the computation time limitations. We recommend the use of the search algorithm for low k values if there are not time restrictions. For high k values the enumeration-based approach could provide better results due to the diagonals neighbourhood computation (if the interaction between constraints is not high). When there are time limitations, we recommend the enumeration-based approach iff the interaction between constraints is very low and there are not tight spacial limitations. Otherwise, the search algorithm would be the best choice. It would be also the best option for problems with a very high number

of valid tuples associated to the constraints. These usability advices represent general patterns, so they should not be taken as undeniable statements. Hence, depending on the topology and characteristics of the dynamic problem, the performance of these approaches can vary.

10.3 Future Work

In this section we mention two further lines of work that could be carried out in the future as an extension of the work presented in this dissertation. The first one is related to the dynamism knowledge associated with the problem. Specifically, it consists in adapting the uninformed approaches presented in this dissertation for considering certain extra dynamism data. Thus, these techniques would be more accurate when this dynamism knowledge is available. The other future line of work consists in combining optimality criteria, where two of them are the robustness and stability of the solutions.

10.3.1 Considering Certain Dynamism Information

We presented three uninformed approaches in this thesis. The fact that they do not require extra dynamism data is one of their main advantages and it has represented one of the main challenges of this dissertation. However, the adaptation of these approaches for considering certain information about the changes could represent an interesting future line of work. For such variation, we could consider how the robustness is modeled/calculated by each type of approach. The enumeration-based approaches model the robustness for each bound (constraints and domains) of the CSP. The search algorithm computes the robustness according to the neighbourhood of each assignment of a variable that composes the solution. In this way, the enumeration-based approaches could consider dynamism data about the constraints and/or domains, whilst the search algorithm could consider dynamism data associated with the variables. In the rest of this section, we analyze the extension of these approaches for dealing with probabilistic dynamism extra data.

For the uninformed enumeration-based approaches presented in this thesis, the dynamism information could consist on probabilities of the restrictions that the bounds can undergo. The latter dynamism parameter, as well as a parameter that measures the magnitude of change, have already been considered by the informed approach presented in this dissertation. For the enumeration-based approach by coverings, this informed extension could consist on pondering the costs of the bounds of the modeled WCSP. For instance, the costs of all the tuples of a bound would be multiplied by its probability of undergoing a restrictive change. Furthermore, the k-covering computed for each bound would be fixed according to the magnitude of change of such bound. Thus, the higher the magnitude of change is, the higher the k-covering computed is.

Subsequently, we analyze a possible extension of the search algorithm for considering, as extra dynamism data, the likelihood that the assignment of a variable is invalidated by a restriction of a bound in the future. This dynamism information represents a different point of view than the previous dynamism information analyzed. An example of such dynamism can be found in scheduling problems. In this type of problems, some tasks might have higher likelihoods of undergoing delays than the others. Equation 10.1 is an extension of the objective function introduced in Chapter 7 (see Equation 7.2) that considers such probabilities. Thus, in the extended equation, each variable $x \in \mathcal{X}$ has a likelihood $p(x) \in [0, 1[$ that the assignment of x is invalidated by a restriction of a bound in the future. By multiplying the probability of each variable by its neighbourhood, we are prioritizing the variables with higher probabilities.

$$f(s,k,\oplus) = \{ \sum_{x \in \mathcal{X} \setminus \mathcal{X}_s} p(x)\ max\{|\mathcal{N}_k(x,v,s,\oplus)|, \forall v \in D_s(x)\} + \sum_{y \in \mathcal{X}_s} p(y)\ |\mathcal{N}_k(y,s(y),s,\oplus)|\} \quad (10.1)$$

10.3.2 Combining Optimality Criteria

In this dissertation, we developed approaches that deal with CSPs. A future line of work, would be the extension of these techniques for handling constraint satisfaction and optimization problems (CSOP). Thus, our models could be extended by including other kinds of optimization criteria (in addition to the robustness and stability). For instance, minimizing the time, maximizing the profit, etc. In a scheduling problem, the desirable multi-objective criteria could be not only to increase the stability and the robustness of the schedule but also to decrease its makespan. We consider that this step would increase the versatility of our technique, since we could potentially find robust and stable solutions while meeting several other optimality criteria.

A proposal for this extension for the enumeration-based approaches would be to combine the costs associated with the tuples of the modeled WCSPs generated with other penalizations according to the other criterion evaluated. For instance, if some combinations of tuples are worse for a certain objective function, an extra cost, representing a penalization, would be added to them. Therefore, for these tuples, the likelihood of being selected as a part of the best solution, would decrease.

As mentioned, the search algorithm is highly adaptable, and therefore, an extension for adding other optimality criterion is quite straight forward. It consists in adding the other criterion to the objective function to be maximized by the solutions

that the search algorithm finds. If a criterion is positive, it is added to the objective function. Otherwise, it is subtracted to the objective function.

For instance, in scheduling problems, a very well known criterion to minimize is the makespan. Equation 10.2 represents a possible representation of this multi-criterion objective function. The global objective function (F) is a combination of the robustness neighbourhood function f (see Equation 7.2) and the makespan (or its upper bound estimation) of the partial/complete assignment s. These two criteria are weighted by means of the $\alpha \in [0, 1]$ parameter. By means of this parameter, it is possible to balance the priority of these criteria.

$$F(s, k, \oplus) = \alpha\, f(s, k, \oplus) - (1 - \alpha)\, makespan(s) \tag{10.2}$$

10.4 Final Conclusions

In this dissertation we defend that it is possible, reasonable and justifiable the development of approaches for a dynamic, uncertain and ordered framework that fulfill the four objectives stated in the abstract of this thesis:

First *"Limit as much as possible the need for successive online problem solvings."*

Second *"Limit as much as possible changes in the produced solution."*

Third *"The production of solutions that are at the same time robust and flexible, that have every chance to resist changes and can be easily adapted when they did not resist, is obviously a desirable objective."*

Fourth *"Ideally, no additional knowledge over the data used to build the classical constraint network is required and no more expertise than for solving the problem without taking uncertainty into account."*

In this thesis we formalize a dynamism framework associated with problems for which the order is significant. We also extend the definitions of robustness and stability for such framework. Furthermore, in this context, we present theoretical and practical approaches. They allow to distinguish between the robustness of the solutions of CSPs with ordered domains that do not have additional detailed information about the dynamism. Finding solutions located far from the bounds is important when we face restrictive modifications over the bounds of the solution space. Moreover, certain level of stability in the obtained solutions is achieved. In cases where a value is lost, it is important to replace it by a nearby value in order to have a solution as similar as possible to the original one. Therefore, achieving both features in the solutions obtained is extremely useful and practical in many real life situations where problems can undergo restrictive changes, especially if there is the added difficulty that information about the possible future changes is limited or non-existent.

10.5 Associated Publications

In this section we list the publications associated with this dissertation. They are classified as follows: journals, international conferences and national conferences.

Journals

- L. Climent, Richard J. Wallace, M. A. Salido , F. Barber
  Robustness and Stability in Constraint Programming under Dynamism and Uncertainty
  Journal of Artificial Intelligence Research (JAIR), ISSN:1076-9757 (JCR-2012: 1.056) (in press).

- L. Climent, Richard J. Wallace, M. A. Salido , F. Barber.
  Finding Robust Solutions for Constraint Satisfaction Problems with Discrete and Ordered Domains by Coverings
  Artificial Intelligence Review (AIRE), ISSN: 0269-2821, pp:1-26, 2013 (DOI 10.1007/s10462-013-9420-0). (JCR-2012: 1.565).

- L. Climent, M. A. Salido , F. Barber
  Robustness in Dynamic Constraint Satisfaction Problems
  International Journal of Innovative Computing, Information and Control (IJICIC), ISSN 1349-4198, Volume 8, Number 4, pp:2513-2532, April 2012. (JCR-2010: 1.664).

International Conferences

- L. Climent, Richard J. Wallace, M. A. Salido , F. Barber
  Modeling Robustness in CSPs as Weighted CSPs
  The 10th International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming (CPAIOR), 2013. (CORE B).

- L. Climent, M. A. Salido, F. Barber
  Reformulating Dynamic Linear Constraint Satisfaction Problems as Weighted CSPs for Searching Robust Solutions
  Symposium on Abstraction, Reformulation and Approximation (SARA). pp.43-42, ISBN: 978-1-57735-543-4, 2011. (CORE B).

- L. Climent, Richard J. Wallace, M. A. Salido , F. Barber
A Constraint Programming Approach to Solve Scheduling Problems under Uncertainty ICAPS 2013, Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS), 2013. (CORE B).

- L. Climent, Richard J. Wallace, M. A. Salido , F. Barber
An Algorithm for Finding Robust and Stable Solutions for Constraint Satisfaction Problems with Discrete and Ordered Domains
The 24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), pp: 874-879. ISBN: 978-0-7695-4915-6, 2012. (CORE B).

- L. Climent, M. A. Salido, F. Barber
Finding Stable Solutions in Constraint Satisfaction Problems
The 15th International Conference on Principles and Practice of Constraint Programming (CP), Doctoral Consortium. pp.25-30, 2009.

- L. Climent, M. A. Salido, F. Barber
Robust Solutions in Changing Constraint Satisfaction Problems
The 23rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE). Part I, pp. 752-761, LNAI 6096, ISBN: 978-3-642-13021-2 ISSN: 0302-9743, 2010. (CORE B, CSC: top 5 in Industrial Engineering Conference Ranking).

- L. Climent, M. A. Salido, F. Barber, Richard J. Wallace
Solution Robustness in Constraint Satisfaction Problems with Discrete and Ordered Domains
The 23rd Irish Conference in Artificial Intelligence and Cognitive Science (AICS). pp. 74-83. ISBN: 978-3-8325-3240-6, 2012.

- L. Climent, M. A. Salido, F. Barber, L. Ingolotti
Robustness and capacity in scheduling: Application to railway timetabling
IBERAMIA 2008, Workshop on Planning, Scheduling and Constraint Satisfaction (PSCS'08), ISBN: 972886207-5, pp.87-98, 2008.

National Conferences

- M. A. Salido, L. Climent, F. Barber
A Tool for Finding Stable Solutions in Constraint Satisfaction Problems
CAEPIA 2009, Workshop on Planning, Scheduling and Constraint Satisfaction, 2009.

- L. Climent, M. A. Salido, F. Barber
  Stable Solutions in Constraint Satisfaction Problems
  The 13th Spanish Conference for Artificial Intelligence (CAEPIA), Doctoral Consortium. 2009.

- L. Climent, M. A. Salido, F. Barber
  Stability of Solutions in Constraint Satisfaction Problems
  The 12th International Conference of the Catalan Association for Artificial Intelligence (CCIA). pp.301-309, ISBN: 978-1-60750-061-2 ISSN: 0922-6389, 2009.

Appendix A

Weighted CSPs File Format Specification

In this appendix, it is described the file format Weighted CSP specification used in this work for modeling the WCSPs: the WCSP file format. First, a description of this format is introduced and subsequently a detailed example is described.

A.1 File Format Description

In this section, we explain the format used in this thesis for defining the WCSPs generated by the enumeration-based approaches proposed in this thesis. The format used is the WCSP file format, which can be found in Dechter's group web page [1]. The WCSP format is a simple format which should be easy to parse by WCSP solvers. It is composed of a list of numerical terms, except for the first one which defines the name of the problem, separated by space, tabulation or end of line. Instead of using names for making reference to variables, variable indices are employed. The same for domain values. All indices start at zero. All the constraints are defined in extension, by their list of tuples. A default cost value is defined per constraint in order to reduce the size of the list. Only tuples with a different cost value should be given. All the cost values must be positive. The structure of the format is: first, the name of the problem and dimensions, then the definition of the variables, and finally, the definition of the constraints. Files typically have the .wcsp extension.

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

- $<N>$ is the number of variables (integer).
- $<K>$ is the maximum domain size (integer).

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

- $<C>$ is the total number of constraints (integer).
- $< U >$ is the global upper bound of the problem (long integer).

The prologue is followed by the variable specifications:
<domain size of variable with index 0> ... <domain size of variable with index N-1>

The constraints are specified as follows (in one line):
<Arity of the constraint>
<Index of the first variable in the scope of the constraint>
...
<Index of the last variable in the scope of the constraint>
<Default cost value>
<Number of tuples with a cost different than the default cost>

And for every tuple (again in one line):
<Index of the value assigned to the first variable in the scope>
...
<Index of the value assigned to the last variable in the scope>
<Cost of the tuple>

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

A.2 Example

This section describes the WCSP file generated for the WCSP model described in Section 4.5. The original informed CSP (*P*) (see Example 4.5) is represented in Table A.1 (left) and its WCSP model ($modP$) is described in Table A.1 (right).

Recall that the original CSP represented in Table A.1 (left) is composed by two variables of domain size five, and four constraints. We would like to highlight that for this modeling example, one new constraint is generated for each original constraint. The rest of the parameters (variables and domains) remain the same. The first line of Table A.1 (right) shows this information. In addition, a huge number is specified for the parameter U because the costs associated with valid tuples must be negligible

Table A.1: Informed CSP (*P*) (left) and WCSP modeled ($modP$) (right).

P	$modP$
Variables:	filename.wcsp 2 2 8 8 100000000
$X_0 \dots X_1$	8 7
Domain:	
$D_0 : 3 - 7$	
$D_1 : 2 - 6$	
$C_1(0.2, 0.2) : x_0 + x_1 \leq 12$	2 0 1 100000000 24 {(3 2 0), (3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (5 6 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0)} 2 0 1 20 19 {(3 2 0), (3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (6 2 0), (6 3 0), (6 4 0), (7 2 0), (7 3 0)}
$C_2(0.8, 0.4) : x_1 + x_0 \geq 6$	2 1 0 100000000 24 {(3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (5 6 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)} 2 1 0 80 14 {(4 5 0), (4 6 0), (5 4 0), (5 5 0), (5 6 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)}
$C_3(0.4, 0.3) : x_1 - x_0 \leq 2$	2 1 0 100000000 24 {(3 2 0), (3 3 0), (3 4 0), (3 5 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (5 6 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)} 2 1 0 40 16 {(3 2 0), (4 2 0), (4 3 0), (5 2 0), (5 3 0), (5 4 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0), (6 6 0), (7 2 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)}
$C_4(0.2, 0.4) : x_0 - x_1 \leq 4$	2 0 1 100000000 24 {(3 2 0),(3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 2 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 2 0), (5 3 0), (5 4 0), (5 5 0), (5 6 0), (6 2 0), (6 3 0), (6 4 0), (6 5 0),(6 6 0), (7 3 0), (7 4 0), (7 5 0), (7 6 0)} 2 0 1 20 14 {(3 3 0), (3 4 0), (3 5 0), (3 6 0), (4 3 0), (4 4 0), (4 5 0), (4 6 0), (5 4 0), (5 5 0), (5 6 0), (6 5 0), (6 6 0), (7 6 0)}

with respect to U. For this example we fixed $U = 100000000$. The second line of Table A.1 (right) fixes the maximum domains of the variables.

The rest of lines of Table A.1 (right) represent the constraints of $modP$. For the clarification of their representation, one original constraint (C_1) and one new generated constraint (C_{41}) are explained in further details in the following.

The first original constraint is C_1 and it is composed by the variables x_0 and x_1. This is the respective order of its tuples (the first number refers to the assigned value to x_0 and the second one to the assigned value to x_1). This constraint has 24 valid tuples. Since this constraint is hard, the rest of the tuples are not allowed to be part of a solution. This fact is represented in $modP$ by assigning a default cost of U to all of them. The first valid tuple of C_1 is $(x_0 = 3, x_1 = 2)$ and the last one is $(x_0 = 7, x_1 = 5)$.

The last new generated constraint of $modP$, called C_{41}, has also the same two variables (with the same tuples representation order than C_1). However, C_{41} has only 14 tuples that satisfy it (see the calculation of $|T(C_{41})|$ above). For this reason, the cost assigned to them is zero (since they do not have to be penalized). The tuples that do not satisfy the new constraint C_{41} have a default cost of 20 because $p(C_4) = 0.2$ for *P* (see Example 4.5). Note that this cost is much lower than the cost U. For this reason these tuples are allowed to be part of a solution (soft constraint), however any solution that is composed by any of these invalid tuples has associated a penalization cost.

Bibliography

Bellicha, A. (1993). Maintenance of solution in a dynamic constraint satisfaction problem. In *Proceedings of the 8th Applications of Artificial Intelligence in Engineering*, pages 261–274.

Bessiere, C. (2006). Constraint propagation. *Foundations of Artificial Intelligence*, 2:29–83.

Boussemart, F., Hemery, F., Lecoutre, C., and Sais, L. (2004). Boosting systematic search by weighting constraints. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI-04)*, volume 16, page 146.

Climent, L., Barber, F., Salido, M., and Ingolotti, L. (2008a). Robustness and capacity in scheduling: Application to railway timetabling. In *Workshop on Planning, Scheduling and Constraint Satisfaction (IBERAMIA-08)*, pages 87–96.

Climent, L., Barber, F., Salido, M. A., and Ingolotti, L. (2008b). Algorithms for robust and on-line railway optimization: Improving the validity and reliability of large scale systems (ARRIVAL-TR-0213).

Climent, L., Salido, M., and Barber, F. (2009a). Finding stable solutions in constraint satisfaction problems. In *Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming Doctoral Program*, page 25.

Climent, L., Salido, M., and Barber, F. (2009b). Stability of solutions in constraint satisfaction problems. In *Proceedings of the 12th International Conference of the Catalan Association for Artificial Intelligence (CCIA-09)*, pages 301–309.

Climent, L., Salido, M., and Barber, F. (2009c). Stable solutions in constraint satisfaction problems. In *Doctoral Consortium of the 13th Spanish Conference for Artificial Intelligence (CAEPIA-09))*.

Climent, L., Salido, M., and 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-10)*, pages 752–761, Part 1.

Climent, L., Salido, M., and Barber, F. (2011). Reformulating dynamic linear constraint satisfaction problems as weighted CSPs for searching robust solutions. In *Proceedings of the 9th Symposium of Abstraction, Reformulation, and Approximation (SARA-11)*, pages 34–41.

Climent, L., Salido, M. A., and Barber, F. (2012a). Robustness in dynamic constraint satisfaction problems. *International Journal of Innovative Information and Control (IJICIC)*, 8(4):2513–2532.

Climent, L., Salido, M. A., Barber, F., and Wallace, R. J. (2012b). Solution robustness in constraint satisfaction problems with discrete and ordered domains. In *23rd Irish Conference in Artificial Intelligence and Cognitive Science (AICS-12)*, pages 74–83.

Climent, L., Wallace, R., Salido, M., and Barber, F. (2012c). An algorithm for finding robust and stable solutions for constraint satisfaction problems with discrete and ordered domains. In *24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-12)*, pages 874–879.

Climent, L., Wallace, R. J., Salido, M. A., and Barber, F. (2013a). A constraint programming approach to solve scheduling problems under uncertainty. In *Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS-13) in ICAPS 2013*, page 28.

Climent, L., Wallace, R. J., Salido, M. A., and Barber, F. (2013b). Finding robust solutions for constraint satisfaction problems with discrete and ordered domains by coverings. *Artificial Intelligence Review (AIRE), DOI 10.1007/s10462-013-9420-0*.

Climent, L., Wallace, R. J., Salido, M. A., and Barber, F. (2013c). Modeling robustness in CSPs as weighted CSPs. In *Proceedings of the 10th International Conference on Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR-13)*, pages 44–60.

Climent, L., Wallace, R. J., Salido, M. A., and Barber, F. (2014). Robustness and stability in constraint programming under dynamism and uncertainty. *Journal of Artificial Intelligence Research (JAIR), (in press)*.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

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

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

Escamilla, J., Rodriguez-Molins, M., Salido, M., Sierra, M., Mencía, C., and Barber, F. (2012). Robust solutions to job-shop scheduling problems with operators. In *24th IEEE International Conference on Tools with Artificial Intelligence (ICTAI-12)*, pages 209–306.

Fargier, H. and 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)*, pages 97–104.

Fargier, H., Lang, J., and Schiex, T. (1996). Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, pages 175–180.

Fowler, D. W. and Brown, K. N. (2003). Branching constraint satisfaction problems and markov decision problems compared. *Annals of Operations Research*, 118(1-4):85–100.

Fu, N., Lau, H., Varakantham, P., and Xiao, F. (2012). Robust local search for solving RCPSP/max with durational uncertainty. *Journal of Artificial Intelligence Research*, 43:43–86.

Gao, J. and Adviser-Revesz, P. (2006). *Adaptive Interpolation Methods for Spatiotemporal Data*. PhD thesis.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability. A Guide to the Theory of NP-completeness*. WH Freeman.

Ghédira, K. and Dubuisson, B. (2013). *Constraint Satisfaction and Optimization Problems*. Wiley Online Library.

Goles, E. and Martínez, S. (1990). *Neural and Automata Networks: Dynamical Behavior and Applications.* Kluwer Academic Publishers.

Grimes, D. (2012). *Identifying Sources of Global Contention in Constraint Satisfaction Search*. PhD thesis, University College Cork.

Hays, W. (1973). *Statistics for the Social Sciences (2nd. edit.)*, volume 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.

Hebrard, E., Hnich, B., and Walsh, T. (2003). Super CSPs. Technical report.

Hebrard, E., Hnich, B., and Walsh, T. (2004). Super solutions in constraint programming. In *Proceedings of the International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR-04)*, pages 157–172.

Hebrard, E., O'Sullivan, B., and Walsh, T. (2007). Distance constraints in constraint satisfaction. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 106–111.

Herrmann, H., Schneider, C., Moreira, A., Andrade Jr, J., and Havlin, S. (2011). Onion-like network topology enhances robustness against malicious attacks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(1):P01027.

Herroelen, W. and Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials. *European Journal of Operational Research*, 165(2):289–306.

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

Kitano, H. (2007). Towards a theory of biological robustness. *Molecular Systems Biology*, 3(1).

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

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

Lecoutre, C. (2013). *Constraint Networks: Targeting Simplicity for Techniques and Algorithms*. Wiley.

Leon, V., Wu, S., and Robert, H. (1994). Robustness measures and robust scheduling for job shops. *IIE transactions*, 26(5):32–43.

Lhomme, O. (1993). Consistency techniques for numeric CSPs. In *Proceedings of the 13th the International Joint Conference on Artificial Intelligence (IJCAI-93)*, volume 13, pages 232–238.

Mackworth, A. (1977a). Consistency in network of relations. *Artificial Intelligence*, 8:99–118.

Mackworth, A. (1977b). On reading sketch maps. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence (IJCAI-77)*, pages 598–606.

Miguel, I. (2004). *Dynamic Flexible Constraint Satisfaction and its Application to AI Planning*. Springer.

Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58(1):161–205.

Mohr, R. and Henderson, T. C. (1986). Arc and path consistency revisited. *Artificial Intelligence*, 28(2):225–233.

Pinedo, M. (2012). *Scheduling: Theory, Algorithms, and Systems*. Springer.

Ran, Y., Roos, N., and van den Herik, J. (2002). Approaches to find a near-minimal change solution for dynamic CSPs. In *the 4th international workshop on integration of AI and OR techniques in Constraint Programming for combinatorial optimisation problems*, pages 373–387.

Rosenfeld, A., Hummel, R. A., and Zucker, S. W. (1976). Scene labeling by relaxation operations. *Systems, Man and Cybernetics, IEEE Transactions on*, 6:420–433.

Rossi, F., Van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier Science Ltd.

Sabin, D. and Freuder, E. (1994). Contradicting conventional wisdom in constraint satisfaction. In *Proceedings of European Conference of Artificial Intelligence (ECAI-94)*, pages 125–129.

Sadeh, N. and Fox, M. (1996). Variable and value ordering heuristics for the job shop scheduling constraint satisfaction problem. *Artificial Intelligence*, 86(1):1–41.

Sakkout, H. and Wallace, M. (2000). Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388.

Salido, M., Climent, L., and Barber, F. (2009). A tool for finding stable solutions in constraint satisfaction problems. In *Workshop on Planning, Scheduling and Constraint Satisfaction (CAEPIA-09)*.

Sam, J. (1995). *Constraint Consistency Techniques for Continuous Domains*. PhD thesis, École polytechnique fédérale de Lausanne.

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

Schiex, T. and Verfaillie, G. (1994). Nogood recording for static and dynamic constraint satisfaction problems. *International Journal of Artificial Intelligence Tools*, 3(2):187–207.

Schmidt, G. (2000). Scheduling with limited machine availability. *European Journal of Operational Research*, 121(1):1–15.

Schröder, B. S. W. (2003). *Ordered Sets: An Introduction*. Springer.

Surico, M., Kaymak, U., Naso, D., and Dekker, R. (2008). Hybrid meta-heuristics for robust scheduling. *ERIM Report Series Reference No. ERS-2006-018-LIS*, Available at SSRN: http://ssrn.com/abstract=902747.

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

Tsang, E. (1993). *Foundations of Constraint Satisfaction*, volume 289. Academic press London.

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

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

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

Wallace, R. J. and 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. (1999). Search in a small world. In *Proceedings of the International Joint Conference on Artificial Intelligence*, volume 16, pages 1172–1177.

Walsh, T. (2002). Stochastic constraint programming. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-02)*, pages 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.

Wu, H. and Van Beek, P. (2007). On universal restart strategies for backtracking search. In *Proceedings of the Principles and Practice of Constraint Programming (CP-2007)*, pages 681–695.

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