HAL
open science

Constraint games revisited

Anthony Palmieri

To cite this version:

Anthony Palmieri. Constraint games revisited. Computer Science and Game Theory [cs.GT]. Normandie Université, 2019. English. NNT : 2019NORMC207 . tel-02366533

HAL Id: tel-02366533

https://theses.hal.science/tel-02366533v1

Submitted on 16 Nov 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Normandie Université

THÈSE

Pour obtenir le diplôme de doctorat

Spécialité INFORMATIQUE

Préparée au sein de l'Université de Caen Normandie

Constraint Games Revisited

Présentée et soutenue par
Anthony PALMIERI

Thèse soutenue publiquement le 15/05/2019 devant le jury composé de		
M. SIMON DE GIVRY	Chargé de recherche à l'INRA, INRA de Toulouse	Rapporteur du jury
M. LAKHDAR SAIS	Professeur des universités, Université d'Artois	Rapporteur du jury
M. NARENDRA JUSSIEN	Professeur, ENSTIMAC Albi-Carmaux	Président du jury
M. ARNAUD LALLOUET	Professeur des universités, Huawei Technologue Ltd	Membre du jury
Mme JULIETTE MATTIOLI	Docteur, Thales Research & Technology	Membre du jury
M. JEAN-CHRALES REGIN	Professeur des universités, Université de Nice Sophia Antipolis	Membre du jury
M. PATRICE BOIZUMAULT	Professeur des universités, Université Caen Normandie	Directeur de thèse

Thèse dirigée par PATRICE BOIZUMAULT, Groupe de recherche en informatique, image, automatique et instrumentation

UNIVERSITÉ CAEN NORMANDIE

GREYC

Résumé
Nouvelles techniques pour les Contraints Games

Cette thèse présente de nouvelles techniques pour les Constraint Games. La manière de résoudre un Constraint Game est repensée en terme de propagation de contraintes. Les préférences des joueurs sont maintenant considérées comme des contraintes globales permettant une intégration transparente dans les solveurs de contraintes ainsi que d'améliorer l'efficacité du framework. Notre nouveau solveur ConGA est diffusé en open source [1]. Celui-ci est plus rapide que les travaux connexes et est capable de trouver tous les équilibres de Nash, et cela même dans des jeux avec 200 joueurs voir 2000 pour certains jeux graphiques.

Grâce à cette perspective, le framework a pu être utilisé pour résoudre un problème de routage dans le domaine des télécommunications. Les aspects centralisé et décentralisé ont été étudiés. La comparaison de ces derniers est très importante pour évaluer la qualité de service dans les applications multi-utilisateurs. L'évaluation de cette dernière peut être très coûteuse, c'est pourquoi nous proposons plusieurs techniques permettant d'améliorer la résolution de ce problème et ainsi d'améliorer la résolution du problème.

Constraint Games revisited

This thesis revisits the Constraint games framework by rethinking their solving technique in terms of constraint propagation. Players preferences are considered as global constraints making transparently the integration in constraints solvers. It yields not only a more elegant but also a more efficient framework. We release our new solver ConGA in open source[1]. Our new complete solver is faster than previous state-of-the-art and is able to find all pure Nash equilibrium for some problems with 200 players or even with 2000 players in graphical games.

This new perspective enables us to tackle real-worlds Telecommunication problems. This problem is solved with a centralized perspective and a decentralized one. The comparison of the two last approaches is really important to evaluate the quality of service in multi-users application, but computationally consuming. That is why, we propose new techniques in order to improve the resolution process.

Mots-clefs— Constraint Programming, Game theory, Nash Equilibrium, Search strategy

1. https ://github.com/palmieri-a/CONGA

Table des matières

1 Introduction **1**
1.1 Motivations and context 1

I Background **5**

2 Constraint Programming **7**
2.1 Modeling 10
2.2 Propagation and filtering algorithms 11
2.3 Exploration 15

3 Game theory **23**
3.1 Taxonomy of game's Categories 24
3.2 Games' representations 27
3.3 Solution concept and their algorithms 33
3.4 Games example 40

II Contributions **43**

4 Constraint games **45**
4.1 Constraint Games framework 45
4.2 Variable control 50
4.3 Hard constraints 57
4.4 Conclusion 60

5 Nash equilibrium as a Global constraint **61**
5.1 Preferences as Global Constraints 61
5.2 Nash propagator applicability 65
5.3 Related work 66
5.4 Experiments 67
5.5 Graphical Aspects 68
5.6 Conclusion 69

6 Constraint games application **71**
6.1 Constraint model 76
6.2 Heuristics and problem's relaxation 78
6.3 A ILP model 87
6.4 Constraint Games 94
6.5 Related work 95
6.6 Instances and experimental results 96
6.7 Conclusion 106

7 Conclusion and perspectives **107**
7.1 Conclusion 107
7.2 Perspectives 108

A Objective criteria for robust search strategies **111**
A.1 Objective Function and Search Strategy 111
A.2 Experiments 115
A.3 Related Work 122
A.4 Conclusion 123

B ConGa : Constraint Game solver **125**

Bibliographie **131**

Table des figures

2.1 A maps and its optimal coloration 9
2.2 A graph view of the map coloring problem 9
2.3 An optimal solution for the map coloring problem into graph domain 10
2.4 An example of filtering 13
2.5 Search tree 17
2.6 Run-time of strategies on multiples problems 19

3.1 Tic tac toe possibilities 25
3.2 An example of bi-matrix representation, also called normal form . . . 28
3.3 WLC game in normal form 29
3.4 Dependency graph of the WLC game. 30
3.5 Wolf Lamb Cabbage Game in AGG 31
3.6 Best response of WLC game in normal form 34
3.7 A toy example showing a SNE and Pareto efficiency 37
3.8 Mixed Nash equilibrium computation 38

4.1 Players' deviations in the location game 48
4.2 Location game with undetermined behavior 51
4.3 Location game with beliefs 53
4.4 Location game with a shared stand 55
4.5 One shared binary variable under the normal form. 56
4.6 X's payoff 58
4.7 Y's payoff 58
4.8 Z's payoff 58
4.9 Hard constraint simulation in normal form 58
4.10 Location game's solutions without and with hard constraints 59

5.1 Nash constraints for Lamb in extension 62
5.2 An example of inconsistent filtering of preferences 66

6.1 Software Defined Networking 72
6.2 A plot of the congestion function for $MaxC = 1000$ and $cong'(0.2) = 1$ 75
6.3 Encoding of a path 77
6.4 Residual graph updates examples 80

6.5 Labeling of paths for two demands 81
6.6 Selection process of MB strategy 83
6.7 Selection process of CO strategy 83
6.8 Selection process of CO1 strategy 84
6.9 Problem's relaxation example for SDN 86
6.10 Column generation procedure 89
6.11 Three pieces linear approximation of the exponential function 90
6.12 All pairs of parameters are covered by 4 tests 98
6.13 Comparison of different CP heuristics on synthetic benchmarks 99
6.14 Comparison of the strategies for finding the first solution 101
6.15 Comparison of different initialization of the Column Generation model 102
6.16 Comparison Nash and and the different heuristics on synthetic benchmarks 105
6.17 Price of Anarchy and Price of Stability for small synthetic instances . 105

7.1 An instance of the meeting scheduling 110

A.1 (left) A search failing to consider the objective value. (right) An objective based search. 112
A.2 Comparison of the search tree by a lexicographic search (top) and an objective based search (bottom) 114
A.3 Comparison of the number of solved instances for different *OBS* configuration. 117
A.4 Comparison of the objective quality between different OBS configurations. 117
A.5 Running time comparisons of the hybrid strategies, with respect to the hybridization parameters. From left to right and the top to the bottom, the configurations of *ABS* with (0.25), (0.5), (0.75) and (0). 118
A.6 Comparison of the original search against their hybridized version . . 118
A.7 Comparison of the objective quality between search strategies and their hybridized versions. 119
A.8 Number of instances where each search strategy has found a strictly better objective compared to all the other. 121
A.9 Comparison of the number of instances solved by the different strategies as a function of time. 122

Liste des tableaux

3.1 Complexity to find a PNE for different games' representations 34

5.1 Runtime of the different methods on Gamut games 67
5.2 Runtime of graphical games on different topologies 70

6.1 Parameters of the generator 97
6.2 Results of the Constraint Programming model on real-world instances from SNDlib 100
6.3 Comparison of solutions on the unsolved instances 101
6.4 Column Generation results on real-world instances from SNDlib . . . 102
6.5 Comparison between Column Generation and CP 103
6.6 Comparison between Column Generation and CP 103
6.7 Constraint Games results on real-world instances from SNDlib 104

A.1 Number of timeout in some families of instances. 120

Chapitre 1

Introduction

1.1 Motivations and context

Artificial intelligence has a long story of modeling and solving problems. Most of the time, the problems tackled are not easy. Implementing them, cleanly and efficiently can be challenging. Multiples frameworks having each their own specificities and domains of applicability facilitating this solving process exist. Each framework has its own way to express problems. Three of the most important frameworks are Boolean satisfiability (or SAT), Integer Linear Programming (or ILP) and Constraint Programming (or CP). SAT and ILP allow expressing problems in a uniform way using either clauses or linear (in)equations. Their languages are restricted to a few instructions set. Constraint Programming works a bit differently. It has an extensible language where a basic element can be any relation. More generally, Constraint Programming is a successful generic paradigm to solve combinatorial problems [220]. Through time, Constraint Programming has shown its ability to solve problems of the frontier with Operational Research. Especially large-scale problems from industry like scheduling for industrial product line [186, 16], planning [9], nurse scheduling in hospitals [53, 197], Bin-Packing [198] and even crew management in transportation [138]. Vehicle Routing Problems (VRP) [203] is one of the famous applications of CP. In VRP a set of drivers have to visit a set of customers, while some regulation constraints have to be fulfilled on vehicles, customers, drivers and so on. The goal of VRP is to find a configuration which minimizes the cost. The VRP is somehow the same as the problem solved by GPS navigation softwares such like Waze [224]. A navigation software proposes routes to users to guide them from a starting point to a destination. The routes are recommended in real time, taking into account the current roads' congestion and so the others users' routes. A navigation software targets a different kind of solutions than the ones sought in VRP. While VRP is a mono-objective optimization, routing applications' goal is different due to the multiples objectives intrinsically present through users. Their goal is to make all users satisfied with their routes. A

user is going to be satisfied if he does not see any visible improvement to the route recommended by the system (i.e. if the system gives his best possible options). For instance, anyone using this application does not want to follow a route which is not optimal for himself, even if it is for a common interest.

The need to provide a high quality of service for navigation software companies is intensified by a competitive environment. Many applications using different systems are available. For instance, if a user can get a better route from a competitor, then the next times he will probably use the other application to be routed. The global problem of such routing application is related to game theory [217], which is a mathematical field modeling interactions between multiples agents (also called players). Game theory has found many applications such in scheduling [65], economics [192] and even biology [89]. A *game* is the central concept of game theory. The latter models a situation composed of players having each a possible set of actions, which, gives a certain level of satisfaction (or utility). Players' utilities are often expressed with numerical preferences but are not limited to it.

Representing a game is not an easy task. Historically, a game is modeled by providing a matrix giving a value to all the possible situations for each agent. This representation requires a lot of memory. Especially, its exponential memory requirement makes it quickly intractable in practice. For example, given 1000 users with two routes each, requires 2^{1000} integers which is about 10^{300}. In comparison, the number of fundamental particles in the observable universe is about 10^{80}. That is why compact representation like *Constraint Games* [150], were developed . Constraint Games is a way to represent games by using Constraint Programming. Constraint Programming provides a high level language to model problems like the VRP, or even some other domain problems such data mining[178], model checking[57] and QCSP[70, 15]. This high level language is an advantage to model complex problems and to revise the solutions sought when the model changes. For example, model users' choices on a dynamic network can be complicated. And especially when an unexpected event happens. For instance in, the closed roads have to be removed from the potential solutions. Under the matrix form, all algorithms need to be revised which is a potential source of errors and can be tedious. While with Constraint Programming, it is a bit different : only a constraint forbidding this situation has to be added. The declarative way to express problems makes the revisions easier.

Routing applications try to propose optimal routes to users in order to provide a high level of services. Mono-objective optimization is not suited for this kind of problems. An optimal solution based on one objective may not satisfy the users. In general users care more about their own preferences than a global one. This situation corresponds to a game solution concept which is a Nash equilibrium. A situation is said to be at Nash equilibrium when no player can improve its utility by changing his strategy alone.

Finding a Nash equilibrium is not easy and belongs to the class of NP-complete problems [78]. This complexity issue makes the brute force methods

totally intractable. However, this same brute force method was not improved until recently in Constraint Games [149] where the authors avoid some re-computation by maintaining counters. This method uses Constraint Programming to model players' preferences. The computation of a Nash equilibrium relies on a ad-hoc algorithm which is not directly integrated into Constraint Programming solvers. This is an issue because it cannot benefit from the recent solvers' advances and can be difficult to maintain and to handle by users.

Thesis contributions

The main thesis's contribution is to improve the current Constraint Game model from users' experience, by providing a simple and unified representation in constraint solvers. The state of the art algorithm for finding Nash equilibriums is also improved in practice. This contribution comes with a solver which has shown also is applicability to real problem instances in network optimization. In detail, the following outcomes have been achieved :

— A new view of Constraint Games by transparently embedding it into Constraint Programming solvers [159, 122]. We propose to construct for each player a global constraint, which encapsulates its preferences and removes the states which cannot be a Nash equilibrium. In other hand, this transparent view eases the solver maintainability and enables to reuse all recent advances in the field for free. The second contribution is to improve the algorithm from [149]. The problem's complexity and more precisely the arc consistency is characterized. Afterwards, we identify the tractable approximations of the problem to define an algorithm which helps to the detection of non solution. The practical interest of such approach has been shown on some classical game theory benchmarks.

— A problem coming the from telecommunication area has been solved using the Constraint Games solver. In it, a set of demands (for instance network packets) has to be routed through a network. The goal was to provide a routing minimizing the network's congestion and cost, while each demand has to be routed to its minimum cost individually. Also, networking problems are a large playground for game theory. For instance, in modern applications such the internet, many customers buy bandwidth in the network. The goal is thus to provide a good service to all the users by giving them a high level of satisfaction. Nash equilibriums are well suited for this kind of problems thanks to its stable and its local optimality for all agents. In addition, game theory allows computing measures evaluating the potential of decentralized algorithms against a global optimization problem. We used these measures to evaluate our problems the potential of decentralization. These contributions were published in [160] and in [122].

— A contribution outside this thesis's scope which consists of the study of a heuristic search for constraint optimization problems [161]. In this article,

we have shown that using the objective variable within a decision process helps to make better choices.

Thesis organization

The thesis organization is quite classical. A background's overview including Constraint programming and Game theory is given in the two first chapters. Afterward, the following chapters constitute the contributions and can all be read separately. The reader is encouraged to first have a glance at the background before reading the contributions. Three chapters are composing the contributions. The first one focuses on Constraint Games and some modelling aspects. The second contribution presents an elegant way to integrate games in constraint solvers and a new algorithm avoiding to explore non-solutions and its complexity. Finally, the last chapter presents an industrial use case in telecommunication networks. We also provide two appendices. One is presenting the constraint games solver released in open source. The second appendix contains a work which has been done during this thesis but outside its scope. It concerns the usage of the objective variable in constraint optimization problems. The objective variable has been shown as an advantage to solve problems efficiently.

Première partie

Background

Chapitre 2

Constraint Programming

Contents

2.1 Modeling **10**
2.2 Propagation and filtering algorithms **11**
2.3 Exploration **15**
Systematic search and heuristics 15
Other backtracking search and methods 20
Local search 21

Our work relies on Constraint Programming, the necessary background with examples to understand the contributions is provided in this chapter.

Constraint programming is a declarative programmatic paradigm approach, enabling to solve efficiently problems. A problem is defined by **variables** and **constraints**. Variables are the unknown parameters of the problem. The variables are each defined by a *domain* which specifies its possible values. A constraint stipulates the relationship between a set of variables. A constraint embeds a **filtering algorithm** which aims to detect the unfeasible parts of the problem, and so, to remove inconsistent values from the variables' domain. Formally a Constraint Satisfaction Problem (or a CSP) is defined as follows :

Definition 1 (Constraint Satisfaction Problem) *A CSP is a pair* $P = (X, C)$ *where* $X = \{x_1, x_2, ..., x_n\}$ *is a set of variables and* $C = \{C_1, C_2, ..., C_m\}$ *is a set of constraints. A variable* x_i *is associated with a domain* $D(x_i)$*, representing all of its possible values. A constraint* C_i *contains a set of all its allowed tuples defined over a subset* $S_{C_i} \subseteq X$ *of variables.*

A constraint can either be defined explicitly (also called by extension) by providing all the tuples characterizing the constraint solution, or with a formula

which defines the constraint. A simple example is the difference constraint which can be expressed in extension with a formula (see Example 1). A solution is a tuple of values $(a_1, a_2, ..., a_n)$ such that the assignments $x_1 = a_1$, $x_2 = a_2$ ..., $x_n = a_n$ respect all the constraints.

Example 1 (Constraint example) *Suppose you have a problem with 2 variables x_1 and x_2 having as domain the set $\{0, 1, 2\}$. A difference constraint is set over x_1 and x_2. Thus x_1 and x_2 cannot take the same value. Two ways are possible to specify this constraint :*

1. *in extension by providing all possibles tuples(i.e. constraint' solutions) :*

$$\{x_1, x_2\} \in \{\{0, 1\}, \{0, 2\}, \{1, 0\}, \{1, 2\}, \{2, 0\}\}, \{2, 1\}$$

2. *by a formula : $x_1 \neq x_2$*

An objective which has to be optimized can be added to a *CSP*. A *CSP* with an objecive is called a Constraint Optimization problem (*COP*). A COP is useful to rank the solutions or to add preferences between them. A solution with a better objective value is going to be preferred against another one which has a lower objective value.A Constraint Optimization problem (*COP*) is a pair (P, F_O), where P is a *CSP* and F_O is an objective function that has to be optimized. All solutions to a COP are not equivalent, as their overall quality is determined by the objective value $F_O(sol)$.

Example 2 (Map coloring) *Imagine we wish to color the map shown in Figure 2.1 with 5 colors or less. This map is made up of 9 countries which have to be colored.*

To model this problem we need 9 variables denoted by $X_{\{1,2,...,9\}}$ to determine each country's color. The domain of each X variable is the set $\{1, 2, 3, 4, 5\}$.

The maps' coloration has to respect some constraints : the color of each country has to be different from its neighbors :

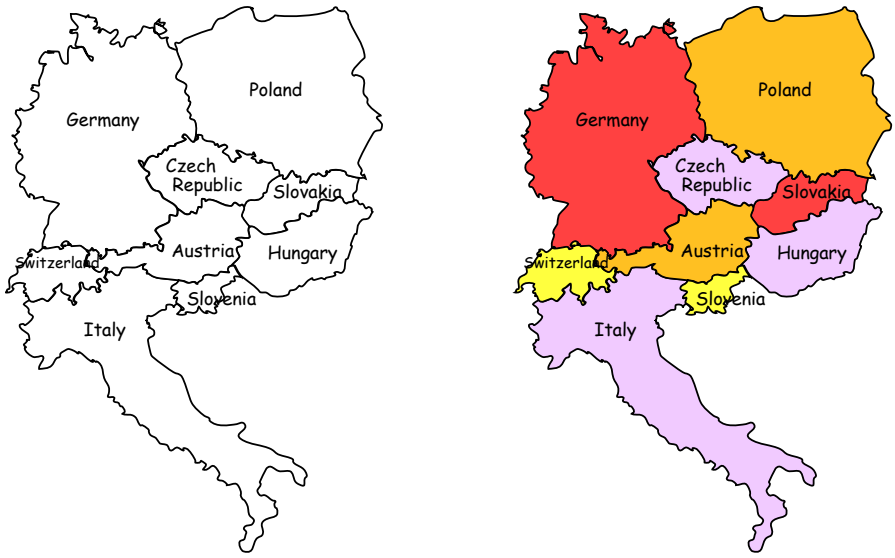
$$\forall i, j, \quad i \neq j \in |X|^2 : X_i \neq X_j$$

For example, Italy's color has to be different from the country where it has a border such as Switzerland, Austria, and Slovenia.

The problem contains an optimization condition : the number of colors used has to be minimized :

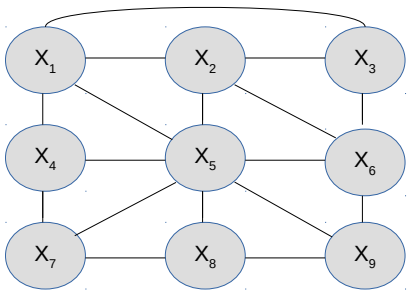
$$minimize \quad (max(X))$$

This problem can be translated into another domain which is graph theory. This perspective makes the problem easier to understand and to represent. To transfer the problem into graph domain, an incompatibility graph is built (see Figure 2.2). The graph is constructed such that a vertex represents a country and each border between two countries is symbolized by an edge. More precisely, the



(a) Data of map coloration problem (b) An optimal coloration

FIGURE 2.1 – A maps and its optimal coloration

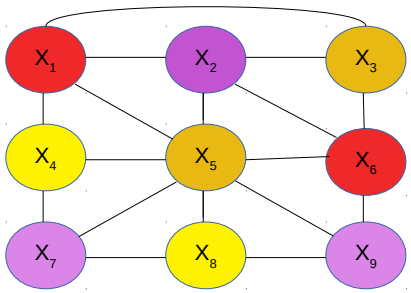


#var	1	2	3	4	5	6	7	8	9
domain	1,2,3, 4,5	1,2,3, 4,5	1,2,3, 4,5	1,2,3, 4,5	1,2,3, 4,5	1,2,3, 4,5	1,2,3, 4,5	1,2,3, 4,5	1,2,3, 4,5

FIGURE 2.2 – A graph view of the map coloring problem

nodes $X_1, X_2, ..., X_9$ respectively represent the countries Germany, Czech Republic, Poland, Switzerland, Austria, Slovakia, Italy, Slovenia, and Hungary. The two representations in Figure 2.1a and 2.2 are equivalent.

The solving process for a *CSP* or a *COP* generally involves two mechanisms : the filtering and the exploration. The filtering mechanism removes inconsistent variables' values while the exploration enumerates by a search tree the remaining possibilities.



#var	1	2	3	4	5	6	7	8	9
domain	2	3	1	4	1	2	3	4	3

Figure 2.3 – An optimal solution for the map coloring problem into graph domain

When a problem is solved with constraint programming three mechanisms have an effect on the performances : the model which determines the search space which has to be explored, the way to explore it and how the inconsistent values are detected by the filtering mechanism. We describe each of these mechanisms and what has been done to improve the efficiency in the following sections.

2.1 Modeling

Efficient models drastically impact the solving performances by reducing the search space and also the way to tackle the problem [62, 185, 207]. Building efficient models is not an exact science but it resembles more to art. In other words, given two models solving the same problem, it is impossible to determine which one is going to perform better a priori. The only way to get a comparison is to obtain it experimentally. Nonetheless some general advice about modelling can be given : "less variable is better" or "less constraints is better". Reducing the number of variables is going to curtail the problem size and potentially the search space to explore, while reducing the number of constraints is going to reduce the number of required propagation, which slow down the global process.

However while building a model, this one can still be analyzed to improve it. A way to do that is to capture symmetries.

Symmetries. In mathematics symmetries are part of group theory. A good analogy between the two fields can be found in [69]. A symmetry corresponds to identify identical permutation or search state thereafter removing the redundant part. In other words, a problem P can have variables which express the same

things or which just corresponds to the same solution or inconsistency, this is called a symmetry (see Example 3).

Example 3 (difference constraint and symmetries) *X and Y are two variables which have to be different and have both as domain the set* $\{0, 1, 2\}$*. The possibles solutions are the couples :*

$$\{\{0, 1\}, \{0, 2\}\{1, 0\}\{1, 2\}, \{2, 0\}, \{2, 1\}\}$$

Suppose now that your variables are interchangeable. Then the order does not matter. In that case, you can identify a symmetry over the two variables. For instance the solutions $\{0, 1\}, \{1, 0\}$ *are symmetric and equivalent. A simple way to break this symmetry is to force an order among the variables. This order can be imposed by the constraint :* $X < Y$

Many people have been interested to identify and remove new kinds of symmetries. This includes the exposure of new kind of symmetries [206, 72, 171, 125] and new methods to handle it [179, 223]. Specialization of filtering algorithms were proposed too, to handle symmetries during the filtering step [184].

Model relaxation Sometimes building an efficient model is not enough. Constraints can make the problem impossible or too difficult to be satisfiable. It is said that the problem is over-constrained [104]. A famous application example is the Radio Link Frequency Assignment problem [34]. From an industrial point of view, finding the optimal solution may not be the first target. Instead, finding at least one solution is very important, and even if this solution may do not be exact, due to the hardness of constraints. Multiples methods have been envisaged to solve such problems like maximizing the number of satisfied constraints (often called *MaxCSP* [63]). This method was refined by associating a weight to each constraint making the objective to maximize the weighted sum over the satisfied constraints (also called *Weighted CSP* [123]). Also, in the literature the problem can be found as constraint satisfaction with preferences [191]. The violation's cost can be also put on the variable assignment itself [14]. Another way to handle this, is Possibilistic CSP [199]. A possibilistic CSP assigns a possibility degree to each constraint expressing how desirable its satisfaction is. More information about these relaxations can be found in [143].

2.2 Propagation and filtering algorithms

The core reasoning of constraint programming is the propagation. It consists into removing inconsistent values of the problem letting only the feasible part remaining.

Filtering algorithm. A filtering algorithm is associated with a constraint. It aims to remove inconsistent values from variables' domain (see Example 4) given the constraint relation. A filtering algorithm has to catch the constraint property, to ensure solution's coherence. For instance, the difference constraint in map coloring problem (see Example 4) has to remove the inconsistent value (i.e. the one already taken) and to ensure that a solution respects the difference property.

The global problem's consistency is ensured by the **propagation mechanism**. The propagation mechanism ensures no more deduction can be made by reaching a fixed point. This is ensured by transmitting those domain reductions through the constraints network. It consists in arranging the propagators' execution, in order to find new value to remove. Usually, it corresponds to store events in a queue in order to check if these newly modifications impact the CSP's state. This process is iterated until no new modification arises, then a fix-point is reached. The propagation mechanism can be variable or constraint oriented. It depends on which event is triggered by the propagation and thus which need to be revised. The variable oriented propagation is shown in algorithm 1. In this algorithm, when a variable is removed from the Queue, all its propagators have to be triggered. This action is done sequentially by the *revise* function. Then, the newly modified variables have to be added to the queue. The process is stopped when no more variables are in the queue.

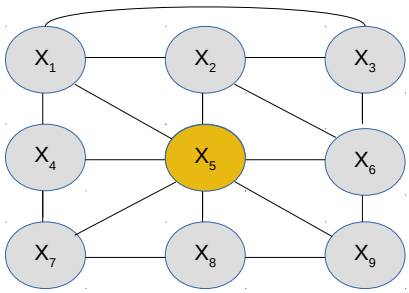
Algorithm 1 Variable oriented propagation algorithm

1: Q $\leftarrow$ $init_queue_with_all_variables()$
2: **while** $Q \neg \emptyset$ **do**
3: $\quad v \leftarrow remove(Q)$
4: $\quad modified_variables \leftarrow trigger_propagators(v)$
5: $\quad Q.enqueu(modified_variables)$
6: **end while**

Propagators also known as filters are filtering algorithm implementation. Propagators have been studied in a formal way to determine what is a correct behavior and which properties they have to ensure (see definition 2, more details can be found in [200]).

Example 4 (Example 2 continued : filtering algorithm)
The Figure 2.4 summarizes the filtering of the difference constraint. To begin, orange color (value 1) was given to the node 5. This modification is propagated to the other variables through the constraints network in order to remove inconsistent values. Since the connected vertices have to be different, then the value 1 is removed from the domain of $X_1, X_2, X_4, X_6, X_7, X_8$ *and* X_9.

Definition 2 (Propagator properties) *In order to make constraints propagation well-behaved propagators are decreasing and monotonic.*



#var	1	2	3	4	5	6	7	8	9
domain	~~1~~,2,3,4,5	~~1~~,2,3,4,5	1,2,3,4,5	~~1~~,2,3,4,5	1	~~1~~,2,3,4,5	~~1~~,2,3,4,5	~~1~~,2,3,4,5	~~1~~,2,3,4,5

FIGURE 2.4 – An example of filtering

— *A propagator p must be a decreasing function over all domains D. This property guarantees that constraint propagation only removes values.*

— *A propagator p must be a monotonic function : given two domains D_1 and D_2 : $D_1 \sqsubseteq D_2$ then the filtering operation should keep this order : $p(D_1) \sqsubseteq p(D_2)$. That is, application of p to stronger domains also yields stronger domains.*

— *Propagators must faithfully implement constraints. A propagator p is correct for a constraint c iff it does not remove any assignment for c. That is, for all domains D.*

Also, different filtering algorithm may have different filtering power. The filtering algorithm can be simple such as a simple deletion for impossible values or may be more complex like the AllDiff constraint (also called difference constraint) [181]. The latter is implemented by a matching algorithm in a bipartite graph. When the filtering algorithm has been triggered, the cost to run it is a bit more but it allows to prune sometimes exponential parts of the search space. The Alldiff constraint [181] was the first global constraint. A constraint is said to be global if its scope has an unfixed number of variables. The difference constraint has shown that a global point of view can help to find a stronger algorithm to filter inconsistent values and thus be more efficient while solving problems. However, it is not always the case and depends on how the constraint can be decomposed [145, 19]. Decomposing a global constraint can bring some benefits from multiples point of view : developer does not need to implement new constraints and can reuse the previous codes. Also, a decomposition can be more efficient than the original global constraint because the constraint is going to be awaken fewer times. All these concepts rely on the filtering power (also called *consistency*) of the decomposition which itself depends on the constraint's decomposability.

Consistency Filtering algorithms can act differently to provide different filtering power. This concept is called **consistency** [139, 18]. The consistency is a widely studied topic. Most of the time, it is a compromise between exploration's speed of new nodes and a stronger but slower filtering algorithm which is going prune more the search space. Multiples kinds of consistencies exist. The most known are the bound consistency and arc consistency.

Definition 3 (Arc consistency) *A CSP is said to be arc consistent if for all couple of variables* $(X_i, X_j), i, j \in X, i \neq j$*, and* $\forall v_i \in D(X_i)$*, it exists a value* $v_j \in D(X_j)$ *such that the instantiation* $\{(X_i, v_i), (X_j, v_j)\}$ *satisfies all the constraints.*

Note that if you consider each constraint alone (i.e you do not intersect constraint together, or add objective in consideration), then the arc consistency is the highest reachable consistency.

Definition 4 (Bound consistency) *A constraint is said bound consistent iif* $\forall x \in X$*, the domain's upper bound* $\overline{x}$ *and the domain's lower bound* $\underline{x}$ *are a support in* $C(X)$*. A constraint network is said bound consistent if all its constraint in it are bound consistent.*

Higher degree of consistencies have shown promising results to solve some problems like in graphs [22, 96]. Ones of the most famous are the k-consistency, Path consistency which is a special case of k-consistency [139] and the Singleton Arc consistency (SAC) [173]. This last one has received more attention this recent years by being practically improved by restricting the consistency procedure [221, 163].

Definition 5 (Singleton arc consistency) *The value a of a variable* x_i *is singleton arc consistency if and only if the problem restricted to* $x_i = a$ *is arc consistent. A CSP is singleton arc consistent if every value of every variable is singleton arc consistent.*

The consistency for satisfaction is not the only topic studied for propagators. Instead, while solving a COP the objective value can be back-propagated to prune more the search space. Specialized filtering algorithms taking into account the objective value have been proposed. For instance the global cardinality constraint [183], in routing problems [201, 84], in problem where the objective is express through a sum constraint [187] and for the round robin problem [100]. A more general approach was proposed in [61] and relies on global constraint linear relaxation in order to have a bound provided by the dual of linear programming.

2.3 Exploration

When solving a problem, the propagation mechanism alone maybe not powerful enough to find a solution or to prove a dead end. In such case, an exploration of the remaining space has to be done. Many algorithmic techniques and their variants exists to explore the search space : backtracking search, local search and dynamic programming. We review here the different main techniques used to explore the search space.

Systematic search and heuristics

A backtrack search [76] is a search procedure to solve combinatorial problems. The simplest version of backtracking search builds a search tree associated with a Dept First Search algorithm (DFS). This search tree begins at the root node with an empty set of *decisions*. Then the successors nodes are construct with respect to the decisions. A decision can be anything which reduces the problem, three ways are most used :

— *enumeration* : creates as many branches as the number of possible value for the selected variable

— *binary choice* : instantiates a variable to one of its possible value. Then, when the decision is backtracked, the opposite choice is considered, e.g. if the decision in an assignment to a value, then the opposite choice is going to be the current domain without this value.

— *domain splitting* : separates the variable's domain into two sets which do not necessarily instantiate a variable.

Note that a variable which is already instantiated cannot be part of a decision. A naive way to use backtracking search is just to consider lexicographic order to select the couple (variable/value). This ordering is called *search strategy*, *heuristic* or even sometimes *branching rule*. An example of the execution of a backtracking algorithm is given in Example 5.

Example 5 (Search tree exploration) *This example does not show a complete tree but instead how the exploration and propagation work. It is built on the map coloring example (see example 2). The search tree (see Figure 2.5(g)) is built by giving the references to the corresponding images. The initial step (Figure 2.5(a)), shows the triggering of the initial propagation which detects no inconsistent values. Since a fixed point is reached, and nothing more can be deduced, an exploration is done in order to solve the problem.*

The first strategy's decision is shown in Figure 2.5(b). It selects the variable X_5 and assigns it a value of 1. Then, this decision is propagated, removing the impossible values : 1 is removed from all the neighbors of X_5. Note that the negated decision and its propagation corresponds to the Figure 2.5(c). This negated decision

is taken into account when the DFS algorithm is going to backtrack from the state shown in Figure 2.5(b). Again, a fix-point is reached conducting the exploration be continued since not all variables are instantiated. Afterwards, X_1 *is instantiated to the value* 2 *(see Figure 2.5(d)), and again this decision is propagated through the neighbourhood of the variable yielding to remove the value 2 from the domain of the variable* X_2, X_3 *and* X_4*. Few steps later (see 2.5(f)) , a solution is found. This solution corresponds to the optimal one like shown in example 2.*

The ordering has a huge impact on the resolution time either to find a solution or to completely explore the search space [209, 162]. Heuristics can be divided into two categories : dedicated searches which are specialized for one given problem. Often dedicated searches exploit problems structures and problems' knowledge unknown by the solver. This knowledge is required to solve industrial problems. Dedicated heuristics have been applied to a wide range of problems including scheduling [216], network design [164], traveling salesman problem [59], software defined network [160], rectangle packing [204] and many others. However, such problem's knowledge, while appealing, is not always available nor possible. That is one of the main motivations for the development of black-box constraint solvers. In such solvers, a generic search is provided, letting only as a concern to users to build an efficient model. The first autonomous search was based on the first fail principles [90, 208] where the goal is to try to fail as soon as possible in the exploration in order to reduce the search tree size. Knowing how to fail as soon as possible cannot be computed efficiently in practice and relies on criterion like selecting the variable having the smallest domain size. Notwithstanding the experimental efficiency of such criterion, the complexity of CSP, which are generally NP-complete makes theses criterion being heuristic. In other words, it does not exist a unique criterion : some are going to perform well in some kind of problems while others will have better performances in others. A lot of attention has been put to find new criterion to solve more efficiently some problems[128, 170, 68, 162, 230, 60, 161]. Notably, in Constraint Programming, activity-based search (ABS) [144], impact-based search (IBS) [180] and weighted degrees (Wdeg) [27] are well known state-of-the-art search strategies for combinatorial problems. Three state-of-the-art Search Strategies are briefly described here. All these strategies share a common procedure, the value of variable is selected such that a solution can be quickly found, while the variable selection tries to minimize the search tree. For a more complete description please refer to their original publications.

Impact Based Search (*IBS*) [180] selects the variable whose choice is expected to provide the largest search space reduction. *IBS* considers the cardinality ratio reduction of the Cartesian product of the domains (called the impact). Thus the main feature of this Search Strategy uses variables' domains.

More formally, let x be a variable, and v be a value belonging to the current domain $D(x)$. Let P_{before} (resp. P_{after}) be the cardinality of the Cartesian product

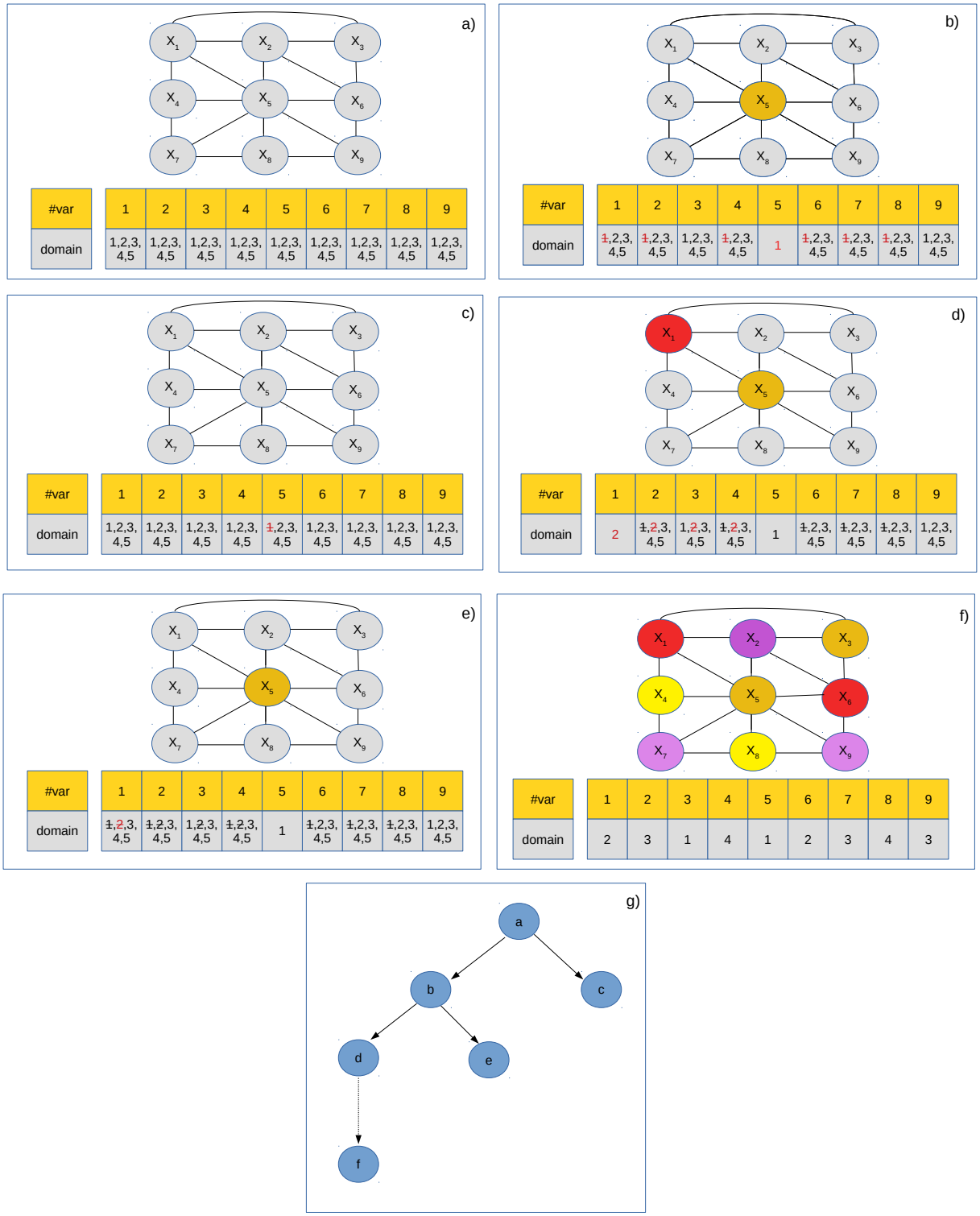


FIGURE 2.5 – Search tree

of the domains before (resp. after) the application of the decision $x = v$. The impact of a decision is :

$$I(x = v) = 1 - \frac{P_{after}}{P_{before}}$$

Let $\bar{I}(x = v)$ be the average impact of the decision $x = v$. Then, this impact of a variable x with current domain $D(x)$ is computed by the following formula :

$$\bar{I}_x = \sum_{v \in D_x} 1 - \bar{I}(x = v)$$

At each node, the free variable having the largest impact is assigned to its value having the smallest impact. Note that this search is an adaptation of pseudo-cost-based search from mixed integer programming.

Activity Based Search (*ABS*) [144] selects the most active variable per domain value. A variable's activity is measured by counting how often its domain is reduced during the search. Thus, once again, the feature of this Search Strategy uses the domains of the variables. More formally, the number of modification of the variable x is monitored and stored in $A(x)$. $A(x)$ is updated after each decision with the following rule :

$$\forall x \in X s.t. |D(x)| > 1 : A(x) = A(x) \times \gamma$$
$$\forall x \in X_0 : A(x) = A(x) + 1$$

X_0 represents the set of variables reduced by the decision and $\gamma \in [0, 1]$ is the decay parameter. *ABS* maintains an exponential moving average of activities by variables' value. At each node, *ABS* selects the variable with the highest activity and the value with the least activity.

Weighted Degree (*WDeg*) [27] uses the constraint graph to make decisions. *WDeg* counts the number of failures ω_c for each constraint c . *WDeg* features are the constraint graph and the fail counters. *WDeg* first computes, for each variable x, the value $wdeg(x)$, which is the weighted (ω) sum of the constraints involving at least one non-assigned variable. *WDeg* then, selects the variable having the highest ratio $\frac{|D(x)|}{wdeg(x)}$.

Portfolio method. New heuristics' interest is not limited to only solve more efficiently some classe of problems. Actually, a practical interest coming from economics exist. In economics, a portfolio is a financial product which tries to provide the best and safest investment by betting on multiples stocks option. Portfolios methods were built because the trading market is not predictable so it is unknown how to invest money. To mitigate an investment and decrease the possible losses bet is done on multiples options. In combinatorial optimization, this kind of reasoning is applied and was first brought in [107].

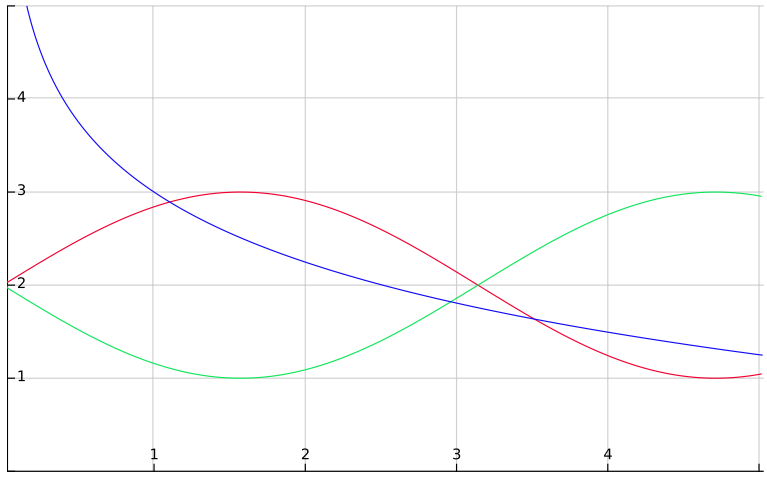


FIGURE 2.6 – Run-time of strategies on multiples problems

Example 6 (Portfolio) *Suppose you have a problem to solve. A priori, it is not possible to determine which strategy performs better. To minimize the risk (i.e. the possible runtimes), a portfolio method is going to launch many heuristics in parallel(or by simulating it, e.g. by short runs). This technique allows solving problems with the minimal run-time of the used strategies. For instance, suppose that 3 strategies are available to solve a large range of problems. The performances of the different strategies are reported in the Figure 2.6. The first strategy is drawn in blue, the second in red and the first one in green.*

The strategies have different performances on different problems. The minimal possible run-time of the three strategies is obtained by taking the minimum run time of all of them at each time. Basically by running these three strategies in parallel the run time corresponds to the minimum of all the strategies. What is important to underline is that no strategy is dominating another one.

The expected speedup can be supra linear [88]. With this method, higher performances are reaches when the chosen heuristics are orthogonal. This technique was brought to the fore when SATZilla [226] the first solver adopting a portfolio approach has won many prices at the SAT competition. An excellent survey about algorithm selection problem can be found in [121]. Also this variability among the different method brought different studies, notably one which underlined the heavy tail phenomena [77]. In fact, some researches have put in evidence that ideally, it exists a subset of variables to instantiate making the problem easier to solve [116]. This has motivated the ideas of *restarts*. A restart consists in make multiples short runs during the search in order to learn features or criterion about the problem. This learning step is then useful to consider earlier the variables making the problem harder [116]. Also *nogood* can be learned from failures. During the exploration, the search strategy detects inconsistencies (i.e.

a set of couples (variable/value)). This chain of affectations is called *nogood*. Nogood can help to solve more efficiently a problem by detecting inconsistencies which have been learned previously. Learning nogoods in general is a difficult task. The required space is exponential. Researches have been done in that sense to compress the representation as well as to deduce more thing from conflicts [52, 115, 127]. Coordination between the search, nogoods learning and restart have been actively studied to improve searches efficiency [129, 75].

Other backtracking search and methods

Since the first backtracking algorithm [76] multiples revisions and possible improvements of backtracking algorithms were proposed. Dynamic backtracking [74] is a way to avoid to explore the same dead end in the search tree by using nogoods. Ginsberg's dynamic backtracking algorithm (DBT), always puts the variable that has most recently been assigned a value on the right-hand side of the implication and only keeps nogoods whose left-hand sides are currently true. In this method, a nogood is deleted once the left-hand-side implication contains more than one variable-value pair that does not appear in the current set of assignment. The required space to implement dynamic backtracking is $o(n^2d)$. A generalization was proposed in [114] to the ith variable. The complexity analysis states that storing all nogoods is not tractable due to the space requirement.

Limited Discrepancy Search(LDS) [94] is another backtracking search. Given a heuristic, *LDS* explores the search tree by considering first the heuristic choices which are not the negations. LDS explores the search space until the limit of negation allowed have been explored. For a given binary heuristic, it is assumed that a left branch is the choice made by the heuristic while the right branch is the negation and thus a default choice. *LDS* considers that a heuristic guides well the exploration and thus explores first the part which is the most promising regarding the heuristics decisions. In other words, the negation are not the good decisions to continue. The explored branches are going to be explored by increasing number of negations. The first branch is the one having only left branches, the seconds are going to be the ones having chosen exactly one right branch etc. Another version of *LDS* but bounded was proposed in [222].

Other backtracking techniques were proposed, notably an Hybrid Best-First Search [2] which combine Depth first search usually done by simple backtracking and Best first search.

MDD solvers In recent years, another sub-paradigm of search space exploration was proposed. Multivalued Decision Diagram (MDD) is a compressed data structure. MDDs are interesting because they provide fast operations [167] which most of the time are linear. Fast operation, in addition with efficient construction [168] has shown promising result in solving multiples real-life problems [17, 119, 194].

Local search

Many world applications cannot be solved by a complete and systematic search. Local search methods are very famous to solve Constraint Satisfaction Problems, because of their efficiency and thus despite their incompleteness (see [105, 99]). In opposition to a complete and systematic search used by the combination of a search strategy and a *DFS* algorithm, a local search is able to explore more diverse space in a given amount of time. While a complete and systematic search has to explore an entire sub-tree before exploring another part, a local search is able to modify at any time, any choice during its decision process. Therefore, a local search can revise earlier a bad decision compared to a systematic search.

Many applications have been successfully solved with local search methods, including vehicle routing [203] and job-shop scheduling [12].

Local Search techniques are based on a simple and general idea : For starting, an initial point which is a complete variables assignment is randomly chosen. Then it iteratively moves to a neighbor solution. Deciding which neighbor will be chosen is performed by an evaluation of a heuristic function. The process is iterated until a termination criterion is reached. It could be the detection of a solution or user-specific such as the max steps or time exceeded.

Chapitre 3

Game theory

Contents

3.1 Taxonomy of game's Categories **24**
Players interactions : static and dynamic 24
Cooperation and selfishness 26
Game's knowledge 26
3.2 Games' representations **27**
Extensive representations 27
Combinatorial games 31
3.3 Solution concept and their algorithms **33**
Pure Nash equilibrium 33
Best response dynamic 35
Nash equilibrium relaxations 37
Quantifying the efficiency of equilibrium 39
3.4 Games example **40**

Game theory is a successful paradigm to model interactions between multiples **agents**. Originally formalized in [229, 217], game theory has found early some applications in economy [112], politics [30], wireless network [41]. Even a biologist, John Maynard Smith was rewarded for his work using game theory [210]. The central notion in game theory is *game* in which the strategic behaviors are represented. Game theory provides an analytic framework to analyze it. The analysis of strategic behavior(like in games) happens more than it is though. Daily life choices involve decision making process (like in Example 7). This process depends on the knowledge about the situation, the possibles cooperation etc. In this chapter, game theory background is provided to understand our contributions.

3.1 Taxonomy of game's Categories

A **game** is the central notion in game theory. It represents a situation in which a set of **players** (also called agents) have to decide an **action** (or strategy) among a possible set. When all players have chose their strategy, they are given a **reward** (also called utility).

Definition 6 (Game)
A game *is a 3-tuple* $G = (\mathcal{P}, A, u)$ *where :*

- $\mathcal{P}$ *is a finite set of players.*
- $A = (A_i)_{i \in \mathcal{P}}$ *where* $A_i \neq \emptyset$ *is the set of actions that can be played by player* i*. We call* strategy *the choice of an action by Player* i *and* strategy profile *a tuple* $s = (s_i)_{i \in \mathcal{P}}$ *where* $s_i \in A_i$*. The set of strategy profiles is denoted by* $A^{\mathcal{P}}$*.*
- $u = (u_i)_{i \in \mathcal{P}}$ *where* $u_i : A^{\mathcal{P}} \to \mathbb{R}$ *is the utility function of player* i*.*

Example 7 (Introductory example : Battle of the sexes) *The Battle of the sexes[157] is a game composed of two players. A couple has planned to go out at evening. The wife would like to go to the opera, while the husband wants to watch soccer on TV. They can go to different places, but they rather prefer to stay together. They did not choose yet where to go and cannot communicate together. Knowingly of this situation they have to go somewhere and hopping to find each other.*

Besides the mathematical definition of a game, its nature and how it can be studied is dependent on multiples concepts. The different point of views such the interactions and the possible games' knowledge are described here.

Players interactions : static and dynamic

Games can be played with different interactions modes between players. The interactions specify how a game is played. Two mains modes exist : **static** and **dynamic** games.

A game is said to be *static* (also called *simultaneous*) if the players' strategies are played only once, and simultaneously. The Example 8 shows the battle of the sexes (Example 7 continued) as a static game.

Example 8 (Example 7 continued as a static game)
The battle of the sexes shown previously is traditionally a static game. It is played once and players chose the actions at the same time. The players' payoffs and the possibles actions characterizing the game are described here.

- $\mathcal{P} = W, M$

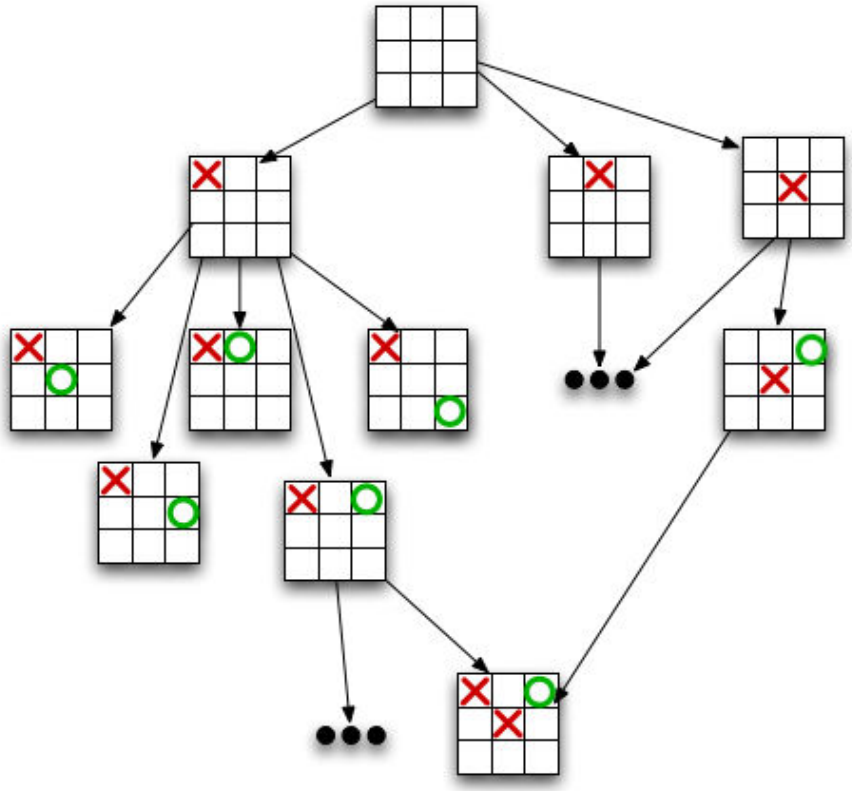


FIGURE 3.1 – Tic tac toe possibilities

— *Both player can choose among the following strategy :* $\{O, S\}$ *where* O *denotes Opera and* S *Soccer*

— *The utilities of the game are the following :* $u_W(O, O) = u_M(S, S) = 3$, $u_W(S, S) = u_M(O, O) = 2$, $u_W(O, S) = u_M(S, 0) = 0$

Unlike static games, dynamic games have a sequence of actions, i.e. a player will chose an action before the other ones choose theirs.

A simple and well know dynamic game is the "Tic Tac Toe". In this game, two players have to choose a sequence of moves in order to win. The first player succeeding to align 3 circles or crosses in a row, a column or a diagonal wins. The game can be finished on a draw, if no player can put 3 elements contiguously. An example of "Tic Tac Toe" progress is shown in the Figure 3.1. First the player playing cross begins, then it is the turn to the one playing circle and so on, until the end of the game is reached.

Cooperation and selfishness

Game theory focuses on studying problems in which players' goals depend on others players' actions. A crucial characteristic to identify is thus cooperation between players. Mainly two kinds of behaviors have been studied : **cooperation** and **selfishness**. In cooperative games, several players may collaborate to achieve their goals together. In opposition, in non-cooperative games, all players are self-interested and focus only on their goals. Each player just wants to optimize his own utility no matter what happens with the others *Non-cooperative games*, which are a crucial class in game theory, capture many research interests. A simple example of non-cooperative game has been shown in Example 7.

Cooperative game theory (also called coalition game) provides an approach to analyze and to describe a structural behavior of players and their payoffs. Cooperative game theory analyzes the behavior of player when they can build coalitions. A *coalition* is simply a subset of the set of players coordinating their strategies in order to play. In a coalition, the players agree on how the total payoff is to be divided among the members. A cooperative game [54] consists of a set of players plus a characteristic function specifying the value created by the different coalitions in the game.

Example 9 (Battle of the sexes as a collaboration game)

The Battle of the Sexes shown in Example 7 is adapted here as a cooperative game :

- $\mathcal{P} = \{Husband, Wife\}$
- *There are 3 possible coalitions :* $C_1 = \{wife\}$ *and* $C_2 = \{man\}$ *and* $C_3 = \{wife, man\}$
- *the utility function of each coalition is :*
  - $v(C_1) = (v_{Wife})$ *where* $v_{Wife} = 0 \vee v_{Wife} = 2 \vee v_{Husband} = 3$
  - $v(C_2) = (v_{Husband})$ *where* $v_{Husband} = 0 \vee v_{Husband} = 2 \vee v_{Husband} = 3$
  - $v(C_3) = (v_{Wife}, v_{Husband})$ *where* $v_i = 2 \vee v_i = 3$ *=3*

As opposition to the classical battle of the sexes, when a coalition is formed by the man and the wife, then they do not consider anymore to go somewhere where the other is not. Going alone somewhere, is dominated by going anywhere together.

Game's knowledge

Another important point is how the players know what happens in the game. In fact, two levels of knowledge exist : the one about the others players and the knowledge about the game itself.

In a game with **imperfect information**, players are simply unaware of the actions chosen by the other players. However, it is known who is participating to the game, the possibles strategies and the possible players' preferences. In **incomplete information games**, players may or may no the other players' strategies and preferences. The Example 10 illustrates these by different situations based on the Battle of the sexes

Example 10 (Battle of the sexes with different knowledge) *Four different situations of the battle of the sexes are given here to understand the impact of imperfect and incomplete information in games.*

- ***Imperfect and incomplete information*** *: The Wife(resp Husband) does not know if the Husband(resp Wife) is part of the game and his preferences.*
- ***Perfect and incomplete information*** *: The Wife(resp Husband) is part of the game but her(resp his) preferences are not known.*
- ***Imperfect and complete information*** *: The Wife(resp Husband) does not know if the Husband(resp Wife) is part of the game but knows her (resp his) preferences.*
- ***Perfect and complete information*** *: Everything is known : who is part of the game and the preferences.*

This section has given the different games models and situation which are well known, in the next section we are going to see how to represent games.

3.2 Games' representations

Uniform representations are required to use algorithm in games. Until now, the representation of games was not evoked. In this section, the techniques used to represent games are listed. Especially the ones which are fully expressive, which means that any game can be represented in it.

Extensive representations

In this section we list the extensive representation (i.e. where all the possibilities are listed).

Normal form The standard game representation of a static game is normal form [64], which is an n-dimensional matrix stating all utilities of all players for all joint strategy profiles in the game. Also, in a sense, the normal form is the most fundamental representation in game theory, because of the others representations of finite games can be encoded in it.

		H	
		O	S
W	O	$(3,2)$	$(0,0)$
	S	$(0,0)$	$(2,3)$

Figure 3.2 – An example of bi-matrix representation, also called normal form

An example of normal form representation is given in the Figure 3.2. It corresponds to the Battle of the sexes as shown in Example 7. The possible actions of the Wife (name W here) are O and S on the columns, while the possible actions of Husband (named H here) are the ones on the lines. In an informal way, the player M is going to choose the column, while the W one is choosing the line. Their decisions are going to characterize their rewards. For instance, when W and H choose O and P, they are going to be respectively rewarded by 3 and 2. The Wolf Lamb Cabbage Game is described into numerical preferences in the Example 11.

Example 11 (Wolf Lamb Cabbage game (WLC)) *Three agents, Wolf (W), Lamb (L) and Cabbage (C) receive an invitation for a party. Each of them has the choice to come or not at this event. Each agent has his own preferences about meeting the others participants. Wolf would be happy to see Lamb but is indifferent about Cabbage's presence. Lamb would like to see Cabbage but only if Wolf is not coming. And Cabbage is a plant and is indifferent to everything. The action of P coming to the party corresponds to p and the reverse to $\overline{p}$.*

- *$\mathcal{P}$ ={ W, L, C} where W, L and C represent respectively the Wolf, Lamb and Cabbage.*
- *Players can either choose to come to the party which corresponds to p or the reverse $\overline{p}$. For each agent we respectively denote for the Wolf, the Lamb and the Cabbage their actions by w, l, c.*
- $u_W(w,l,c) = 1, u_L(w,l,c) = 0, u_C(w,l,c) = 0...$

The action of P coming to the party corresponds to p and the reverse to $\overline{p}$. For instance w, l and c represent respectively the action of coming for Wolf, Lamb and Cabbage. The complete representation of the game requires $3 \times 2^3 = 24$ integers. We intentionally do not combine all the players into on big matrix in order have a better understanding of the example and the required size.

The normal form is fully expressive but not compact. For instance, to represent a game composed of 100 players and all having only 2 strategies, the normal form requires to store 100^2 numbers. But it is not the only weakness, games become totally unstructured and flat complicating their modelling especially for big games.

	l, c	$l, \overline{c}$	$\overline{l}, c$	$\overline{l}, \overline{c}$
w	1	1	0	0
$\overline{w}$	0	0	0	0

Wolf's payoff

	w, c	$w, \overline{c}$	$\overline{w}, c$	$\overline{w}, \overline{c}$
l	0	0	1	0
$\overline{l}$	1	1	0	0

Lamb's payoff

	w, l	$w, \overline{l}$	$\overline{w}, l$	$\overline{w}, \overline{l}$
c	0	0	0	0
$\overline{c}$	0	0	0	0

Cabbage's payoff

FIGURE 3.3 – WLC game in normal form

As mentioned earlier, representing a game can be challenging in term of memory. This need brought people to find new compact representation like Graphical games [117] or Action Graph Games [113]. These representations factorize the normal form by catching a property and are able to compact it.

Graphical game : A graphical game only represents the useful interactions between players. In other words, if two players' actions are independent, then, in a graphical game, the action are not directly stored. Graphical games are kind of reduced normal form embedding a graph indicating how and which players interact together. The graph which represents the interaction is constructed such that each vertex is a player and a player is linked into another one if its utility depends on the action of the other player. Any node of the graph (or player) has a local matrix which depends on the interactions between the other players (i.e. its neighbors).

Example 12 (Wolf Lamb Cabbage game as a graphical game) *Since Wolf only depends on Lamb and Lamb on Wolf and Cabbage, the WLC game is actually a graphical game, as depicted in Figure 3.4. By using this dependency scheme, we are able to reduce the size of the matrices to one* 1×2*, one* 2×2 *and one* $2 \times 2 \times 2$*, for a total amount of 14 integers. Also, there is no more need to store the preferences of the cabbage since it has always the same preference.*

The required matrices are now :

The memory consumption of graphical games is reduced only if the largest neighbors degree d is inferior to the number of players n. Hence, if d $\ll$ n, then the graphical representation is significantly more compact than the normal form.

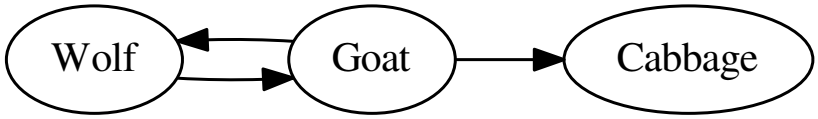


Figure 3.4 – Dependency graph of the WLC game.

	l	$\bar{l}$
w	1	0
$\overline{w}$	0	0

Wolf's payoff

	w, c	$w, \overline{c}$	$\overline{w}, c$	$\overline{w}, \overline{c}$
l	0	0	1	0
$\bar{l}$	1	1	0	0

Lamb's payoff

However, it also means that this representation does not fit well to games that are not separable, i.e. when there exists a full interaction between all players. In this games, the size would exactly the same as the normal form size.

Action graph game

Action Graph Game (AGG) is a representation fully expressive and at least as compact as graphical games. AGGs catch symmetries of interactions between players and compacts their identical strategies. This is useful for specific kind of game where the players' utilities do not depend on who is taking the action but instead on the number of players choosing an action. These kind of games are called anonymous. AGGs are represented thought a directed graph with a set of nodes A, a set of edges E, and a set of agents $N = \{1, ...n\}$. Identical tokens are given to each agent $i \in N$. To play a game, each agent i simultaneously places his token on a node $a_i \in A_i$, where $A_i \in A$. Each node in the graph corresponds to an action available to one or more of the agents. Each agent's utility is calculated according to an arbitrary function of the node he chooses and the numbers of tokens placed on the neighbor'nodes. The WLC game in AGG form is given in Example 13.

Example 13 (Wolf Lamb Cabbage as an AGG) *The Wolf, Lamb, and Cabbage have to choose among their respective actions on their set of nodes A_W, A_l and A_C. They can either choose to come (e.g. W) or not (e.g. $\neg W$ by placing a token on their corresponding node. Then a function on each node is going to compute the utility knowing the actions of the neighbors. For instance, in the figure 3.5 the set of actions for the Wolf denoted by A_W are $\{W, \neg W\}$. He can either place a token on W or on $\neg W$ to determine if he comes or not. Afterwards, his reward is going to be computed according to the choices of its*

neighbors nodes. For instance if the Wolf choose W then his rewards depend on l or ¬l, depending on where the Lamb has put his token.

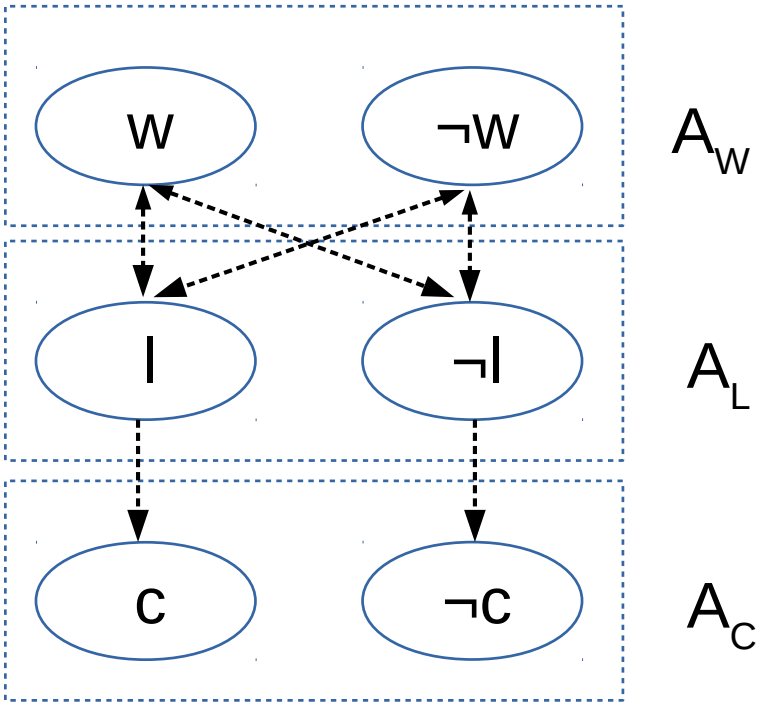


FIGURE 3.5 – Wolf Lamb Cabbage Game in AGG

Combinatorial games

In this part, game's representations coming from combinatorial techniques are presented. Two main approaches exist : Boolean Games based on Boolean satisfiability (or SAT) theory and Constraint Games based on constraint programming (see Chapter 2). Only Boolean games are presented, instead, the chapter 4 is dedicated to the Constraint Games which is the approach used in our contributions. We also refer the *Distributed Constraint Optimization Problems*(DCOP) which can sometimes resemble or solve the same problematic as in game theory. Compactness is one of the main advantages of such approaches. The players' utilities are not directly stored but instead formulas are provided to compute it on fly. Also, the resolution system relies on solvers, which are optimized to be efficient at solving combinatorial problems.

Boolean games

Boolean games were initially proposed in [92]. In such games, each player own a set of variables and a SAT problem which defines its satisfaction.

Definition 7 (Boolean games) *A boolean game is a 3-tuple* $(\mathcal{P}, V, \Sigma)$ *where :*

- $\mathcal{P} = \{1, ..., n\}$ *is a finite set of players*
- $V = (V_i)_{i \in \mathcal{P}}$ *such that* $i \neq j \leftrightarrow V_i \cap V_j = \emptyset$ *where* V_i *is a set of propositional variables controlled by player i*
- $\Sigma = \{\phi_1, ..., \phi_n\}$ *is the set of goals, where each* ϕ_i *is a satisfiable formula for player i*

The WLC game in a Boolean game form is presented in the Example 14.

Example 14 (Example 11 as Boolean game)
The WLC game can be express by a Boolean Game. First, it can be underlined that each player has only two strategies and dichotomic preferences, thus making this problem expressible with boolean games [92, 25]. We name x_W, x_L *and* x_C *the respective optimization variables of Wolf, Lamb and Cabbage. Then we can state using boolean algebra notations :* $x_W = wl$, $x_L = \overline{w}lc + w\bar{l}$ *and* $x_C = 0$.

Logical propositions can sometimes be not enough to properly provide a compact representation, that's why boolean games preferences were extended with CP-net [28, 23]. Boolean games concentrate a lot of attention these last years. Especially by adapting some well known results of game theory. For instance dependencies of graphical games to Boolean games [24], cooperative Boolean games [56], iterative Boolean games [85], incomplete or imperfect information [80, 46] or even possibilitic games [43]. New concepts were also envisaged. The structure provided by combinatorial games made people think differently and that why some concepts emerged like partial cooperation between player [83] or the concept of variables' control [73, 13, 103] which is also related to imperfect information or partial collaboration.

Distributed Constraint Optimization Problem (DCOP). Distributed Constraint Optimization Problem consists of a distributed optimization problem through agents. The main concern of such approach is to keep privacy of agents' preferences (or variable assignment). In such representation, a search is done by passing message (usually preferences) between agents in order to agree on a solution.

A Distributed Constraint Optimization Problem is defined by a set of agents controlling each a set of variables defined over a domain. Each variable assignment is defined by a cost and an operator that aggregates all of the individual costs for all possible variable assignments. The objective of a *DCOP* is to have each agent assigned values to its associated variables in order to either minimize or maximize a cost for a given assignment of the variables. Multiples questions related to game theory were tackle by DCOP and its asymmetric variant like the solution sought : Nash equilibrium [135, 82, 81, 132]or even mechanism design [132].

3.3 Solution concept and their algorithms

So far, the solutions in games have not been expressed. In games, solution concepts (also called equilibrium) have been discussed for decades, resulting in many solutions concepts with different properties [157]. One of the most fundamental equilibria is the Nash equilibria [146], which models a situation where no player has an incentive to change his decision unilaterally. Finding a Nash equilibrium is a satisfaction problem in which players are in a best response statement. In other words, it consists in finding a state where no players can improve its current satisfaction by changing alone its strategy. In this thesis, we focus only on Pure Nash equilibrium because it has the advantage of giving a deterministic decision for the players and thus being easily applicable in real world applications. Various other solution concepts have been proposed in the literature. Among them the most popular are : the mixed Nash equilibrium [146], Bayesian Nash equilibrium [93] and correlated equilibrium [5]. However, we do not detail these concepts since they are not directly related to this thesis.

In this section, classical solution concepts are listed and described.

Pure Nash equilibrium

The basic solution concept for a static game is called *Nash equilibrium* and corresponds to a state where each player has no incentive to change his strategy assuming the other players do not change theirs. A best strategy for a player is usually called a *best response*. Given the choice of a strategy by the other players, a best response for a player is the choice of the (or one of the) strategy which has the best reward for this player.

Definition 8 (Best Response)
A strategy profile s is a best response *for player i if and only if* $\forall s'_i \in A_i,\ u_i(s) \geq u_i((s'_i, s_{-i}))$.

Pure Nash equilibrium (PNE) can be rephrased in term of better response : a situation is at Nash equilibrium if it does not exist any better response for any player, in other words it does not exist a strategy giving a better utility.

Example 15 (Pure Nash equilibrium in WLC game) *In Figure 3.6 a box is colored in red if it is a best response for the given player. For instance, let's take the Lamb's payoff, and a configuration such that the Wolf and Cabbage are coming (column w, c in the Lamb's payoff). The action not coming is a best response in this situation since it has a better payoff than coming. The intersection of the sets of best responses gives us the set of PNEs. The WLC game has 3 PNEs. The first one happens when all players do not come to the party ($\overline{wlc}$). The two others when Wolf chooses to come and Lamb to skip ($w\overline{lc}$ and $w\bar{l}c$).*

	l, c	$l, \overline{c}$	$\overline{l}, c$	$\overline{l}, \overline{c}$
w	1	1	0	0
$\overline{w}$	0	0	0	0

Wolf's payoff

	w, c	$w, \overline{c}$	$\overline{w}, c$	$\overline{w}, \overline{c}$
l	0	0	1	0
$\overline{l}$	1	1	0	0

Lamb's payoff

	w, l	$w, \overline{l}$	$\overline{w}, l$	$\overline{w}, \overline{l}$
c	0	0	0	0
$\overline{c}$	0	0	0	0

Cabbage's payoff

Figure 3.6 – Best response of WLC game in normal form

For finding PNE in games, a generic algorithm is described in Algorithm 2. The *Enum* procedure calls for each strategy profile a function *isNash* which checks for each player with the function *Deviation* whether the current strategy is a best response or not for this player. The main interest of this algorithm is that it provides a complete search which outputs all equilibria. This naive algorithm has not been improved until recently by the *Conga 1.0* algorithm [149] for Constraint Games. *Conga 1.0* is a tree-search algorithm which memorizes the best responses already found and uses a counter for pruning some actions that are never best responses. For reasoning on basic Boolean games, the first methods for computing all PNE (or core elements) have been proposed by [44] by using disjunctive answer set programming [31]. Their idea is to transform a Boolean game into an answer set program. Henceforth, by using the saturation techniques [10], the obtained answer set will coincide with the set of PNE (or the cores).

A summary of complexities for finding a PNE for different representations is shown in table 3.1. In this table s corresponds to the number of strategy, n to the number of players and d to the maximum degree in the graph of graphical games.

Representation	**PNE complexity**	**space complexity**
normal form game	NP-complete [78]	s^n
graphical game	NP-complete [78]	s^{d+1}
boolean games	Σ_2^P [55]	unknown
constraint games	Σ_2^P [148]	unknown

Table 3.1 – Complexity to find a PNE for different games' representations

Algorithm 2 Enum

```
procedure ENUM(game : G = (P, A, u))
    for all s ∈ A^P do
        if isNash(s) then
            print(s)
        end if
    end for
end procedure
function ISNASH(strategy profile : s) : boolean
    for all i ∈ P do
        if Deviation(s, i) then return false
        end if
    end for
    return true
end function
function DEVIATION(strategy profile : s, player : i) : boolean
    for all s'_i ∈ A_i, s'_i ≠ s_i do
        if u_i(s_i, s'_{-i}) > u_i(s) then return true
        end if
    end for
    return false
end function
```

Definition 9 (Pareto Efficiency) *A strategy profile $s \in A_i$ of a game is said to be weakly Pareto efficient if there does not exist any $s' \in A_i$ such that $u_i(s) > u_i(s')$, for all $i \in \mathcal{P}$.*

Note on best response concept : The concept of best response can be abstracted in multiples ways. The best responses simply state if a player is satisfied or not. In general, numerical preferences are given, but it can be extended to any notion. For instance, if a player has multiple objectives, then the best responses could be the Pareto frontier which corresponds to the states where each objective alone are not improvable without making another one worse (see definition 9).

Best response dynamic

Best Response Dynamic also called Iterated Best Response (IBR) is an incomplete method to find a Nash equilibrium, it is analogous to local search in Constraint programming (see Section 2.3). This algorithm is quite simple (see Algorithm 3), and is very efficient in practice to find a PNE if it exists. The idea of this algorithm is to first take a random configuration of players' strategies

and then to make randomly deviates a player to its best response until a Nash Equilibrium is reached.

Algorithm 3 IBR

```
1: procedure IBR(game : G = (P, A, u))
2:     Select a random configuration S
3:     while not isNash(S) do
4:         S = makeDeviateAPlayerToItsBestResponse(P)
5:     end while
6: end procedure
```

Pareto Nash equilibrium

In a game even if a Nash equilibrium is found, it is not always a desirable state for the players. Concepts on the top of Nash equilibrium have been studied. For instance Pareto efficiency (see definition 9), or Pareto optimality, which is a state of resources allocation in which it is impossible to make any one individual better off without making at least one individual worse. That is, a Pareto Optimal outcome cannot be improved upon without damaging at least one player. Pareto optimality has been widely used as the most fundamental solution concept for multi-objective optimization problems and has also been applied for games [202].

Definition 10 (Pareto Nash Equilibrium [78]) *A Pure Nash Equilibrium is a Pareto Nash Equilibrium if there does not exist a PNE s' such that $\forall i \in \mathcal{P}, u_i(s') > u_i(s)$*

A Pareto Nash Equilibrium is simply a multi-objective optimization problem on the top of the Nash equilibrium. In other words, the Pareto efficiency is applied to the players' objectives at Nash equilibrium. Obviously, if it exists a Nash Equilibrium then a Pareto Nash exists too. An example of Pareto Nash equilibrium is given in Example 16. In games it filters the equilibriums which are dominated according to the Pareto criterion.

Strong Nash equilibrium

A Strong Nash equilibrium (SNE) is a special case of Nash equilibrium. A strategy profile is a SNE means no coalition (including the grand coalition, i.e., all players collectively) can profitably deviate from the prescribed profile. This immediately implies that any SNE is both Pareto efficient and a Nash equilibrium. Also, it is stable with regard to the deviation of any coalition. Note that a SNE is also Pareto Nash. An example of a Strong Nash equilibrium is given in Example 16.

Example 16 (Pareto Nash and Strong Nash equilibrium) *Consider the toy game in the Normal form, depicted in Figure 3.7. This example shows two games and aims to underline the differences between Strong Nash Equilibrium and Pareto Nash Equilibrium. Both situations have 3 PNEs which are on the diagonal. In the first matrix (see Figure 3.7a) a SNE is reached when both players are playing s_1. This SNE is also a Pareto Nash since it corresponds to the best possible choice of the players among all possible combinations. The other equilibriums are only PNEs and not Pareto PNE because the SNE dominates the other equilibriums.*

In the second matrix (see Figure 3.7a) the utility of the SNE has changed. The 3 PNEs still exist but not the SNE anymore. At the same time, two Pareto-PNEs (colored in orange) exist and are reached when both players play either simultaneously s_1 or s_2.

		P_1		
		$s_1(P_1)$	$s_2(P_1)$	$s_3(P_1)$
	$s_1(P_2)$	$(3,3)$	$(0,0)$	$(0,0)$
P_2	$s_2(P_2)$	$(0,0)$	$(1,3)$	$(0,0)$
	$s_3(P_2)$	$(0,0)$	$(0,0)$	$(0,0)$

(a) Strong Nash equilibrium example

		P_1		
		$s_1(P_1)$	$s_2(P_1)$	$s_3(P_1)$
	$s_1(P_2)$	$(3,1)$	$(0,0)$	$(0,0)$
P_2	$s_2(P_2)$	$(0,0)$	$(1,3)$	$(0,0)$
	$s_3(P_2)$	$(0,0)$	$(0,0)$	$(0,0)$

(b) Pareto Nash equilibrium

FIGURE 3.7 – A toy example showing a SNE and Pareto efficiency

Nash equilibrium relaxations

In general cases, finding a Pure Nash equilibrium is not an easy task. That why people try to find relaxations to make the problem more tractable or to ensure that the problem has a solution.

Mixed Nash equilibrium

In a PNE, players choose a unique strategy. This choice is sometimes a bit restrictive and especially because it can makes games solutionless. Nonetheless, PNEs are good since they force players to play in a deterministic way which is easier to take into consideration to build real applications. A Mixed Nash Equilibrium is less restrictive than a PNE it allows to put a probability distribution over the

set of strategies for each player. The relaxation is done over the choices players' strategies. A player can select multiple strategies if it improves its expected payoff.

A wonderful result which was found by Nash [146] is showing that in any game it exists a Mixed Nash equilibrium. Latter, it was shown that finding a Nash equilibrium belongs to *PPDA* complexity class [50] and even if the game has only 2 players. PPDA complexity means that in it always exist a solution but might be hard to find.

Definition 11 (expected utility) *The expected utility of player i under the mixed strategy profile σ, denoted by $u_i(\sigma)$, is*
$u_i(\sigma) = \Pi_{s \in S}\ u_i(s)\ \Sigma_{j \in \mathcal{P}}\ \sigma_j(s_j)$ *where $\sigma_j(s_j)$ denotes the probability that j plays s_j*

Definition 12 (Mixed Nash Equilibrium) *A mixed strategy profile σ is a Mixed Nash Equilibrium if $\forall i \in \mathcal{P}$, $\sigma_i \in argmax_{\sigma_i} u_i(\sigma_i, \sigma_{\text{-}i})$, where $\sigma_{\text{-}i}$ is a tuple of mixed strategy of the other players, except i.*

Example 17 (Battle of the sexes continued) *In this game, there are 2 PNEs which are reached when both players are going to the same place, which can be either Opera or Soccer. Let's check if it exists another mixed Nash equilibrium. This method relies on a simple linear algebra. To do so, we are going to solve a*

	q	1-q
p	$(3, 2)$	$(0, 0)$
1-p	$(0, 0)$	$(2, 3)$

Figure 3.8 – Mixed Nash equilibrium computation

linear system. The unknown part of the problem is going to be the probability distribution over each strategy. Suppose that column's mixed strategy assigns probability weight q to O and (1-q) to S. The figure 3.8 shows how the distribution of probability is done. Then :

$$\textit{row's expected payoff from } M \textit{ against } (1, 1-q) = q \times 3 + (1-q) \times 0 = 3q \quad (3.1)$$

$$\textit{row's expected payoff from } W \textit{ against } (1, 1-q) = q \times 0 + (1-q) \times 2 = 2 - 2q \quad (3.2)$$

These expected payoff have to be equal, we must have $3q = 2 - 2q$ *or* $q = \frac{2}{5}$. *The probability distribution over the second player can be obtained in the same way.*

To summarize, if now a Mixed Nash equilibrium is sought, then the probability distribution over the row is $(\frac{2}{5}, \frac{3}{5})$.

Best responses relaxation. A way to relax Nash equilibrium is to weaken the best response concept. This relaxation is also known as $\epsilon - Nash$ and falls in the category of *approximate Nash equilibrium*[32]. It corresponds to a strategy profile that approximately satisfies the condition of Nash equilibrium.

Definition 13 (ϵ-approximate Nash equilibrium) *A strategy profile* $(x_1, ..., x_n$ *is an* ϵ*-approximate if for every player* i *and every action* a *in the support of* x_i, $E[u_i(a, x_i)] \geq E[u_i(b, x_i)] - \epsilon$ *for any action* $b \in A_i$

In other words, the concept of best response is relaxed by enlarging the zone of players' satisfaction. Many researches have been done to find good algorithms for this approximations schema using different techniques [8, 97, 45]. The potential of such method an if the approximation scheme was possible was also studied [133, 51, 195, 205].

Game relaxations Recently another approach was proposed, this one consists into relax the Nash equilibrium concept by weakening the number of satisfied players. The authors call it a *local equilibrium* because it consists into maximizing the number of satisfied players [87] in a game.

Quantifying the efficiency of equilibrium

Some equilibriums are more desirable than others. In many cases, the efficiency of a solution can be evaluated by an external measure called *social welfare* function which should be optimized. This global function is used to compute the best-centralized solution (by discarding the players' objectives). Afterwards, it is possible to quantify the loss of efficiency induced by the selfish behavior of the players. These measures are computed by considering the ratio "best-centralized solution / best equilibrium" for the Price of Stability (PoS) and "best-centralized solution / worst equilibrium" for the Price of Anarchy (PoA). The best equilibrium is the one giving the highest value for the social welfare function among all equilibriums.

$$PoA = \frac{min_{s \in equilibriums} Welfare(s)}{max_{s \in S} Welfare(s)}$$

$$PoS = \frac{max_{s \in equilibriums} Welfare(s)}{max_{s \in S} Welfare(s)}$$

Where s is a state, S the set of all possible states and $welfare(s)$ corresponds to the social welfare function associated to a state s.

Example 18 (Computation of PoA and PoS) *The toy example showed in Figure 3.7aa is considered again. This example has 3 PNEs and 1 SNE. Suppose now that the social welfare function is simply the sum of all the players' utility over plus one. As recall, the worse equilibrium regarding to the social welfare function is the oen obtained when the players* 1 *and* 2 *play both the strategy* s_2*. In comparison, according to the social welfare function the best centralized solution which is also an equilibrium is reached when the players play all* s_1*. Then the PoA is equal to* $\frac{1+0+0}{1+3+3} = \frac{1}{7}$ *and the PoS is equals to* $\frac{1+3+3}{1+3+3} = 1$*.*

3.4 Games example

This section provides few examples of classical games. Most of them are part of the Gamut library [152]. When a game comes from another source, the original publications are given as reference. We separate the description of classical games and the graphical ones. Note that a list of games can be found on Wikipedia [1]

Classical games

— **Minimum Effort Game (MEG) :** is a coordination game which demonstrates the coordination with multiple equilibrium. The equilibrium will be reached if all players choose the same strategy. In this game, given an identical strategy set A for each player, his payoff is determined by the formula $a + b \times M - c \times E$ where E is the player's effort and M is the minimum effort of all the players. a,b, c are the parameters of the game.

— **Traveler Dilemma (TD) :** An airline loses n suitcases belonging to n different travelers. All the suitcases are identical and contain the same items. All the travelers are told to claim the value of their suitcase between 2 and 100 (They can not discuss each other). If any travelers write down the lowest value, he will get an extra value $n, and the remaining travelers will get an minus value $n. All travelers would like to maximize the value they would be reimbursed by the airline.

— **Collaboration Game (CG) :** is a game where players have to collaborate to get the highest utility. Players have to chose among a set of actions. The highest payoffs are for all outcomes in which every player chooses the same action. PNEs are the situations where the players choose the same strategy.

— **Dispersion Game (DG) :** is a game where players do not want to collaborate and is the opposite of the Collaboration game. Players have to

1. https ://en.wikipedia.org/wiki/List_of_games_in_game_theory

chose among a set of actions. In this game the players want as much as possible to not share a strategy. The highest payoffs are for all outcomes in which every player chooses a distinct strategy/

— **Arm Race (AR) :** is a common game in economics literature. An arms race, in its original usage, is a competition between two or more states to have the best armed forces. Each party competes to produce more weapons, larger military, superior military technology. In this game Payoffs in this game are symmetric and calculated by using the formula -$C(x) + B(x\text{-}y)$ where x is the level of arms the player in question has chosen, y is the level of arms his opponent has chosen, and C and B are user-specified functions.

— **ElFarol Bar Game (EFBG)[4] :** is a kind of coordination game. In this game, there is a particular, finite population of people. Every Thursday night, all of these people want to go to the El Farol Bar. However, the El Farol is quite small, and it's no fun to go there if it's too crowded. So much so, in fact, that the preferences of the population can be described as follows with k an integer with a possible range between 1 and 100 :

  — If less than k% of the population go to the bar, they'll all have a better time than if they stayed at home

  — If more than k% of the population go to the bar, they'll all have a worse time than if they stayed at home.

  It is necessary for everyone to decide at the same time whether they will go to the bar or not.

— **Colonel Blotto (CB)[188] :** is a type of two-person zero-sum game in which the players are tasked to simultaneously distribute limited resources over several objects (or battlefields). In the classic version of the game, the player devoting the most resources to a battlefield wins that battlefield, and the gain (or payoff) is then equal to the total number of battlefields won.

Graphical games :

— **Public Good Game(PGG)[112] :**This game is useful to analyze the behavior of players when they can share a good. For instance, the action might be learning how to do something, where that information is readily communicated ; or buying a book or other product that is easily lent from one player to another. Taking action 1 is costly, and a player would prefer that a neighbor take the action rather than having to do it himself or herself ; but taking the action and paying the cost is better than having nobody take the action.

— **Threshold Game of Complement (TGC)[112] :**This game is a bit in opposition with the PGG. In some situations, a player has an increasing incentive to take a given action as more neighbors take the action. In particular, consider situations in which the benefit to a player from taking

action 1 compared to action 0 (weakly) increases with the number of neighbors who choose action 1.

— **Road Game (RG)[215] :** A road is being built from north to south through undeveloped land. n agents have purchased plots of land along the road. As the road reaches each agent's plot, the agent needs to choose what to build on his land. His utility depends on what he builds, on some private information about the suitability of his land for various purposes, and on what is built north, south, and across the road from his land. The agent can observe what has already been built immediately to the north of his land (on both sides of the road), but he cannot observe further north ; nor can he observe what will be built across from his land or south of it. Then he has to chose what to build in order to maximize his utility

Deuxième partie

Contributions

Chapitre 4

Constraint games

Contents

4.1 Constraint Games framework **45**
Constraint games examples 49
4.2 Variable control **50**
Uncontrolled variables 50
Shared ownership of variables 54
4.3 Hard constraints **57**
4.4 Conclusion **60**

This chapter presents the Constraint Games framework and gives related extensions. The extensions such as the hard constraints, the shared variables and the uncontrolled variables are useful to provide a higher power of expressivity in Constraint games. The new concepts provided concern mainly modelling part of games. Especially, we study the impact on equilibriums of these new concepts and how to compute it.

4.1 Constraint Games framework

Constraint Games [149, 148] are a way to give games a compact representation by using Constraint Programming to represent utility functions.

Definition 14 (Constraint Satisfaction Game)
A Constraint Satisfaction Game *(or CSG) is a 4-tuple* $(\mathcal{P}, V, D, G)$ *where* $\mathcal{P}$ *is a finite set of players,* V *is a finite set of variables composed of a family of disjoint non empty sets* $(V_i)_{i\in\mathcal{P}}$ *for each player and a set* V_E *of* existential *variables disjoint*

of all the players variables, D are the domains' variable defined as for CSP, and $G = (G_i)_{i \in \mathcal{P}}$ *is a family of CSP on* V *representing the* goal *of each player.*

In Constraint Games, the set of players' actions are represented by the joint values of the variables he controls, and its utility by a function valuing players' actions.

Definition 15 (Strategy) *A strategy* s_i *for player* i *is an assignment of the variables* V_i *which he is controlling.*

Definition 16 (Strategy profile) *A strategy profile* $s = (s_i)_{i \in \mathcal{P}}$ *is the given of a strategy for each player*

In constraint satisfaction games, players' satisfactions are expressed by Boolean utility. In other words, the preferences are either "yes" or "no". The Boolean utility function of player i over a strategy profile s is set to 1 if s satisfies the goal of i and to 0, otherwise.

Definition 17 (CSG's utility) *Let* $u_i(s)$ *be the utility function of player* i *over* s, $u_i(s) = 1 \leftrightarrow s \in sol(G_i)$ *and* $u_i(s) = 0 \leftrightarrow s \notin sol(G_i)$.

Note that [149] has introduced satisfaction and optimization variants of Constraint Games. A Constraint Optimization Game (COG) is a variant $(\mathcal{P}, V, D, G, opt)$ where $opt = (opt_i)_{i \in \mathcal{P}}$ and $\forall i \in \mathcal{P}, opt_i \in V$ is the variable whose value defines the utility function u_i of Player i. In a constraint optimization game, all players want to maximize their utility.

Definition 18 (COG's utility) *Let* $u_i(s)$ *be the utility function of player* i *over* s, *then* $u_i(s) = -\infty \leftrightarrow s \notin sol(G_i)$ *and* $u_i(s) = s|x_i \leftrightarrow s \in sol(G_i)$ *where* $opt_i = max(x_i)$.

In other words, when a profile s satisfy the goal G_i of Player i, the utility u_i of this player is given by the value of the variable to be maximized : $u_i(s) = opt_i$. By changing the definition of utility for COGs, the others notions remain the same as for CSGs

Example 19 (Location game) *This game comes from [148] and is an adaptation from [106]. A group of* n *ice cream vendors would like to choose a location numbered from 1 to* m *for their stand in a street. Each vendor* i *wants to find a location* l_i. *He already has fixed the price of his ice cream to* p_i *and we assume there is a customer's house at each location. The customers choose their vendor by minimizing the sum of the distance between their house and the vendor plus the price of the ice cream. Note that if two vendors have the same final price then a customer is going to choose the first one by lexicographic order. This game can be modeled by the following constraint game model :*

A set of players :

$$\mathcal{P} = \{1, .., n\}$$

For each player i, the location of his stand is determined by a variable l_i with a domain the $\{1, ..., m\}$:

$$\forall i \in \mathcal{P}, V_i = \{l_i\}, D(l_i) = \{1, ..., m\}$$

A variable $cost_{ij}$ determines the cost for the customer j to come to the vendor i. The cost is computed as the addition between the ice cream's price and the cost to come to the vendor :

$$\forall c \in \{1, .., m\}, \forall i \in \mathcal{P},\ cost_{ij} = p_i + dist(l_i, m),$$

Each customer selects the vendor with the minimal cost. An indicator boolean variable min_c, for all customer c determines the minimal cost giving all vendors. The first constraint specifies which cost is minimal, while the second ensures that the indicator variable is set to 1 if the cost is minimal.

$$\forall c \in \{1, .., m\},\ min_c = min(cost_{1c}, ..., cost_{nc})$$
$$\forall c \in \{1, .., m\},\ min_c = min(cost_{ic}) \leftarrow (choice_{ic} = 1)$$

The customers can go only to one vendor :

$$\forall c \in \{1, .., m\},\ \sum_{i \in \mathcal{P}} choice_{ic} = 1$$

The goal of each vendor is to maximize their utility which is computed as their price times the number of customers coming :

$$\forall i \in \mathcal{P}, G_i = max(p_i \times \sum_{c=1}^{m} choice_{ic})$$

We denote by *s-i* the projection of s on $V_{-i} = V \backslash V_i$. When a player can improve his satisfaction by changing the assignment of the variables he can control, then he has a beneficial deviation.

Definition 19 (Beneficial deviation) *Given a strategy profile s, a player i has a beneficial deviation if $\exists s'_i \in S_i$ such that $u_i(s'_i, s_{-i}) > u_i(s_, s_{-i})$*

A tuple s is a best response for player i if this player is not able to make any beneficial deviation. In other words, it corresponds to the tuples which give the the maximum value to $sol(G_i)$.

Definition 20 (Best response) *A strategy profile $s(s_i, s_{-i})$ is a best response (abbreviated BR) for player i if and only if $\forall s'_i, u_i(s_i, s_{-i}) \geq u_i(s'_i, s_{-i})$*

The main solution concept defined for Constraint Games is pure Nash equilibrium.

Definition 21 (Nash equilibrium) *A strategy profile s is a Pure Nash Equilibrium (abbreviated PNE) of the Constraint games if and only if no player has a beneficial deviation, i.e. s is a best response of all players.*

Example 20 (Example 19 continued) *In the Figure 4.1, an instance of Location Game with 3 sellers and 14 customers is shown. The sellers are represented by the triangles, and their price are shown inside it. One customer is located at each position. Each green arrow represents a customer choice.*

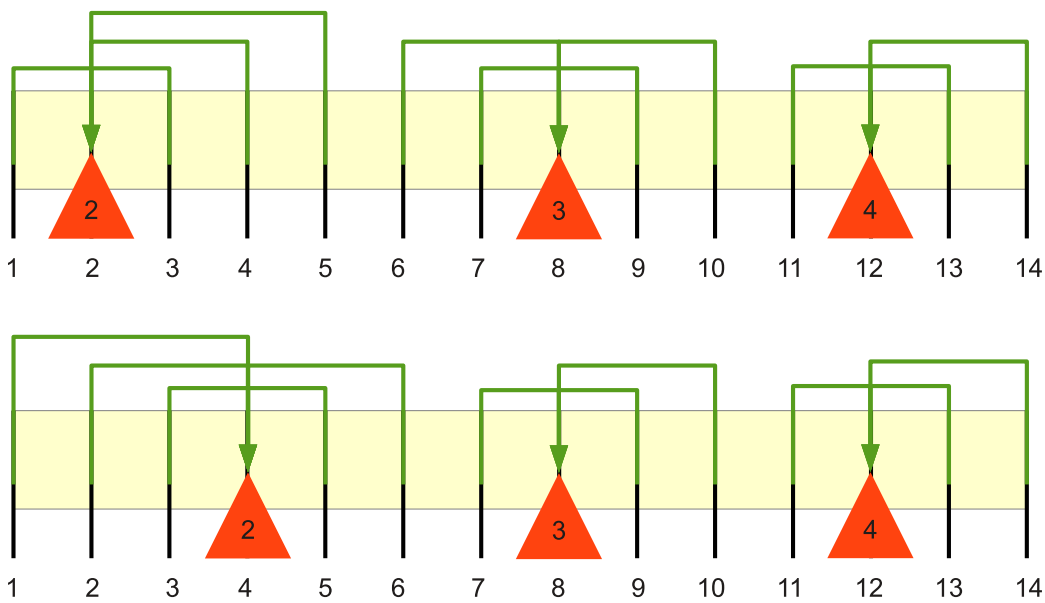


Figure 4.1 – Players' deviations in the location game

In the top figure, the most left player has a deviation. He can shift of 2 positions on the right in order to get a new customer, as shown in the bottom picture.

A variable $x \in V_i$ is said to be *controlled* by player i. The meaning of a Constraint Game $(\mathcal{P}, V, D, G, opt)$ is a game $(\mathcal{P}, A, u)$ in which the set of actions of a player i is defined by the different assignments of his controlled variables : $A_i = D^{V_i}$. We denote by $V_C = V \backslash V_E$ the set of controlled variables and by $A_{-i} = D^{V_C \backslash V_i}$ the set of states of all players but i.

Constraint games examples

We give briefly here some examples of games expressed with Constraint games.

Example 21 (Minimum effort game (MEG) [152]) *It is a coordination game which demonstrates the coordination with multiple equilibriums. The equilibriums are reached when all players choose the same strategy. In this game, given an identical strategies set A for each player, his payoff is determined by the formula $a + b \times M - c \times E$ where E is the player's effort and M is the minimum effort of all the players. a, b, c are the parameters of the game.*

The following existential variables are used to model this game by constraint games.

- *M stands for the minimum effort of all players.*
- *$\forall i \in \mathcal{P}, p_i$ stands for the payoff of player i.*

MEG can be modeled as follows :

- $\mathcal{P} = \{1, \ldots, n\}$
- $\forall i \in \mathcal{P}, V_i = \{e_i\}$
- $\forall i \in \mathcal{P}, D(e_i) = A$
- *$\forall i \in \mathcal{P}$, G_i contains the following constraints :*
  - $M = \min(e_1, \ldots, e_n)$
  - $p_i = a + b \times M - c \times e_i$
- *$\forall i \in \mathcal{P}$, the optimization condition $opt_i = \max(p_i)$*

Example 22 (Travelers Dilemma (TD)[152]) *An airline loses n suitcases belonging to n different travelers. All the suitcases are identical and contain the same items. All the travelers are told to claim the value of their suitcase between 2 and 100 (They can not discuss each other). If any travelers write down the lowest value, he will get an extra value $\$n$, and the remaining travelers will get an minus value $\$n$. All travelers would like to maximize the value they would be reimbursed by the airline. TD has only one PNE when all players take the minimal number as their strategy.*

The following existential variables are used to model this game by constraint games.

- *y stands for the minimal value chosen by all the travelers.*
- *$\forall i \in \mathcal{P}, choice_i$ is a boolean variable which is set to 1 if the value of player i is minimal, otherwise, it is set to zero.*
- *$\forall i \in \mathcal{P}, p_i$ stands for the payoff of player i.*

TD can be expressed as follows :

- $\mathcal{P} = \{1, \ldots, n\}$
- $\forall i \in \mathcal{P}, V_i = \{x_i\}$
- $\forall i \in \mathcal{P}, D(x_i) = \{2, \ldots, 100\}$
- *$\forall i \in \mathcal{P}$, G_i contains the following constraints :*
  - $y = \min(x_1, \ldots, x_n)$

  - $\forall i \in \mathcal{P}, choice_i = 1 \Leftrightarrow x_i = y$
  - $choice_i = 1 \rightarrow p_i = x_i + n$
  - $choice_i = 0 \rightarrow p_i = x_i - n$
- $\forall i \in \mathcal{P}$, *the optimization condition* $opt_i = \max(p_i)$

Example 23 (Collaboration Game (CG) [152]) *In a collaboration game, the highest payoffs are for every player who chooses the same action. Each player has to choose an action between 1 and m. Then for each player his rewards corresponds to the number of players having chosen the same strategy as him. CG has as many PNE as player actions*

The following existential variables are used to model this game by constraint games.

- $\forall i \in \mathcal{P}, c_i$ *stands for the action choose by each player.* c_i *is a variable having a domain the set* $\{1, ..., m\}$
- $\forall i, j \in \mathcal{P}, e_{ij}$ *is a boolean variable which takes the value 1 if* $c_i)c_j$ *stands for the action choose by each player.* c_i *is a variable having a domain the set* $\{1, ..., m\}$
- $\forall i \in \mathcal{P}, p_i$ *stands for the payoff of player* i.

CG can be expressed as follows. In this model, the constraint count specifies how many times the value of

- $\forall i, j \in \mathcal{P}$, $e_{ij} = 1 \leftrightarrow c_i = c_j$
- $\forall i \in \mathcal{P}$, $p_i = \sum_{j \in \mathcal{P}} e_{ij}$
- $\forall i \in \mathcal{P}$, *the optimization condition* $opt_i = \max(p_i)$

4.2 Variable control

Uncontrolled variables

So far, the handling of Nash equilibrium in constraint games is related to the precise definition of the *ceteris paribus* principle [20] (i.e. games where players' actions and rewards are clearly defined). However, in full generality, this principle might be violated. In real-world, often the information available is incomplete and or imperfect. This arises, as soon as a state is not well defined, because of for example of unknown parameters in a game. A simple and famous example is the poker Texas Hold'em. In this game, the information is incomplete and imperfect. A player does not know opponents' cards and few others on the table are hidden. Each player knows only his own cards, the number of tokens he has, the amount of token of each other player and the two cards on the table. Given that, each player has to take decisions in order to bet on his possible victory (i.e. the best card) and win the maximum amount of tokens.

The location game depicted in Example 24 is another game with incomplete information. Until now, in the location game, if a customer has the choice between two vendors having the same price, it was settled that he goes to the first one

by lexicographic order. This assumption simplifies the situation and could be unrealistic. Especially, in reality, a customer's choice can be arbitrary and many possibilities can be considered. If the assumption is no longer considered, then, he may go to the vendor having the best ice cream or the nicest storefront. This is associated with customers' belief but is subject to each player's interpretation. As formulated in the game, customers' choices might be unknown from players. The variables associated with the customers' choice remain free and are not instantiated and that even if all players have chosen their strategies. Most of the time, to valuate the utility functions, it is required to know everything about the players' choices and the environment. An unknown could affect the utility function making a player uncertain about customers' choice and thus about his own choice.

Example 24 (Location games with unknown variables' values) *In the location game previously exposed, it is supposed that if two vendors' cost are the same for a customer, then, the latter is going to the first one by lexicographic order. However, in this example, this assumption is not true anymore. The players know the other players strategies but not all the customers choices making potentially dilemma like in the Figure 4.2.*

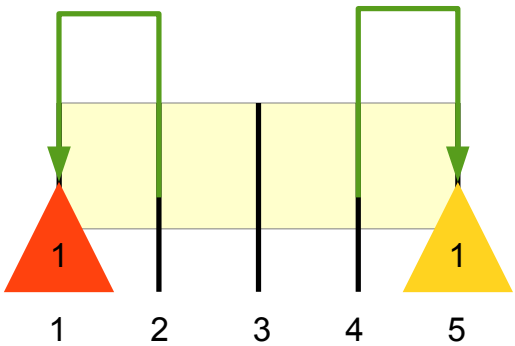


FIGURE 4.2 – Location game with undetermined behavior

In this figure, the two players : the orange and the yellow have to choose where to put their stand. There is a customer at each position and the green arrows correspond to the customers' choice. An example of dilemma is exposed in this Figure : the player orange has the position 1 and the yellow the position 5, then sellers do not know where the customer at the position 3 wish to go. The cost is the same and therefore this criterion is not enough to make a decision. As it is, the players cannot completely compute their best responses. The outcome result has multiples issues and thus is not determined. The customer can either goes to the orange's stand or to the yellow one.

Given a strategy profile, it is possible that it remains non instantiated variables. These variables are called *free variables*. These variable can represent unknown from the nature or hidden players strategies.

In fact, two kinds of free variables exists :

— unknown players' strategies e.g. Texas Hold'em. In other words, a player may not know the variables' values of the others players.

— unknown environment variables e.g. the location game. This situation corresponds to an unknown variable which is not controlled by any player like how the customers behave in the location game.

In constraint games, the unknowns correspond to unknown variables values during the resolution. In other words, even if all the decision variables have been instantiated, it remains some variables which are not. Players have to decide their actions without knowing completely the game's issue. They give a semantic to the variables which are not instantiated. Often, the resolution of such problems relies on prior knowledge like beliefs, if it is available. The beliefs may take different forms like a probability distribution over the unknown variables values or a symbolic behavior. For instance, a player may think that another vendor has better advertising than him or a stand which looks better. Beliefs impact the utility function's computation and thus because it decides how to consider the free variables values which possibly change the objective value.

Symbolic behaviors can be often simulated by computing statistics measures like average, means, min, max on the player's utility function. The possibles statistics are computed on the free variables knowing a strategy profile. For instance, a pessimistic player would take the minimum reward over all the possibles values which are not possibly controlled by himself.

Example 25 (Example 24 continued with beliefs) *While in the previous cases, the game was assumed to be totally defined. It can arise that players cannot decide where a customer goes. Instead, each of them may have beliefs about customers behaviors. For instance, the yellow player can be optimistic, while the orange is pessimistic. Therefore, both are believing that if a customer has a choice, then he would go to the yellow player.*

The Figure 4.3 presents some situations where the beliefs modify the utility of each player. The Figures 4.3a,c and d present how the players are going to interpret the customers' behavior. Each player's belief is symbolized by a colored arrow. The yellow (resp. orange) player's belief on customers behavior is symbolized by a yellow (resp. orange) arrow. In the Figure 4.3a and Figure 4.3c, it is unknown where the customer in position 3 would go. Instead both players believe that the customer is going to the yellow vendor located in position 5. That is why, the orange player deviates (Figure 4.3a) of one position to the left to get another customer (Figure 4.3b).

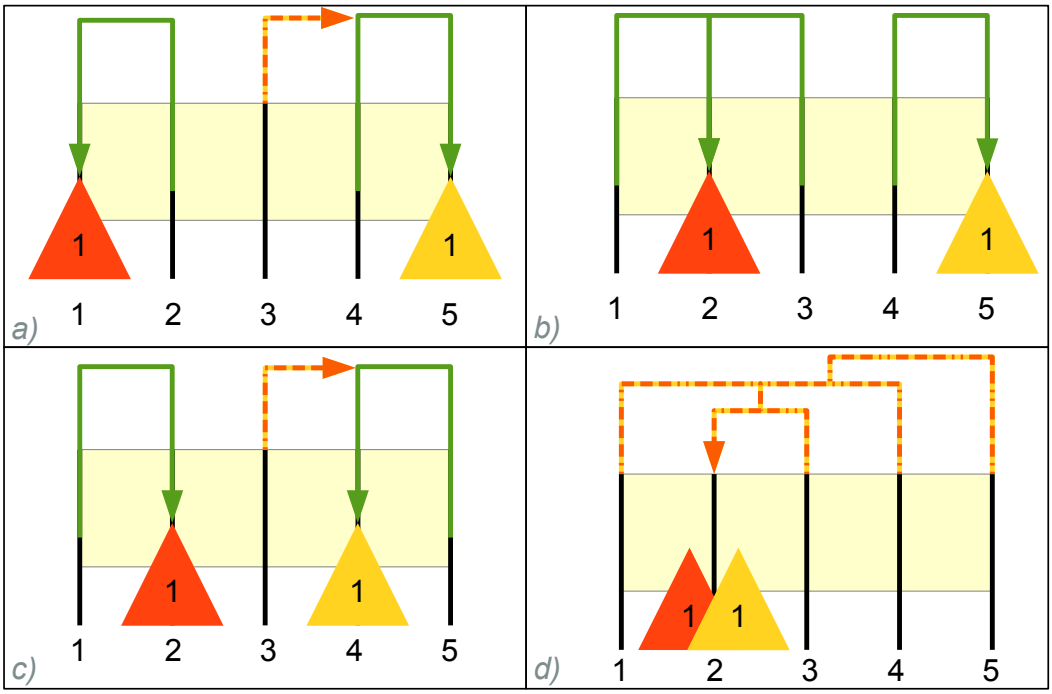


FIGURE 4.3 – Location game with beliefs

In the same way, the yellow player is going to deviate to the same position as the orange player (Figure 4.3c to Figure4.3d). In that case, the yellow player is going be visited by all the customers according to the players' beliefs.

Players' beliefs impact the players' rewards because free variables may change the players' utility making the best responses beliefs dependent, as well as the set of Nash equilibriums.

More generally, a belief can be abstracted by a function f. f takes as input a strategy profile, a player plus the existential variables of a games and output a preference. The f-function is used to evaluate the subspace of the existential variable plus the strategy profile.

A f-beneficial is a beneficial deviation for games with free variables. Note that a f-beneficial deviation without free variable is a beneficial deviation and thus a f-beneficial deviation is a generalization of a beneficial deviation.

Definition 22 (f-beneficial deviation) *Let f be a function such that $f : (s, V_E) \to \mathbb{R}$. Given a strategy profile s, and V_E the set of existential variables, a player i has a f-beneficial deviation if $\exists s'_i \in S_i$, such that $f((s'_i, s_{-i}), V_E) > f((s, s_{-i}), V_E)$*

The other concepts remain the same by changing a deviation by a f-beneficial deviation. A Nash equilibrium with free variables has to respect the concept of

f-beneficial deviation. In other words, a situation is at Nash equilibrium if no player has a f-beneficial deviation.

f-function : The f function is an arbitrary function which can simulate any kind of behavior from a cautious player by a using a *minimum* function to enthusiastic one by using a *maximum* function. Probabilistic behaviors can also be includeed within the f function.

Related work The question about knowledge and uncertainty in games has already been addressed in Boolean games and in different manner. In [80] the authors Especially by the possibility to manipulate games when the information in not complete.

In [1], the authors model uncertainty by extending the framework of Boolean games with a set of observable action variables for every agent (i.e. not all variables are observable). Possibilistic logic has been also considered in [46]. Or more recently by considering incomplete information for negotiation between agents [86, 79, 33].

Shared ownership of variables

Originally in games, players exercise a unique control over a set of variables (or strategies) in order to achieve a unique goal.

It may happen however that the unique control is too restrictive as for example in a modified version of the location game (see Example 26). The motivation in location game to share variable come from the situation in which a shopping mall is built. The stakeholders represented by each vendor in the project have to agree on a location. This location is going to depend on the other stores' location as well as on the stakeholders' preferences. The location's choice is going to be decided by a consensus (i.e. a group of people). Shared control of variables has been firstly proposed in boolean games in [73]. In this article, the authors propose an extension to the CL-PC logic [166] as well as a complete axiomatization of an extension including the shared control of boolean propositions. This concept has been also studied and extended in [13] to iterated boolean games.

Example 26 (Location game with shared variables) *In this example of location game, the vendors can have multiple stores and can store can be shared by multiple players.*

When a store is shared, the benefits are redistributed equally among the owners. Figure 4.4 presents an instance of Location game with one shared store. Two vendors : the orange and the yellow have each one store and share one. The stores are still symbolized by the triangle, and fill by the owners' color. Now a triangle might have multiple colors. This is the case for a shared store like the one in position 7 in the Figure 4.4a.

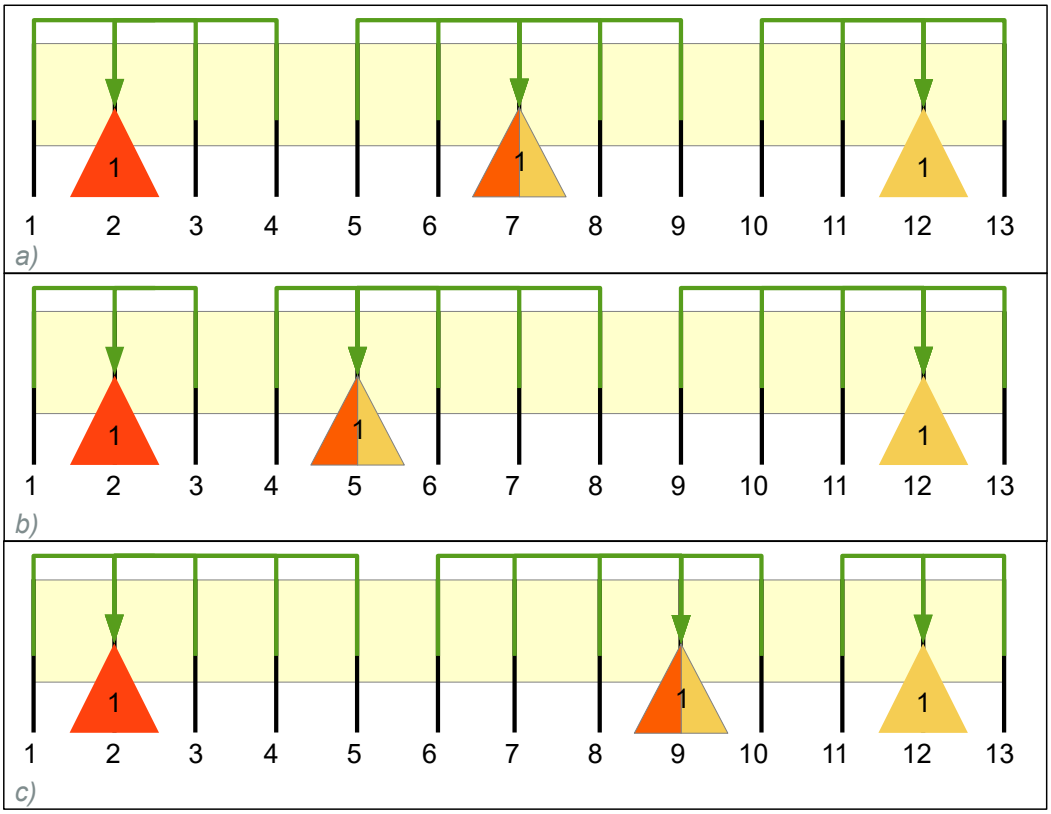


FIGURE 4.4 – Location game with a shared stand

Each player chooses the value of the variables he owns, including the shared variables too. A Nash equilibrium with a shared variable has to respect the preferences of each player. In the Example 26, there is no Nash equilibrium. Each player would like to shift closer to the other player's store in order to increase its utility. For instance, in Figure 4.4a the yellow players would prefer to shift the shared store to the left as shown in Figure 4.4b. Whereas, the orange one prefers to shit on the other side as shown in Figure 4.4c. The players who own a shared variable need to agree on the value taken by the shared variable. That is why the concept of partial cooperation or negotiation [103, 83, 228, 231] is related to the concept of a shared variable. This concept has been mostly studied in DCOP but it is different since the players cannot share a variable and have privacy concern.

Shared variable implementation

Besides, the fact that this concept has been already proposed, we review here the possibles implementation of shared variables and its impossibility to translate into normal form. The concept of shared variable is not easily representable inside a matrix. For instance, suppose the simplest game, where a binary variable is

shared by two players. A matrix representing the possible strategies is presented in Figure 4.5. A single variable which can either take the value symbolized by S_1 or S_2, is depicted here. Each player can either choose S_1 or S_2. However, nothing constraints the players to choose the same strategy at the same time. An implicit constraint exists : the strategy's value of P_1 has to be the same as P_2. This is a hard constraint and cannot be simply added in the normal form (see section 4.3). As it is, the matrix does not respect the semantic of a shared variable, since its value has to be the same for all the players. For instance, players preferences are different : P_1 wishes to play S_1 while P_2 would prefer S_2. In the matrix form, this impossibilities impact the equilibriums and creates inconsistencies as for example by the strategy profile (S_1,S_2) which is inconsistent : the shared variable cannot take two different values.

		P_1	
		S_1	S_2
P_2	S_1	$(0, 15)$	$(0, 0)$
	S_2	$(15, 15)$	$(15, 0)$

Figure 4.5 – One shared binary variable under the normal form.

While the normal form cannot represent shared variables, in constraint games two implementations are considered here : one with duplicated variables and another one where the variables are not duplicated but shared among players.

Model 1 : duplicating the variables. The first simplest model is to duplicate the shared variables to retrieve the case of unique control. Then, to make the agents agree on their choices an equality constraint is added over the duplicated variables to ensure that it does not exist any inconsistency. An inconsistency for a shared variable would be to have multiple values (e.g. the shared store is going to be at two locations at the same time). However, this model is not valid. It does not compute the Nash equilibriums anymore. When the check for players' deviations is computed, the controlled variables which include the shared one need to be uninstantiated in order to compute valid deviations. Instead, during the search procedure, if one of the players has a shared variable instantiated (or a restricted domain), then the equality constraint is going to force the other variables' copy to take the same range of values. Therefore, while seeking the best responses, the search space is constrained by this equality constraint and not valid anymore because incomplete. The player has to check his deviation is forced to have the same value for the shared variables and cannot compute properly his deviations. For instance, let's take the situation depicted in Figure 4.4a. The orange player wants to check his possible deviation knowing the choices of the yellow player. With the current model, the yellow variables values are reported. However, the equality constraint imposes to the shared store to be at

position 7 because of the instantiation of the copy of the variable (own by the yellow). Instead, the orange player wishes to shift the shared store to the right like in the position as in Figure 4.4c. The best responses are then only computed on the non shared variable making the solutions inconsistent. This simple model does not respect game with the shared variable semantic.

Model 2 : sharing the variables. The second model is to relax the assumption of unique control and to share the ownership of variable among multiples players. Now, the shared variables are not duplicated in the model but remain unique. While the players need to check their deviations, the shared variable being controlled by the player is relaxed and part of the subspace of deviations check. The constraint games semantics is kept. For instance, let's take the situation depicted in Figure 4.4b. The orange player wants to check his possible deviation knowing the choices of the yellow player. With the current model, the variables own by the player are relaxed in order to check the deviations. Then the player is free to choose the location of the shared store and thus to compute his best responses correctly.

4.3 Hard constraints

Only fully expressive representations (i.e. which can represent any games) have been presented until here. However, it is sometimes not enough to model a problem. For example in the location game, some physical constraints may restrict the possible players' positions. Two stores cannot be located exactly at the same position.

Modeling these physical constraints in games requires to forbid positions by adding new constraints to the model. *Hard constraints* define situations which are globally not possible or forbidden [189]. It is easy to prove that they provide a large increase in expressivity since it is impossible to find a matrix representation for a game with hard constraints by giving unsatisfiable profiles any numerical value for utility as shown in the Example 4.9.

Example 27 (Hard constraints and normal form) *This example is taken and adapted from [148]. We consider the following CG :*

- *the set of players is* $\mathcal{P} = \{X, Y, Z\}$.
- *each player owns one variable :* $V_X = \{x\}$,$V_Y = \{y\}$ *and* $V_Z = \{z\}$ *with* $D_x = D_z = 0, 1, 2$, $D_y = 0, 1$.
- *one hard constraint is post :* $C = \{alldiff(x, y, z)\}$

In this example, the notation x_i *is used when the variable* x *takes the value* i*. To ease the understanding, the players have not any goal. When a strategy profile is*

reachable then their utility is equal to 1. The strategies which are not possible, an utility of $-\infty$ is given. The normal form associated to this game is as follows :

	y_0, z_0	y_1, z_0	y_0, z_1	y_1, z_1	y_0, z_2	y_1, z_2
x_0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1
x_1	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1	$-\infty$
x_2	$-\infty$	1	1	$-\infty$	$-\infty$	$-\infty$

Figure 4.6 – X's payoff

	x_0, z_0	x_1, z_0	x_2, z_0	x_1, z_0	x_1, z_1	x_1, z_2	x_2, z_0	x_2, z_1	x_2, z_2
y_0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1	$-\infty$	1	$-\infty$
y_1	$-\infty$	$-\infty$	1	$-\infty$	$-\infty$	$-\infty$	1	$-\infty$	$-\infty$

Figure 4.7 – Y's payoff

	x_0, y_0	x_0, y_1	x_1, y_0	x_1, y_1	x_2, y_0	x_2, y_1
z_0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1
z_1	$-\infty$	$-\infty$	$-\infty$	$-\infty$	1	$-\infty$
z_2	$-\infty$	1	1	$-\infty$	$-\infty$	$-\infty$

Figure 4.8 – Z's payoff

Figure 4.9 – Hard constraint simulation in normal form

The only possible strategies are when the players have all a different value for their variables. This happens only for 4 profiles : $(x_0, y_1, z_2), (x_1, y_0, z_2), (x_2, y_0, z_1), (x_2, y_1, z_0)$. *These strategy profiles are the PNEs of the game. However, the hard constraint encoding creates equilibriums which should not exist. For instance when the strategy profile* (x_0, y_0, z_0) *is an equilibrium whereas it should not.*

PNEs and best responses are defined in the satisfiable part of the hard constraints. Hard constraints may change the set of best responses and the set equilibriums (see Example 28). The two spaces : with and without hard constraints can be totally different. With and without hard constraints the best responses are changed.

It is not easy to characterize a priory how the equilibriums are going to be impacted by hard constraints. Some of them may be forbidden while new ones

may appear whether the best responses that would have occurred without hard constraints are actually unsatisfiable.

Example 28 (Example 19 continued : hard constraints' impacts) *In*

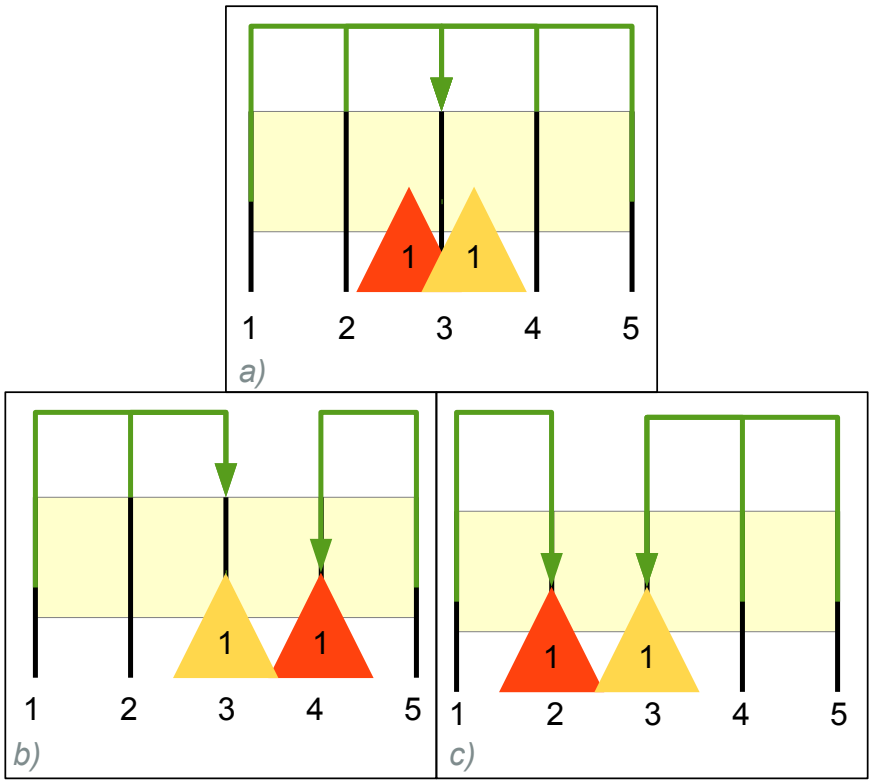


FIGURE 4.10 – Location game's solutions without and with hard constraints

this example, we are considering a simple version of Location game. Instead of have to choose a vendor, the rewards is going to be share among them. The figure 4.10 shows the effects of hard constraints on the Nash equilibrium for the Location Game. In this instance, two vendors : the orange and the yellow choose a position between 1 and 5. A constraint forbidding the two vendors to have the same position has been added. The Figure 4.10a shows how the players would have been at Nash equilibrium without the hard constraints addition. Their best option is to share the location 3. This is going to ensure them a reward of 2.5. In this picture their best responses are computed on the entire space. In other words, the players while computing their possibilities are considering all the positions from 1 to 5. Hence, their highest utility is reached when they are going to choose the position 3.

In the Figures 4.10b and 4.10c, it is for the players forbidden to have the same position. Their best responses space is reduced. For instance in the Figure 4.10b,

the orange player cannot take the position 3. Instead, when he is computing his best responses, then the only possible strategy evaluated are going to be $\{1, 2, 4, 5\}$. *The position 3 is not anymore considered. Therefore, the best responses are changed. The best options for the orange player are now the positions 4 and 2. The equilibriums are this changed with the addition of the hard constraint. While without hard constraint there is only one equilibrium, with the hard constraint this one does not exist anymore and 4 new equilibriums appear. Two are shown in the Figure 4.10b and c. The two other can be obtained by inverting the players.*

4.4 Conclusion

In this chapter we have shown the power of constraint games, their power of modeling. We have shown extensions providing more expressivity to constraint games such as the hard constraints. Also, we have shown that the uncontrolled variables are related to incomplete and imperfect information. In addition, we investigate the concept of shared variables by showing that this concept is not easily representable by a matrix. Instead this concept can easily integrated into the framework thanks to the best response computation mechanism.

Chapitre 5

Nash equilibrium as a Global constraint

Contents

5.1 Preferences as Global Constraints **61**
5.2 Nash propagator applicability **65**
5.3 Related work **66**
5.4 Experiments **67**
5.5 Graphical Aspects **68**
5.6 Conclusion **69**

In this chapter, we present a new way to see the games. This perspective consider a Nash equilibrium as a set of global constraints embedding players' preferences. An efficient method to compute the Nash equilibrium and the characterization of its complexity is also proposed.

5.1 Preferences as Global Constraints

Although more efficient than the *Enum* algorithm used in Gambit, the *Conga 1.0* algorithm can be still improved by using constraint propagation instead of an ad-hoc pruning algorithm. We propose a new solver for Constraint Games also based on a tree search where a player can own multiple variables which are instantiated separately by a regular Constraint Programming solver. Note that there is a main search tree which defines the region of the search space where an equilibrium is sought. Like in Constraint Programming, a search state S is defined by giving each variable a current domain : $S = (S_x)_{x \in V}$ with $S_x \subseteq D_x$.

w	c	l
0	0	0
0	0	1
0	1	1
1	0	0
1	1	0

(a) Nash Constraint N_{Lamb}

w	c	l	utility
0	0	0	0
0	0	1	0
0	1	1	1
1	0	0	1
1	1	0	1

(b) Ext. Nash constraint eN_{Lamb}

Figure 5.1 – Nash constraints for Lamb in extension

We use the notations $S_i = S|_{V_i}$ and $S_{-i} = S|_{V_C \setminus V_i}$. Search consists in a series of domain reductions with an alternation of deterministic consistency steps and non-deterministic branching steps [190].

We propose to implement as constraints the preferences of the players. For this, it is useful to consider how preferences are defined from the goal of the players. Preferences have been widely studied in the literature from the point of view of knowledge representation, especially using logic [218, 20]. Generally, it is widely accepted that a preference is a preorder $\succcurlyeq$ on a set of outcomes Ω. It does not have to be total. We have $s \succcurlyeq s'$ whenever s is preferred to s'. A utility function $\Omega \to \mathbb{R}$ as defined in games defines a natural preference where $s \succcurlyeq s' \leftrightarrow u(s) \geq u(s')$. But games also add a notion of controllability : an outcome can only be compared to a controllable one, thus making the preorder partial.

For a Constraint Game $(\mathcal{P}, V, D, G, opt)$, the set of outcomes is defined by the search space D^V, and the preference associated to a player is induced by the utility given by the value of his optimization variable. Since Player i only controls the variables V_i, we can fully describe his preference relation by giving the preferred outcomes for each uncontrollable situation defined by the other players, i.e. the set of best responses in A_i of Player i associated to each partial state of A_{-i}. This relation has been introduced in [78] in the context of graphical games under the name of *Nash constraint*. The Nash constraint N_i of Player i is defined by $N_i = (V_C, \{(s_i, s_{-i}) \mid s_{-i} \in A_{-i} \wedge \forall s'_i \in A_i, u((s_i, s_{-i})) \geq u((s'_i, s_{-i}))\})$. Theorem 4.3 of [78] adapted to Constraint Games states that $\mathbb{NE} = sol(\bigcup\{N_i \mid i \in \mathcal{P}\})$.

We propose to implement players preferences using global constraints and study filtering algorithms for them. The Nash constraint N_i contains at least one entry for each partial profile in A_{-i}. This property makes it unsuitable for filtering values from the other players' variables. Moreover, its representation in extension is exponential.

Example 29 (Example 11 continued)
The Nash constraint for Lamb is defined in Figure 5.1(a). Note that if Wolf and Cabbage are not coming, Lamb is indifferent and thus has both coming *and* not coming *as best responses.*

Since the Nash constraints N_i contains all best responses of Player i for all the uncontrollable situations defined by the other players joint strategies, any strategy for i that does not belong to the projection $N_i|_{V_i}$ is called a *never best response*. We first state an intractability result :

Proposition 1 *In a Constraint Satisfaction Game, deciding whether an assignment $nbr \in A_i$ for Player i is a never best response is Π_2^P-complete.*

Proof 1 *Membership is immediate.*

For hardness, we reduce a QCSP [26] $Q = \forall X \exists Y C$ to a 2-players 0-sum CSG $G = (\{1,2\}, V_1 = X \cup \{x,a\}, V_2 = Y \cup \{y,b\}, G = (a = b) \vee (C \wedge (Y \neq nbr) \wedge (x = y))$ where $Y \neq nbr$ stands for the assignment of Y is different of the tuple nbr and x, y, a, b are new variables whose domain contains at least two elements. Since the game is 0-sum, we only need to specify the goal G for Player 1 and take Player 2's goal as the negation of G.

If Q is valid, then for all $s^X \in D^X$, there is a $s^Y \in D^Y$ such that C is true. Let v_x and v_a be the respective values of x and a set by Player 1. If $s^Y = nbr$, then Player 2 assigns $b \neq v_a$. If $s^Y \neq nbr$, then Player 2 assigns $y = v_x$. Hence nbr is a never best response.

Conversely, if Q is not valid, then it does exist a tuple $s^X \in D^X$ such that forall $s^Y \in D^Y$, C is false. If $s^Y = nbr$, then Player 2 assigns $b = v_a$. If $s^Y \neq nbr$, then Player 2 assigns $b \neq v_a$. Hence nbr is a best response.

Arc-consistency filtering amounts to removing all values involved only in tuples which are never best responses. From Proposition 1 and the exponential representation argument, we infer that filtering the Nash constraint to arc-consistency is intractable even for dichotomic preferences. Thus we introduce three approximations in order to keep the problem tractable. We recall that a propagator for a constraint is a function that is *i)* correct, *ii)* contracting, *iii)* monotonic, and *iv)* singleton complete [3]. The respect of these properties ensures the correct behavior of the propagator when placed in the solver.

The first approximation consists in using only the objective value. We call *extended Nash Constraint* $eN_i = (V_C \cup \{opt_i\}, \{(s, o_i) \mid s \in N_i\})$ the Nash constraint N_i augmented with the value of the player's objective (see Figure 5.1(b)). Instead of removing exact inconsistent values, we will remove inconsistent objective values. The actual pruning on decision variables is done by back-propagation of arc-consistency through the constraints defining the objective. It means that we can approximate eN_i by ooN_i (objective optimal Nash constraint) defined by $ooN_i = (V_C \cup \{opt_i\}, \{(s, o_i) \mid \exists (s', o_i) \in eN_i \wedge o_i \in eN_i|_{opt_i}\})$. However, only looking at the objective value is not enough. In different regions of the search space, two tuples s and s' may have the same objective value o_i for Player i despite s is a not a best response for s_{-i} and s' is one for s'_{-i}, hence the approximation.

Still, computing the exact objective values can be difficult because the subspace generated by the variables V_i of Player i may be huge. In our second approximation,

we replace the set of exact objective values by the interval where they lie. This is defined by the constraint $mooN_i = (V_C \cup \{opt_i\}, \{(s, o_i) \mid \exists (s', o'_i) \in eN_i, o_i \geq o'_i\})$. Since $sol(eN_i) \subseteq sol(ooN_i) \subseteq sol(mooN_i)$, any filtering of $mooN_i$ is correct with respect to eN_i. Note that we only need to update the lower bound because we have a maximization problem.

Since consistent objective value correspond to one best response, the minimum bound of the objective should be set to the maximum value of the minimum bound of the best responses on the current search state S :

$$Mm(S) = \max_{s_i \in A_i} \min_{s_{-i} \in S_{-i}} (s_i, s_{-i})|_{opt_i} \qquad (1)$$

Thus we need to find (or approximate) the *maximin* of the objective on the current subspace. The exact computation of this maximin on a search state S requires a traversal of the subspace S_{-i}, and for each tuple, a test for deviation in A_i. Unfortunately, a traversal of S_{-i} is too costly. The third approximation consists in using arc-consistency to reject impossible objective values.

To implement this filtering, we use an auxiliary solver with Branch & Bound on a tree-search limited to Player i's variables to maximize the minimum value of the objective on the subspace at a node n of the main search tree. In order to compute deviations for Player i, all variables of V_i are reset to their original domain A_i, *including* those which were assigned before n. The domain of the variables of the other players are given by S_{-i} (some are already assigned at node n and some are not). Whenever the minimum bound of the objective is pruned to a value b, a new constraint $opt_i > b$ is posted for the rest of the search. For a search state S, we replace in formula (1) the traversal of the search space S_{-i} by an arc-consistency check. This amounts to compute $Mm_{AC}(S)$, the best estimation arc-consistency can provide for the maximum value of the minimum bound of the objective : $Mm_{AC}(S) = \max_{s_i \in A_i} \; lb(AC((s_i, S_{-i})))$ where AC denotes arc-consistency applied to a search state. Formally, the operator $BB_i : D^V \to D^V$ implemented by the global constraint is defined by :

$$BB_i(S) = \begin{cases} S_{opt_i} & = & S_{opt_i} \cap [Mm_{AC}(S)..ub(S_{opt_i})] \\ S_x & = & S_x, \forall x \neq opt_i \end{cases}$$

When all variables are assigned, computing deviations amounts to checking whether a complete assignment is a best response for Player i.

Proposition 2 *BB is a propagator for eN_i.*

Proof 2 *Let S and S' be two search states. First, because AC is correct, we have $Mm_{AC}(S) \leq Mm(S)$. Then, if some value is pruned by Mm_{AC}, then it has to be pruned also by Mm.*

— *correctness : $S \cap \mathbb{NE} \subseteq BB(S)$. Take $s \in S \cap \mathbb{NE}$ and suppose $s \notin BB(S)$. Let $s|_{opt_i} = o_i$ and let $\alpha_i = Mm_{AC}(S)$. Since $s \notin BB(S)$, we have $o_i < \alpha_i$.*

Then there exists $s'_i \in A_i$ such that $(s'_i, s_{-i})|_{opt_i} = o'_i > o_i$. Hence we have $o_i < \alpha_i \leq o'i$ and (s'_i, s_{-i}) is a deviation for s and $s \notin \mathbb{NE}$, contradiction.

— *contractness : $BB(S) \subseteq S$. Since the propagator only remove values, contractness is always true.*

— *monotony : $S \subseteq S' \rightarrow BB(S) \subseteq BB(S')$. Take $s \in BB(S)$, we have $s \in S$ since BB is contracting and $s \in S'$ by definition. Let $s|_{opt_i} = o_i$, $\alpha_i = Mm_{AC}(S)$ and $\beta_i = Mm_{AC}(S')$. Since $s \in BB(S)$, we have $\alpha_i \leq o_i$. Since AC is monotonic, we have $\beta_i \leq \alpha_i$. Thus $\beta_i \leq o_i$ and $s \in BB(S')$.*

— *singleton completeness : ensured by construction.*

Example 30 (Example 11 continued)
Interestingly, the WLC game owns a dynamic dependency : when Wolf is coming, Lamb does not depend on Cabbage anymore. It is reflected in the Lamb formula $x_L = \overline{w}lc + w\bar{l}$ for which $\bar{l}$ is a best response to w whatever the value of c. By checking earlier the possible deviation, our propagator is able to find this kind of dependencies in order to prune the search space.

5.2 Nash propagator applicability

We discuss here about the Nash constraint. Especially about its filtering algorithm completeness and validity. It has been said, that hard constraints modify the search space. Especially, Nash equilibriums can be moved, created, or even removed (see section 4.3).

Actually, when a game contains hard constraints, those make in general the Nash constraint no longer valid. The inference made by the Nash constraint, it is made by supposing that the other players cannot affect a player strategy (i.e. making it impossible).

The outputted solutions are still Nash equilibrium but the completeness is not anymore guaranteed.

Example 31 (Nash propagator invalidity) *This example shows a game where the Nash constraint does not guarantee anymore the completeness of the search. A Nash equilibrium is removed by the constraints. Two players X and Y respectively own the variables x and y which can either take values 0 or 1. The goal of player X is to maximize $3X + Y$ while Y's goal is to maximize $Y - X$. A hard constraint forbids the players to chose the same value for their variables. The game is expressed with the following constraints and variables :*

— $\mathcal{P} = \{X, Y\}$

— $D(x) = D(y) = \{0, 1\}$

— $G_x = Max(3X + Y)$

— $G_y = Max(Y - X)$

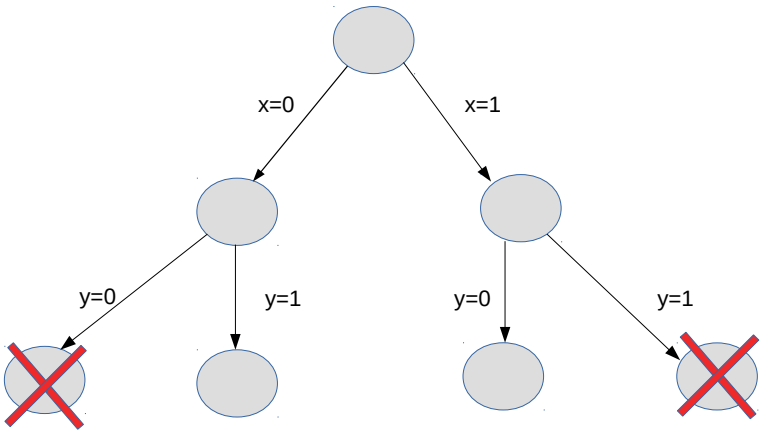


Figure 5.2 – An example of inconsistent filtering of preferences

— *sb* $X \neq Y$

The entire search tree is developed in Figure 5.2. When a node is crossed, it means that it is forbidden by a hard constraint (x=0 ∧ y=0 or x=1 ∧ y=1). The problem here is the initial propagation deletes a Nash equilibrium. When X tries different configuration for its variable then he realizes that he has an incentive to always assign 1 to x (when he plays 1, he has at least 3 while by playing 0 the minimal utility is going to be 0. Thus the propagator updates the objective's lower bound to 1. The problem is that Y does the same reasoning and both consider that the minimum utility reachable is higher than what they can get. Therefore, both objectives bounds are updated making all states inconsistent. It also, removes the two Nash equilibriums.

5.3 Related work

A related scheme is the one of Asynchronous *DCOP* [219] in which all players share the same constraints but with different costs. Also, in this work, the authors propose to work directly on the action themselves, while we restrict the action based on the objective. Furthermore, we characterize how to reach arc consistency for the Nash constraint. From this, we propose a tractable and efficient filtering for the Nash constraint.

Our work can be also seen as an opening of Singleton arc consistency but based on the objective value. In the original singleton arc consistency, each variable's value is checked whether the constraints are still satisfiable. In games, it is a bit different since without hard constraints all the search space is satisfiable. Instead, an inference on the objective value is feasible. This method is also valid in single objective optimization but a strong condition which is to have a problem without hard constraints.

5.4 Experiments

We have implemented a new solver for Constraint Games on top of the constraint solver Choco v4 [174]. This new solver is composed of an interface to post goals for the different players and the implicit post of the new global constraints as defined in Section 5.1. An important aspect is that it does not require a modification of the classical Constraint Programming framework (search and consistency).

Game	Param.	#PNE	*enum*	*Conga 1*	*Conga 2*
MEG	7.7	6	30.60	10.63	**0.10**
	8.8	7	–	169.92	**0.13**
	70.70	70	–	–	**480.08**
TD	6.6	1	5.95	2.80	**0.06**
	7.7	1	105.13	39.80	**0.07**
	300.300	1	–	–	**514.73**
DG	6.6	720	2.46	**1.53**	2.02
	7.7	5040	43.36	**21.86**	35.42
	8.8	40320	–	**443.11**	–
CG	7.7	7	32.78	11.92	**2.08**
	8.8	8	–	225.31	**27.85**
	9.9	9	–	–	**469.15**
AR	2.250	1	173.68	**0.64**	1.51
	2.2000	1	–	**457.42**	567.15
	2.2500	1	–	**596.30**	–
EFBG	16.2	12870	116.88	115.44	**87.87**
	17.2	24310	256.77	249.04	**188.80**
	18.2	48620	549.69	539.73	**414.38**
CB	2.3.32	0	474.86	450.24	**54.21**
	2.3.55	0	–	–	**564.96**
	2.4.13	0	388.37	329.25	**19.79**
	2.4.20	0	–	–	**244.75**
	2.6.6	0	278.57	240.43	**16.89**
	2.6.10	0	–	–	**599.01**

TABLE 5.1 – Runtime of the different methods on Gamut games

The scope of the preference constraint for Player i is the set of all controlled variables V_C plus Player i's optimization variable opt_i. However, the propagator is called whenever the objective of the player has been updated. Upon the call, it

revises opt_i by launching a new search tree in a distinct solver only on the variables of V_i, having copied the current state S_{-i} of all the other players' variables. During this search, Branch & Bound is applied to the lower bound of the objective and the approximation of its maximin value on the current subspace S_{-i} is returned. For all experiments, we have applied a lexicographic heuristics on the choice of variables and a min value heuristics on the choice of the values.

We have performed experiments on classical games of the Gamut suite [152] : Minimum Effort Game (MEG), Travelers Dilemma (TD), Dispersion Game (DG), Collaboration Game (CG), ArmRaces (AR), ElFarol Bar Game (EFBG) [4] and Colonel Blotto (CB) [188]. We invite the reader to refer to the sections 4.1 and 3.4 and to the original papers to get a full description of the games. They represent a wide range of games, with many, few or no equilibria. In all games, the parameters are given by two numbers : the number of players and the number of actions. For example, "MEG 5.5" denotes the Minimum Effort Game with 5 players, each one having 5 actions. One exception is the Colonel Blotto game for which the second number is the number of battlefields and the last one the number of troops each player can deploy.

We have compared our new approach (called *Conga 2*) to the complete enumeration of the search space (hereafter called *enum*) as implemented in the Gambit solver [141] and to a custom re-implementation of the Conga algorithm (hereafter called *Conga 1*) as described in [149]. The results are presented in Table 5.1. All times are given in seconds and we applied a maximum run-time of 10 minutes to get the complete set of PNE for each problem. All instances for which the given solver has reached a timeout are indicated by "–". Experiments have been run on a Intel Xeon E5-1660 with 32 GB of RAM, Java 8 and Windows 7.

We can see that Conga 2.0 is most of the time better than Conga 1.0, reaching up to three orders of magnitude on MEG. Exceptions are the Dispersion Game and Arm Race for which there is little propagation. We have presented the results in order to show the limits of each technique when staying under the time limit of 10 minutes.

5.5 Graphical Aspects

When the utility of a player only depends on a subset of the other players, the game is called *graphical* [118]. More formally, for a profile $s \in A^{\mathcal{P}}$ and two players i and j, we say that $u_i(s)$ is independent of j if $\forall s'_j \in A_j, u_i(s) = u_i(s_{-j}, s'_j)$. If this is not true, then i depends on j for s, denoted by $i \lhd_s j$. We denote by $dep_i(s) = \{j \mid i \lhd_s j\}$ the set of dependencies of Player i for the state s. For $E \subseteq A^{\mathcal{P}}$, we note $dep_i(E) = \bigcup_{s \in E} dep_i(s)$. Finally, Player i *statically depends* on Player j, denoted by $i \lhd j$ if $j \in dep_i(A^{\mathcal{P}})$. A minimal dependency which occurs only in a strict subspace of $A^{\mathcal{P}}$ is called dynamic.

The main purpose of graphical games is to save space in the matrix repre-

sentation, and also to speed-up the computation of equilibria [215, 156]. We can build the oriented graph of dependencies between player $\mathcal{D} = (\mathcal{P}, \{(i, j) \mid i \triangleleft j\})$. If this graph is not a clique, then the game is graphical. In this case, it is possible to limit the scope of each global constraint to its associated player and his set of dependencies. It ensures an increased performance in the checking of deviations without relying on a specialized algorithm. When checking for deviations, only the state of the dependent variables is injected in the second solver. In addition, because our global constraint is triggered on updates of the objective, it is able to detect on-the-fly dynamic dependencies. Note that this kind of dependencies may also occur in non-graphical games.

In our implementation, we provide the dependency graph as a part of the model. We have performed experiments on classical graphical games : Public Good Game (PGG), Threshold Game of Complement (TGC) [112] and Road Game (RG) [215]. All games have a fixed number of strategies, which is 2 for PGG and TGC, and 4 for RG. Each model has been run on different topologies with different node degrees. In the circle topology (C), each node is of degree 2 and the only parameter is the number of players. For the tree topology (T), T 21.4.3 means that the game has 21 players connected by a tree of maximal degree 3 on 4 levels. The last topology is the complete bipartite graph (B) whose parameter is the number of players. The results are shown in Table 5.2, also with a timeout of 10 minutes. In all cases but one, adding the graphical information is highly beneficial. It demonstrate the efficiency of the approach even for graphs of relative high degrees.

5.6 Conclusion

In this chapter, we have proposed a more elegant and efficient vision of Constraint Games. We have fully modeled the potential constraint pruning available in Game Theory, and proved its intractability. We have proposed an efficient filtering algorithm in the general and graphical cases and demonstrated experimentally its efficiency over the state-of-the-art game solver Conga 1.0 [149]. However, the main interest of this new approach is to bring game theory closer to the elegant framework of Constraint Programming by viewing agent preferences as constraints.

So far, we limited ourselves to the design of a complete solver because only this type of solver is able to be used as a basis for computing more specialized Nash equilibria, like those optimizing a Social Welfare function or Pareto efficient. Also completeness is required to compute Price of Anarchy and Price of Stability. Another open question is the extension of our filtering to games with hard constraints. As is, our filtering is incorrect in presence of hard constraints because the expected deviation may not be satisfiable. As a consequence, the solver becomes incomplete.

Game	Topology	#PNE	*enum*	*Conga 1*	*Conga 2*
PGG	C.30	4610	347.60	332.02	**15.21**
	C.40	76725	–	–	**330.50**
	T.21.4.3	0	522.65	339.56	**2.51**
	T.91.9.3	512	–	–	**319.20**
	B.22	2	73.08	9.17	**0.25**
	B.27	2	–	54.16	**0.33**
	B.250	2	–	–	**485.85**
TGC	C.150	2	460.45	311.54	**1.69**
	C.170	2	–	470.89	**2.21**
	C.2000	2	–	–	**479.91**
	T.57.7.3	2	538.98	80.82	**1.04**
	T.1555.7.5	2	–	–	**182.89**
	B.22	2	131.68	15.48	**2.31**
	B.27	2	–	91.50	**10.23**
	B.33	2	–	–	**96.97**
RG	C.10	1119	16.20	10.92	**3.45**
	C.13	9230	167.16	142.43	**97.95**
	C.16	76004	–	–	**519.14**
	T.31.5.3	244	162.42	50.18	**9.49**
	T.57.7.3	2174	–	–	**216.60**
	B.11	96	16.19	**8.90**	10.11
	B.12	3728	140.54	105.18	**63.74**
	B.15	384	–	–	**266.48**

Table 5.2 – Runtime of graphical games on different topologies

Chapitre 6

Constraint games application

Contents

Problem statement 74
Congestion model 75
Optimization 76
6.1 Constraint model **76**
Path modelling 76
Graph model 77
6.2 Heuristics and problem's relaxation **78**
Residual graph 78
Search strategies 79
Value selection 79
Variable Selection 82
Problem's relaxation 84
6.3 A ILP model **87**
Master Problem 87
Column generation 88
Linearisation of the master problem 90
Piecewise linear approximation of the exponential function. 90
Linearization of the products. 91
Pricing problem 92
Solution 93
6.4 Constraint Games **94**
Constraint games for SDN 94

6.5 Related work **95**
6.6 Instances and experimental results **96**
Generator 96
Settings and implementation issues 97
Experimentation settings 97
A note on implementations 98
Experimental results 99
Constraint programming model 99
ILP model with Column Generation 101
Constraint programming against Column Generation 103
Constraint games 104
6.7 Conclusion **106**

With the internet of things, all kinds of devices are going to communicate, from washing machines, lightbulbs to autonomous cars. By 2020, the forecasts estimate the number of connected devices to the internet is growing to over 31 billion [158]. The amount of data transfer increases with the rise in the number of connected devices. Recently, Software Defined Networking (or SDN) is replacing traditional network routing because it allows fast and remote network reconfiguration, which enables a plethora of flexible architectures, like the upcoming network slicing [214]. SDN (see Figure 6.1) allows centralized control over a network of commuters in order to increase the overall performance. A full SDN controller is a nice source for many optimization problems [130] including online ones. Due to this dynamic aspect and the increasing size of the controlled networks, it is very likely that decentralized algorithms will be mandatory to provide both the expected quality of service and short time response.

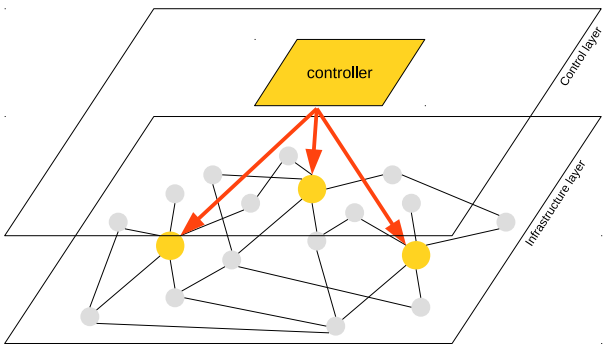


Figure 6.1 – Software Defined Networking

In this paper, we consider the independent routing of multiple demands across a network, also called *the multicommodity flow routing problem*. Each demand

requires to be routed from a source to a destination in a network with limited capacity (i.e. each link has a limited capacity). The overall goal is to assign a route to each demand that minimizes the global cost of routing. This problem and other variants such as robust SDN networks have been studied for a long time [58, 35] with some computational approaches including linear programming [7]. A survey can be found in [142]. Interesting theoretical results have been found, like the one which states that when the problem has a sufficient size and capacity, all flows are actually routed along single-paths [175]. This justifies the modern interest in unsplittable routing of demands. The demands are routed in a network with limited capacity constraining the shortest paths computations, which is known as a NP-complete problem. In our approach, we do not consider other side constraints such as must-pass/cannot-pass or redundant routing, although they can be easily introduced in our constraint model. However, we consider a congestion model increasing the cost of a link according to the traffic routed.

Furthermore, SDN is based on a centralized vision of networking but this does not mean that all algorithms have to be centralized [154]. Indeed, with the growth of the size of the controlled zone and the large increase in the volume of the demands, decentralized algorithms will be necessary to achieve the expected level of performance for future SDN with millions of demands coming online. A common way of modeling agreement between a set of agents is to reach a Nash equilibrium. Also, some instances of the problem correspond to networks of aggregated traffic for which the users (often network providers) are very sensitive to the quality of service. This is why an allocation at Nash equilibrium is desirable as it ensures the user that his quality of service cannot be improved by any selfish move. While a centralized approach is sure to converge to an optimal solution, it is not guaranteed for Nash equilibriums. The equilibriums costs can be far from the global optimum. Braess's paradox [29] is a good illustration. It states that in congested roads network, building a new route creates even more congestion due to the selfishness of agents. This degenerative behavior is one of the motivations to compute Price of Anarchy [64] which allows to evaluate the potential loss of efficiency of decentralized algorithms (i.e. the loss of being at Nash equilibrium).

We propose two approaches for the centralized approach : a Constraint Programming (or CP) model and an Integer Linear Programming (or ILP) model using Column Generation. Only the Constraint Programming model is able to model closely the congestion problem and can be used to solve the problem to optimally. For this, we use a natural and dedicated heuristic based on increasing paths and a relaxation based on shortest path to prune efficiently the search space. Note that increasing path have been introduced as CP heuristics in [164].

The Constraint Programming model is implemented using the Choco constraint solver [174] and the ILP one with CPLEX [49]. The model computing the Nash equilibriums uses Constraint Games framework : ConGa which is an extension of the Choco solver for Constraint Games [159]. In the benchmarks, we show networks with hundreds or even thousands of commodities solved to optimality

including the Nash equilibriums computations. These results show that practical use of game theory is now possible at industrial scale.

The chapter is organizing as follow : first we introduce the problem in Section 6, then in Section 6.1 and 6.2 we present the CP model and the heuristics used to compute a solution in practice. In section 6.3 we present the ILP model, in Section 6.4 the Constraint Games framework used to compute selfish routing, the Section 6.5 gives a literature review, in Section 6.6 the evaluation on a set of benchmarks on real-world and synthetic instances and lastly we present the conclusion.

Problem statement

A multicommodity path routing problem (MCPRP) consists of a graph defining a network and a set of commodities (flow demands) to be routed on this graph. We consider in this article the problem in which we compute for each demand a single route from the source to the destination node such that the sum of bandwidth routed by a link does not exceed its capacity. Congestion occurs when a link is taken and is reflected by a congestion cost which helps to ensure a homogeneous distribution of the routes. The overall objective is to minimize the sum of costs of the routed demands.

We assume we have a network $N = (V, E)$, which is a directed graph composed of a set of vertices (or nodes) V and a set of edges (or links) $E \subseteq V^2$. For each edge $e = (x, y) \in E$, we associate a cost $cost(e) \in \mathbb{R}^+$ and a capacity $cap(e) \in \mathcal{R}^+$. Let D be the set of demands to be routed. For a demand $d \in D$, we define $src(d) \in V$ and $dst(d) \in V$ to be respectively the source and destination node, and $bw(d) \in \mathcal{R}^+$ to be the required bandwidth for this demand.

A *path* is a sequence of nodes $p = (v_i)_{i \in [0..n]}$ such that $\forall i \in 0..n-1, (v_i, v_{i+1}) \in E$. We denote by $src(p)$ the node v_0 and by $dst(p)$ the node v_n. We consider here only acyclic paths, i.e. such that $i \neq j \rightarrow v_i \neq v_j$. By a slight abuse of notation, we write $(x, y) \in p$ to denote that the arc (x, y) is taken in the path p.

A *solution* for the MCPRP is the assignment of a path $path(d)$ to each demand d such that we ensure correctness :

$$\begin{aligned} \forall d \in D, \quad src(path(d)) &= src(d) \\ \forall d \in D, \quad dst(path(d)) &= dst(d) \end{aligned}$$

and admissibility with respect to the capacity constraints :

$$\forall e \in E, \left(\sum_{\{d \in D \mid e \in path(d)\}} bw(d) \right) \quad \leq \quad cap(e)$$

Congestion model

In order to ensure a good balance over the network, we incorporate to the model a model of congestion. Basically, congestion will increase the cost of a link when this link is close to saturation. For this, we define the load of an edge e to be :

$$load(e) = \left(\sum_{\{d \in D \mid e \in path(d)\}} bw(d) \right) \; / \; cap(e) \tag{6.1}$$

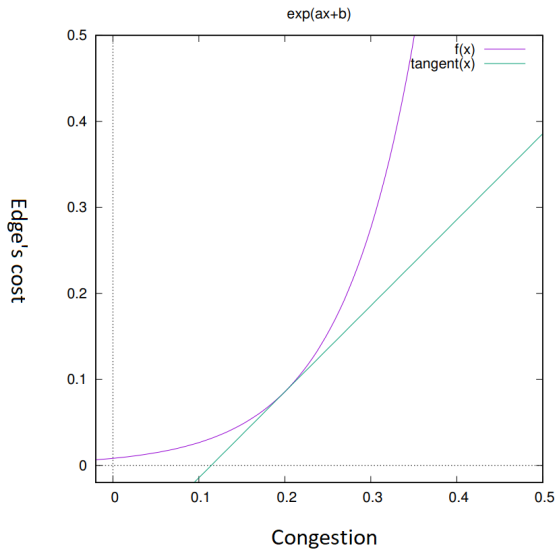


FIGURE 6.2 – A plot of the congestion function for $MaxC = 1000$ and $cong'(0.2) = 1$

The congestion model we use for a given arc e has an exponential increase of the form :

$$cong(e) = \exp\left(a \times load(e) + b\right) \tag{6.2}$$

In order to choose the parameters a and b, we pose some conditions on the function. First we should have a sufficiently high value of $cong(e)$ when the load is 1. By sufficiently high we mean that a demand should not prefer to take a heavily congested link while there are some (maybe longer) available paths. It can be done by fixing this limit to the highest link cost of the network $MaxC$. We then have the equation $e^{a+b} = MaxC$. Then, in order to set when the exponential starts to overtake on a linear increase, we impose a condition on the derivative to be 1 at a given point α. The derivative of the congestion function is given by

$cong'(x) = ae^{ax+b}$. If we impose that the derivative should be 1 for $x = \alpha$, we get the equation $ae^{a\alpha+b} = 1$. By solving numerically these equations we get the values of a and b for a given problem. For example, in Figure 6.2 is a plot of the congestion function for $MaxC = 1000$ and $cong'(0.2) = 1$. We assume that the same values of a and b are set for all the links of the network, although this can be easily changed.

Optimization

Solving a MCPRP P to optimality means finding a solution minimizing the global cost of the demands. For this, we first define the cost to route a demand. It is obtained by aggregating the cost of each traversed arc with the cost coming from congestion :

$$cost(d) = bw(d) \times \sum_{e \in path(d)} (cost(e) + cong(e)) \tag{6.3}$$

Then the cost of the whole problem is given by :

$$cost(P) = \sum_{d \in D} cost(d) \tag{6.4}$$

Note that this function is strictly monotonic, resulting in that each addition of demand increases the edge cost.

6.1 Constraint model

In order to implement this problem as a constraint program, we need to first represent paths, which will be the solutions of our problem. Then we need to link the computed paths to the network data : costs, capacity and provide a support to compute congestion.

Path modelling

A path is represented by an array *path* of $|V|$ variables which correspond to the set of arcs in the path. Each variable's value corresponds to the node's successor (i.e. the next node along the path). The initial domain of a variable associated with a node v is given by the set of neighbors of v in the graph. In order to ensure the correct representation of a path, we use the global constraint $subPath(path, src, dst)$ which ensures that the node from *src* to *dst* form a valid subpath of the graph. This constraint is a variant of *subCircuit*. Unused nodes of the path point to themselves and an extra variable is appended to the array to indicate which vertex starts the path.

Example 32 (Path model) *The figure 6.3 describes the model for finding a path having the node* 2 *as a source and the node* 5 *as the destination. In the beginning, the variables domains are filled with all the possibles neighbors including itself. For example, the node* 0 *can have as successor the nodes :* $0, 1 or 2$. *Only the source and the destination are treated differently. Since a path can be seen as a circuit between the source and the destination nodes. That is why their domains are adapted : no self-loop for the source node(i.e. a successor is required) and the destination's successor is the source. A solution to the problem instance is depicted in the array line labeled by path. This array encodes the path* $(2, 1, 4, 3, 5)$. *It has to be read as follow : the node 0 has 0 as successor (not in the path), the successor's of 1 is 4, the successor's node of 2 is 1 ...*

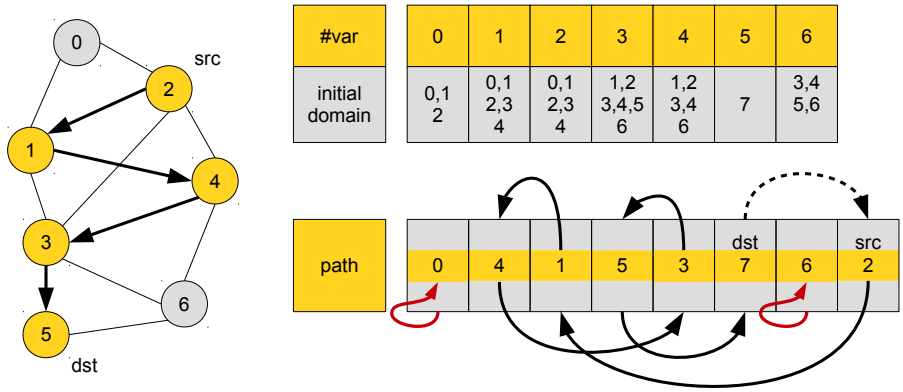


FIGURE 6.3 – Encoding of a path

For each demand $d \in D$ we associate an array $path^d = (v_i^d)_{i \in V}$ constrained by :

$$subPath([v_1^d, \ldots, v_n^d], src(d), dst(d))$$

Graph model

In order to ensure that no link is overloaded and in order to compute congestion, we need to know which demands are routed by a given arc. In this model, we use a Boolean variable $EdgeIsUsed_{(i,j)}^d$ which is true if the path $[v_1^d, \ldots, v_n^d]$ assigned to demand d uses the arc (i, j). This connection is made with the following channeling constraints :

$$\forall (i,j) \in E, \forall d \in D, \quad EdgeIsUsed_{(i,j)}^d \leftrightarrow v_i^d = j$$

We compute the amount of bandwidth routed by an arc in a variable $f(e)$ with the constraint :

$$\forall e \in E, \quad f(e) = \sum_{d \in D} EdgeIsUsed_e^d \times bw(d)$$

We ensure that the capacity of each arc is not exceeded :

$$\forall e \in E, \quad f(e) \leq cap(e)$$

Then we can compute the congestion of a given edge in a variable $cong(e)$:

$$\forall e \in E, \quad cong(e) = e^{a \times \frac{f(e)}{cap(e)} + b}$$

The cost $cost(d)$ of routing a demand by a given path is given by the constraint :

$$\forall d \in D, \quad cost(d) = \sum_{e \in E} EdgeIsUsed_e^d \times bw(d) \times (cost(e) + cong(e)) \tag{6.5}$$

A variable $ProblemCost$ sums the costs to route all demands :

$$ProblemCost = \sum_{d \in D} cost(d) \tag{6.6}$$

We shall minimize this variable.

This model is quite standard and intuitive. It defines one Boolean variable by edge and by demand. Since the number of edges is quadratic in the number of vertices, this number may grow a lot for some large networks.

6.2 Heuristics and problem's relaxation

We have tried a variety of combinations of search strategy (or SS) and problem's relaxation to improve the resolution of this problem. In this chapter, we will refer to a particular combination by `A/B/C` where `A` is the variable selection strategy, `B` the value selection strategy and `C` the type of relaxation to compute the problem's bound, as explained below. At a given node of the search tree, some demands or some partial paths may already be assigned. Apart from classical CP heuristics, all heuristics and lower bound computations use the *residual graph* obtained by considering this part already fixed.

Residual graph

For each demand, a residual graph is maintained all along the search. This graph is the cornerstone to solve efficiently this problem. It is used by the search heuristics and the relaxation technique. A residual graph is modified incrementally at each search tree node. We refer as `path` the edges belonging to the path as it is in the current search tree (i.e. the instantiated variables) and as `future path` the path's part which is not instantiated in the search tree but computed by the Dijkstra algorithm. A residual graph is built such that :

— It exists a directed edge from the node i to $j \leftrightarrow j \in D(V_i)$

— The cost of an edge is dynamically set and updated by the variables' values. When a variable is instantiated(i.e. an edge is added to the path), the edge's minimal cost is updated by updating its bandwidth. For instance, when a demand goes through an edge, its congestion cost is automatically updated with the current demand's bandwidth and that for all residual graphs. However, it is not possible to take into account the congestion in future path. It means that two demands whose future path would take the same link act like if they do not create congestion in their future paths.

The residual graphs are constructed with the CP variables, therefore the graphs are modified at each decision or propagation automatically.

Example 33 (residual graph) *Two demands d_1 and d_2 having each a bandwidth of 2, have to be routed in the 4 nodes networks shown in Figure 6.4. In this network, the cost of each edge is 0 and the congestion parameters are respectively $a = 1$ and $b = -0.5$. At first, the residual graphs are constructed at the root of the search tree (the edge sets representing the paths are empty). The costs are initialized only with the bandwidth induced by the demand. For instance in Figure 6.4a, the residual graph costs of d_1 are computed only knowing the bandwidth of d_1, no assumption can be done about d_2. The costs are thus 2 obtained by :$2 \times e^{\frac{2}{4}-0.5}$. In the next step shown in Figure 6.4b, d_1's path is continued with the edge between the node 0 and 1. This decision updates the cost of the residual graph of d_2. The new cost is $2 \times e^{0.5}$, it corresponds to the cost when the two demands take the same edge. In Figure 6.4b, d_2 is going through the same edge as d_2, thus the residual graph of d_1 is updated.*

Search strategies

Path-oriented problems are particularly sensitive to SS, and not surprisingly, a standard dynamic CP SS (denoted by `CP` in this chapter) like impact or activity would be of weak efficiency for this type of problem. Indeed, it is likely that this SS will label any node in the path without knowing if it could be linked to the source or destination. Therefore, we propose a variable's value selection strategy as well as three variable selection strategy, all dedicated to this SDN problem.

Value selection

For each variable, the value SS determines the path's direction. Since the goal is to find the best path for each demand, it would be inefficient to start the path in a wrong direction. We have chosen to label path variables in order of *increasing* path cost. In order to start with the most promising path, we maintain at each node of the search tree the shortest path to the destination in the residual

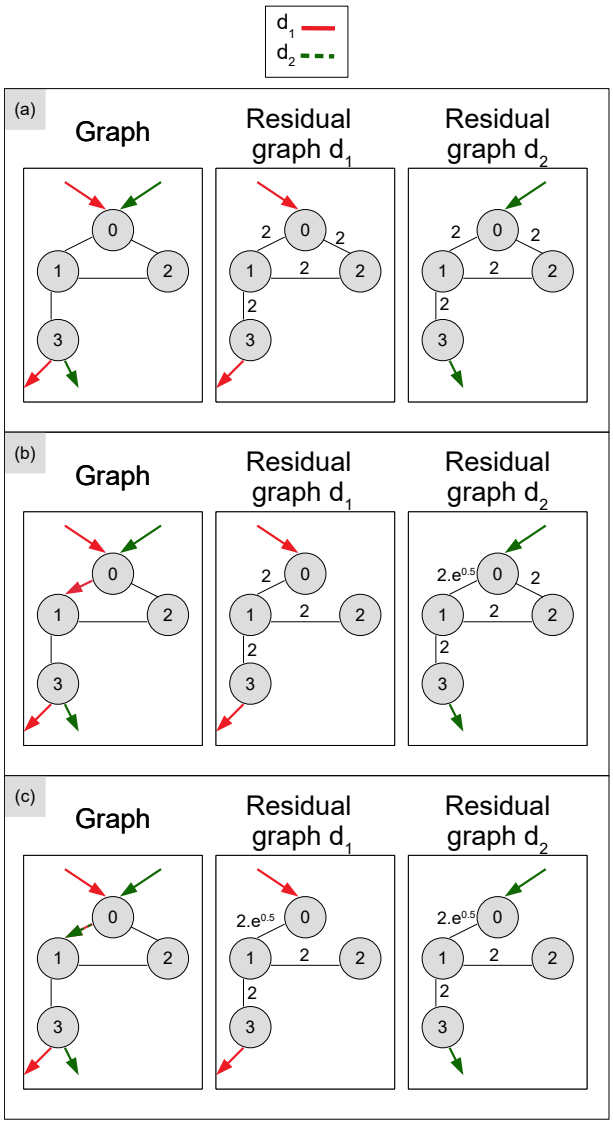


Figure 6.4 – Residual graph updates examples

network for each demand in isolation. In other words, given a variable v_i^d and the shortest path SP_d for the demand d the variable is going to be instantiated as follow :

$$v_i^d = SP_i(v_i^d), \quad \text{if } v_i^d \in SP_d$$
$$v_i^d = i, \quad \text{otherwise}$$

Where $SP_d(v_i^d)$ gives the successor of the node v_i^d for the demand d 's shortest path.

We call this value strategy `SP` (for *Shortest_Path*). It is done with Dijkstra's algorithm, considering the progression of the already assigned part of the other demands. This information on the best future path is used to choose the next node of the path when needed(i.e. the variable value). Note that the Dijkstra algorithm only considers the nodes of the paths already assigned at a given point of the search tree for computing the congestion. In particular, the congestion is not cumulative for two demands which share the same future link. The same idea has been implemented in [40] but with specific path variables.

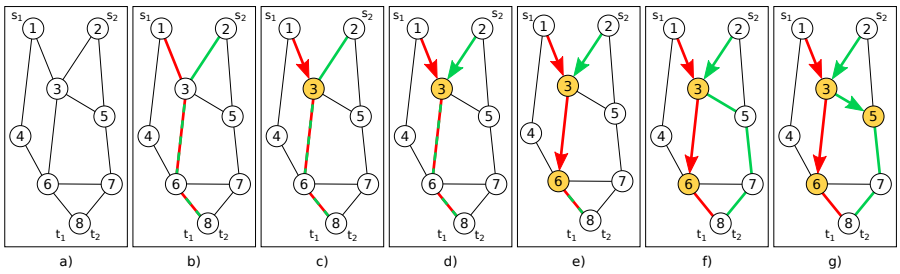


FIGURE 6.5 – Labeling of paths for two demands

Example 34 *In Figure 6.5 is represented a small example of two demands being routed on a 8-nodes network (6.5a) by shortest path heuristic. This example aims to show the dynamical aspect of the shortest path computation. The demand d_1 has to be routed from the nodes 1 to 8 and the demand d_2 from the nodes 2 to 8. The source and destination of d_1 (resp. d_2) are the nodes labeled by respectively s_1 and d_1 (resp. s_2 and d_2). Actual paths taken by the demands are depicted by solid arrows while shortest paths computed by the SS are with hatched lines. In other words, the solid lines represent the variables already instantiated in the problem while the hatched ones are representing the current shortest path computed by the value search strategy. At first (6.5b), the two demands compute their shortest paths : $(1-3-6-8)$ for d_1 and $(2-3-6-8)$ for d_2. The shortest future paths correspond to hatched lines. In (6.5c), one labeling step is performed for d_1. Since there is no change on d_2's path, no update of d_2's shortest path is necessary. Hence in (6.5d) one step is performed for d_2. In (6.5e), the next move of d_1 causes congestion on the link from 3 to 6. Thus d_2 updates its shortest path to $(2-3-5-7-8)$ in (6.5f) to lower its minimal cost. It yields a next move by d_2 in the direction of node 5 in (6.5g). Then the last edge is selected, resulting the complete paths instantiation(6.5h).*

Variable Selection

Once again, a variable strategy selecting the variables outside a path scope would be very inefficient, that why we choose to select the variables all along the paths. In other words, our variable selection respects the path order, it selects the next uninstantiated successor variable along the path. Note that this is a partial variable selector since it is only once the demand is chosen that the actual variable is determined by the next step to be extended. For the variable selection to be completely defined, we have considered three strategies for choosing the demand. The first one, called `MB` (for *Max_Bandwidth*), consists in routing the next remaining demand with the maximum bandwidth up to its completion. Then we have defined two strategies based on conflicts analysis. The strategies react on a solution (by `MB` for the first one) or when a fail occurs. For each demand and each link, we compute the marginal cost (with congestion) induced by the presence of the demand on this very link. The marginal cost corresponds to the difference between the cost with and without routing the demand, all being equals. Then, we sum up all these numbers for each demand along the taken path to obtain a score. The first one, called `CO` (for *Conflict*), chooses the demand of the highest score and develop its path up to the destination. The second one, called `CO1` (for *Conflict_1_Step*), also chooses the demand of the highest score but only develops one step in the path before reconsidering the situation. In `CO1`, the conflicts are stored for each path variable for each demand and score are only computed for the uninstantiated variables.

Example 35 (Strategies in action) *In this example, we show the selection process of the three strategies with SP value SS. Three demands d_1, d_2 and d_3 with respectively a bandwidth of $4, 3$ and 2 have to be routed in the 6-nodes network. To keep things easy, all edges have a capacity of 7, a cost of 0 and congestion parameters are set to $a = 1$ and $b = -0.5$. The selection process of* `MB`, `CO` *and* `CO1` *are shown respectively in Figure 6.6, Figure 6.7 and Figure 6.8.*

MB Strategy. *Given the network in Figure 6.6a),* `MB` *selects the demand having the highest bandwidth to be routed. At first, d_1 is chosen and is instantiated from its source to its destination (see Figure 6.6b). Afterwards, the next demands to be instantiated are going to be iteratively d_2 and d_3 (Figure 6.6c and d).*

CO strategy. *Given the network in Figure 6.7a), where a first solution has been found.* `CO` *analyses the conflicts in the solution to try to redirect the conflicting demands. In this solution the demands d_1 and d_2 are in conflict. The marginal*

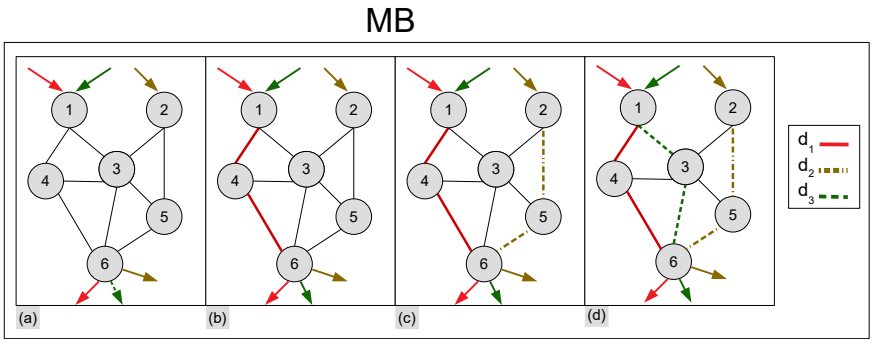


FIGURE 6.6 – Selection process of MB strategy

costs are computed for the demands as follow :

$$\Delta(price_{d_1}) = 2 \times (7 \times e^{\frac{3}{7}+\frac{4}{7}-0.5} - 3 \times e^{\frac{3}{7}-0.5}) = 20.19$$
$$\Delta(price_{d_2}) = 2 \times (7 \times e^{\frac{4}{7}+\frac{3}{7}-0.5} - 4 \times e^{\frac{4}{7}-0.5}) = 17.47$$

The demand d_1 is the one having the highest score and thus selected to be routed (see Figure 6.7c). And then to finish the demand d_2 is the last one to be routed until its destination (see Figure 6.7d).

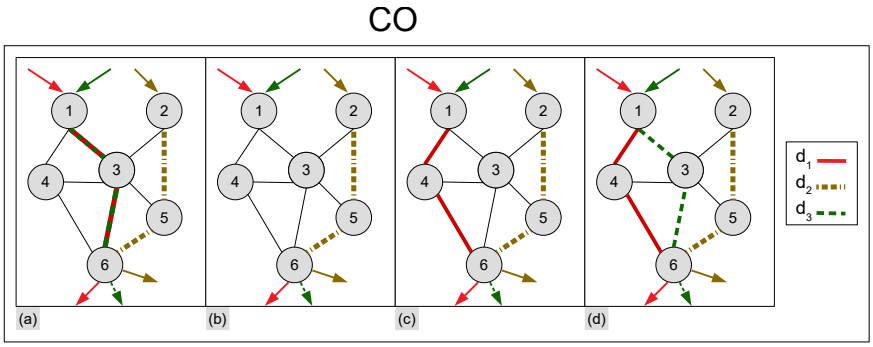


FIGURE 6.7 – Selection process of CO strategy

CO1 strategy. *The initial situation of* `CO1` *is the same as* `CO` *: a first solution has been found.* `CO1` *analyses the conflicts as well as* `CO` *but instantiates only edge by edge while selecting the demands with the highest conflict score on the non instantiated variables. The marginal costs for a demand d_i, given an edge from the nodes i to j, named $\Delta(price_{d_i}(n_i, n_j))$ are computed as follow :*

$$\Delta(price_{d_1}(1,3)) = (7 \times e^{\frac{3}{7}+\frac{2}{7}-0.5} - 3 \times e^{\frac{3}{7}-0.5}) = 10.09$$
$$\Delta(price_{d_1}(3,6)) = (7 \times e^{\frac{3}{7}+\frac{2}{7}-0.5} - 3 \times e^{\frac{3}{7}-0.5}) = 10.09$$
$$\Delta(price_{d_2}(1,3)) = (7 \times e^{\frac{4}{7}+\frac{3}{7}-0.5} - 4 \times e^{\frac{4}{7}-0.5}) = 8.74$$
$$\Delta(price_{d_2}(3,6)) = (7 \times e^{\frac{4}{7}+\frac{3}{7}-0.5} - 4 \times e^{\frac{4}{7}-0.5}) = 8.74$$

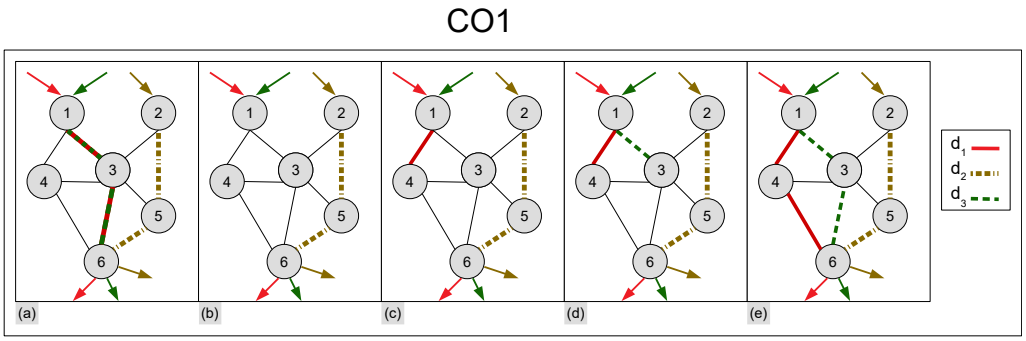


Figure 6.8 – Selection process of CO1 strategy

After a solution was found, the solver backtrack until the Figure 6.8b. Then it selects the demands with the highest score and instantiates its first element on the path, which corresponds to the edge between the node 1 and 4 (see Figure 6.8c). Afterwards, d_3 has the highest score since the edge between the nodes 1 and 4 is not anymore considered for the demand d_1. d_3 is routed by the edge between the nodes 1 and 3 (see see Figure 6.8d). This process is continued until all the destinations are reached (see Figure 6.8e)

Problem's relaxation

Relaxation techniques are commonly used in constraint optimization. However, CP solvers offer a restricted and uninformed version. When minimizing the variable $ProblemCost$ and after having found a solution of value A, it simply adds to the remainder of the search the constraint $ProblemCost < A$. The CP solver is unaware of the problem structure. While efficient, it requires that the lower bound of $ProblemCost$ to exceed A to cut the search tree and backtrack. In our case, the possible values of $ProblemCost$ are strongly constrained by the current branch of the search tree leading to a node, but very loosely for the remaining part of the problem. In order to cut earlier, we need a better estimation of the lower bound of $ProblemCost$. This is done by adding to the lower bound the cost of individual routing along the path computed by the Dijkstra algorithm used for the value SS. We use the previously defined *residual graph* in which congestion is

taken into account to estimate the cost lower bound of the current search tree state. We need this to provide a better yet safe estimate of the lower bound which does not exceed the future real cost. We call the classical CP branch & bound `CP` and the one which uses the bound provided by the shortest path `SP`.

Let $[a_1^d, \ldots, a_i^d, \ldots, a_{n_d}^d]$ be the path computed by the Dijkstra algorithm for a demand d from node a_i^d. We have $a_1^d = src(d)$ and $a_{n_d}^d = dst(d)$ and $\forall j < i$, the value of a_j^d is given by the instantiated part of the path in $[v_1^d, \ldots, v_n^d]$ (up to the current node of the search tree). The cost contribution of demand d is given by :

$$\begin{aligned} cost(d) \quad = \quad & \sum_{\{e=(a_j^d, a_{j+1}^d) \ \mid \ j<i\}} bw(d) * (cost(e) + cong(e)) + \\ & \sum_{\{e=(a_j^d, a_{j+1}^d) \ \mid \ i \leq j < n_d\}} bw(d) * cost(e) \end{aligned} \tag{6.7}$$

Proposition 3 *Given a monotonic cost function (see equation (6.3)), the bound given in equation 6.7 is sound.*

Proof 3 *Suppose by contradiction that the proposition is not correct and the equation is not sound. This statement implies that it exists at least one node's cost which is overestimated by the Dijkstra algorithm. The latter is either located on the instantiated nodes or on the future path. This is impossible because the given costs corresponds to the lower bound and are at worst underestimated. That is why Dijkstra algorithm and thus the computed path is computing correct lower bound for the shortest path algorithm.*

Note that, due to the presence of link capacity constraints, a fail is triggered when Dijkstra algorithm is unable to find a path from the source to the destination [201].

Example 36 *The example in Figure 6.9 illustrates how the relaxation technique based on shortest paths works. Two demands : d_1 and d_2 have to be routed through a 5 nodes network (see Figure 6.9(a)). Each demand has a bandwidth of 2 and each arc in the network can transport 4 units of bandwidth. The two demands d_1 and d_2 have both the node 0 as source and respectively the nodes 3 and 4 as destination. The parameters of the congestion cost function are $a = 1$ and $b = -0.5$. To simplify the problem, each edge's cost is 0.*

In each subfigure is depicted on the left the state of the current graph with the decisions already taken and on the right the residual graphs of d_1 and d_2. The shortest path algorithm of each demand is computed on its own residual graph. Because it is implemented on the CP variables, the SP computation is aware of the current bandwidth and the successor variables in order to consider only feasible paths.

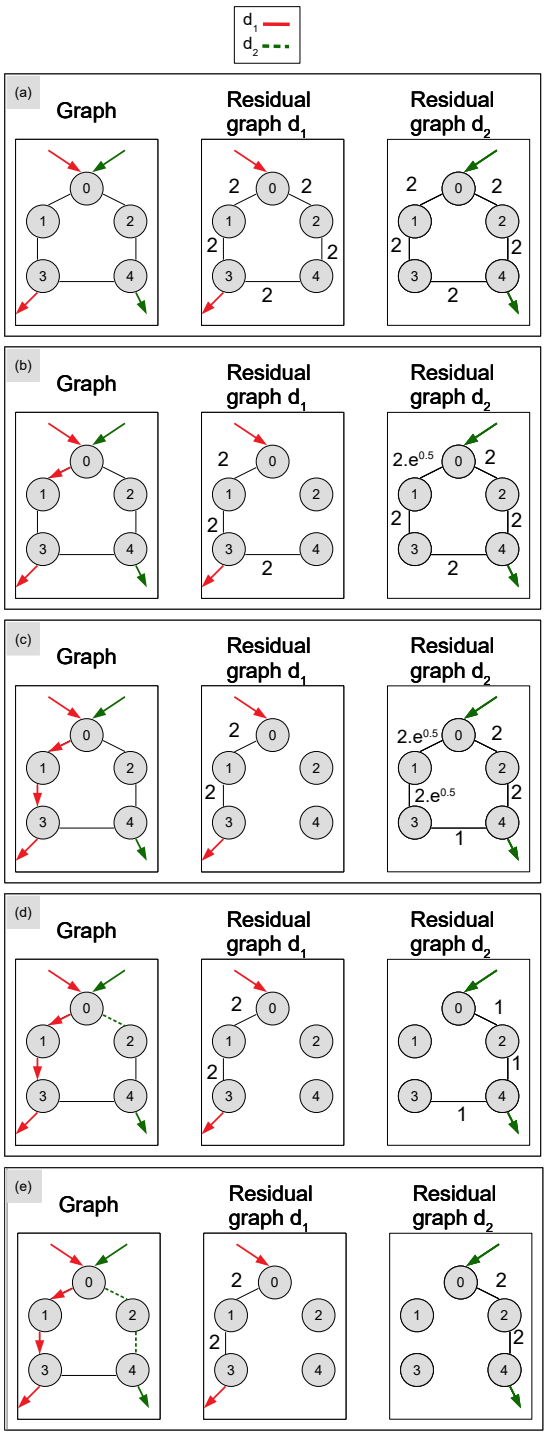


Figure 6.9 – Problem's relaxation example for SDN

In beginning of the problem resolution, the initial propagation is triggered updating the minimal reachable global cost. To do so, the cost of each demand is evaluated (see Figure 6.9a). The minimal possible cost corresponds to the demand shortest path without any added congestion due to other demands. For d_1 and d_2 it is obtained by the following computation : $2 \times 2 \times e^{\frac{2}{4}-0.5} = 4$. For each demand, the shortest path's cost is 4. Then the problem is explored by instantiating a first edge for d_1 (Figure 6.9b). The cost is updated in the residual graphs. Taking the edge from the nodes 0 and 1 costs now $2 \times e^{0.5}$, this update is done in the residual graph of d_2. In the residual graph of d_1 only the possible path are updated : the node 2 cannot be taken anymore. The same process is repeated when the path of d_1 is continued (Figure 6.9c). Afterwards, it is the second demand which is routed (Figure 6.9d and Figure 6.9e). While taking these decisions, the residual graph of d_2 is updated by removing some edges. The edges of the residual graph of d_1 are not impacted since d_2 does not take the same edges. The solution found has a cost of 8. The Dijkstra relaxation help to state that it does not exist better solution since at the beginning the lower bound for the problem was also 8. The problem's exploration is thus finished.

6.3 A ILP model

ILP techniques are commonly used to solve multicommodity flow problems [11], even in the context of SDN [165]. However, the model we presented in Section 6.1 is not suited to an ILP formulation because it is very difficult to model paths as in CP. Instead, most formulations either use a flow model or use a pre-computation of paths for the different demands and associate a Boolean variable to each possible path. We will use this technique despite it yields an exponential number of variables because they can be generated on the fly using column generation.

Master Problem

First we reformulate the multicommodity flow problems with Boolean path variables in what we call a Master Problem, then we provide a linearization and the pricing problem used to introduce new columns. For each demand $d \in D$, we associate the set P_d of all paths from $src(d)$ to $dst(d)$. By a slight abuse of notation, we also call p a Boolean variable associated to a path $p \in P_d$ because paths are only manipulated through their Boolean variable. Because a path is statically defined and because we need to sum up the bandwidths associated to the various arcs of the network in order to enforce the capacity constraints, we associate to a path variable p and each arc $e \in E$, a variable p_e which is true if the arc e is taken by the path p. Note that this variable p_e is used just to simplify the notation and does not belong to the implemented model. We ensure that

exactly one path is chosen for each demand :

$$\forall d \in D, \quad \sum_{p \in P_d} p \geq 1 \tag{6.8}$$

The capacity constraints become :

$$\forall e \in E, \quad \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \leq cap(e) \tag{6.9}$$

We aggregate all costs in the following expression to be minimized :

$$\min \quad \sum_{e \in E} \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \times (cost(e) + cong(e)) \tag{6.10}$$

Where the congestion is defined by equations 6.1 and 6.2. There are two sources of non-linearity in these formulas. First the load of an arc uses an exponential function. It yields that it is easier to break up equation 6.10 in two for its linearization. A first part we call *cost with congestion* $cwc(e)$ for a given arc e and a subsequent aggregation on the set of demands :

$$cwc(e) \geq \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \times (cost(e) + cong(e)) \tag{6.11}$$

Note that since we deal with a minimization problem, only the $\geq$ part of the equation is mandatory to enforce equality. Then the expression to be minimized is :

$$\min \sum_{e \in E} cwc(e) \tag{6.12}$$

But then a more subtle source of non-linearity is that, since the cost depends on the load and the load depends on the path chosen for each demand, we have to consider for the cost the cases where two or more demands are routed by the same arc. It yields a product between the Boolean variables p_e^d and $p_e^{d'}$ for any pair $d, d' \in D$. We now address these two relaxations.

Column generation

The problem we get with a model based on paths and its subsequent linearization involves an exponential number of variables (since there are exponentially many paths between a source and a destination). Moreover, only one variable for each demand will be set to 1 because we seek a single path for each demand. It is impossible to represent all these variables but fortunately they can be generated on the fly (along with the constraints they are subject to) using Column Generation (see Figure 6.10).

Column Generation alternates between solving the linear Restricted Master Problem with a limited number of variables (or columns) and generating new

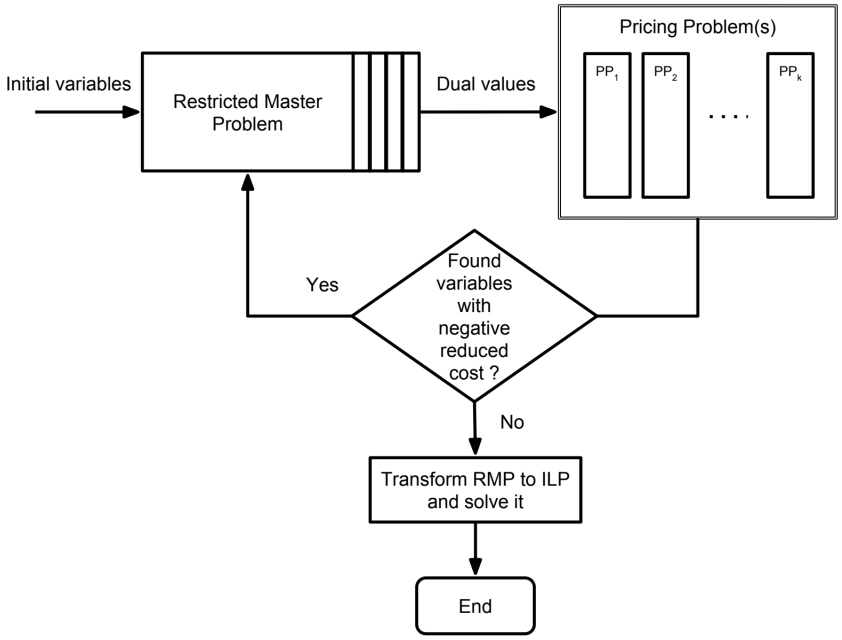


FIGURE 6.10 – Column generation procedure

variables by solving a sequence of subproblems called Pricing Problems. The first step before iterating is to initialize the linear Restricted Master Problem (or RMP) with initial columns. It is then possible to get dual values and to compute reduced costs. A reduced cost is associated to a dual variable and tells how much the objective changes if this variable increases by a small amount. In other words, it is the first derivative from a certain point on the polyhedron that constrains the problem.

Column generation methods were invented from the observation that often in problems many variables do not belong to the optimal solution and thus their values are set 0 and not used. The idea is to try to generate only the columns useful to solve optimally the problem. For instance in our problem, often only few paths are needed to find and prove the optimal solution.

A Pricing Problem is used to determine which column should be introduced. It yields either to add a new variable or to ensure that there are no further variables with negative dual feasibility i.e. which can potentially improve over the current solution. When no more columns can be generated, the linear solution is rounded to give an integer one.

We consider only column generation at the root node. This method can be incomplete unlike Branch and Price which is a bit different since it considers a tree obtained by solving the ILP problem for different sets of columns.

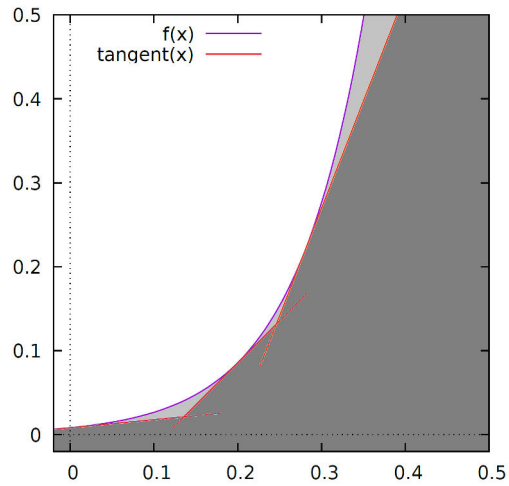


Figure 6.11 – Three pieces linear approximation of the exponential function

Linearisation of the master problem

What we call Linearized Master Problem (or LMP) is essentially a linear approximation of the Master Problem introduced above. It means that the solutions we will find with ILP are solutions to the approximate model and not exactly solutions of the original problem. However, if the linearization is good, it is likely that the solution paths for the demands will be the same as if the exact model was solved, although it cannot be ensured in all cases. In practice, we have not observed any difference.

The first thing to come is to transform the Boolean variables into continuous ones in the interval [0..1].

Piecewise linear approximation of the exponential function.

The second relaxation concerns the exponential function. We approximate it with multiples tangents. Let I be a set of numbers in [0..1]. For each point of the exponential curve $(i, cong(i))_{i \in I}$, a tangent $t_i(x) = a_i x + b_i$ is computed. In Figure 6.11 is depicted a 3-points approximation of an exponential function. The red lines correspond to the computed tangents approximating the function. The difference between the approximation and the real function is shown in light gray. In this zone, the congestion is underestimated and may induces less filtering. For the sake of simplicity, we use the same set I for all links of the network.

The approximation of the exponential correspond to the maximum value of these tangents $\max_{i \in I} t_i(load(e))$. In order to get a linear formulation of the maximum, we can introduce for each arc e and each $i \in I$ a variable $cong_i(e)$ giving the value of each tangent for a given load and one variable $cong(e)$ for the

maximal value. The $cong_i(e)$ reuses the definition of the load given in equation 6.1 :

$$\forall e \in E, \forall i \in I, \quad cong_i(e) = \left(\frac{a_i}{cap(e)} \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \right) + b_i$$

And for each link, the following constraints are added :

$$\forall e \in E, \forall i \in I, \quad cong(e) \geq cong_i(e) \tag{6.13}$$

Linearization of the products.

Unfortunately, when computing the cost with congestion $cwc(e)$ of an arc e with equation 6.11, all demands crossing this arc actually cause the congestion to increase. If we develop the formula by mixing equations 6.11 and 6.13 with respect to each $i \in I$, it yields for a given edge e :

$$\forall i \in I, \quad cwc_i(e) \geq$$
$$\sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \times \left(cost(e) + \left(\frac{a_i}{cap(e)} \sum_{d' \in D} \sum_{p' \in P_{d'}} p'_e \times bw(d') \right) + b_i \right)$$

By splitting the linear and non-linear part we get :

$$\forall i \in I, \quad cwc_i(e) \geq$$
$$(cost(e) + b_i) \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) +$$
$$\frac{a_i}{cap(e)} \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) \times \sum_{d' \in D} \sum_{p' \in P_{d'}} p'_e \times bw(d')$$

The last expression is quadratic because it contains a product between p_e and p'_e. To get a linear formulation, we introduce new Boolean variables pp'_e for each arc e and each path p for d and each path p' for demand d' such that pp'_e is true if and only if p and p' share e as common arc. Since we model only one simple path by demand, we can use another trick by summing all the path for each demand. The meaning of the pp'_e can be reformulated as : pp'_e is true if and only if it exists p and p' for respectively d and d'.

We implement the logical `AND` (see chapter 7 of [21]) by this set of linear constraints :

$$pp'_e \leq \sum_{p \in P_d} p_e, \quad \forall e \in E \tag{6.14}$$

$$pp'_e \leq \sum_{p' \in P_{d'}} p'_e, \quad \forall e \in E \tag{6.15}$$

$$pp'_e \geq \sum_{p \in P_d} p_e + \sum_{p' \in P_{d'}} p'_e - 1, \quad \forall e \in E \tag{6.16}$$

Then the cost constraints can be reformulated as follows :

$$\forall e \in E, \forall i \in I, \quad cwc_i(e) \geq (cost(e) + b_i) \sum_{d \in D} \sum_{p \in P_d} p_e \times bw(d) + \frac{a_i}{cap(e)} \sum_{d \in D} \sum_{p \in P_d} \sum_{d' \in D} \sum_{p' \in P_{d'}} pp'_e \times bw(d) \times bw(d')$$

As in equation 6.13 we aggregate all costs for the different tangents :

$$\forall e \in E, \forall i \in I, \quad cwc(e) \geq cwc_i(e)$$

And thus the expression to be minimized as in equation 6.12 becomes :

$$\min \sum_{e \in E} cwc(e)$$

Pricing problem

The reduced cost for a given variable determines how the objective changes if the variable increase of one unit. A Linear problem is optimal if its reduced cost is 0. However, if the reduced cost is negative, the solution can enter the basis as a new column. If the reduced cost is greater or equal than zero, the lower bound for the optimal solution has been found, although this may not be an integer solution. Note that the reduced cost can be computed on each edge individually. In order to find an improving path for each demand, we could perform a shortest path computation with Dijkstra's algorithm on the graph where arcs are labeled with reduced costs. The new variable of the discovered path already implicitly exists, and we just compute it on the fly. When it is not possible to improve the LP solution, it will be also not possible to find a path such that the reduced costs are negative.

Note that in our problem the decision variables (i.e. the paths) are not directly present with a coefficient in the objective function but instead appear though pp'_e. And thus, the coefficients of the decision variable do not appear in the pricing problem.

In order to formulate the dual, let us give names to the constraints of the problem. We consider only the constraints that are potentially affected by the introduction of a new column. Let us call ONE_d the constraint given in equation 6.8, CAP_e the capacity constraint given in equation 6.9, and $AND1_e$, $AND2_e$, and $AND3_e$ respectively the constraints in equations 6.14, 6.15 and 6.16. By using these dual values found when solving the RMP, we are able to define the graph of reduced costs for a given demand d. For each edge e and demand d, we have :

$$rcost_d(e) = -CAP_e \times bw(d) + (\sum_{d' \in D} AND1_e + AND2_e - AND3_e)$$

Then the pricing problem for each demand d become now finding a shortest path in the graph of reduced costs, i.e. which minimizes the following formula for a path p defined by its Boolean variables p_e :

$$\min \left(ONE_d + \sum_{e \in E} p_e \times rcost_d(e) \right)$$

Unfortunately, the network labelled with reduced costs has negative cycles and thus Dijkstra's algorithm cannot be used to find a shortest path. Since we are only interested in simple path (i.e. a path without cycle), the pricing problem can be solved through a new Integer Linear Problem by the following flow model. Like before, let p_e be the (continuous) variable associated to the arc e.

The following constraints ensure that only one unit of flow comes out from the source of the demand d and nothing enters in, and the reverse for the destination.

$$\sum_{e=(src(d),y) \in E} p_e = 1 \qquad \sum_{e=(x,src(d)) \in E} p_e = 0$$

$$\sum_{e=(x,dst(d)) \in E} p_e = 1 \qquad \sum_{e=(dst(d),y) \in E} p_e = 0$$

Here are the flow conservation constraints :

$$\forall v \in V, \sum_{e=(x,v) \in E} p_e - \sum_{e'=(v,y) \in E} p_{e'} = 0$$

Then we state non-splittability and no-cycle constraints :

$$\forall v \in V, \sum_{e=(x,v) \in E} p_e \leq 1 \qquad \forall v \in V, \sum_{e=(v,y) \in E} p_e \leq 1$$

The objective becomes :

$$\min \left(ONE_d + \sum_{v \in V} \sum_{e=(v,y) \in E} p_e \times rcost(e) \right)$$

We extract from this flow the minimum path and introduce the corresponding variable.

Solution

A solution for the ILP model when using Column Generation is not equivalent to a solution with the Constraint Programming model. First the approximation introduced by the linearization of the exponential function tend to underestimate the congestion. Thus the value of the objective may be lower for the ILP model even if the solution paths are the same. Second, we solve the ILP problem only when the Column Generation procedure has ended. It may happen that in some cases this procedure does not terminate in a reasonable time. Then the integer solution is not computed and we get no solution.

6.4 Constraint Games

Constraint games for SDN

The MCPRP defined in section 6 can be simply extended to a game by considering each demand as a player who wants to find the best route from source to destination. Then each player wants to minimize her/his own cost as defined in equation 6.3.

If we denote by $S = D^V$ the total search space and by N the set of Nash equilibria, we can define formally the welfare of the best centralized solution adapted to our cost minimization problem by $W^* = \min\{w(s) \mid s \in S\}$. The welfare of the best Nash equilibrium is defined in a similar way by $N^* = \min\{w(s) \mid s \in N\}$ and the one of the worst one by $n^* = \max\{w(s) \mid s \in N\}$. Thus the Price of Stability is simply $PoS = W^*/N^*$ and the Price of Arnarchy $PoA = W^*/n^*$. Note that usually the classical definitions of PoS and PoA yield a result greater than 1, this is not the case here because we have a minimization problem. Note that, we have used IBR as a heuristic to go from the first solution to the first equilibrium In our problem, the social welfare function is simply the global cost to be minimized as defined in equation 6.4. We proceed in two steps. First the best centralized solution is computed as a Constraint Optimization Problem, then the Nash equilibria using our Constraint Games solver. We can immediately see that PoS and PoA are asymetric in term of the relaxation technique we can implement. For PoS, the problem is still a minimization. Thus we can use the same relaxation technique as the one we use in the centralized version (equation 6.7).

For the PoA, we have a maximization problem. But each player still wants to minimize her/his cost. The situation is then to find a set of *shortest* paths of *maximal* global cost. The standard relaxation technique provided by the CP solver provides a loose upper bound for this problem by summing up all upper bounds of the costs of the edges. But we know that the upper bound is at most the cost of the longest path in the residual network. Unfortunately, computing the longest path is NP-complete in the general case, since it corresponds to determine if it exists an Hamiltonian cycle, which is NP-complete [66]. This problem has been already addressed in CP [172] where the authors propose a model and a local search algorithm to solve this problem. In our case, we are interested in a polynomial sound algorithm. This is why we propose to approximate the longest path by a Maximum Spanning Tree (MST) in the residual graph. The MST is computed by considering the upper bound value of the cost of the edges. The algorithm is like Prim's algorithm, we add to all remaining edges the cost of the demand to compute congestion, then we start by taking the most costly edge and add edges linking a new node in descending order of cost. It is clear that the cost of the MST is always greater than the cost of the longest path.

6.5 Related work

Combinatorial methods. SDN allows fast and remote network reconfiguration and thus is a nice source for many optimization problems [130] including online ones. Due to this dynamic aspect and the increasing size of the controlled networks, it is very likely that decentralized algorithms will be mandatory to provide both the expected quality of service and short time response. A survey of most techniques can be found in [134]. Since then, many extensions have been considered like the very important case of demands coming online [91, 165], service provisioning [108], energy-aware routing [109], controller placement [176, 196], fault prevention [213, 36] or even congestion-aware algorithm [211].

Concerning the specific CP framework and to our best of our knowledge, only a few works have been considered : a problem of Service Function Chaining deployment [136] and a general framework providing through CP a high-level programming language to model SDN problems [126].

Our article differs from these methods because first, we propose a CP model taking into account non-linear congestion. Then, we optimize our model by proposing a relaxation technique based on Dijkstra algorithm as well as fast heuristics to solve the problem to optimality.

Quality of service and Game theory

Quality of service (or QoS) is an important problem in SDN and has been addressed in multiples ways. From combinatorial methods with for example with genetic algorithms [177], or even multiples linear programs [212] optimizing multiples criteria such as bandwidth or energy consumption. These criteria concern the whole network which is different from game theory wherein each flow is considered as a criterion. A mechanism design method for multicommodity flow games has been proposed [48] . Nonetheless, Game theory studies have been mainly concentrated with routing games [193] to model uncapacited networks in order to determine how selfish behaviors impact solutions and to quantify it by the price of anarchy[48]. Other mathematical studies on more general networks and solution's degeneration have been done such as on on capacited network [47], unsplittable flow [6]. And even on different model based on distributed games [120, 95].

Our approach is different since we propose a model to compute the PNEs and POA. The two Relaxations are given to fast the exact computation of the PNEs and POA giving a more pratical way to this kind of problems.

6.6 Instances and experimental results

We have tested our framework on a library of instances called SNDlib [155] and a personal problem generator that is able to generate instances close to real ones. This problem and other variants such as robust SDN networks have been studied for a long time [58, 35] with some computational approaches including linear programming [7]. A survey can be found in [142]. Interesting theoretical results have been found, like the one which states that when the problem has a sufficient size and capacity, all flows are actually routed along single-paths [175].

Generator

We have designed a generator to create synthetic problems that allow to test the algorithms against the different hypothesis. Several parameters allow obtaining a great variety of graphs. The generation process is mainly constituted of two phases :

— Generation of the topology, that is nodes as well as arcs and their respective costs ;

— Generation of demands, along with their bandwidth requests, which also determines the capacities of the arcs.

During the generation of the topology, N_{nodes} nodes are created. Each of these nodes n_i with $i \in [1, N_{nodes}]$ is assigned to random coordinates in a fixed size space of $topologyDimension$ dimensions. In case the boolean $topologicalCost$ is set to $true$, the cost of an arc is given by the distance between the source and destination node. Note that for dimension 2, this is not sufficient to ensure that the resulting graph is planar. The size of the space in one given dimension is irrelevant, as we refer to it only with percentage. Each node is also assigned to a degree, randomly chosen in an interval $[deg_{min}, deg_{max}]$. To obtain graphs similar to actual networks, we introduce hubs which are nodes of higher degree than regular nodes. Each node has a probability P_{hub} of being a hub. If a node is a hub, then its degree is randomly chosen in a different interval $[degh_{min}, degh_{max}]$.

We first build a spanning tree over all nodes to ensure that the graph is fully connected, then we create the remaining links in the graph. For each node, we look for candidates, so the desired degree is reached. For a link to be created, we ensure that a) the other node is not already connected with this edge and b) its distance in the space is not greater than $maxDistance$, expressed as a percentage of the space size ($\sqrt{topologyDimension}$ is the maximum). Using this process, it is possible that certain nodes do not reach the desired degree, but as the network grow larger, this situation becomes less and less likely to happen.

Once the topology is generated, $N_{demands}$ are generated. For each of these, a starting node is randomly selected, as well as a bandwidth in a $[bw_{min}, bw_{max}]$ interval. We then generate what we refer to as an "initial path". For that purpose, different strategies are available. The first strategy, called random generation,

Parameter	**Type**	**Range used in the benchmarks**
N_{nodes}	integer	$50, 75, 100, 120, 140, 160, 180, 200, 500$
$topologyDimension$	integer	2
$topologicalCost$	boolean	$true, false$
deg_{min}, deg_{max}	integer	$[1, 2, 4, 8]$, $[2, 5, 7, 10]$
P_{hub}	integer	$0, 1, 5, 10, 20$
$degh_{min}, degh_{max}$	integer	$[25, 50, 75, 100]$, $[25, 50, 75, 100]$
$initbw_{min}, initbw_{max}$	integer	$[50, 100, 200]$, $[50, 100, 200]$
$initcost_{min}, initcost_{max}$	integer	$[100, 200]$, $[200, 500]$
$maxDistance$	integer	$25, 50, 100$
$N_{demands}$	integer	$30, 50, 100, 120, 130, 150, 200$
bw_{min}, bw_{max}	integer	$[50, 100, 200]$, $[50, 100, 200]$
hop_{min}, hop_{max}	integer	$[0], [10]$
P_{bw}	integer	$[10, 30, 50, 70, 90, 100]$

Table 6.1 – Parameters of the generator

consists in selecting a random number of hop h in the $[hop_{min}, hop_{max}]$ interval, and randomly navigating in the graph for h hops, starting from the initial node. The last node is then considered to be the destination node of the request. During the navigation, we only make sure to never reach a node that is already in the initial path. The second strategy consists in randomly selecting a destination node, and applying a shortest path algorithm to find the path from the source node to the destination node with the least number of hops. The path yielded by the algorithm is considered to be the initial path. Regardless of how the initial path is constructed, for each of its arc, there is a probability P_{bw} that we increase its capacity of the amount of bandwidth of the request. The list of all generation parameters, as well as short description can be found in Table 6.1.

Settings and implementation issues

Experimentation settings

Due to the large number of parameters of the generator, we have applied a benchmark method called combinatorial testing [151, 102] using the ACTS software [227]. This technique allows for p parameters and a size c to generate a set of instances where all possibilities of combinations of parameters of cardinality c are inside the set. For example, if we have 3 Boolean parameters a, b and c, a complete test of all possibilities would require $2^3 = 8$ tests. But if we decide to test only all combinations of pairs, we can achieve this with only 4 instances (see

Figure 6.12). With our generator, by choosing an appropriate sampling of the

a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

a	b
0	0
0	1
1	0
1	1

a	c
0	0
0	1
1	1
1	0

b	c
0	0
1	1
0	1
1	0

Figure 6.12 – All pairs of parameters are covered by 4 tests

intervals described in 6.1, we get roughly 500 instances to get a covering of all 3-sets of parameters. From these 500 instances, we have discarded those whose resolution lead to a timeout for all techniques. This gives a total of 123 instances which give a meaningful picture of the range of problems that can be solved.

The tests have been performed on a cluster of Intel Xeon E5-2690, each having 10 cores sequenced at 3GHz and 256 GB of RAM. We have computed experimental results for the CP approach described in Section 6.1 and the Constraint Game model of Section 6.4 with a timeout fixed at 1 hour.

A note on implementations

The CP model has been implemented using the Choco solver [174], including the Constraint Game through our Choco extension called Conga [159]. Besides the search techniques and different heuristics and the relaxation technique described in Section 6.2, our first implementation was using the Ibex solver [38, 39] in association with Choco. Real variables linked to Ibex were used to model the computation of the load, congestion and costs while discrete decision variables remained in Choco. However, this was not efficient because the two solvers need to communicate through Java Native Interface. In addition, many auxiliary real variables and constraints(e.g. constraint for the congestion cost and auxiliary variable for the sum) were used to compute intermediate values through constraint propagation despite this part is purely functional. In addition, the cost is also obtained as a by-product of the shortest path algorithm since at the end of the search tree the sum over the demands costs computed by the Dijkstra algorithm is the real cost. Therefore, we have replaced all the auxiliary variables and constraints computing the objective value by a single global constraint which also encapsulates the Dijkstra algorithm. At the end, the model only contains path and capacity constraints and the global constraint computing the objective.

The Column Generation model has been implemented using CPLEX [49] version 12 with its Python interface. At first we tried to post all constraints at the problem initialization for all possible paths. However, it was not efficient since

it takes a lot of time to initialize. Since most of the constraints (i.e. consider all the edges in the graph) are useless to solve the problem, we instead choose to post the constraints on the fly along each path when it was required after having generated the columns.

Once again, in the following, when we are saying that an instance is solved it has different meaning when we are talking about CP or CG. For CP, an instance is solved when the optimal solution has been found and proved. For CG, the instance is considered as solved when the generation procedure is finished and the ILP problem solved within the generated columns. Because of that, Column Generation does not prove optimality. Because of the linearization, the objective values are most of the time different between the two techniques.

Experimental results

Constraint programming model

For the synthetic benchmarks, we have displayed the results in Figure 6.13. As a preliminary test, we have tried the pure `CP` heuristic based on impact [180] to measure the gap with the `SP` value heuristics. A problem which should be easy (13 nodes and 9 demands) is solved in less than 1 second by using the shortest path strategy, whereas the impact strategy took 878.969 seconds. Due to this, we have not displayed this `CP/CP/CP` heuristics in the figure and we only present results for the `SP` strategy.

For each instance, we have run the combinations `MB/SP/CP` and `MB/SP/SP`, and the two conflict variants `CO/SP/SP` and `CO1/SP/SP`. The plot in Figure 6.13 shows how many instances are solved in a specific delay. Clearly, the `MB/SP/SP`

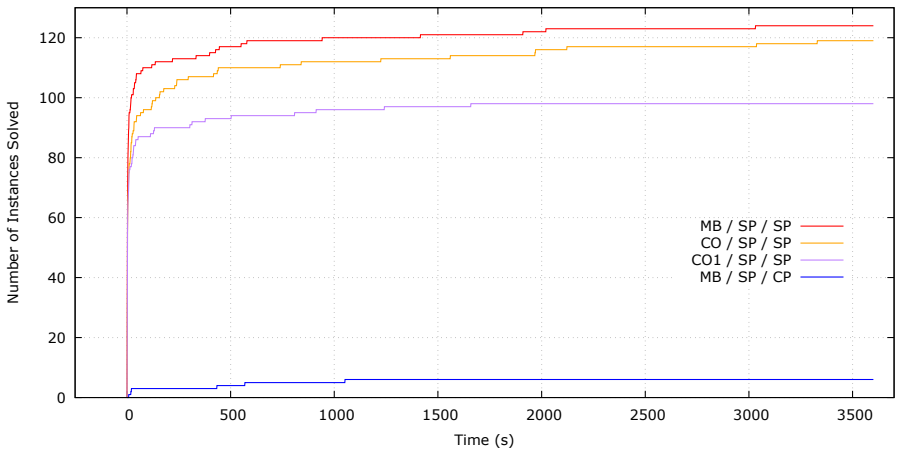


FIGURE 6.13 – Comparison of different CP heuristics on synthetic benchmarks

instance	#Demands	#Nodes	#Edges	MB/SP/CP	MB/SP/SP
SAT					
dfn-bwin	90	10	45	*TO*	0.670
dfn-gwin	110	11	47	*TO*	0.746
di-yuan	22	11	42	*TO*	0.472
giul39	1471	39	172	*TO*	26.554
india35	595	35	80	*TO*	8.739
newyork	240	16	49	*TO*	3.486
nobel-eu	378	28	41	*TO*	4.919
norway	702	27	51	*TO*	7.962
pdh	24	11	34	*TO*	0.553
UNSAT					
geant	462	22	36	2.729	2.550
germany	662	50	88	6.489	6.241
janos-us	650	26	84	3.508	3.605
janos-us-ca	1482	39	122	25.926	25.926
UNKNOWN					
france	300	25	15	*TO*	*TO*
pioro40	780	40	89	*TO*	*TO*
polska	66	12	18	*TO*	*TO*

Table 6.2 – Results of the Constraint Programming model on real-world instances from SNDlib

heuristics outperform the other ones. This is not surprising compared with the CP-style B&B, but it shows that a more dynamic heuristic based on conflicts is not effective on this type of problems. We also compared the performances of the different strategies on the unsolved instances. Since, the solver did not finish either because it did not prove the solution's optimality, or because it did not find any solution, the only valuable comparison is the current solution and the time to find a first solution.

The performances for finding a first solution of the different strategies are shown in Figure 6.14. This figure presents the time required to find a first solution for all instances and given the three CP strategies. The instances are sorted by increasing time. As we can see, the strategies are very good at finding a first solution and on most of the instances. The strategies provide comparable performances when the goal is to find a first solution. Another interesting comparison is about the performances of the strategies on unsolved instances for getting the best solution. The table 6.3 presents how many times a strategy has found the best current solution while it timeout. `MB` is the strategy finding most of the time the best solution after a timeout.

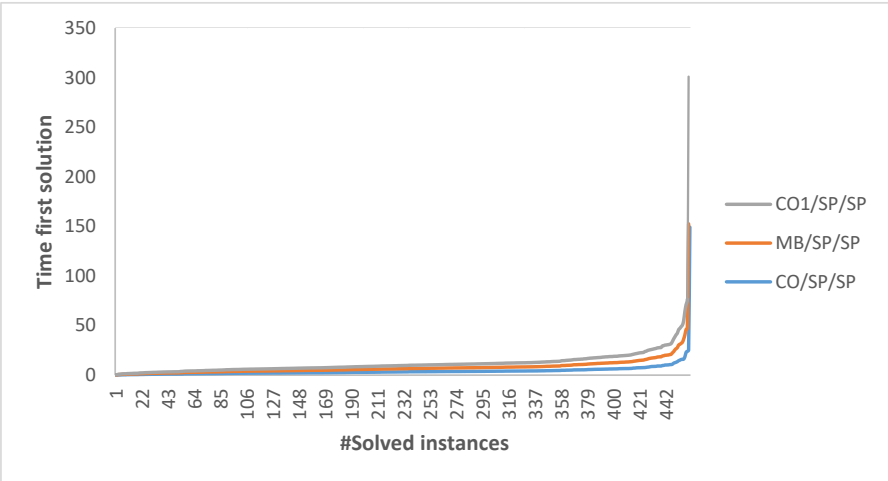


FIGURE 6.14 – Comparison of the strategies for finding the first solution

	MB/SP/SP	CO/SP/SP	CO1/SP/SP
# best sol	340	321	303

TABLE 6.3 – Comparison of solutions on the unsolved instances

It appears that these instances are hard for multiples reasons. First it not a simple parameter which makes those harder. An instance can be hard even with 30 nodes and 30 demands. A problem become hard when the back-propagation of the relaxation is not enough and requires a lot of search. This effect is visible in the Figure 6.13, while comparing the instance which benefit from the SP relaxation against the one using the CP one. Most of the time the strategies take time to prove the optimality of a solution.

In the following, we only compare with the `MB/SP/SP` heuristic. The time in seconds for real instances from the SNDlib are shown in Table 6.2 only for two heuristic combinations. The improved relaxation technique allows to solve many instances to optimality. Each instance is described by its name (which corresponds usually to a network in a particular country), then the number of demands, nodes and links of the network. The SAT and UNSAT instances are the one for which we can find the optimal solution or prove unsatisfiability. For some instances depicted as UNKOWN, our method was unable to find the optimal routing. But still the best solution can be reported.

ILP model with Column Generation

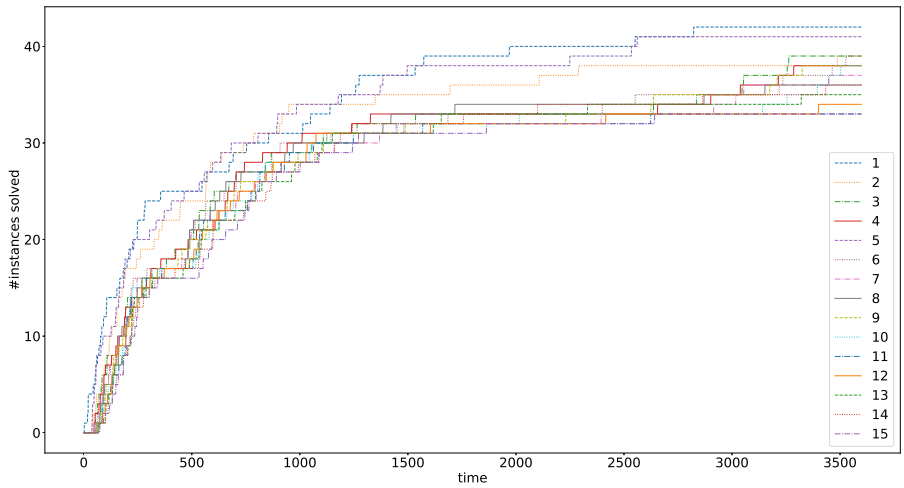


Figure 6.15 – Comparison of different initialization of the Column Generation model

Instance	#Demands	#Nodes	#Edges	CG	MB/SP/SP	Obj $\frac{CP}{CG}$
SAT						
dfn-bwin	90	10	45	456.742	0.670	1,000067618
dfn-gwin	110	11	47	*TO*	0.746	N/A
di-yuan	22	11	42	*TO*	0.472	N/A
giul39	1471	39	172	*TO*	26.554	N/A
india35	595	35	80	*TO*	8.739	N/A
newyork	240	16	49	*TO*	3.486	N/A
nobel-eu	378	28	41	*TO*	4.919	N/A
norway	702	27	51	*TO*	7.962	N/A
pdh	24	11	34	108.3804	0.553	1
UNSAT						
geant	462	22	36	*TO*	2.550	N/A
germany	662	50	88	*TO*	6.241	N/A
janos-us	650	26	84	*TO*	3.605	N/A
janos-us-ca	1482	39	122	*TO*	25.926	N/A

Table 6.4 – Column Generation results on real-world instances from SNDlib

We have run the ILP model on the same synthetic instances as the CP model. In our model, the number of initial columns can be parametrized. We show in Figure 6.15 a cumulative plot comparing the number of instances solved with different initializations. Each method starts with a different number of path from 1 to 15. It appears that starting with an unique path gives better performances.

A reason which can explain this behavior it that if too many path are generated at start, many constraints have to be added and it slows down the initial simplex iterations and the next ones for each demands.

The real instances from SNDlib are shown in Table 6.4 along with the ratio of objective value between CG and CP. Due to many timeouts, there is no meaningful conclusion to be analyzed.

Constraint programming against Column Generation

To get a very synthetic insight of the respective strengths of the two approaches, we have depicted a set of comparisons in Table 6.5. It simply shows how many times each method has found a better solution or finished the resolution before the other. From this table, we can see that in general the CP model performs better, but not all the time.

	#better run-time	**#better solution**	**#better run-time and s**
Column generation	11	17	0
Constraint Programming	124	382	11

Table 6.5 – Comparison between Column Generation and CP

#Node	**#demands**	**time CP**	**time CG**	**solution CP**	**solution CG**
120	80	*TO*	1906.63	707329	704229
180	90	*TO*	2245.03	956690	897772
180	50	*TO*	2095.51	708083	683188
100	30	2.29	*TO*	4564010	*TO*
500	10	3.812	547.638	131182	133039
40	20	1.87	208.979	84291.7	86397.5
75	200	*TO*	182.183	914526	*TO*
200	40	*TO*	1651.83	707515	713809

Table 6.6 – Comparison between Column Generation and CP

Furthermore, we extracted some meaningful synthetic instances presented in Table 6.6. These instances present different kinds of behavior. In a few instances, the CG approach is able to find a better solution in shorter time. Sometimes the CG generation procedure times out and we are not able to find an integer solution in the given time. But interestingly, even if the CG has finished his generation procedure and the CP model times out and fails to prove optimality, it happens

that the integer solution found by CG is still worse than the one returned at the end by CP.

We can see that very often CP performs better. One possible explanation about the bad result of Column Generation is how the RMP is linearized. When a new variable is entering the problem, the objective constraint is not modified. And thus the computed reduced cost are less efficient to improve the linear solution. This in turn slows down the global resolution by forcing the generation procedure while it is not required. In addition, the generation of one column needs to solve a NP-complete problem and it has to be embedded in ILP since no shortest path algorithm is applicable due to the negative cycles (see section 6.3). In contrast, the CP model benefits from a good heuristic which guides well the search space exploration, and moreover it has a good relaxation to bound the objective value when the search tree is explored in order to close the nodes.

Constraint games

For the Constraint Game model, we have only used the combination `MB/SP/SP`, with and without improvement of the first solution by IBR. Results show that IBR improves relaxation technique by giving quickly a good first solution which is also an equilibrium.

instance	#Demands	#Nodes	#Edges	MB/SP/SP	[NASH] MB/SP/SP
SAT					
dfn-bwin	90	10	45	0.670	3.871
dfn-gwin	110	11	47	0.746	5.681
di-yuan	22	11	42	0.472	2.012
giul39	1471	39	172	26.554	1571.197
india35	595	35	80	8.739	215.716
newyork	240	16	49	3.486	18.173
nobel-eu	378	28	41	4.919	41.861
norway	702	27	51	7.962	154.520
pdh	24	11	34	0.553	2.016
UNSAT					
geant	462	22	36	2.550	2.92
germany	662	50	88	6.241	6.783
janos-us	650	26	84	3.605	5.174
janos-us-ca	1482	39	122	25.926	50.486

Table 6.7 – Constraint Games results on real-world instances from SNDlib

We present in Table 6.7 the run-time in second of the different strategies on the SNDlib instances. It is interesting to see that games of unprecedented size

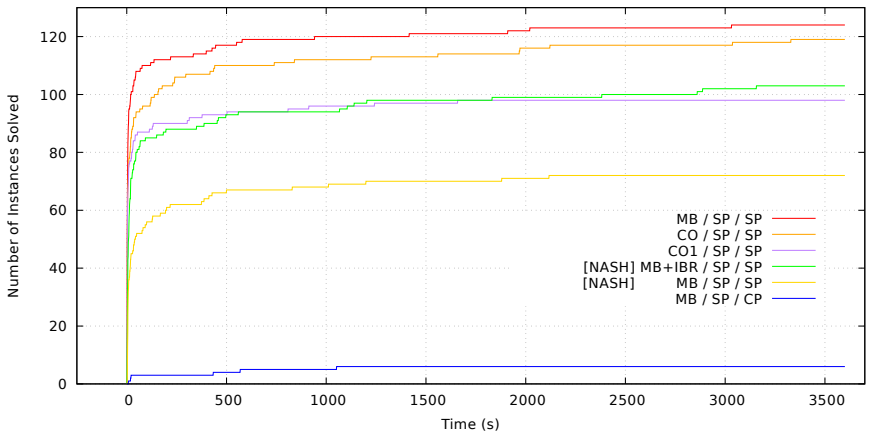


FIGURE 6.16 – Comparison Nash and and the different heuristics on synthetic benchmarks

(up to 1482 players in the *janos-us-ca* instance) can be solved to optimality by Conga [159]. Interestingly and in contrast with the synthetic instances, we have observed that IBR slightly degrades the computation time, this is why we did not include the column in the table. We believe that in these problems, most first solutions computed by the `MB` heuristics were already at equilibrium, and thus adding IBR only adds another check.

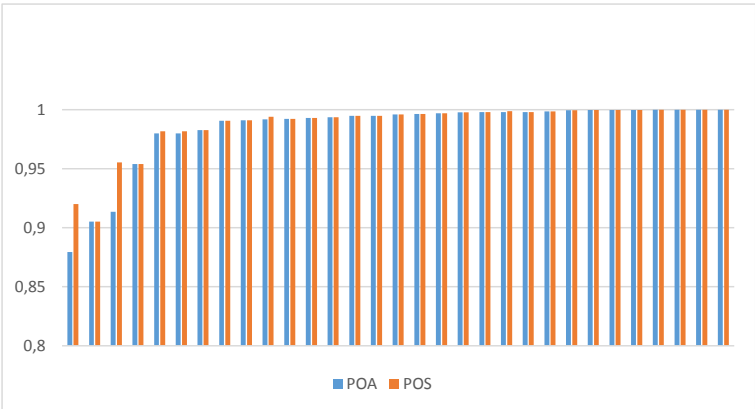


FIGURE 6.17 – Price of Anarchy and Price of Stability for small synthetic instances

We report the results for the computation of PoA and PoS for small synthetic instances in Figure 6.17. In most instances, we observe that the PoA and PoS are very close, and also very close to the centralized optimum. It means that on these problems, a decentralized algorithm would be very interesting to implement if we assume it scales up to larger problems. We have used much smaller instances

because the PoA is very difficult to reach. The upper bound computed for the Maximal Spanning Tree overestimates the longest path which also overestimates the longest shortest path. We pay these two approximations by a limited pruning of the search tree which has a major impact on the computation time.

6.7 Conclusion

This chapter includes two practical contributions. First we have modeled and solved efficiently the unsplittable multi-commodity flow routing problem with congestion in Constraint Programming and in ILP with Column Generation. We have provided an accurate branch and bound technique that allows to solve real-world size instances up to optimality. Our third contribution is a Constraint Game model that allows to evaluate the potential of decentralized routing in this context. We have found all Nash equilibria for problems with thousands of player thanks to the Constraint Game solver Conga. This is the first time that such large instances are solved up to optimality by a general-purpose Game Theory solver.

Chapitre 7

Conclusion and perspectives

Contents

7.1 Conclusion **107**
7.2 Perspectives **108**

In this chapter, we summarize the contents and the contributions of the thesis by discussing the results along with some possible directions for the future work.

7.1 Conclusion

This thesis revisits the constraint game framework which was firstly studied in [148]. The baseline formalism of constraint games was provided through this previous work. In this thesis we are going further by studding two axes of work : the computation of Nash equilibrium and the resolution of real world application. This thesis provides a more practical interest in multiples points :

— **Modelling and solving games.** Many frameworks have been proposed to solve games. From Dedicated approaches like Action-Graph Games [113], or more recently others paradigms like Boolean games [92] or Constraints Games [150] based on combinatorial methods have been proposed. Besides a concentration of interest into these approaches, modeling and solving games is not an easy task. All the cited methods rely on ad-hoc algorithms. They do not benefit from the recent advances of solvers. The way to see Constraint Games as a global constraint presented in this thesis(see chapter 5) allows modeling transparently games into solvers. In addition, to this new view, we have characterized the complexity of the arc consistency for the Nash constraint, and we have proposed an improvement to the algorithm for finding Nash equilibriums in a game. This new incremental

algorithm infers never best responses as soon as possible in the search tree by back-propagating the objective value of each player.

— **Applications.** Constraint games solver : ConGa has been used to solve real networking problem (see chapter 6). Especially, in such applications, a Nash equilibrium solution is really desirable. Its properties make the users satisfied with their situation, hence seen as a high quality of service for multi-user applications. In order to speed the resolution, we have proposed dedicated techniques such as a specialized branch and bound and two heuristics based on the problem's structure. These techniques have helped us to solve games with many players and strategies by improving the solving process's efficiency.

— **Solver release** [1]**.** A solver for constraint games with its manual has been released. It includes explanations to use it and how to extend it. This is the first release of such solver in the community.

Several parts of the work in this thesis have been validated by accepted papers in proceedings of international peer-reviewed conferences :

— Palmieri, Anthony, and Arnaud Lallouet. "**Constraint games revisited**." International Joint Conference on Artificial Intelligence, IJCAI 2017. 2017.

— Lallouet, Arnaud, and Anthony Palmieri. "**Constraint Games for Modeling and Solving Pure Nash Equilibria, Price of Anarchy and Pareto Efficient Equilibria**." 13th European Meeting on Game Theory, SING 2017. 2017 .

— Palmieri, Anthony, and Guillaume Perez. "**Objective as a Feature for Robust Search Strategies**." International Conference on Principles and Practice of Constraint Programming. Springer, Cham, 2018.

— Palmieri, Anthony, Arnaud Lallouet and Luc Pons. "**Constraint Games for stable and optimal allocation of demands in SDN**." International Conference on Principles and Practice of Constraint Programming. Springer, Cham, 2018.

— Palmieri, Anthony, Arnaud Lallouet and Luc Pons. "**Constraint Games for stable and optimal allocation of demands in SDN**." Constraints Journal 2019, to appear.

In conclusion, our contributions are first to give a new view of constraint games and in a second hand to prove the solver's efficiency by tackling a networking problem.

7.2 Perspectives

This thesis thesis has been guided by practical aspects yielding to the solver's implementation and multiples improvements to model and to solve games. During

1. https ://github.com/palmieri-a/CONGA

this work, we had think about multiples future work. For instance, in the Location Game (see Example 19), what happen if two vendors have to share a stand. This can arise if a conglomerate decide to build a shopping mall. The location of both store have to be decided conjointly. The computation of one equilibrium, emphasizing the role of a good heuristics and opening comparisons with best response dynamics [37]. Dedicated heuristics taking advantage of games structure is still an open question and belongs to the perspectives of this thesis. Also, the extension to new solution concepts like the mixed version of the Nash equilibrium is an important step for the solver adoption and for Constraint games.

Mixed Nash equilibrium. Finding a mixed Nash equilibrium is a hard task and has been shown recently NEXP-hard [111]. Like underlined in [153] computing a Mixed Nash equilibrium requires to have all the valuation of each action in order to apply well-known methods. However, it requires an exponential memory space which is intractable for large games that why constraint games are a new way to think about that. The Mixed Nash equilibrium can be an extension of the solver. A mixed Nash equilibrium is simply a maximization of the expectation for all players separately. A way to do that would be to estimate the expectation, by sampling for instance (as in [8]). A challenge could be the complete enumeration of the mixed Nash equilibrium with this technique which has to be complete for this purpose. Also, a good compromise between the accuracy of the estimation and the time spent has to be found in order to propose a competitive method. Additionally, the other solution concepts could be investigated, like the one for Cooperation or Strong Nash equilibrium.

Shared variables. Originally in games, players exercise unique control over a set of variables (or strategies) in order to achieve a unique goal. It may happen however that the unique control is too restrictive as in the meeting scheduling game (see Example 37).

Example 37 (Meeting scheduling) *This problem is an adaptation from [140] A set of agents want to schedule a set of meetings. Each has a set of compulsory attendees while a set of other participants are encouraged to assist. Each meeting has a starting time and a fixed duration. The starting time of each meeting is defined by the agents participating in it. The meetings have to respect some ordering constraints. For instance, a meeting has to start before noon due to participants in other time zones. A natural way is to let the attendees of a meeting control a unique decision variable. The variable ownership is going to be shared. Each agent is part of the decision process to specify the starting time of each meeting where he is involved. But many agents can be required for the same meeting and thus have to share the same variable.*

However, a model expressed with a shared variable has no solution most of the time. The problem is that players sharing a variable have to agree completely on a solution. Which is very rare in practice.

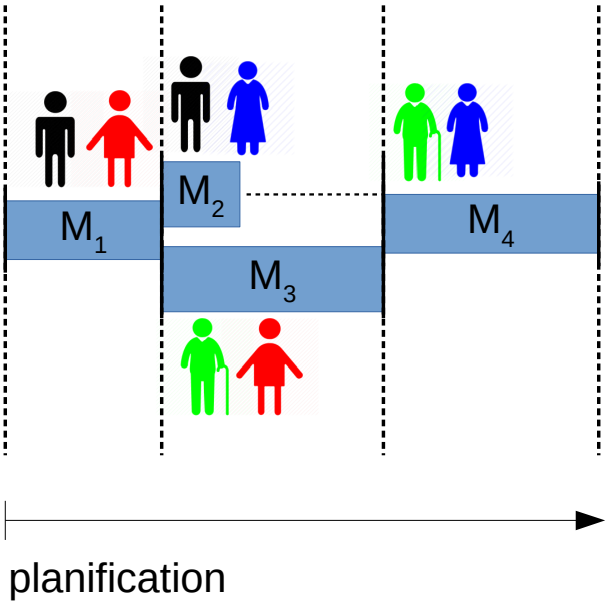


Figure 7.1 – An instance of the meeting scheduling

For example, the simple instance of meeting scheduling shown in Figure 7.1 does not have a Nash equilibrium. In this situation, if the players do not compromise together then the meeting M_2 is problematic. The black player would like to put it just after M_1, while the blue one wishes to put it next to M_4. Relaxation such as voting systems, partial cooperation can be studied in order be able to provide a solution. The challenge of this new concept is not limited to only the solution concept since it requires to be computed as fast as possible because of the potential large instances. The propagators can be studied in order to prune the search space and give better performances.

In summary, we are confident that an efficient solver as much as possible complete, open source, would be very appreciated in practice and by the community.

Annexe A

Objective criteria for robust search strategies

In this section we give the work done in this thesis but not directly related to the subject.

A.1 Objective Function and Search Strategy

Search strategies aim to reduce the search space, but additionally aim to find good solutions as quickly as possible. Most SSs choose the hardest variables to satisfy first, the main challenge being to find such variables. While most SSs decisions were based on variables domains, the constraint graph, etc, objective-value based decisions are rarely done in CP. One of the reasons is that, in CP, we cannot easily back-propagate the objective to the variables to make decisions as done in Mixed Integer Programming. But even if we can not have such exact information, not taking into account the variables impacting the objective value can lead to an exponential loss in time. This is shown by the following synthetic example.

Example Consider a COP having $n + m$ variables and whose objective is the sum of the last m variables. This problem has an AllDifferent constraint [182] over all the variables. Ignoring the objective value can lead to the search tree shown in Figure A.1 (left). In this example, the strategy focuses only on other features, without taking the objective into account. Whereas a strategy that considers the objective detects variables having high impact on the objective, and consider them earlier enabling a potential reduction of the search tree.

Moreover, we can find high quality solutions earlier and these solutions prune the search space more efficiently. As we can see, the processing of the m variables

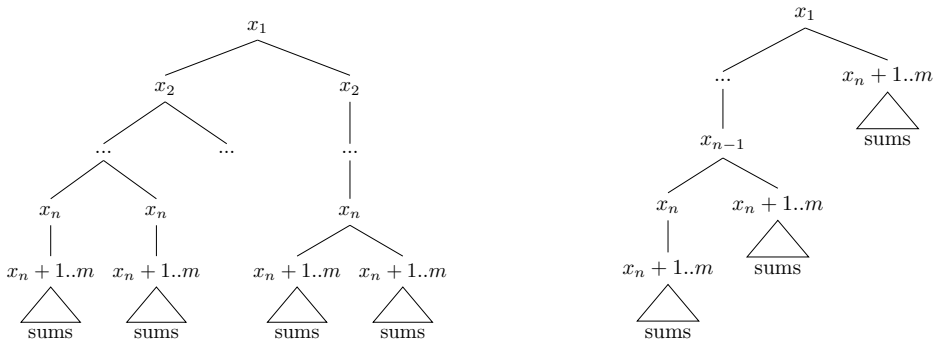


Figure A.1 – (left) A search failing to consider the objective value. (right) An objective based search.

is repeated an exponential number of times (d^n). This is because the variables that impact the objective are chosen too late leading to a bigger search tree.

The search tree using the objective value as a feature is shown in Figure A.1 (right). The last m variables are selected higher in the search tree, yielding better solutions faster and allowing to close the search using the objective sooner. Finally, by using the objective value, we obtain a smaller search tree.

This simple example shows that the objective value allows to assign variables having high influence on the objective earlier and thus can help the solver to avoid considering useless parts of the tree. The idea is to consider as soon as possible the variables impacting the objective. We now define a new feature based on the objective, which we will use to define an objective-based search strategy.

Objective modifications as a feature

The proposed feature focuses on the objective bounds modifications by using a function Δ_O. The upper and lower bounds are separately considered as two different pieces of information. Let O be the objective variable to optimize. Let s and $s-1$ be respectively the current and the previous node of the search tree. Let $\overline{\Delta_O}$ (resp. $\underline{\Delta_O}$) be the upper (resp. lower) bounds difference between its value before and after the decision propagation. The function is defined as follows :

$$\Delta_O(s) = a \times \underline{\Delta_O} + b \times \overline{\Delta_O}$$

We choose to consider the upper and lower bounds separately. The choice of the parameters a and b defines the function behavior. The coefficients can take any value and correspond to the importance (positive or negative) given to each bound. For instance, in minimization problems, the coefficient a of lower bound modification corresponds to the weight for the consideration of removing the best

potential solutions. While, the upper bound modification coefficient b, represents the weight to consider the deletion of the worst potential solutions.

Note that this function has a more fine-grained description of the objective than usual measures used in search strategies. Classic SSs monitor the modifications of the decision variables, but in general, treating differently the lower and upper bounds, has no meaning for such variables.

Objective-Based Selector (OBS)

We propose a new variable selector based on the Δ_O function : *OBS*. OBS first selects the variables having the highest impact on the objective with regard to the Δ_O function. To do so, the weighted sum of the Δ_O function values for each $x \in X$ is monitored through $\widetilde{\Delta_O}(x)$, and updated after each decision involving the variable x. The parameter γ is the degree of weighting decrease of the exponential moving average. The updated value $\widetilde{\Delta_O}'(x)$ is processed as follow :

$$\widetilde{\Delta_O}'(x) = \frac{\widetilde{\Delta_O}(x) * (1-\gamma) + \gamma * \Delta_O(x)}{\gamma}$$

At each decision, OBS selects the variable $x \in X$ such that $\forall y \in X, \widetilde{\Delta_O}(x) \geq \widetilde{\Delta_O}(y)$.

Example

Consider the didactic COP defined by the variables (x_1, x_2, x_3, x_4) having each as domain $D = [1, 4]$ and an *AllDifferent* constraint on the 4 variables. The COP's objective is $\min x_3 + x_4$. We use the parameters ($a = -1$,$b = 1$) for the Δ_O function, in order to penalize lower bound modifications and reward upper bound modification.

The tree search from Figure A.2 shows the application of the objective based search strategy versus a lexicographic search. In this example, when a variable is selected, it is assigned to its minimum value. The lexicographic search on the left has more decisions than OBS on the right because it cannot identify which variable are important to satisfy the constraints and improve the objective.

At the beginning of the exploration, in the right tree showing *OBS* search, the variables x_1, x_2 and x_3 are selected and set to their minimum values. Each of these assignments has an effect on the objective's bounds and thus modifies Δ_O. When the decision $x_1 = 1$ is propagated, $\Delta_O(x_1)$ is set to -2 because of the changes of the objective domain from $[2, 8]$ to $[4, 8]$. The propagation of $x_2 = 2$ reduces the objective's domain from $[4, 8]$ to $[6, 8]$ implying $\Delta_O(x_2) = -2$. When the variable x_3 is selected, the objective is instantiated to 7. This implies a $\Delta_O(x_3) = 0$. A solution is found with the value 7, so the next solution has to be smaller than 7. Afterwards the decision $x_3 = 3$ is refuted and the search tree is backtracked to the decision $x_2 = 2$ which is refuted, implying $x_2 \neq 2$. Then x_3 which has the

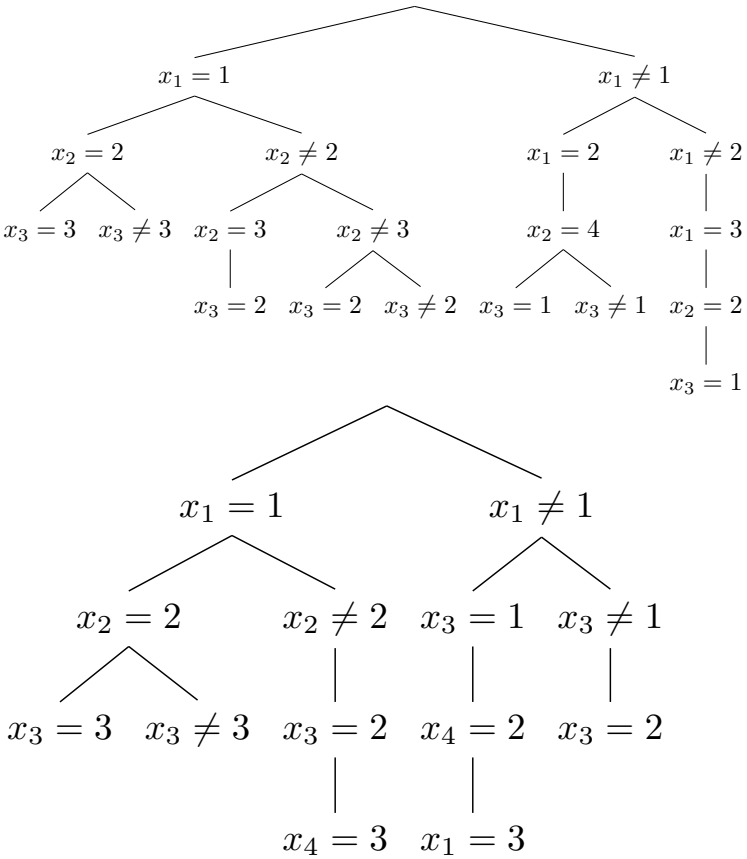


Figure A.2 – Comparison of the search tree by a lexicographic search (top) and an objective based search (bottom)

highest Δ_O value is selected and instantiated to its domain's minimal value : 2. Then x_4 is the next free variable with the highest Δ_O value, 0. We thus select x_4 and assign it to its smallest value, 3. We find a solution equal to 5. Finally, when this branch is closed, the decision $x_1 = 1$ is refuted and by applying the *OBS* selection, the branch leading to the best solution is explored. An important aspect of this search is that it is close to human intuition to choose first variables x_3 and x_4 since they belong to the objective.

Note that the maintenance of $\widetilde{\Delta_O}$ values and the selection process are simple and not intrusive in solvers. Moreover, *OBS* does not need to change the constraints implementations.

Hybridization of search strategies

In this section we show how the objective and classical features can be combined together, to benefit from both. But most strategies should not be

directly combined due to the range differences of their feature. For example, the *IBS* strategy has a value range between $[0, 1]$, while *ABS* one is between $[0, n]$. We propose to normalize all these values to fit in the interval $[0, 1]$ in order to combine them. Note that this applies to the Δ_O function as well.

Let $\widetilde{S^n}(x)$ be the normalized value of a search strategy S based on a classical feature. And let $\widetilde{\Delta_O^n}(x)$ be the normalized values for *OBS*. We combine the two pieces of information with the following formula :

$$S_O(x) = \alpha * \widetilde{S^n}(x) + (1 - \alpha) * \widetilde{\Delta_O^n}(x)$$

The hybrid search strategy selects the variable maximizing S_O. The values α and $(1 - \alpha)$ represent the importance given to each feature. Note that α is in $[0, 1]$.

Example with ABS_O : While the *ABS* strategy uses the $\widetilde{A}$ values, storing the activities involved by the variables, our modification of the value associated with each variable is the sum :

$$ABS_O(x) = \alpha * \widetilde{A^n}(x) + (1 - \alpha) * \widetilde{\Delta_O^n}(x)$$

This $ABS_O(x)$ value contains both pieces of information : the activity and the objective modifications.

Remarks : The hybridization of many others strategies is as simple as for *ABS*. For the following sections, we respectively denote the hybridized versions of *ABS*, *IBS* and *WDeg* by ABS_O, IBS_O and $WDeg_O$.

A.2 Experiments

The Experimental Setting

Configurations All experiments were done on a Dell machine having four Intel E7-4870 Intel processors and 256 GB of memory, running Scientific Linux. We implemented these new strategies in the Choco 4 CP solver [174]. Each run used a time limit of 30 minutes. The strategies were warmed up with a diving step, using up to 1000 restarts, or by ensuring a certain number of decisions. The same warm up (method and seed) was used for all the methods, in order to avoid any bias.

Benchmarks The experimental evaluation used on the MiniZinc Benchmark library[147], with benchmarks that have been widely studied, often by different communities, including *template design*, *still life*, *RCPSP*, *golomb ruler*, etc. Many problem specifications can be found in [71]. Every class of optimization problems from the MiniZinc library has been considered. Since the number of instances per family is huge, and has a large variance between families, we have randomly

selected up to 10 instances per family. Such subset selection preserves the diversity of instances, and do not favor a specific kind of family in plots. The problems have been translated into the FlatZinc format, using the MiniZinc global constraints library provided by Choco-solver, which preserves the global constraints.

Plots The scatters and curves presented in this section are in log scale. A scatter plot shows the comparison of two strategies instance by instance. The diagonal separates the instances where each method has performed better than the other. The points above (resp under) the line correspond to the instances where the ordinate (resp the abscissa) strategy is less efficient. Larger is the gap between the axis line and the point, bigger is the difference between the strategies. Extreme points above and on the right correspond to the timeouts.

Terminology An instance is said to be *solved*, when the best solution has been found and its value proved to be optimal. The term *solution quality* is used when the search is incomplete, and only the best found solution can be judged.

OBS evaluation

Once again, the *OBS* selector is highly configurable : each bound can have its own coefficient impacting the selection process. The running time of several configurations with different bounds importance have been profiled. The values -1, 1 and 0 have been tested to respectively give : negative, positive, or no importance to the considered bound. All possible pairs of (a,b) from $(-1, 0, 1) \times (-1, 0, 1) \backslash \{(0, 0)\}$ have been tested.

The performance of different *OBS* parameters are shown in Figure A.3. This cumulative plot shows how many instances can be solved by each method, for a given time limit. This plot shows that a negative cost to the lower bound outperforms zero or positive cost, regardless of the upper bound.

Configurations weighting the lower bound negatively solve approximately 50 more instances than the alternatives. The solution quality has been compared as well : Figure A.4 shows how many time a search has found the best solution (not necessarily optimal) compared to its alternatives. Once again, the searches weighting the lower bound negatively show better results. One intuitive explanation is that choosing the variables impacting the less the bound which has to be optimized, concentrates the search into the most promising parts and like shown in Example A.2 helps to back-propagate the objective to prune the tree search. Furthermore, the upper bound in optimization problems (here minimization, without loss of generality) does not have a big impact on resolution time. In addition to our previous intuitions, the upper bound seems to be very sensitive to initialization and to propagation. For instance in some constraints such as *sum*, which often determines the objective value, no arc consistency can be achieved in polynomial time. But, often, only the bounds are filtered, making less

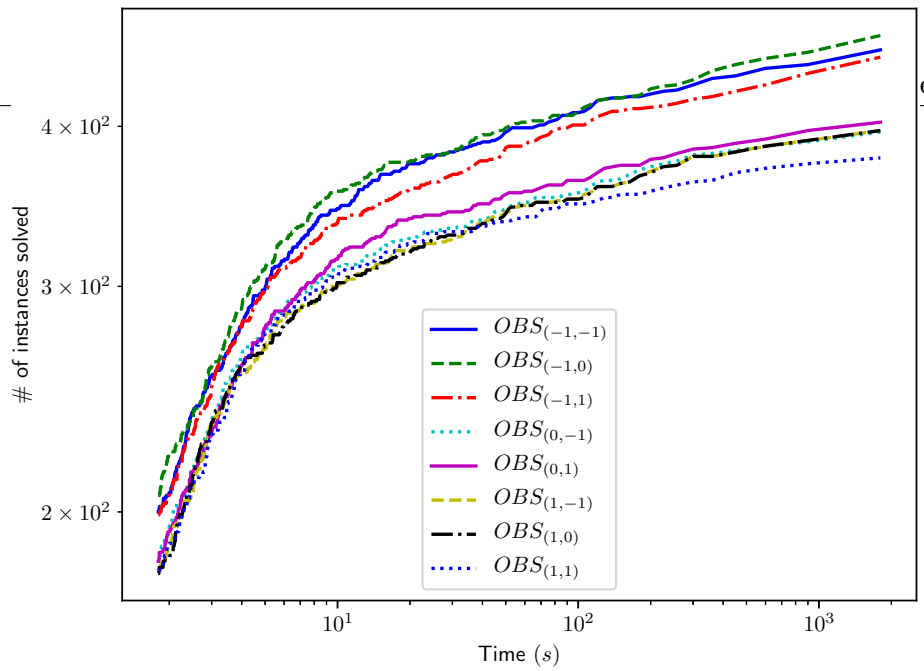


FIGURE A.3 – Comparison of the number of solved instances for different *OBS* configuration.

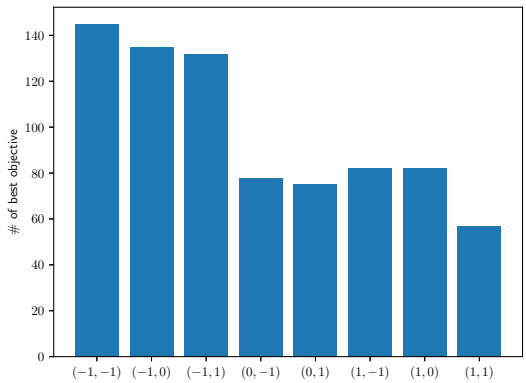


FIGURE A.4 – Comparison of the objective quality between different OBS configurations.

consistent the variations of this variable. Based on different OBS experiments, the configurations *(a=-1,b=0)* and *(a=-1,b=-1)* seem to be the most promising.

Evaluation of Hybrid Strategies

We tried the hybridization with all the *OBS* configurations in order to select the most promising one. The configuration $(a = -1, b = 0)$ got better results

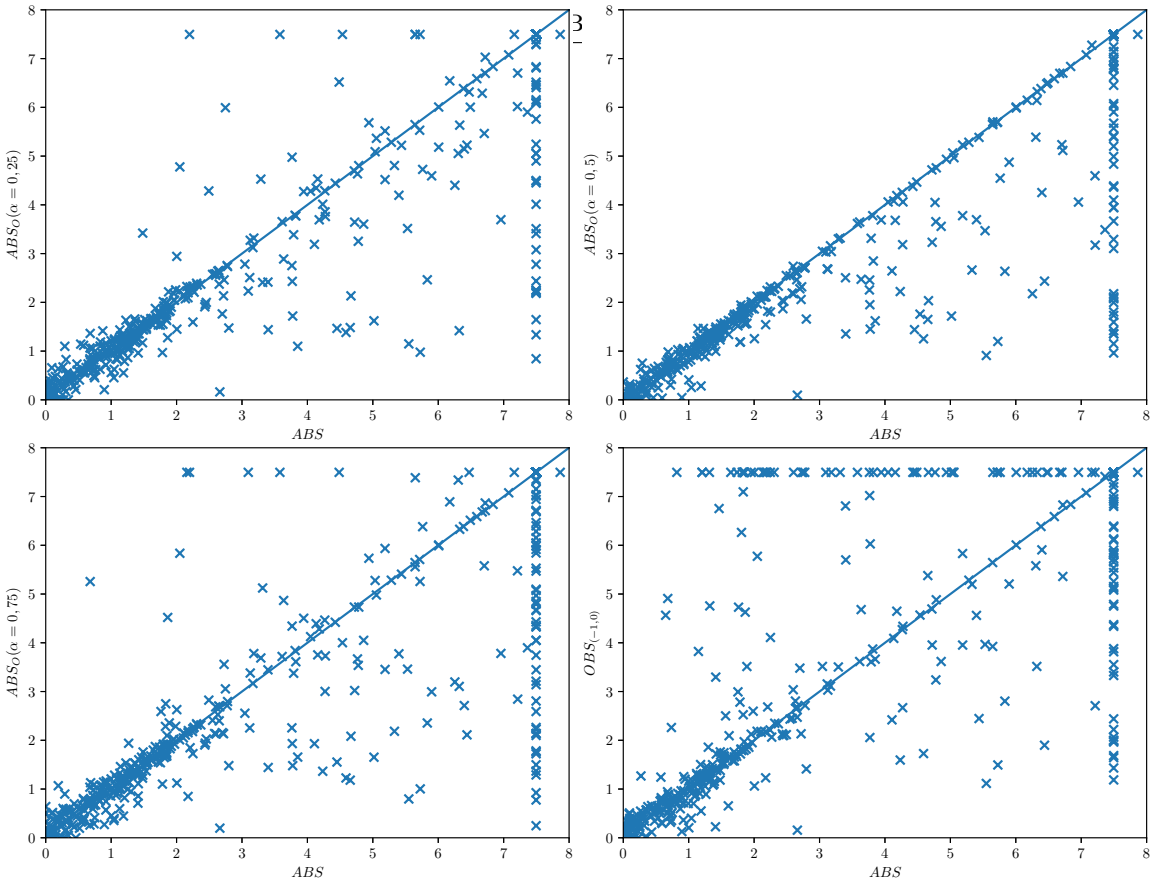


FIGURE A.5 – Running time comparisons of the hybrid strategies, with respect to the hybridization parameters. From left to right and the top to the bottom, the configurations of ABS with (0.25), (0.5), (0.75) and (0).

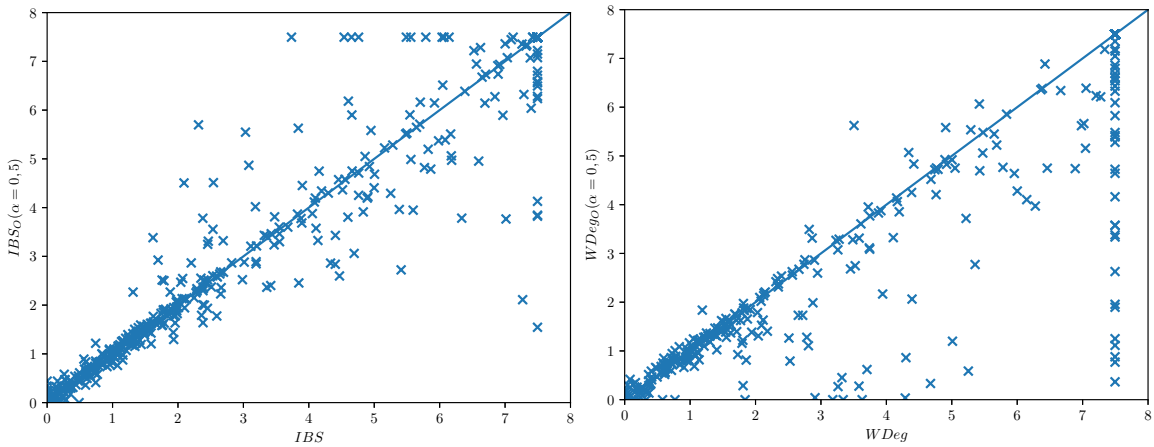


FIGURE A.6 – Comparison of the original search against their hybridized version

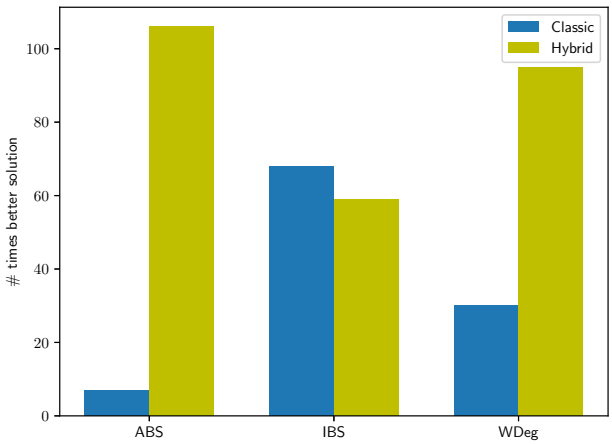


FIGURE A.7 – Comparison of the objective quality between search strategies and their hybridized versions.

within the hybridization, both in run-times and best objectives, which confirms our previous results. In the following, when no configuration is specified for *OBS*, then it means that the configuration $(a = -1, b = 0)$ has been used. Like *OBS*, the hybridization method is configurable in different ways. More or less importance can be given to the objective, or to the classical feature. In order to find the best parameter α, different experiments have been done. Figure A.5 shows the comparison of different values of α on *ABS*. When *OBS* and *ABS* are not hybridized ($\alpha = 1$ and $\alpha = 0$), they clearly show orthogonal behaviors : the timeout are observed on different instances. By looking at the Figure A.5, it can be seen that $ABS_O(0.5)$ dominates the others : less timeout and better run-times are observed. Only a full comparison of *ABS* is presented here. We intentionally omit the remaining combinations to preserve clarity, but similar results are observed with the others hybridized strategies. The best combinations are reached when $\alpha = 0.5$. Thus in the following, when we are going to talk about a hybridized version, it will be always with $\alpha = 0.5$.

The run-time and timeout comparisons between the others searches and their hybridized version are shown in Figure A.6. It is import to remark that $WDeg_O$ seems to outperform its original version, unlike IBS_O which has an orthogonal behavior. The figure A.7 shows how the objective feature impacts the search to find good solutions. It compares the number of times a search against its hybridized version has found a better solution. Unlike *IBS*, *ABS* and *WDeg* seem to benefit from the objective features, since their hybridized versions often find better solution than the original ones. For instance, the classical *ABS* find less than 10 better solution compared to its hybridized version which find more than 100 times.

To support again the interest of the hybridization method, we have extracted some interesting problem families in the Table A.1. In this Table, even if *OBS* is

Family	*OBS*	*ABS*	*WDeg*	*IBS*	ABS_O	$WDeg_O$	IBS_O
tdtsp	**2**	5	5	5	5	5	5
prize-collecting	**2**	7	7	7	7	**2**	8
2DBinPacking	7	8	8	8	**6**	8	8
mrcpspmm	**0**	3	1	1	**0**	**0**	**0**
mario	**0**	4	2	**0**	4	**0**	3
tpp	7	7	10	10	**5**	10	10
depot placement	3	7	7	**1**	4	6	4
p1f	2	**1**	7	**1**	2	7	3
table-layout	**0**	**0**	10	4	**0**	3	5
filters	2	**1**	5	**1**	**1**	5	**1**
amaze	4	**3**	5	4	**3**	5	4
open stack	8	5	10	7	**4**	9	6
talent scheduling	8	7	8	7	**4**	8	7

Table A.1 – Number of timeout in some families of instances.

not the best strategy, it is often able to solve problems where classical strategies do not. Furthermore, this table shows the interest of the hybridization, which most of the time takes advantage and improve the search considering only one feature. A good example is talent scheduling problem, *OBS* has 8 timeouts and *ABS* 7, but the hybridized version have only 4.

From the different plots and table presented, we remark that *IBS* is an exception because neither the original nor the hybridized version dominates each other and thus does not benefit as much as other search strategies from the hybridization. Actually, *IBS* contains already some information about the objective bounds modifications. The impact is computed over all variables including the objective. This is why the combination of the two features does not lead to a domination, but only an improvement in several problems and a decrease in some others. The resulting search is an orthogonal search to *IBS*.

Overall evaluation

Figure A.9 shows how many instances were solved as a function of time over all strategies. Without any hybridization *IBS* is the best strategy. However, with the hybridization, *ABS* shows the best improvements and so ABS_O become the best strategy. ABS_O has the largest number of solved instances under the allotted time. Furthermore, the hybridized versions are very competitive and improve the number of solved instances. Such a result confirms that using the objective as feature leads to strong improvement in solving time.

Most of the time, in real life problems, the optimal solution cannot be found

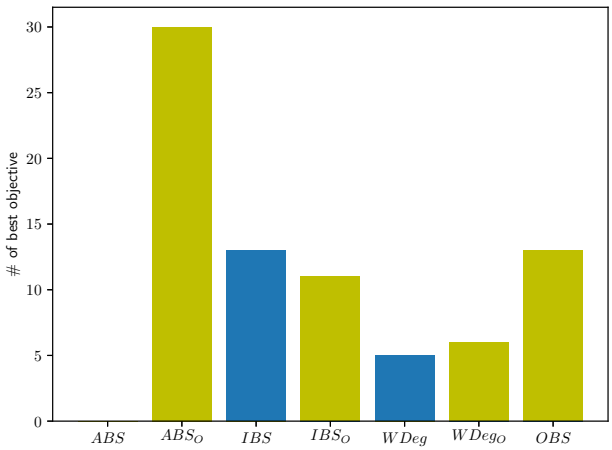


FIGURE A.8 – Number of instances where each search strategy has found a strictly better objective compared to all the other.

or proved due to time limits. That is why we now compare the capabilities of *OBS* and the hybridized versions to find good solutions under an allotted time. The new hybrid strategies are very competitive in finding good solutions under a given amount of time as well. Figure A.8 shows how many times a search strategy has found a strictly better solution than all the others. Searches using the objective feature are depicted in yellow and the others in blue.

ABS_O surpass the others and was able to find 30 times a strictly better objective than the others, while its original version *ABS* never finds a better solution. *IBS* and *OBS* seem to be the second best search strategies in terms of score. The hybridization shows again its advantages since ABS_O is strictly better than *ABS*. $WDeg_O$ slightly dominates *WDeg* and *OBS* has a good rank.

Miscellaneous discussions The objective can be monitor in many different ways. The $\Delta_O(t)$ was not our only trial, we tried to monitor the changes through a qualitative function counting how many times a variable modifies either the lower or the upper bounds. On the Minizinc Library, the qualitative function was dominated by the quantitative one.

Furthermore the $\Delta_O(t)$ function was used to designed a value selector. Different variants have been tried : first to select the value minimizing $\Delta_O(t)$, with possibly different values for a and b. Second to select the value using the new value heuristic from [60]. However, even if on some instances such as *ghoulomb* or *openstack* these selector showed a real improvement, they seem to globally be dominated in the Minizinc problems set by *minVal*. The definition of a good value heuristic seems to still be a challenge to solve.

Our experimental section shows that combining classic search strategies with our objective-based feature leads to better performances and the ability to solve

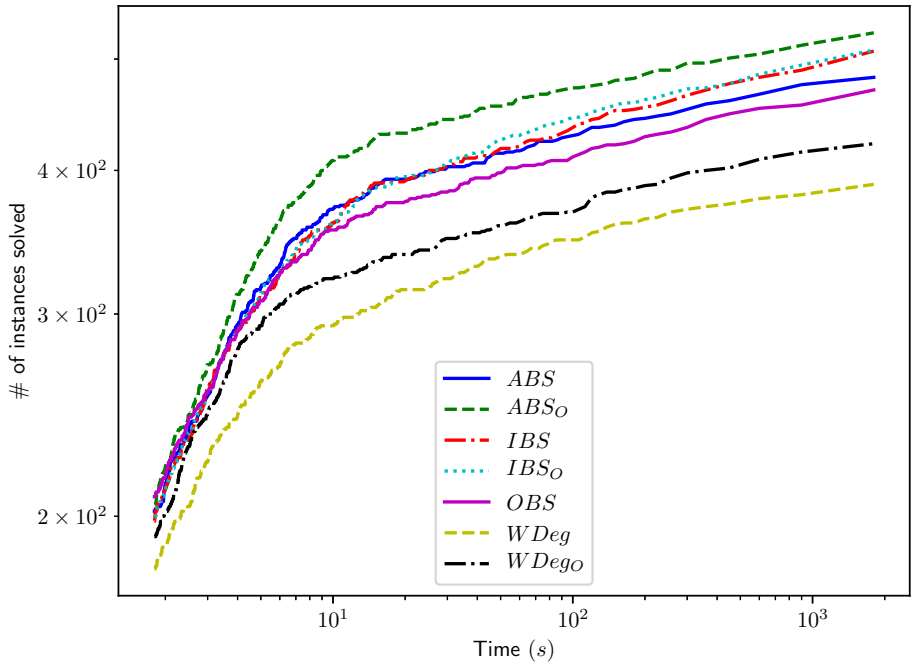


Figure A.9 – Comparison of the number of instances solved by the different strategies as a function of time.

new problems. It shows that for *ABS* and *WDeg* adding an objective-based feature seems to dominate their performance. Finally it shows that the objective as a feature can play an important role in finding a good solution faster, as already claimed [60].

A.3 Related Work

The objective variable in COPs has already been considered in other fields such as max-SAT [101] to choose which literal to select, or in Soft-CSP for the decision value [124]. Large Neighborhood Search (LNS) framework also consider the objective : for example by changing the term of the weighted sum to minimize [137]. In constraint programming, the objective information is not yet well used. In [131], the authors propose a heuristic for weighted constraint satisfaction problems based on the solution quality to guide the value selection during the search. In [42], the authors propose a machine learning approach to learn the objective function from the variables' values, but not directly on the variables themselves.

More recently, counting based search has been adapted for optimization problems [169], the main idea being to consider objective-based solution density instead of a simple solution density. This is done by adding to each objective-based

constraint an additional algorithm processing these values. Also, in MIP, the objective is widely used in the heuristic [67] : the variables having the best impact on the objective value of the relaxed problem are selected first. This approach differs from our, since CP does not have good relaxation as MIP and we consider the hybridization of the search strategies. A recent work [60] uses the objective information in order to select the variable value, leaving the variable selection to another strategy. Our method differs from [60] since we propose a variables selector, while [60] proposed a value selector. Secondly, we are trying to learn on-the-fly all along the search tree which variable seems to be the most promising, unlike [60]. In [60] the value is selected by testing all the possible assignments of the variable's domains to determine after the propagation which value is the best. Moreover, our feature is more fine-grained because it can be determined how strongly to emphasize bound modifications, using positive or negative parameters. In addition, in this appendix we propose an hybridization of existing searches with the objective feature. More particularly our new strategies can be added into the set of available strategies to choose to solve a problem, even in online fashion [225].

A.4 Conclusion

We have demonstrated the need for using the objective variable as a feature for decisions within search strategies in constraint programming. We have defined a fine grain feature based on objective bound modifications. By using this new feature, we have designed a new variable selector named OBS. This new variable selector is not the most efficient, but it is able to overpass the existing ones on some class of problems. Moreover, we have proposed a hybridization method to combine our proposed objective-based feature with many existing search strategies. Our evaluation has shown that the hybridized searches give great results and are better than the original strategy in term of run time and solution quality. Some searches are dominated by their hybrid versions. Through this new perspective, we have shown that using the objective as a feature to make decisions can lead to strong results. In addition, further work can be done, for example, with non valued SSs like the ones using a ranking criteria such as COS [68]. Directly applying this work on such SSs is not trivial, and should be a next step.

For both the $\Delta_O(t)$ function and the hybridization, we consider here only a linear combination of the values. More complex combination scheme can be considered. For example, non linear function or ranking function could be studied.

Finally, parameter optimization methods [110] could be used in order to find the best values of a, b and α for a given family of problem while solving it.

Annexe B

ConGa : Constraint Game solver

This project is a minimal working version of **Con**straint **Ga**me solver (or Conga). Conga is used to model games and especially to find Pure Nash equilibrium in it. The project can be found at https://github.com/palmieri-a/CONGA. Conga is based on Choco Solver which is a Constraint programming solver.

Using Conga should be easy for the ones who have already used a constraint solver. For the readers who have never used a Constraint solver, it is encouraged to read the Choco solver's documentation and its tutorial.

In the following, an example is provided to illustrate the new internal functions and behaviors in Conga. In this tutorial, the basic notions of constraint programming and game theory are supposed to be known. The tutorial is organized as follow, we first describe the general principles of the solver, then we show how to model a game, afterwards the possible customization and to finally present how to extend the solver.

Requirements

Tools :

- JDK8+
- Maven 3+

Libraries :

- Choco Solver
- JUnit (tests only)
- Gambit (tests only)

Installation

To use Conga a jar including all the dependencies is provided on this repository. Otherwise the sources can be compiled directly with the following instructions.

Compiling from sources

After having downloaded the sources, the project can be compiled by launching the following command at the root of the project :

```
mvn clean install -DskipTests
```

The above command compiles the sources and creates a jar including the dependencies. This jar can either be imported into another maven project, in your favorite IDE...

Functioning principles

Conga Solver is not a solver itself, but instead, it is an interface to create a model in Choco solver for games. Games or multi-agents systems are generally composed of two sides : the global situation and the agents' situation. That is why Conga is built around two solver instances. One is used to compute the players' deviations, while the second one explores all the possible global situations. A model is created in both solvers. To create and solve a game, two classes are crucial : **AbstractGameModel*** and ***CongaSolver***.

— ***CongaSolver*** is an interface to solve and to manage the solving process of games. It is used to customize games' model, but only the game part. Further information are provided in the solver extensions.

— ***AbstractGameModel*** is the interface to implement games. It contains the necessary routines to create games' semantic.

Modeling your first game

This section presents how to model a simple game with Conga. The Wolf Lamb Cabbage (WLC) game is a simple game (see description above) used to illustrate the internal functions and how to find the Pure Nash Equilibriums.

Modeling

To model a game, the class ***AbstractGameModel*** has to be overridden. It imposes to implement the ***buildModel*** and to call to the super constructor which takes as parameter the number of players.

buildModel method is the internal way to construct a game. This method provides as a parameter a model object in order to build the variables and the constraints. Also, when a game model is built, it has to call the super constructor. This call defines the number of players in the game. Then, when it associated with an instance of **CongaSolver**, it builds the players data-structures and then create the players objects. These players instances are used to define players behaviors, specify the players search strategies, their goals...

Note that to build a game you need absolutely to use these functions. Conga builds the constraints, data-structures, and players through it.

An optional method ***defineObjective*** can be overridden. This function enables to define a function which has to be optimized, while seeking for Nash Equilibriums. Note also that the players are indexed from 1 to n in an array provided by the *AbstractGameModel*. The player 0 is a flag for the data structures or can be used to put any new behavior which cannot enter in the current framework (like a new player with a different behavior).

Wolf Lamb Cabbage Game (WLC)

Three agents, Wolf (W), Lamb (L) and Cabbage (C) receive an invitation for a party. Each of them has the choice to come or not at this event. Each agent has his own preferences about meeting the others participants. Wolf would be happy to see Lamb but is indifferent about Cabbage's presence. Lamb would like to see Cabbage but only if Wolf is not coming. And Cabbage is a plant and is indifferent to everything.

The game 's model code is provided in the following and can be found in the examples folder of the project.

```
public class WolfLambCabbage extends AbstractGameModel {

        private final Short WOLF = 1, LAMB = 2, CABBAGE = 3;

        public WolfLambCabbage() {
                super(3);
        }


        @Override
        public void buildModel(Model s) {
                BoolVar[] isComing = s.boolVarArray("coming", players.length - 1);
                IntVar[] utilities = s.intVarArray(players.length - 2, 0, 2);

                for (int i = 0; i < isComing.length; i++) {
                    players[i + 1].own(isComing[i]);
```

```
            }


        players[WOLF].setObjective(
            ResolutionPolicy.MAXIMIZE,utilities[WOLF - 1]);
        players[LAMB].setObjective(
            ResolutionPolicy.MAXIMIZE,utilities[LAMB - 1]);

        Constraint lamb_s = s.arithm(isComing[LAMB - 1], "=", 1);
        Constraint cabbage_s = s.arithm(isComing[CABBAGE - 1],
            "=", 1);
        Constraint wolf_s = s.arithm(isComing[WOLF - 1], "=", 1);

        s.arithm(s.and(wolf_s, lamb_s).reify(),
                 "=", utilities[WOLF - 1]).post();
        s.sum(new BoolVar[] { s.and(
            wolf_s.getOpposite(), lamb_s, cabbage_s).reify(),
        s.and(wolf_s, lamb_s.getOpposite()).reify() },
              "=", utilities[LAMB -1]).post();
        }


}
```

The code above shows the basic routines to model a game. The Choco's API is used to build the variables and the constraints. Then the game semantic is given by the function ***own(Variable...)*** which is callable from a player. The players are a protected field in the class **AbstractGameModel**. For instance the line above specifies that the *player i* own the variable *isComing[i]*.

```
players[i + 1].own(isComing[i]);
```

In the same way, for each player, an objective can be added by using the function ***setObjective(ResolutionPolicy, Variable)***. For instance the following code specifies that the *WOLF* wants to maximize the value of the variable *utilities[WOLF-1]*.

```
players[WOLF].setObjective(
            ResolutionPolicy.MAXIMIZE,utilities[WOLF - 1]);
```

Solving

We show here how to solve a game. The basic idea is to provide to an instance of the class *CongaSolver* a model to then compute the Nash equilibriums in it.

This last instruction is done by the function ***prepareAndGetSolver()*** which return a solver in the Choco sense.

Most of the time, to ease the modelization part, factories are provided. These factories can be found in the folder *src/factories*

The following example shows how to find the Pure Nash Equilibrium in the ***WolfLambCabbage*** game using the ***LAST_LEVEL*** constraint. This last one is a version of the propagator proposed in [131]. Another propagator named ***BOUND_CONSTRAINT*** can be used. This propagator is the new algorithm proposed in [142]. Also a possibility which can be helpful while debugging is to specify that the game has no constraints with ***NO_CONSTRAINT***.

Finally, the function solve from the solver to find the Nash equilibriums of the game can be called.

```
public static void main(String[] args) {
        CongaSolver cg = new CongaSolver(new WolfLambCabbage());
        cg.setConstraintBuilder(ConstraintFactory.LAST_LEVEL);
        Solver s =cg.prepareAndGetSolver();
        s.solve();
}
```

Multiples interfaces can be redefined to customize the solver behavior. A non exhaustive list is given here :

— ***ConstraintBuilder*** : defines how to build for instance a Nash constraint (see ***ConstraintFactory*** for multiples examples)

— ***IEquilibriumConcept*** : defines the concept which has to be computed. Most of the time it is the Nash equilibrium. It can be useful to extend the solver to other multi agent problems.

— ***ISearchPolicy*** : defines the search policy : how the exploration is made in the players' deviations search space. This Interface can be useful to define incomplete search or any new kind of search like Pareto Best response.

— ***PlayerDependenciesUpdater*** : this interface allows to create a graphical game and limit the number of time a player's constraint is awakened.

Going further

We list here some important class which may be useful to modify the solver behavior.

— ***VarHelperImpl*** is the implementation of the interface ***Varhelper*** which provides data structures to retrieves the variables and objectives of the different players. It can be useful to know if a variable or an objective is shared. You can then ask within this class if such case happens in the game.

— ***Executable*** is the interface to implement more general algorithms. You can find the examples in the enum ***AlgorithmFactory*** with for instance the algorithms **NASH, IBR, PARETO_NASH**, and the pricing function such **POA** or **POS**.

Gambit reader

If you are really attached to gambit solver, we provide a way to solve gambit representation through our solver. To use gambit you first have to define the global variable *GAMBIT_PATH* to specify where gambit executable is located. More information of this interface can be found in the class ***GambitNFGParser*** which take as input a normal form game as defined in gambit, and output a model in constraint game. The transformation used is quite simple, the matrix is transformed into table constraints.

Bibliographie

[1] Thomas Ågotnes et al. "Verifiable Equilibria in Boolean Games". In : *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. Sous la dir. de Francesca Rossi. IJCAI/AAAI, 2013, p. 689–695. URL : http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6921.

[2] David Allouche et al. "Anytime Hybrid Best-First Search with Tree Decomposition for Weighted CSP". In : *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*. Sous la dir. de Gilles Pesant. T. 9255. Lecture Notes in Computer Science. Springer, 2015, p. 12–29. DOI : 10.1007/978-3-319-23219-5_2. URL : https://doi.org/10.1007/978-3-319-23219-5%5C_2.

[3] Krzysztof R. Apt. "The Essence of Constraint Propagation". In : *Theor. Comput. Sci.* 221.1-2 (1999), p. 179–210. DOI : 10.1016/S0304-3975(99)00032-8. URL : http://dx.doi.org/10.1016/S0304-3975(99)00032-8.

[4] Brian W. Arthur. "Complexity in Economic Theory. Inductive Reasoning and Bounded Rationality". In : *American Economic Review* 82.2 (1994), p. 406–411.

[5] Robert J Aumann. "Correlated equilibrium as an expression of Bayesian rationality". In : *Econometrica : Journal of the Econometric Society* (1987), p. 1–18.

[6] Baruch Awerbuch, Yossi Azar et Amir Epstein. "The Price of Routing Unsplittable Flow". In : *SIAM J. Comput.* 42.1 (2013), p. 160–177. DOI : 10.1137/070702370. URL : https://doi.org/10.1137/070702370.

[7] Abdelhadi Azzouni, Raouf Boutaba et Guy Pujolle. "NeuRoute : Predictive dynamic routing for software-defined networks". In : *13th International Conference on Network and Service Management,*

CNSM 2017, Tokyo, Japan, November 26-30, 2017. IEEE Computer Society, 2017, p. 1–6. ISBN : 978-3-901882-98-2. DOI : 10.23919/CNSM.2017.8256059. URL : https://doi.org/10.23919/CNSM.2017.8256059.

[8] Yakov Babichenko et Ron Peretz. "Approximate Nash Equilibria via Sampling". In : *CoRR* abs/1307.4934 (2013). arXiv : 1307.4934. URL : http://arxiv.org/abs/1307.4934.

[9] Philippe Baptiste et al. "Constraint-Based Scheduling and Planning". In : *Handbook of Constraint Programming*. 2006, p. 761–799. DOI : 10.1016/S1574-6526(06)80026-X. URL : https://doi.org/10.1016/S1574-6526(06)80026-X.

[10] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2010. ISBN : 978-0-521-14775-0. URL : http://www.cambridge.org/de/academic/subjects/computer-science/artificial-intelligence-and-natural-language-processing/knowledge-representation-reasoning-and-declarative-problem-solving.

[11] Cynthia Barnhart, Christopher A. Hane et Pamela H. Vance. "Using Branch-and-Price-and-Cut to Solve Origin-Destination Integer Multicommodity Flow Problems". In : *Operations Research* 48.2 (2000), p. 318–326. DOI : 10.1287/opre.48.2.318.12378. URL : https://doi.org/10.1287/opre.48.2.318.12378.

[12] J Christopher Beck, TK Feng et Jean-Paul Watson. "Combining constraint programming and local search for job-shop scheduling". In : *INFORMS Journal on Computing* 23.1 (2011), p. 1–14.

[13] Francesco Belardinelli et al. "Relaxing Exclusive Control in Boolean Games". In : *Proceedings Sixteenth Conference on Theoretical Aspects of Rationality and Knowledge, TARK 2017, Liverpool, UK, 24-26 July 2017*. Sous la dir. de Jérôme Lang. T. 251. EPTCS. 2017, p. 43–56. DOI : 10.4204/EPTCS.251.4. URL : https://doi.org/10.4204/EPTCS.251.4.

[14] Nicolas Beldiceanu et Thierry Petit. "Cost Evaluation of Soft Global Constraints". In : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, First International Conference, CPAIOR 2004, Nice, France, April 20-22, 2004, Proceedings*. Sous la dir. de Jean-Charles Régin et Michel Rueher. T. 3011. Lecture Notes in Computer Science. Springer, 2004, p. 80–95. DOI : 10.1007/978-3-540-24664-0_6. URL : https://doi.org/10.1007/978-3-540-24664-0%5C_6.

[15] Marco Benedetti, Arnaud Lallouet et Jérémie Vautard. “QCSP Made Practical by Virtue of Restricted Quantification.” In : *IJCAI*. 2007, p. 38–43.

[16] David Bergman, André A. Ciré et Willem Jan van Hoeve. “MDD Propagation for Sequence Constraints”. In : *J. Artif. Intell. Res.* 50 (2014), p. 697–722. doi : 10.1613/jair.4199. url : https://doi.org/10.1613/jair.4199.

[17] David Bergman et al. *Decision Diagrams for Optimization*. Artificial Intelligence : Foundations, Theory, and Algorithms. Springer, 2016. isbn : 978-3-319-42847-5. doi : 10.1007/978-3-319-42849-9. url : https://doi.org/10.1007/978-3-319-42849-9.

[18] Christian Bessiere. “Constraint Propagation”. In : *Handbook of Constraint Programming*. 2006, p. 29–83. doi : 10.1016/S1574-6526(06)80007-6. url : https://doi.org/10.1016/S1574-6526(06)80007-6.

[19] Christian Bessiere et al. “Circuit Complexity and Decompositions of Global Constraints”. In : *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. Sous la dir. de Craig Boutilier. 2009, p. 412–418. url : http://ijcai.org/Proceedings/09/Papers/076.pdf.

[20] Meghyn Bienvenu, Jérôme Lang et Nic Wilson. “From Preference Logics to Preference Languages, and Back”. In : *KR 2010, Toronto, Ontario, Canada, May 9-13, 2010*. Sous la dir. de F. Lin, U. Sattler et M. Truszczynski. AAAI Press, 2010. url : http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1360.

[21] Johannes Bisschop. *AIMMS optimization modeling*. Lulu. com, 2006.

[22] Christian Bliek et Djamila Sam-Haroud. “Path Consistency on Triangulated Constraint Graphs”. In : *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*. Sous la dir. de Thomas Dean. Morgan Kaufmann, 1999, p. 456–461. url : http://ijcai.org/Proceedings/99-1/Papers/066.pdf.

[23] Elise Bonzon, Marie-Christine Lagasquie-Schiex et Jérôme Lang. “Compact Preference Representation for Boolean Games”. In : *PRICAI 2006 : Trends in Artificial Intelligence, 9th Pacific Rim International Conference on Artificial Intelligence, Guilin, China, August 7-11, 2006, Proceedings*. Sous la dir. de Qiang Yang et Geoffrey I.

Webb. T. 4099. Lecture Notes in Computer Science. Springer, 2006, p. 41–50. doi : 10.1007/11801603_7. url : https://doi.org/10.1007/11801603%5C_7.

[24] Elise Bonzon, Marie-Christine Lagasquie-Schiex et Jérôme Lang. "Dependencies between players in Boolean games". In : *Int. J. Approx. Reasoning* 50.6 (2009), p. 899–914.

[25] Elise Bonzon et al. "Boolean Games Revisited". In : *ECAI*. 2006, p. 265–269.

[26] Lucas Bordeaux et Eric Monfroy. "Beyond NP : Arc-Consistency for Quantified Constraints". In : *CP 2002, Ithaca, NY, USA, September 9-13, 2002*. Sous la dir. de P. Van Hentenryck. T. 2470. LNCS. Springer, 2002, p. 371–386. isbn : 3-540-44120-4. url : http://link.springer.de/link/service/series/0558/bibs/2470/24700371.htm.

[27] Frédéric Boussemart et al. "Boosting systematic search by weighting constraints". In : *ECAI*. T. 16. 2004, p. 146.

[28] Craig Boutilier et al. "CP-nets : A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements". In : *J. Artif. Intell. Res. (JAIR)* 21 (2004), p. 135–191.

[29] Dietrich Braess, Anna Nagurney et Tina Wakolbinger. "On a Paradox of Traffic Planning". In : *Transportation Science* 39.4 (2005), p. 446–450. doi : 10.1287/trsc.1050.0127. url : https://doi.org/10.1287/trsc.1050.0127.

[30] Steven J Brams. *Game theory and politics*. Courier Corporation, 2011.

[31] Gerhard Brewka, Thomas Eiter et Miroslaw Truszczynski. "Answer set programming at a glance". In : *Commun. ACM* 54.12 (2011), p. 92–103. doi : 10.1145/2043174.2043195. url : http://doi.acm.org/10.1145/2043174.2043195.

[32] Valentinas Bubelis. "On equilibria in finite games". In : *International Journal of Game Theory* 8.2 (1979), p. 65–79.

[33] Nils Bulling et Koen V. Hindriks. "Boolean Negotiation Games". In : *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*. Sous la dir. de Gal A. Kaminka et al. T. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, p. 1748–1749. doi : 10.3233/978-1-61499-672-9-1748. url : https://doi.org/10.3233/978-1-61499-672-9-1748.

[34] Bertrand Cabon et al. “Radio Link Frequency Assignment”. In : *Constraints* 4.1 (1999), p. 79–89. doi : 10.1023/A:1009812409930. url : https://doi.org/10.1023/A:1009812409930.

[35] Antonio Capone et al. “Detour planning for fast and reliable failure recovery in SDN with OpenState”. In : *2015 11th International Conference on the Design of Reliable Communication Networks (DRCN)*. IEEE. 2015, p. 25–32.

[36] Antonio Capone et al. “Detour planning for fast and reliable failure recovery in SDN with OpenState”. In : *11th International Conference on the Design of Reliable Communication Networks, DRCN 2015, Kansas City, MO, USA, March 24-27, 2015*. IEEE, 2015, p. 25–32. doi : 10.1109/DRCN.2015.7148981. url : https://doi.org/10.1109/DRCN.2015.7148981.

[37] Sofia Ceppi et al. “Local Search Methods for Finding a Nash Equilibrium in Two-Player Games”. In : *IAT*. Sous la dir. de J. Xiangji Huang et al. IEEE Computer Society Press, 2010, p. 335–342. isbn : 978-0-7695-4191-4.

[38] G Chabert et al. *Ibex An iInterval based EXplorer*. 2009. url : http://www.ibex-lib.org.

[39] Gilles Chabert et Luc Jaulin. “Contractor programming”. In : *Artificial Intelligence* 173.11 (2009), p. 1079–1100.

[40] Alain Chabrier et al. “Solving a Network Design Problem”. In : *Annals OR* 130.1-4 (2004), p. 217–239. doi : 10.1023/B:ANOR.0000032577.81139.84. url : https://doi.org/10.1023/B:ANOR.0000032577.81139.84.

[41] Dimitris E. Charilas et Athanasios D. Panagopoulos. “A survey on game theory applications in wireless networks”. In : *Computer Networks* 54.18 (2010), p. 3421–3430. doi : 10.1016/j.comnet.2010.06.020. url : https://doi.org/10.1016/j.comnet.2010.06.020.

[42] Geoffrey Chu et Peter J. Stuckey. “Learning Value Heuristics for Constraint Programming”. In : *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings*. 2015, p. 108–123. doi : 10.1007/978-3-319-18008-3_8. url : https://doi.org/10.1007/978-3-319-18008-3_8.

[43] Sofie De Clercq et al. “Possibilistic Boolean Games : Strategic Reasoning under Incomplete Information”. In : *Logics in Artificial Intelligence - 14th European Conference, JELIA 2014, Funchal, Madeira, Portugal, September 24-26, 2014. Proceedings*. Sous la dir. d’Eduardo Fermé et João Leite. T. 8761. Lecture Notes in Computer Science. Springer, 2014, p. 196–209. doi : 10.1007/978-3-319-11558-0_14. url : https://doi.org/10.1007/978-3-319-11558-0%5C_14.

[44] Sofie De Clercq et al. “Using Answer Set Programming for Solving Boolean Games”. In : *Principles of Knowledge Representation and Reasoning : Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. Sous la dir. de Chitta Baral, Giuseppe De Giacomo et Thomas Eiter. AAAI Press, 2014. url : http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7805.

[45] Sofie De Clercq et al. “Exact and heuristic methods for solving Boolean games”. In : *Autonomous Agents and Multi-Agent Systems* 31.1 (2017), p. 66–106. doi : 10.1007/s10458-015-9313-5. url : https://doi.org/10.1007/s10458-015-9313-5.

[46] Sofie De Clercq et al. “Modelling incomplete information in Boolean games using possibilistic logic”. In : *Int. J. Approx. Reasoning* 93 (2018), p. 1–23. doi : 10.1016/j.ijar.2017.10.017. url : https://doi.org/10.1016/j.ijar.2017.10.017.

[47] José R. Correa, Andreas S. Schulz et Nicolás E. Stier Moses. “Selfish Routing in Capacitated Networks”. In : *Math. Oper. Res.* 29.4 (2004), p. 961–976. doi : 10.1287/moor.1040.0098. url : https://doi.org/10.1287/moor.1040.0098.

[48] José R. Correa, Andreas S. Schulz et Nicolás E. Stier Moses. “Fast, Fair, and Efficient Flows in Networks”. In : *Operations Research* 55.2 (2007), p. 215–225. doi : 10.1287/opre.1070.0383. url : https://doi.org/10.1287/opre.1070.0383.

[49] IBM ILOG CPLEX. “12.6”. In : *CPLEX User’s Manual* (2014).

[50] Constantinos Daskalakis, Paul W. Goldberg et Christos H. Papadimitriou. “The complexity of computing a Nash equilibrium”. In : *Commun. ACM* 52.2 (2009), p. 89–97. doi : 10.1145/1461928.1461951. url : https://doi.org/10.1145/1461928.1461951.

[51] Constantinos Daskalakis, Aranyak Mehta et Christos Papadimitriou. “A note on approximate Nash equilibria”. In : *Theoretical Computer Science* 410.17 (2009), p. 1581–1588.

[52] Rina Dechter. “Learning While Searching in Constraint-Satisfaction-Problems”. In : *Proceedings of the 5th National Conference on Artificial Intelligence. Philadelphia, PA, USA, August 11-15, 1986. Volume 1 : Science.* Sous la dir. de Tom Kehler. Morgan Kaufmann, 1986, p. 178–185. url : http : //www.aaai.org/Library/AAAI/1986/aaai86-029.php.

[53] Sophie Demassey, Gilles Pesant et Louis-Martin Rousseau. “A Cost-Regular Based Hybrid Column Generation Approach”. In : *Constraints* 11.4 (2006), p. 315–333. doi : 10.1007/s10601-006-9003-7. url : https://doi.org/10.1007/s10601-006-9003-7.

[54] Theo SH Driessen. *Cooperative games, solutions and applications.* T. 3. Springer Science & Business Media, 2013.

[55] Paul E. Dunne et Wiebe van der Hoek. “Representation and Complexity in Boolean Games”. In : *JELIA*. Sous la dir. de J. J. Alferes et J. A. Leite. T. 3229. Lecture Notes in Computer Science. Springer, 2004, p. 347–359. isbn : 3-540-23242-7.

[56] Paul E. Dunne et al. “Cooperative Boolean games”. In : *AAMAS (2).* Sous la dir. de L. Padgham et al. IFAAMAS, 2008, p. 1015–1022. isbn : 978-0-9817381-1-6.

[57] Javier Esparza et Stephan Melzer. “Model checking LTL using constraint programming”. In : *International Conference on Application and Theory of Petri Nets.* Springer. 1997, p. 1–20.

[58] S. Even, A. Itai et A. Shamir. “On the Complexity of Time Table and Multi-commodity Flow Problems”. In : *Proceedings of the 16th Annual Symposium on Foundations of Computer Science.* SFCS ’75. Washington, DC, USA : IEEE Computer Society, 1975, p. 184–193. doi : 10.1109/SFCS.1975.21. url : http://dx.doi.org/10.1109/SFCS.1975.21.

[59] Jean-Guillaume Fages, Xavier Lorca et Louis-Martin Rousseau. “The salesman and the tree : the importance of search in CP”. In : *Constraints* 21.2 (2016), p. 145–162. doi : 10.1007/s10601-014-9178-2. url : https://doi.org/10.1007/s10601-014-9178-2.

[60] Jean-Guillaume Fages et Charles Prud’Homme. “Making the first solution good !” In : *ICTAI 2017 29th IEEE International Conference on Tools with Artificial Intelligence.* 2017.

[61] Filippo Focacci, Andrea Lodi et Michela Milano. “Cost-Based Domain Filtering”. In : *Principles and Practice of Constraint Programming - CP’99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*. Sous la dir. de Joxan Jaffar. T. 1713. Lecture Notes in Computer Science. Springer, 1999, p. 189–203. doi : 10.1007/978-3-540-48085-3_14. url : https://doi.org/10.1007/978-3-540-48085-3%5C_14.

[62] Eugene C Freuder. “Modeling : the final frontier”. In : *The First International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP), London*. 1999, p. 15–21.

[63] Eugene C. Freuder et Richard J. Wallace. “Partial Constraint Satisfaction”. In : *Artif. Intell.* 58.1-3 (1992), p. 21–70. doi : 10.1016/0004-3702(92)90004-H. url : https://doi.org/10.1016/0004-3702(92)90004-H.

[64] D. Fudenberg et J. Tirole. *Game Theory*. The MIT Press, 1991.

[65] Martin Gairing et al. “Computing Nash equilibria for scheduling on restricted parallel links”. In : *STOC*. ACM, 2004, p. 613–622.

[66] M. R. Garey et David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. isbn : 0-7167-1044-7.

[67] Jean-Michel Gauthier et Gerard Ribière. “Experiments in mixed-integer linear programming using pseudo-costs”. In : *Math. Program.* 12.1 (1977), p. 26–47. doi : 10.1007/BF01593767. url : https://doi.org/10.1007/BF01593767.

[68] Steven Gay et al. “Conflict ordering search for scheduling problems”. In : *International Conference on Principles and Practice of Constraint Programming*. Springer. 2015, p. 140–148.

[69] Ian P. Gent, Warwick Harvey et Tom Kelsey. “Groups and Constraints : Symmetry Breaking during Search”. In : *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*. Sous la dir. de Pascal Van Hentenryck. T. 2470. Lecture Notes in Computer Science. Springer, 2002, p. 415–430. doi : 10.1007/3-540-46135-3_28. url : https://doi.org/10.1007/3-540-46135-3%5C_28.

[70] Ian P Gent, Peter Nightingale et Kostas Stergiou. “QCSP-Solve : A solver for quantified constraint satisfaction problems”. In : *IJCAI*. T. 5. 2005, p. 138–143.

[71] Ian P Gent et Toby Walsh. “CSPLib : a benchmark library for constraints”. In : *International Conference on Principles and Practice of Constraint Programming*. Springer. 1999, p. 480–481.

[72] Ian P. Gent et al. “Conditional Symmetry Breaking”. In : *Principles and Practice of Constraint Programming - CP 2005, 11th International Conference, CP 2005, Sitges, Spain, October 1-5, 2005, Proceedings*. Sous la dir. de Peter van Beek. T. 3709. Lecture Notes in Computer Science. Springer, 2005, p. 256–270. doi : 10.1007/11564751_21. url : https://doi.org/10.1007/11564751%5C_21.

[73] Jelle Gerbrandy. “Logics of propositional control”. In : *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*. Sous la dir. d’Hideyuki Nakashima et al. ACM, 2006, p. 193–200. doi : 10.1145/1160633.1160664. url : http://doi.acm.org/10.1145/1160633.1160664.

[74] Matthew L. Ginsberg. “Dynamic Backtracking”. In : *J. Artif. Intell. Res.* 1 (1993), p. 25–46. doi : 10.1613/jair.1. url : https://doi.org/10.1613/jair.1.

[75] Gael Glorian et al. “Combining Nogoods in Restart-Based Search”. In : *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*. Sous la dir. de J. Christopher Beck. T. 10416. Lecture Notes in Computer Science. Springer, 2017, p. 129–138. doi : 10.1007/978-3-319-66158-2_9. url : https://doi.org/10.1007/978-3-319-66158-2%5C_9.

[76] Solomon W. Golomb et Leonard D. Baumert. “Backtrack Programming”. In : *J. ACM* 12.4 (1965), p. 516–524. doi : 10.1145/321296.321300. url : https://doi.org/10.1145/321296.321300.

[77] Carla P. Gomes et al. “Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems”. In : *J. Autom. Reasoning* 24.1/2 (2000), p. 67–100. doi : 10.1023/A:1006314320276. url : https://doi.org/10.1023/A:1006314320276.

[78] Georg Gottlob, Gianluigi Greco et Francesco Scarcello. “Pure Nash Equilibria : Hard and Easy Games”. In : *J. Artif. Intell. Res. (JAIR)* 24 (2005), p. 357–406.

[79] Umberto Grandi, Davide Grossi et Paolo Turrini. “Equilibrium Refinement through Negotiation in Binary Voting”. In : *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Sous la dir. de Qiang Yang et Michael Wooldridge. AAAI Press, 2015, p. 540–546. url : http://ijcai.org/Abstract/15/082.

[80] John Grant et al. “Manipulating Boolean Games through Communication”. In : *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. Sous la dir. de Toby Walsh. IJCAI/AAAI, 2011, p. 210–215. doi : 10.5591/978-1-57735-516-8/IJCAI11-046. url : https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-046.

[81] Tal Grinshpoun et al. “Asymmetric Distributed Constraint Optimization Problems”. In : *J. Artif. Intell. Res. (JAIR)* 47 (2013), p. 613–647. doi : 10.1613/jair.3945. url : http://dx.doi.org/10.1613/jair.3945.

[82] Alon Grubshtein et Amnon Meisels. “Finding a Nash Equilibrium by Asynchronous Backtracking”. In : *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*. Sous la dir. de Michela Milano. T. 7514. Lecture Notes in Computer Science. Springer, 2012, p. 925–940. doi : 10.1007/978-3-642-33558-7_66. url : https://doi.org/10.1007/978-3-642-33558-7%5C_66.

[83] Alon Grubshtein, Roie Zivan et Amnon Meisels. “Partial Cooperation in Multi-agent Local Search”. In : *ECAI 2012 - 20th European Conference on Artificial Intelligence. Including Prestigious Applications of Artificial Intelligence (PAIS-2012) System Demonstrations Track, Montpellier, France, August 27-31 , 2012*. Sous la dir. de Luc De Raedt et al. T. 242. Frontiers in Artificial Intelligence and Applications. IOS Press, 2012, p. 378–383. doi : 10.3233/978-1-61499-098-7-378. url : https://doi.org/10.3233/978-1-61499-098-7-378.

[84] Stefano Gualandi et Federico Malucelli. “Resource Constrained Shortest Paths with a Super Additive Objective Function”. In : *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*. Sous la dir. de Michela Milano. T. 7514. Lecture Notes in Computer Science. Springer, 2012, p. 299–315. doi : 10.1007/978-3-642-33558-7_24. url : https://doi.org/10.1007/978-3-642-33558-7%5C_24.

[85] Julian Gutierrez, Paul Harrenstein et Michael Wooldridge. "Iterated Boolean Games". In : *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. Sous la dir. de Francesca Rossi. IJCAI/AAAI, 2013, p. 932–938. URL : http://www.aaai.org/ocs/index.php/IJCAI/IJCAI13/paper/view/6799.

[86] Julian Gutierrez, Giuseppe Perelli et Michael Wooldridge. "Imperfect information in Reactive Modules games". In : *Inf. Comput.* 261.Part (2018), p. 650–675. DOI : 10.1016/j.ic.2018.02.023. URL : https://doi.org/10.1016/j.ic.2018.02.023.

[87] Julian Gutierrez et al. "Local Equilibria in Logic-Based Multi-Player Games". In : *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*. Sous la dir. d'Elisabeth André et al. International Foundation for Autonomous Agents et Multiagent Systems Richland, SC, USA / ACM, 2018, p. 399–406. URL : http://dl.acm.org/citation.cfm?id=3237445.

[88] Youssef Hamadi. *Combinatorial Search - From Algorithms to Systems*. Springer, 2013. ISBN : 978-3-642-41481-7. URL : http://www.springer.com/computer/ai/book/978-3-642-41481-7.

[89] Peter Hammerstein et Reinhard Selten. "Game theory and evolutionary biology". In : *Handbook of game theory with economic applications* 2 (1994), p. 929–993.

[90] Robert M Haralick et Gordon L Elliott. "Increasing tree search efficiency for constraint satisfaction problems". In : *Artificial intelligence* 14.3 (1980), p. 263–313.

[91] Tobias Harks, Stefan Heinz et Marc E. Pfetsch. "Competitive Online Multicommodity Routing". In : *Approximation and Online Algorithms, 4th International Workshop, WAOA 2006, Zurich, Switzerland, September 14-15, 2006, Revised Papers*. Sous la dir. de Thomas Erlebach et Christos Kaklamanis. T. 4368. Lecture Notes in Computer Science. Springer, 2006, p. 240–252. ISBN : 3-540-69513-3. DOI : 10.1007/11970125_19. URL : https://doi.org/10.1007/11970125%5C_19.

[92] Paul Harrenstein et al. "Boolean Games". In :

[93] John C Harsanyi. "Games with incomplete information played by "Bayesian" players, I–III Part I. The basic model". In : *Management science* 14.3 (1967), p. 159–182.

[94] William D. Harvey et Matthew L. Ginsberg. “Limited Discrepancy Search”. In : *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*. Morgan Kaufmann, 1995, p. 607–615. url : http://ijcai.org/Proceedings/95-1/Papers/080.pdf.

[95] Ara Hayrapetyan, Éva Tardos et Tom Wexler. “A network pricing game for selfish traffic”. In : *Distributed Computing* 19.4 (2007), p. 255–266. doi : 10.1007/s00446-006-0020-y. url : https://doi.org/10.1007/s00446-006-0020-y.

[96] Pavol Hell, J Nesetril et Xuding Zhu. “Duality and polynomial testing of tree homomorphisms”. In : *Transactions of the American Mathematical Society* 348.4 (1996), p. 1281–1297.

[97] Sébastien Hémon, Michel de Rougemont et Miklos Santha. “Approximate Nash equilibria for multi-player games”. In : *International Symposium on Algorithmic Game Theory*. Springer. 2008, p. 267–278.

[98] P. Van Hentenryck, éd. T. 2470. LNCS. Springer, 2002. isbn : 3-540-44120-4.

[99] Pascal Van Hentenryck et Laurent Michel. *Constraint-based local search*. The MIT press, 2009.

[100] Martin Henz, Tobias Müller et Sven Thiel. “Global constraints for round robin tournament scheduling”. In : *European Journal of Operational Research* 153.1 (2004), p. 92–101. doi : 10.1016/S0377-2217(03)00101-2. url : https://doi.org/10.1016/S0377-2217(03)00101-2.

[101] Federico Heras et Javier Larrosa. “New inference rules for efficient Max-SAT solving”. In : *AAAI*. 2006, p. 68–73.

[102] Brahim Hnich et al. “Constraint Models for the Covering Test Problem”. In : *Constraints* 11.2-3 (2006), p. 199–219. doi : 10.1007/s10601-006-7094-9. url : https://doi.org/10.1007/s10601-006-7094-9.

[103] Wiebe van der Hoek et Michael Wooldridge. “On the logic of cooperation and propositional control”. In : *Artif. Intell.* 164.1-2 (2005), p. 81–119. doi : 10.1016/j.artint.2005.01.003. url : https://doi.org/10.1016/j.artint.2005.01.003.

[104] Willem-Jan van Hoeve. “Over-constrained problems”. In : *Hybrid Optimization*. Springer, 2011, p. 191–225.

[105] Holger H. Hoos et Edward P. K. Tsang. “Local Search Methods”. In : *Handbook of Constraint Programming*. Sous la dir. de Francesca Rossi, Peter van Beek et Toby Walsh. T. 2. Foundations of Artificial Intelligence. Elsevier, 2006, p. 135–167. doi : 10.1016/S1574-6526(06)80009-X. url : https://doi.org/10.1016/S1574-6526(06)80009-X.

[106] H. Hotelling. “Stability in competition”. In : *Economic Journal* (1929), p. 41–57.

[107] Bernardo A Huberman, Rajan M Lukose et Tad Hogg. “An economics approach to hard computational problems”. In : *Science* 275.5296 (1997), p. 51–54.

[108] Nicolas Huin, Brigitte Jaumard et Frédéric Giroire. “Optimal Network Service Chain Provisioning”. In : *IEEE/ACM Trans. Netw.* 26.3 (2018), p. 1320–1333. doi : 10.1109/TNET.2018.2833815. url : http://doi.ieeecomputersociety.org/10.1109/TNET.2018.2833815.

[109] Nicolas Huin et al. “Energy-Efficient Service Function Chain Provisioning”. In : *Electronic Notes in Discrete Mathematics* 64 (2018), p. 265–274. doi : 10.1016/j.endm.2018.02.001. url : https://doi.org/10.1016/j.endm.2018.02.001.

[110] Frank Hutter, Holger Hoos et Kevin Leyton-Brown. “An evaluation of sequential model-based optimization for expensive blackbox functions”. In : *Proceedings of the 15th annual conference companion on Genetic and evolutionary computation*. ACM. 2013, p. 1209–1216.

[111] Egor Ianovski. “Complexity of mixed equilibria in Boolean games”. In : *CoRR* abs/1702.03620 (2017).

[112] Matthew O Jackson. *Social and Economic Networks*. T. 3. Princeton university press Princeton, 2008.

[113] Albert Xin Jiang, Kevin Leyton-Brown et Navin A. R. Bhat. “Action-Graph Games”. In : *Games and Economic Behavior* 71.1 (2011), p. 141–173.

[114] Roberto J. Bayardo Jr. et Daniel P. Miranker. “A Complexity Analysis of Space-Bounded Learning Algorithms for the Constraint Satisfaction Problem”. In : *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, USA, August 4-8, 1996, Volume 1*. Sous la dir. de William J. Clancey et Daniel S. Weld. AAAI Press / The MIT Press, 1996, p. 298–304. url : http://www.aaai.org/Library/AAAI/1996/aaai96-045.php.

[115] George Katsirelos et Fahiem Bacchus. “Generalized NoGoods in CSPs”. In : *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*. Sous la dir. de Manuela M. Veloso et Subbarao Kambhampati. AAAI Press / The MIT Press, 2005, p. 390–396. url : http://www.aaai.org/Library/AAAI/2005/aaai05-062.php.

[116] Henry A. Kautz et al. “Dynamic Restart Policies”. In : *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada.* Sous la dir. de Rina Dechter, Michael J. Kearns et Richard S. Sutton. AAAI Press / The MIT Press, 2002, p. 674–681. url : http://www.aaai.org/Library/AAAI/2002/aaai02-101.php.

[117] Michael J. Kearns, Michael L. Littman et Satinder P. Singh. “Graphical Models for Game Theory”. In : *UAI ’01 : Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001.* Sous la dir. de Jack S. Breese et Daphne Koller. Morgan Kaufmann, 2001, p. 253–260. url : https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=107&proceeding_id=17.

[118] Michael J. Kearns, Michael L. Littman et Satinder P. Singh. “Graphical Models for Game Theory”. In : *UAI*. Sous la dir. de Jack S. Breese et Daphne Koller. Morgan Kaufmann, 2001, p. 253–260. isbn : 1-55860-800-1.

[119] Brian Kell et Willem Jan van Hoeve. “An MDD Approach to Multidimensional Bin Packing”. In : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 10th International Conference, CPAIOR 2013, Yorktown Heights, NY, USA, May 18-22, 2013. Proceedings*. Sous la dir. de Carla P. Gomes et Meinolf Sellmann. T. 7874. Lecture Notes in Computer Science. Springer, 2013, p. 128–143. doi : 10.1007/978-3-642-38171-3_9. url : https://doi.org/10.1007/978-3-642-38171-3%5C_9.

[120] Peter B. Key et Derek McAuley. “Differential QoS and pricing in networks : Where flow control meets game theory”. In : *IEE Proceedings - Software* 146.1 (1999), p. 39–43. doi : 10.1049/ip-sen:19990154. url : https://doi.org/10.1049/ip-sen:19990154.

[121] Lars Kotthoff. “Algorithm Selection for Combinatorial Search Problems : A Survey”. In : *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*. Sous la dir. de Christian Bessiere et al. T. 10101. Lecture Notes in Computer Science. Springer, 2016, p. 149–190. doi : 10.1007/978-3-319-50137-6_7. url : https://doi.org/10.1007/978-3-319-50137-6%5C_7.

[122] Arnaud Lallouet et Anthony Palmieri. “Constraint Games for Modeling and Solving Pure Nash Equilibria, Price of Anarchy and Pareto Efficient Equilibria”. In : *13th European Meeting on Game Theory, SING 2017*. 2017.

[123] Javier Larrosa. “Node and Arc Consistency in Weighted CSP”. In : *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*. Sous la dir. de Rina Dechter, Michael J. Kearns et Richard S. Sutton. AAAI Press / The MIT Press, 2002, p. 48–53. url : http://www.aaai.org/Library/AAAI/2002/aaai02-008.php.

[124] Javier Larrosa et Thomas Schiex. “Solving weighted CSP by maintaining arc consistency”. In : *Artificial Intelligence* 159.1-2 (2004), p. 1–26.

[125] Y. Law et J. Lee. “Symmetry Breaking Constraints for Value Symmetries in Constraint Satisfaction”. In : *Constraints* 11.2-3 (2006), p. 221–267. doi : 10.1007/s10601-006-7095-8. url : https://doi.org/10.1007/s10601-006-7095-8.

[126] Siamak Layeghy, Farzaneh Pakzad et Marius Portmann. “SCOR : Constraint Programming-based Northbound Interface for SDN”. In : *26th International Telecommunication Networks and Applications Conference, ITNAC 2016, Dunedin, New Zealand, December 7-9, 2016*. IEEE, 2016, p. 83–88. isbn : 978-1-5090-0919-0. doi : 10.1109/ATNAC.2016.7878788. url : http://doi.ieeecomputersociety.org/10.1109/ATNAC.2016.7878788.

[127] Christophe Lecoutre et al. “Recording and Minimizing Nogoods from Restarts”. In : *JSAT* 1.3-4 (2007), p. 147–167. url : https://satassociation.org/jsat/index.php/jsat/article/view/13.

[128] C. Lecoutre et al. “Reasoning from last conflict(s) in constraint programming”. In : *ai* 173 (2009), p. 1592, 1614.

[129] Jimmy H. M. Lee, Christian Schulte et Zichen Zhu. “Increasing Nogoods in Restart-Based Search”. In : *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.* Sous la dir. de Dale Schuurmans et Michael P. Wellman. AAAI Press, 2016, p. 3426–3433. url : http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12357.

[130] Jérémie Leguay et al. “Online and Global Network Optimization : Towards the Next-Generation of Routing Platforms”. In : *CoRR* abs/1602.01629 (2016). arXiv : 1602.01629. url : http://arxiv.org/abs/1602.01629.

[131] Nicolas Levasseur, Patrice Boizumault et Samir Loudni. “A value ordering heuristic for weighted CSP”. In : *Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference on.* T. 1. IEEE. 2007, p. 259–262.

[132] Vadim Levit et Amnon Meisels. “Distributed Constrained Search by Selfish Agents for Efficient Equilibria”. In : *Principles and Practice of Constraint Programming - 24th International Conference, CP 2018, Lille, France, August 27-31, 2018, Proceedings.* Sous la dir. de John N. Hooker. T. 11008. Lecture Notes in Computer Science. Springer, 2018, p. 707–724. doi : 10.1007/978-3-319-98334-9_45. url : https://doi.org/10.1007/978-3-319-98334-9%5C_45.

[133] Richard J. Lipton, Evangelos Markakis et Aranyak Mehta. “Playing large games using simple strategies”. In : *Proceedings 4th ACM Conference on Electronic Commerce (EC-2003), San Diego, California, USA, June 9-12, 2003.* ACM, 2003, p. 36–41. doi : 10.1145/779928.779933. url : https://doi.org/10.1145/779928.779933.

[134] Abdel Lisser et Philippe Mahey. “Multicommodity Flow Problems and Decomposition in Telecommunications Networks”. In : *Handbook of Optimization in Telecommunications.* Sous la dir. de Mauricio G. C. Resende et Panos M. Pardalos. Springer, 2006, p. 241–267. isbn : 978-0-387-30662-9. doi : 10.1007/978-0-387-30165-5_10. url : https://doi.org/10.1007/978-0-387-30165-5%5C_10.

[135] Omer Litov et Amnon Meisels. “Distributed Search for Pure Nash Equilibria in Graphical Games : (Extended Abstract)”. In : *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016.* Sous la dir. de Catholijn M. Jonker et al. ACM, 2016, p. 1279–1280. url : http://dl.acm.org/citation.cfm?id=2937119.

[136] Tong Liu et al. “Constraint programming for flexible Service Function Chaining deployment”. In : *CoRR* abs/1812.05534 (2018). arXiv : 1812.05534. URL : http://arxiv.org/abs/1812.05534.

[137] Michele Lombardi et Pierre Schaus. “Cost impact guided lns”. In : *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2014, p. 293–300.

[138] Xavier Lorca et al. “Using Constraint Programming for the Urban Transit Crew Rescheduling Problem”. In : *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*. Sous la dir. de Michel Rueher. T. 9892. Lecture Notes in Computer Science. Springer, 2016, p. 636–649. ISBN : 978-3-319-44952-4. DOI : 10.1007/978-3-319-44953-1_40. URL : https://doi.org/10.1007/978-3-319-44953-1%5C_40.

[139] Alan K. Mackworth. “Consistency in Networks of Relations”. In : *Artif. Intell.* 8.1 (1977), p. 99–118. DOI : 10.1016/0004-3702(77)90007-8. URL : https://doi.org/10.1016/0004-3702(77)90007-8.

[140] Rajiv T. Maheswaran et al. “Taking DCOP to the Real World : Efficient Complete Solutions for Distributed Multi-Event Scheduling”. In : *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004), 19-23 August 2004, New York, NY, USA*. IEEE Computer Society, 2004, p. 310–317. DOI : 10.1109/AAMAS.2004.10067. URL : http://doi.ieeecomputersociety.org/10.1109/AAMAS.2004.10067.

[141] Richard D McKelvey, Andrew M McLennan et Theodore L Turocy. *Gambit : Software tools for game theory*. Version 16.0.0. 2016. URL : http://www.gambit-project.org.

[142] Alaitz Mendiola et al. “A Survey on the Contributions of Software-Defined Networking to Traffic Engineering”. In : *IEEE Communications Surveys and Tutorials* 19.2 (2017), p. 918–953. DOI : 10.1109/COMST.2016.2633579. URL : https://doi.org/10.1109/COMST.2016.2633579.

[143] Pedro Meseguer, Francesca Rossi et Thomas Schiex. “Soft Constraints”. In : *Handbook of Constraint Programming*. Sous la dir. de Francesca Rossi, Peter van Beek et Toby Walsh. T. 2. Foundations of Artificial Intelligence. Elsevier, 2006, p. 281–328. DOI : 10.1016/S1574-6526(06)80013-1. URL : https://doi.org/10.1016/S1574-6526(06)80013-1.

[144] Laurent Michel et Pascal Van Hentenryck. “Activity-based search for black-box constraint programming solvers”. In : *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2012, p. 228–243.

[145] Nina Narodytska. “Reformulation of global constraints”. Thèse de doct. University of New South Wales, Sydney, Australia, 2011. url : http://handle.unsw.edu.au/1959.4/51366.

[146] J.F. Nash. “Non-cooperative Games”. In : *Annals of Mathematics* 54.2 (1951), p. 286–295.

[147] Nicholas Nethercote et al. “MiniZinc : Towards a standard CP modelling language”. In : *Principles and Practice of Constraint Programming–CP 2007* (2007), p. 529–543.

[148] Thi-Van-Anh Nguyen. “Constraint Games : Modeling and Solving Games with Constraints. (Contrainte Jeux : modélisation et Jeux Résolution avec Contraintes)”. Thèse de doct. University of Caen Normandy, France, 2014. url : https://tel.archives-ouvertes.fr/tel-01138960.

[149] Thi-Van-Anh Nguyen et Arnaud Lallouet. “A Complete Solver for Constraint Games”. In : *CP 2014, Lyon, France, September 8-12, 2014*. Sous la dir. de Barry O’Sullivan. T. 8656. LNCS. Springer, 2014, p. 58–74. doi : 10.1007/978-3-319-10428-7_8. url : http://dx.doi.org/10.1007/978-3-319-10428-7_8.

[150] Thi-Van-Anh Nguyen, Arnaud Lallouet et Lucas Bordeaux. “Constraint Games : Framework and Local Search Solver”. In : *ICTAI*. Sous la dir. d’É. Grégoire et B. Mazure. Special Track on SAT and CSP Technologies. Springer, 2013, p. 963–970.

[151] Changhai Nie et Hareton Leung. “A Survey of Combinatorial Testing”. In : *ACM Comput. Surv.* 43.2 (fév. 2011), 11 :1–11 :29. issn : 0360-0300. doi : 10.1145/1883612.1883618. url : http://doi.acm.org/10.1145/1883612.1883618.

[152] Eugene Nudelman et al. “Run the GAMUT : A Comprehensive Approach to Evaluating Game-Theoretic Algorithms”. In : *AAMAS*. `http://gamut.stanford.edu/`. IEEE Computer Society, 2004, p. 880–887. isbn : 1-58113-864-4. url : http://gamut.stanford.edu/.

[153] Steve Nugent et al. “Behavioural Strategies in Boolean Games”. In : *International Workshop on Strategic Reasoning*. 2017.

[154] Ariel Orda, Raphael Rom et Nahum Shimkin. “Competitive Routing in Multi-User Communication Networks”. In : *Proceedings IEEE INFOCOM ’93, The Conference on Computer Communications, Twelfth Annual Joint Conference of the IEEE Computer and Communications Societies, Networking : Foundation for the Future, San Francisco, CA, USA, March 28 - April 1, 1993*. IEEE, 1993, p. 964–971. isbn : 0-8186-3580-0. doi : 10.1109/INFCOM.1993.253270. url : https://doi.org/10.1109/INFCOM.1993.253270.

[155] S. Orlowski et al. “SNDlib 1.0 – Survivable Network Design Library”. English. In : *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*. http ://sndlib.zib.de, extended version accepted in Networks, 2009. 2007. url : http://www.zib.de/orlowski/Paper/OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC.pdf.gz.

[156] Luis E. Ortiz et Michael J. Kearns. “Nash Propagation for Loopy Graphical Games”. In : *NIPS 2002, December 9-14, 2002, Vancouver, British Columbia, Canada]*. Sous la dir. de S. Becker, S. Thrun et K. Obermayer. MIT Press, 2002, p. 793–800. url : http://papers.nips.cc/paper/2263-nash-propagation-for-loopy-graphical-games.

[157] M.J. Osborne et A. Rubinstein. *A Course in Game Theory*. The MIT Press, 1994.

[158] Afif Osseiran et al. “The foundation of the mobile and wireless communications system for 2020 and beyond : Challenges, enablers and technology solutions”. In : *Vehicular Technology Conference (VTC Spring), 2013 IEEE 77th*. IEEE. 2013, p. 1–5.

[159] Anthony Palmieri et Arnaud Lallouet. “Constraint Games revisited”. In : *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Sous la dir. de Carles Sierra. ijcai.org, 2017, p. 729–735. isbn : 978-0-9992411-0-3. doi : 10.24963/ijcai.2017/101. url : https://doi.org/10.24963/ijcai.2017/101.

[160] Anthony Palmieri, Arnaud Lallouet et Luc Pons. “Constraint Games for stable and optimal allocation of demands in SDN”. In : *Constraints* (2019).

[161] Anthony Palmieri et Guillaume Perez. “Objective as a Feature for Robust Search Strategies”. In : *International Conference on Principles and Practice of Constraint Programming*. Springer. 2018, p. 328–344.

[162] Anthony Palmieri, Jean-Charles Régin et Pierre Schaus. “Parallel strategies selection”. In : *International Conference on Principles and Practice of Constraint Programming*. Springer. 2016, p. 388–404.

[163] Anastasia Paparrizou et Kostas Stergiou. “On Neighborhood Singleton Consistencies”. In : *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Sous la dir. de Carles Sierra. ijcai.org, 2017, p. 736–742. doi : 10.24963/ijcai.2017/102. url : https://doi.org/10.24963/ijcai.2017/102.

[164] Claude Le Pape et al. “Robust and Parallel Solving of a Network Design Problem”. In : *Principles and Practice of Constraint Programming - CP 2002, 8th International Conference, CP 2002, Ithaca, NY, USA, September 9-13, 2002, Proceedings*. Sous la dir. de P. Van Hentenryck. T. 2470. LNCS. Springer, 2002, p. 633–648. isbn : 3-540-44120-4. doi : 10.1007/3-540-46135-3_42. url : https://doi.org/10.1007/3-540-46135-3_42.

[165] Stefano Paris et al. “Controlling flow reconfigurations in SDN”. In : *35th Annual IEEE International Conference on Computer Communications, INFOCOM 2016, San Francisco, CA, USA, April 10-14, 2016*. IEEE, 2016, p. 1–9. isbn : 978-1-4673-9953-1. doi : 10.1109/INFOCOM.2016.7524330. url : https://doi.org/10.1109/INFOCOM.2016.7524330.

[166] Marc Pauly. “A Modal Logic for Coalitional Power in Games”. In : *J. Log. Comput.* 12.1 (2002), p. 149–166. doi : 10.1093/logcom/12.1.149. url : https://doi.org/10.1093/logcom/12.1.149.

[167] Guillaume Perez et Jean-Charles Régin. “Efficient Operations On MDDs for Building Constraint Programming Models”. In : *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Sous la dir. de Qiang Yang et Michael Wooldridge. AAAI Press, 2015, p. 374–380. url : http://ijcai.org/Abstract/15/059.

[168] Guillaume Perez et Jean-Charles Régin. “Constructions and In-Place Operations for MDDs Based Constraints”. In : *Integration of AI and OR Techniques in Constraint Programming - 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29 - June 1, 2016, Proceedings*. Sous la dir. de Claude-Guy Quimper. T. 9676. Lecture Notes in Computer Science. Springer, 2016, p. 279–293. doi : 10.1007/978-3-319-33954-2_20. url : https://doi.org/10.1007/978-3-319-33954-2%5C_20.

[169] Gilles Pesant. “Counting-Based Search for Constraint Optimization Problems”. In : *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA.* 2016, p. 3441–3448. URL : http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12065.

[170] Gilles Pesant, Claude-Guy Quimper et Alessandro Zanarini. “Counting-Based Search : Branching Heuristics for Constraint Satisfaction Problems.” In : *J. Artif. Intell. Res.(JAIR)* 43 (2012), p. 173–210.

[171] Karen E. Petrie et Barbara M. Smith. “Symmetry Breaking in Graceful Graphs”. In : *Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings.* Sous la dir. de Francesca Rossi. T. 2833. Lecture Notes in Computer Science. Springer, 2003, p. 930–934. DOI : 10.1007/978-3-540-45193-8_81. URL : https://doi.org/10.1007/978-3-540-45193-8%5C_81.

[172] Quang-Dung Pham et Yves Deville. “Solving the Longest Simple Path Problem with Constraint-Based Techniques”. In : *Integration of Constraint Programming, Artificial Intelligence and Operations Research, CPAIOR 2012, Nantes, France, May 28 - June1, 2012.* Sous la dir. de Nicolas Beldiceanu, Narendra Jussien et Eric Pinson. T. 7298. LNCS. Springer, 2012, p. 292–306. ISBN : 978-3-642-29827-1. DOI : 10.1007/978-3-642-29828-8_19. URL : https://doi.org/10.1007/978-3-642-29828-8_19.

[173] Patrick Prosser, Kostas Stergiou et Toby Walsh. “Singleton Consistencies”. In : *Principles and Practice of Constraint Programming - CP 2000, 6th International Conference, Singapore, September 18-21, 2000, Proceedings.* Sous la dir. de Rina Dechter. T. 1894. Lecture Notes in Computer Science. Springer, 2000, p. 353–368. DOI : 10.1007/3-540-45349-0_26. URL : https://doi.org/10.1007/3-540-45349-0%5C_26.

[174] Charles Prud’homme, Jean-Guillaume Fages et Xavier Lorca. *Choco Documentation.* TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. 2017. URL : http://www.choco-solver.org.

[175] Anuj Puri et Stavros Tripakis. “Algorithms for the Multi-constrained Routing Problem”. In : *Algorithm Theory - SWAT 2002, 8th Scandinavian Workshop on Algorithm Theory, Turku, Finland, July 3-5, 2002 Proceedings.* Sous la dir. de Martti Penttonen et Erik Meineche Schmidt. T. 2368. Lecture Notes in Computer Science.

Springer, 2002, p. 338–347. doi : 10.1007/3-540-45471-3_35. url : https://doi.org/10.1007/3-540-45471-3%5C_35.

[176] Qiaofeng Qin et al. “SDN Controller Placement at the Edge : Optimizing Delay and Overheads”. In : *2018 IEEE Conference on Computer Communications, INFOCOM 2018, Honolulu, HI, USA, April 16-19, 2018*. IEEE, 2018, p. 684–692. doi : 10.1109/INFOCOM.2018.8485963. url : https://doi.org/10.1109/INFOCOM.2018.8485963.

[177] Pham Tran Anh Quang et al. “Multi-objective multi-constrained QoS Routing in large-scale networks : A genetic algorithm approach”. In : *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*. IEEE. 2018, p. 55–60.

[178] Luc De Raedt, Tias Guns et Siegfried Nijssen. “Constraint programming for itemset mining”. In : *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*. Sous la dir. d’Ying Li, Bing Liu et Sunita Sarawagi. ACM, 2008, p. 204–212. isbn : 978-1-60558-193-4. doi : 10.1145/1401890.1401919. url : http://doi.acm.org/10.1145/1401890.1401919.

[179] Arathi Ramani et Igor L. Markov. “Automatically Exploiting Symmetries in Constraint Programming”. In : *Recent Advances in Constraints, Joint ERCIM/CoLogNet International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2004, Lausanne, Switzerland, June 23-25, 2004, Revised Selected and Invited Papers*. Sous la dir. de Boi Faltings et al. T. 3419. Lecture Notes in Computer Science. Springer, 2004, p. 98–112. doi : 10.1007/11402763_8. url : https://doi.org/10.1007/11402763%5C_8.

[180] Philippe Refalo. “Impact-Based Search Strategies for Constraint Programming”. In : *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*. Sous la dir. de Mark Wallace. T. 3258. Lecture Notes in Computer Science. Springer, 2004, p. 557–571. doi : 10.1007/978-3-540-30201-8_41. url : https://doi.org/10.1007/978-3-540-30201-8%5C_41.

[181] Jean-Charles Régin. “A Filtering Algorithm for Constraints of Difference in CSPs”. In : *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1*. 1994, p. 362–367. url : http://www.aaai.org/Library/AAAI/1994/aaai94-055.php.

[182] Jean-Charles Régin. “A filtering algorithm for constraints of difference in CSPs”. In : *Aaai*. T. 94. 1994, p. 362–367.

[183] Jean-Charles Régin. “Arc Consistency for Global Cardinality Constraints with Costs”. In : *Principles and Practice of Constraint Programming - CP’99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*. Sous la dir. de Joxan Jaffar. T. 1713. Lecture Notes in Computer Science. Springer, 1999, p. 390–404. doi : 10.1007/978-3-540-48085-3_28. url : https://doi.org/10.1007/978-3-540-48085-3%5C_28.

[184] Jean-Charles Régin. “The Symmetric Alldiff Constraint”. In : *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*. Sous la dir. de Thomas Dean. Morgan Kaufmann, 1999, p. 420–425. url : http://ijcai.org/Proceedings/99-1/Papers/061.pdf.

[185] Jean-Charles Régin. “Modeling problems in constraint programming”. In : *Tutorial CP* 4 (2004).

[186] Jean-Charles Régin et Jean-Francois Puget. “A Filtering Algorithm for Global Sequencing Constraints”. In : *Principles and Practice of Constraint Programming - CP97, Third International Conference, Linz, Austria, October 29 - November 1, 1997, Proceedings*. Sous la dir. de Gert Smolka. T. 1330. Lecture Notes in Computer Science. Springer, 1997, p. 32–46. doi : 10.1007/BFb0017428. url : https://doi.org/10.1007/BFb0017428.

[187] Jean-Charles Régin et Michel Rueher. “Inequality-sum : a global constraint capturing the objective function”. In : *RAIRO - Operations Research* 39.2 (2005), p. 123–139. doi : 10.1051/ro:2005007. url : https://doi.org/10.1051/ro:2005007.

[188] Brian Roberson. “The Colonel Blotto Game”. In : *Economic Theory* 29.1 (2006), p. 1–24. issn : 1432-0479. doi : 10.1007/s00199-005-0071-5. url : http://dx.doi.org/10.1007/s00199-005-0071-5.

[189] J. B. Rosen. “Existence and Uniqueness of Equilibrium Points for Concave n-Person Games”. In : *Econometrica* 33.3 (1965), p. 520–534.

[190] Francesca Rossi, Peter van Beek et Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA : Elsevier Science Inc., 2006. isbn : 0444527265.

[191] Francesca Rossi, Kristen Brent Venable et Toby Walsh. “Preferences in Constraint Satisfaction and Optimization”. In : *AI Magazine* 29.4 (2008), p. 58–68.

[192] Alvin E Roth. “The economist as engineer : Game theory, experimentation, and computation as tools for design economics”. In : *Econometrica* 70.4 (2002), p. 1341–1378.

[193] Tim Roughgarden. “Routing Games”. In : *Algorithmic game theory*. Algorithmic game theory. Cambridge University Press, 2007. Chap. 18, p. 461–486.

[194] Pierre Roy et al. “Enforcing Structure on Temporal Sequences : The Allen Constraint”. In : *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*. Sous la dir. de Michel Rueher. T. 9892. Lecture Notes in Computer Science. Springer, 2016, p. 786–801. doi : 10.1007/978-3-319-44953-1_49. url : https://doi.org/10.1007/978-3-319-44953-1%5C_49.

[195] Aviad Rubinstein. “Inapproximability of Nash Equilibrium”. In : *SIAM J. Comput.* 47.3 (2018), p. 917–959. doi : 10.1137/15M1039274. url : https://doi.org/10.1137/15M1039274.

[196] Jean-Michel Sanner et al. “An evolutionary controllers’ placement algorithm for reliable SDN networks”. In : *2017 13th International Conference on Network and Service Management (CNSM)*. IEEE. 2017, p. 1–6.

[197] Pierre Schaus, Pascal Van Hentenryck et Jean-Charles Régin. “Scalable Load Balancing in Nurse to Patient Assignment Problems”. In : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009, Pittsburgh, PA, USA, May 27-31, 2009, Proceedings*. Sous la dir. de Willem Jan van Hoeve et John N. Hooker. T. 5547. Lecture Notes in Computer Science. Springer, 2009, p. 248–262. isbn : 978-3-642-01928-9. doi : 10.1007/978-3-642-01929-6_19. url : https://doi.org/10.1007/978-3-642-01929-6%5C_19.

[198] Pierre Schaus et al. “Cardinality Reasoning for Bin-Packing Constraint : Application to a Tank Allocation Problem”. In : *Principles and Practice of Constraint Programming - 18th International Conference, CP 2012, Québec City, QC, Canada, October 8-12, 2012. Proceedings*. Sous la dir. de Michela Milano. T. 7514. Lecture Notes in Computer Science. Springer, 2012, p. 815–822. isbn :

978-3-642-33557-0. DOI : 10.1007/978-3-642-33558-7_58. URL : https://doi.org/10.1007/978-3-642-33558-7%5C_58.

[199] Thomas SCHIEX. “Possibilistic Constraint Satisfaction Problems or "How to handle soft constraints ?"”. In : *CoRR* abs/1303.5427 (2013). arXiv : 1303.5427. URL : http://arxiv.org/abs/1303.5427.

[200] Christian SCHULTE et Mats CARLSSON. “Finite Domain Constraint Programming Systems”. In : *Handbook of Constraint Programming*. 2006, p. 495–526. DOI : 10.1016/S1574-6526(06)80018-0. URL : https://doi.org/10.1016/S1574-6526(06)80018-0.

[201] Meinolf SELLMANN, Thorsten GELLERMANN et Robert WRIGHT. “Cost-based Filtering for Shorter Path Constraints”. In : *Constraints* 12.2 (2007), p. 207–238. DOI : 10.1007/s10601-006-9006-4. URL : https://doi.org/10.1007/s10601-006-9006-4.

[202] Sandip SEN, Stéphane AIRIAU et Rajatish MUKHERJEE. “Towards a pareto-optimal solution in general-sum games”. In : *The Second International Joint Conference on Autonomous Agents & Multiagent Systems, AAMAS 2003, July 14-18, 2003, Melbourne, Victoria, Australia, Proceedings*. ACM, 2003, p. 153–160. DOI : 10.1145/860575.860600. URL : https://doi.org/10.1145/860575.860600.

[203] Paul SHAW. “Using constraint programming and local search methods to solve vehicle routing problems”. In : *International conference on principles and practice of constraint programming*. Springer. 1998, p. 417–431.

[204] Helmut SIMONIS et Barry O’SULLIVAN. “Search Strategies for Rectangle Packing”. In : *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings*. Sous la dir. de Peter J. STUCKEY. T. 5202. Lecture Notes in Computer Science. Springer, 2008, p. 52–66. DOI : 10.1007/978-3-540-85958-1_4. URL : https://doi.org/10.1007/978-3-540-85958-1%5C_4.

[205] Alexander SKOPALIK et Berthold VÖCKING. “Inapproximability of pure nash equilibria”. In : *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*. Sous la dir. de Cynthia DWORK. ACM, 2008, p. 355–364. DOI : 10.1145/1374376.1374428. URL : https://doi.org/10.1145/1374376.1374428.

[206] Barbara M. Smith. “Symmetry and Search in a Network Design Problem”. In : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings*. Sous la dir. de Roman Barták et Michela Milano. T. 3524. Lecture Notes in Computer Science. Springer, 2005, p. 336–350. doi : 10.1007/11493853_25. url : https://doi.org/10.1007/11493853%5C_25.

[207] Barbara M. Smith. “Modelling”. In : *Handbook of Constraint Programming*. Sous la dir. de Francesca Rossi, Peter van Beek et Toby Walsh. T. 2. Foundations of Artificial Intelligence. Elsevier, 2006, p. 377–406. doi : 10.1016/S1574-6526(06)80015-5. url : https://doi.org/10.1016/S1574-6526(06)80015-5.

[208] Barbara M Smith et Stuart A Grant. “Trying harder to fail first”. In : *RESEARCH REPORT SERIES-UNIVERSITY OF LEEDS SCHOOL OF COMPUTER STUDIES LU SCS RR* (1997).

[209] Barbara M. Smith et Paula Sturdy. “Value Ordering for Finding All Solutions”. In : *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*. Sous la dir. de Leslie Pack Kaelbling et Alessandro Saffiotti. Professional Book Center, 2005, p. 311–316. url : http://ijcai.org/Proceedings/05/Papers/1122.pdf.

[210] John Maynard Smith. *Evolution and the Theory of Games*. Cambridge university press, 1982.

[211] Seungbeom Song et al. “A congestion avoidance algorithm in SDN environment”. In : *2016 International Conference on Information Networking, ICOIN 2016, Kota Kinabalu, Malaysia, January 13-15, 2016*. IEEE Computer Society, 2016, p. 420–423. doi : 10.1109/ICOIN.2016.7427148. url : https://doi.org/10.1109/ICOIN.2016.7427148.

[212] Ted H. Szymanski. “Max-Flow Min-Cost Routing in a Future-Internet with Improved QoS Guarantees”. In : *IEEE Trans. Communications* 61.4 (2013), p. 1485–1497. doi : 10.1109/TCOMM.2013.020713.110882. url : https://doi.org/10.1109/TCOMM.2013.020713.110882.

[213] Mohammad Tajiki et al. “Joint QoS and congestion control based on traffic prediction in SDN”. In : *Applied Sciences* 7.12 (2017), p. 1265.

[214] S. Vassilaras et al. “The Algorithmic Aspects of Network Slicing”. In : *IEEE Communications Magazine* (2017).

[215] David Vickrey et Daphne Koller. “Multi-Agent Algorithms for Solving Graphical Games”. In : *AAAI 2002, July 28 - August 1, 2002, Edmonton, Alberta, Canada.* Sous la dir. de Rina Dechter et Richard S. Sutton. AAAI Press / The MIT Press, 2002, p. 345–351. url : http://www.aaai.org/Library/AAAI/2002/aaai02-053.php.

[216] Petr Vilim, Philippe Laborie et Paul Shaw. “Failure-Directed Search for Constraint-Based Scheduling”. In : *Integration of AI and OR Techniques in Constraint Programming - 12th International Conference, CPAIOR 2015, Barcelona, Spain, May 18-22, 2015, Proceedings.* Sous la dir. de Laurent Michel. T. 9075. Lecture Notes in Computer Science. Springer, 2015, p. 437–453. doi : 10.1007/978-3-319-18008-3_30. url : https://doi.org/10.1007/978-3-319-18008-3%5C_30.

[217] John Von Neumann et Oskar Morgenstern. *Theory of Games and Economic Behavior.* Princeton University Press, 1944. isbn : 0691119937. url : http://jmvidal.cse.sc.edu/library/neumann44a.pdf.

[218] Georg Henrik Von Wright. “The logic of preference”. In : (1963).

[219] Mohamed Wahbi et Kenneth N. Brown. “A Distributed Asynchronous Solver for Nash Equilibria in Hypergraphical Games”. In : *ECAI 2016, 29 August-2 September 2016, The Hague, The Netherlands.* Sous la dir. de G. A. Kaminka et al. T. 285. Frontiers in Artificial Intelligence and Applications. IOS Press, 2016, p. 1291–1299. isbn : 978-1-61499-671-2. doi : 10.3233/978-1-61499-672-9-1291. url : http://dx.doi.org/10.3233/978-1-61499-672-9-1291.

[220] Mark Wallace. “Practical Applications of Constraint Programming”. In : *Constraints* 1.1/2 (1996), p. 139–168. doi : 10.1007/BF00143881. url : https://doi.org/10.1007/BF00143881.

[221] Richard J. Wallace. “Neighbourhood SAC : Extensions and new algorithms”. In : *AI Commun.* 29.2 (2016), p. 249–268. doi : 10.3233/AIC-150696. url : https://doi.org/10.3233/AIC-150696.

[222] Toby Walsh. “Depth-bounded Discrepancy Search”. In : *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes.* Morgan Kaufmann, 1997, p. 1388–1395. url : http://ijcai.org/Proceedings/97-2/Papers/086.pdf.

[223] Toby Walsh. “General Symmetry Breaking Constraints”. In : *Principles and Practice of Constraint Programming - CP 2006, 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006, Proceedings*. Sous la dir. de Frédéric Benhamou. T. 4204. Lecture Notes in Computer Science. Springer, 2006, p. 650–664. doi : 10.1007/11889205_46. url : https://doi.org/10.1007/11889205%5C_46.

[224] GPS Waze Social. “Maps and Traffic”. In : *URL http ://www. waze. com/. Last visited* (2013), p. 08–26.

[225] Wei Xia et Roland H. C. Yap. “Learning Robust Search Strategies Using a Bandit-Based Approach”. In : *AAAI Conference on Artificial Intelligence*. 2018.

[226] Lin Xu et al. “SATzilla : Portfolio-based Algorithm Selection for SAT”. In : *J. Artif. Intell. Res.* 32 (2008), p. 565–606. doi : 10.1613/jair.2490. url : https://doi.org/10.1613/jair.2490.

[227] Linbin Yu et al. “ACTS : A Combinatorial Test Generation Tool”. In : *Sixth IEEE International Conference on Software Testing, Verification and Validation, ICST 2013, Luxembourg, Luxembourg, March 18-22, 2013*. IEEE Computer Society, 2013, p. 370–375. isbn : 978-1-4673-5961-0. doi : 10.1109/ICST.2013.52. url : https://doi.org/10.1109/ICST.2013.52.

[228] Tal Ze’evi, Roie Zivan et Omer Lev. “Socially Motivated Partial Cooperation in Multi-agent Local Search”. In : *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*. Sous la dir. de Jérôme Lang. ijcai.org, 2018, p. 583–589. doi : 10.24963/ijcai.2018/81. url : https://doi.org/10.24963/ijcai.2018/81.

[229] Ernst Zermelo. “Über eine Anwendung der Mengenlehre auf die Theorie des Schachspiels”. In : *Proceedings of the fifth international congress of mathematicians*. T. 2. II, Cambridge UP, Cambridge. 1913, p. 501–504.

[230] Heytem Zitoun et al. “Search strategies for floating point constraint systems”. In : *International Conference on Principles and Practice of Constraint Programming*. Springer. 2017, p. 707–722.

[231] Roie Zivan et al. “Partial cooperation in multi-agent search”. In : *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*. International Foundation for Autonomous Agents et Multiagent Systems. 2012, p. 1267–1268.

Résumé
Nouvelles techniques pour les Contraints Games

Cette thèse présente de nouvelles techniques pour les Constraint Games. La manière de résoudre un Constraint Game est repensée en terme de propagation de contraintes. Les préférences des joueurs sont maintenant considérées comme des contraintes globales permettant une intégration transparente dans les solveurs de contraintes ainsi que d'améliorer l'efficacité du framework. Notre nouveau solveur ConGA est diffusé en open source [1]. Celui-ci est plus rapide que les travaux connexes et est capable de trouver tous les équilibres de Nash, et cela même dans des jeux avec 200 joueurs voir 2000 pour certains jeux graphiques.

Grâce à cette perspective, le framework a pu être utilisé pour résoudre un problème de routage dans le domaine des télécommunications. Les aspects centralisé et décentralisé ont été étudiés. La comparaison de ces derniers est très importante pour évaluer la qualité de service dans les applications multi-utilisateurs. L'évaluation de cette dernière peut être très coûteuse, c'est pourquoi nous proposons plusieurs techniques permettant d'améliorer la résolution de ce problème et ainsi d'améliorer la résolution du problème.

Constraint Games revisited

This thesis revisits the Constraint games framework by rethinking their solving technique in terms of constraint propagation. Players preferences are considered as global constraints making transparently the integration in constraints solvers. It yields not only a more elegant but also a more efficient framework. We release our new solver ConGA in open source[1]. Our new complete solver is faster than previous state-of-the-art and is able to find all pure Nash equilibrium for some problems with 200 players or even with 2000 players in graphical games.

This new perspective enables us to tackle real-worlds Telecommunication problems. This problem is solved with a centralized perspective and a decentralized one. The comparison of the two last approaches is really important to evaluate the quality of service in multi-users application, but computationally consuming. That is why, we propose new techniques in order to improve the resolution process.

Mots-clefs — Constraint Programming, Game theory, Nash Equilibrium, Search strategy

1. https ://github.com/palmieri-a/CONGA