UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Habilitation à Diriger des Recherches

Charlotte TRUCHET

Mémoire présenté en vue de l'obtention de
l'Habilitation à Diriger des Recherches à l'Université de Nantes
sous le sceau de l'Université Bretagne Loire

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique et applications, section CNU 27
Unité de recherche : Laboratoire des Sciences du Numérique de Nantes (LS2N)

Soutenue le *date à définir*

Vers une programmation par contraintes moins contrainte

Outils venant de l'Interprétation Abstraite et de l'Analyse en Moyenne pour définir des solveurs expressifs et faciles à utiliser

JURY

Rapporteurs :	**M^me^ Nadia CREIGNOU**, Professeure, Aix-Marseille Université
	M. Christian SCHULTE, Professeur, KTH Royal Institute of Technology, Suède
	M. Pascal VAN HENTENRYCK, Professeur, University of Michigan, USA
Examinateurs :	**M. Gérard ASSAYAG**, Directeur de Recherches, IRCAM
	M. Frédéric BENHAMOU, Professeur, Université de Nantes
	M. Eric GOUBAULT, Professeur, Ecole Polytechnique
	M. Thomas JENSEN, Directeur de Recherches, INRIA
	M^me^ Brigitte VALLÉE, Directrice de Recherches, CNRS

A Justine et Victor,
mes deux merveilles.

Remerciements

First of all I want to thank Anne Siegel, who offered me her talents as a habilitation planner. Anne is a wonderful ass-kicker. She forced me to clarify my ideas, respect my deadlines and, more importantly, value my work. I was lucky to benefit from her precious lessons and I am extremely grateful to her.

This document was kindly proofread by several colleagues. Their comments were useful and precious, which is no surprise because they are themselves precious : thank you David Cachera, Benoît Delahaye, Damien Eveillard, Laure Gonnord, Xavier Lorca, David Monniaux, Marie Pelleau, Charles Prudhomme, Ghiles Ziat.

I would like to sincerely thank Pascal Van Hentenryck, Christian Schulte, Nadia Creignou, Brigitte Vallée, Gérard Assayag, Eric Goubault, Thomas Jensen and Frédéric Benhamou for accepting to be in my jury. This is the best jury I could dream of !

Supervising PhDs is a hard and stressful task. Fortunately for me, I have been very lucky with my PhD students, who, each in their own way, quickly became hard-working, passionate colleagues. Thank you so much Marie Pelleau, Bruno Belin, Anicet Bart, Ghiles Ziat and Giovanni Lo Bianco for bringing your youth and energy to our researches. I know for sure that each one of you will get the great career he/she deserves.

In 2015, Serge Abiteboul invited me to join the editorial team of the Binaire blog. Only later I discovered that, more than an editorial board, it was a cluster of friends where mutual help and stupid jokes were mandatory. I love being one of them (poil aux lemmes).

Narendra Jussien and Christine Solnon suggested that I should join the board of the French CP association (AFPC). Later, Pierre Flener and Laurent Michel invited me to apply to the executive committee of the international CP association (ACP). Michel Rueher asked me to be a senior PC chair for the CP conference. I would like to thank them for giving me these opportunities, which were very important for me.

Brigitte Vallée, Philippe Flajolet and Danièle Gardy invited me to join the Alea community, in particular through the Boole project. This was a turning point in my career, because it allowed me to discuss with real average-case analysts. Besides, Alea is a great, welcoming, passionate community. Every meeting with them is a pleasure.

I would not pretend that I knew Philippe Flajolet well, yet, when he died, I cried (I never cry). He was obviously a great mind, and more importantly, one of the very few brilliant people who use their cleverness not to show that they are themselves intelligent, but to reveal other's intelligence. The day he died, many computer scientists cried because, I think, we lost an ideal. Thank you so much, Philippe, for being who you were. The afternoon we spent at Rocquencourt trying to figure out how to calculate the expectation of LS markov chains (and failing) is one of my best research memories.

Being a researcher gives the privilege to discuss with brilliant people, and for me, this is one of the most important benefits of the job. For sharing ideas, thoughts and laughs, thank you Antoine Miné, Frédéric

Saubion, Ali Akhavi, Pierre Roy, Camille Coti, Emmanuel Morin, Arnaud Lanoix, Christian Attiogbé, Alan Schmitt, Eric Monfroy, Annie Tartier, Marie-Madeleine Tallineau, Sylvie Cazalens, Philippe Codognet, Marc Christie, Lorraine Goeuriot, Alejandro Arbelaez, and, of course, all of the above. Dear reader, if you have any occasion to discuss with one of them, about anything, anywhere, anytime, just take it !

When I tried twitter, I did not expect it to be so useful, allowing me to discuss with colleagues in physics, math, philosophy, chemistry, litterature, biology, history, law... The French scientific community on Twitter is active, fun, united and supportive. They even gave my name to the scale of bad professional train trips - I am very proud. And I've met IRL plenty of great twittos[1]. Thank you twitter, with a special thought for the Bourricots.

Should I thank Jérémie, best life-partner in the world, wonderful father for our kids, and so much more ? No, not a good idea. These 154 pages would not be long enough.

[1] @alpha42, @DevilleSy, @Blouchtika, @Alexis_Verger, @mixlamalice, @frestagn, @fabricehuet, @Brice_Kauffmann, @CecileMichaut, @nlidgi, @gmoreau_ecn, @Mr_Pepeete, @squintar... all worth following !

Contents

List of Tables

List of Figures

Introduction

1

Introduction

1.1 Constraint Programming Users are Constrained

Constraint Programming (CP) is a research area which on the one hand inherits from Artificial Intelligence, for its declarative programming framework, and on the other hand is closely tied to Operations Research for its efficiency in solving combinatorial problems. Constraints are logical relations between variables, which are the unknowns of a problem. The variables are given a domain, *i.e.* a set of values they can take. On a given problem with constraints, variables and domains, a constraint solver finds values in the domains such that the constraints are satisfied. In theory, this declarative programming process is quite simple: the user only needs to write the constraints, and the solver finds solutions to the user's problem. CP has a wide range of successful applications in many areas such as biology, music, transportation, computer-assisted design, etc. CP has for instance been applied to compute plans for power restoration, reducing the duration of black-outs, after a natural disaster [Hijazi et al., 2015], to automatically find chords on a given bass line in classic Western style harmony [Pachet and Roy, 2001], or even in space, to plan the tasks of the famous Philae comet lander based on its available energy resources [Simonin et al., 2015].

In practice, none of the two steps, modeling and solving, are as easy at they seem. First, modeling the problem may be out of reach for several reasons: constraint languages can be too restrictive, not all models[1] are equivalent in particular in terms of solving time, and of course, modeling real-life problems is sometimes a long and difficult trial-and-error process. Second, solving it is definitely not as easy at the solvers documentation usually promises. The search space, *i.e.* the space of all combinations of values of the domains, has an exponential size in the number of variables (whether the problem was initially NP-hard or not). Constraint solvers explore it, sometimes exhaustively and sometimes not. They embed clever techniques to reduce the search space or to guide the search, but these techniques are very sensitive to their intern configuration. Thus, obtaining a good solving time is often out of reach of a non-specialist user. In the end, while the CP technology is extremely powerful, non-expert users have to overcome several obstacles before using it. As a consequence, there are several constraints holding on constraint users: among others, they need to have a certain level of knowledge of the discipline (not only computer science but also constraint programming), they have to project their problems into a given, possibly rigid, CP framework, they have to know how to tune the solvers and they have to guess if the computation time on their problem, as they have expressed it, is feasible or not.

[1] In this document, the word "model" refers to a constraint model of some real-life problem. It should not be confused with SAT models, which are solutions.

1.2 Contributions

In this thesis, I will focus on two of these limitations. They are motivated by my experience in CP applications to domains where the users are unaware of the subtleties of constraint solvers, and even not computer scientists: music, urban planning and chemistry. These applications are described in Chapter 2, with a highlight on the urban planning application.

1.2.1 Analysis of the Behavior of Constraints Algorithms and Solvers

The need for expertise before using a CP solver of any kind (complete or incomplete) is a strong assumption on CP users. In practice, even in cases where CP could be able to provide nice modeling languages for specific applications (as in music or urban planning), one cannot leave non-expert users facing such complex tools. For instance, constraint-based local search has very nice properties for such applications: it scales quite well, the core algorithm, based on iterated local optimization, is transparent, and, though incomplete, it does naturally handle over-constrained problems.

Yet, when writing a dedicated solver based on local search, a crucial step cannot be automated: the choice of the parameters of the local search algorithm (random restart policy, parameters of the metaheuristic). This has to be left to the user. Machine-learning based techniques exist to auto-tune the algorithms (from the simple idea of Reactive Tabu Search to more sophisticated techniques), but they are still at a research level. More than this: in fact, one has little theoretic knowledge on the behavior of these algorithms. The quantified impact of these parameters is known experimentally at best.

Another example is about the model of the problems: CP offers a very rich language of global constraints which can both express specific properties of the problem, and improve the solving time by reducing the search space, which is called propagation. Some of these global constraints embark heavy algorithms: for instance, nearly all of the cardinality constraints, which express restrictions on the number of values that a set of variables can take, use graph or flow algorithms to reduce the search space. Some of these algorithms may be costly, and computing the propagation of a given constraint can even be NP-hard in some cases (propagation of the lower bound of the value in the `nvalue` constraint). It is thus necessary to use them wisely, either by unplugging the global propagation or by tuning the propagation loop.

In these two examples, the real issue is that we have very little knowledge of what is really happening in the solvers. In fact, constraint solvers include many sophisticated mechanism that are seldom studied as such. But I believe it is really necessary to have a better theoretical characterization of their behavior if we want to build solvers that could be accessible to non-users. Part II describes three works in this area which represent an important part of my research activity: a detailed, quantified analysis of the random restart mechanism in local search algorithm, a probabilistic model for the speed-up of parallel Las Vegas algorithms (which include local search) and an average-case analysis of the behavior of the all-different constraint propagation for bound consistency.

1.2.2 More Expressive Domains

One of the limitations of constraints solvers is that they only deal with a single type of variables: there are several solvers on continuous variables (such as Ibex, Realpaver...) and many solvers on discrete variables (such as Choco, Gecode, Sicstus, etc). But the question of solving problems with both real and integer variables has never been elegantly solved. At best, the user must use *ad hoc* techniques which are closer to hacking than to proper typed programmation: either by adding integrity constraints on some variables in a continuous solver, or by discretizing the real variables in a discrete solver. This is not satisfying, because in this case some of the variables lose their constraint language (for instance, adding integrity constraints in a continuous solver makes it impossible to add global constraints). Other methods consist in linking two solvers, as it is the case with Ibex and Choco, or to develop methods on solver cooperation. However, again, the solving process cannot be shared among the two solvers.

Part III describes a work using Abstract Interpretation notions to properly define a mixed[2] solving method. We first studied the methods developed in Abstract Interpretation for static analysis of programs. This idea came from two very basic remarks: first, Abstract Interpretation and Constraint Programming have a lot in common (they both over-approximate sets which are too costly, or impossible, to compute) ; second, static analyzers do not bother analyzing programs with integer and floating-point variables. In the end, this work, conducted with my PhD student Marie Pelleau, was more fruitful than we thought. We decided to take the notion of abstract domain, as it exists in Abstract Interpretation, as an abstraction for the domains in Constraint Programming. By adding a few operators to these abstract domains (choice and precision), we could define a generic solving method which is very similar to the existing one, except that it can accept any abstract domain. This operation opened several perspectives, not only for mixed constraint solving, but also for using relational abstract domains in CP. The case of Octagons is discussed, as well as recent advances on the Box-Polyhedra and the Integer-Real Reduced Product abstract domain.

1.3 Organization of the document

1.3.1 Organization

This document is organized in three parts. Part I introduces the document. Chapter 2 focusses on some applications I have worked on in music, urban planning and chemistry, and details some specificities of real-life applications of CP. A highlight of the urban planning application is then presented in Chapter 3, based on an article published in the Proceedings of CPAIOR 2012 with Marc Christie and Bruno Belin. I chose to highlight this application because it illustrates really well the difference between CP in theory and CP in practice, where there are no models, no off-the-shelf algorithms, and scaling issues, yet there is a huge needs of combinatorial optimization.

Part II presents my contributions in analysis of some constraint algorithms. Chapter 4 explains why, in my opinion, we need to improve our theoretical knowledge of what happens in constraint solvers.

This work originated in the music application (PhD work), where I used a local search algorithm in a library for musicians. There was little theoretical knowledge on the parameter setting for this algorithms, and it was thus impossible to automatically find a good tuning - this task had to be left to the user (non computer scientist). Thus, I later came up with a specific study of random restart, which quantified its influence on local serach performances. Chapter 5 details this contribution. Chapter 6 is based on an article in Journal of Heuristics with Alejandro Arbelaez, Philippe Codognet and Florian Richoux. This article synthetizes several contributions on the analysis of speed-ups of multiwalk parallel Las Vegas algorithms, including Local Search algorithms. This study originated in observations by Philippe Codognet's team on the strange behavior of a multiwalk version of his Adaptive Search algorithm on different problems: in some cases, the speed-up was excellent (up to linear) and in other cases, multiwalk performed poorly. My contribution here is the probabilistic model used to explain the different behaviors. The model is used *a posteriori* to explain the observed speed-ups, and recent works showed that it can also be used *a priori* to predict the speed-up based on the instances features. Finally, Chapter 7 is based on an article with Jérémie du Boisberranger, Danièle Gardy and Xavier Lorca in the Proceedings of Analco 2013. It presents an average-case analysis of one of the most famous global constraints, `all-different`. I contributed to this work first by producing a first probabilistic model, and then developing a collaboration with experts in average-case analysis to improve it. The main result of the article is, in my opinion, the most impressive outcomes of this series of analysis, as it precisely exhibits two regimes where the propagation of bound consistency for all-different is useful or useless.

Part III describes how we used the idea of abstract domains as defined in Abstract Interpretation in CP. Chapter 9 introduces the general framework, and is partly based on an article with Marie Pelleau, Antoine Miné and Frédéric Benhamou at VMCAI 2013. Chapter 10 presents the abstract domain of octagons, and

[2] in this document, we will use the word "mixed" for "mixing integers and reals", unless specified.

an efficient propagation algorithm for them. It is based on a publication to the CP conférence with Marie Pelleau and Frédéric Benhamou. Chapter 11 details how to define and used abstract domains including two different abstract domains, with a focus on the case of boxes and polyheda. Such a tool can use the adequate propagation for each constraint (*e.g.,* polyheda for linear constraints, boxes for non-linear constraints). This chapter relies on an pre-print written in collaboration with Emmanuel Rauzy, Ghiles Ziat, Marie Pelleau and Antoine Miné. My contribution in this series of work is the definition of CP abstract domains, in particular to identify the limitations of Abstract Interpretation domains and introduce new operators dedicated to CP solving, and the definition of the new domains (Octagons and combined domains).

Part 12 concludes this document and presents several axes for further research. Finally, Part V is a detailed vitae of my scientific and teaching activities.

1.3.2 Articles

This document uses several published articles, with the authorization of the co-authors:

- for Part I, Chapter 3: Bruno Belin, Marc Christie, Charlotte Truchet, *Interactive Design of Sustainable Cities with a Distributed Local Search Solver*, CPAIOR 2014: 104-119.
- for Part II:
  - Chapter, 6, Charlotte Truchet, Alejandro Arbelaez, Florian Richoux, Philippe Codognet, *Estimating parallel runtimes for randomized algorithms in constraint solving*, J. Heuristics 22(4): 613-648 (2016).
  - Chapter 7, Jérémie Du Boisberranger, Danièle Gardy, Xavier Lorca, Charlotte Truchet, *When is it worthwhile to propagate a constraint? A probabilistic analysis of AllDifferent*, ANALCO 2013: 80-90.
- for Part III:
  - Chapter 9, Marie Pelleau, Antoine Miné, Charlotte Truchet, Frédéric Benhamou, *A Constraint Solver Based on Abstract Domains*, VMCAI 2013: 434-454.
  - Chapter 10, Marie Pelleau, Charlotte Truchet, Frédéric Benhamou, Octagonal Domains for Continuous Constraints. CP 2011: 706-720.

For each of these chapters, I add a frame where I recall the reference and explain why I think this work is important, what my contribution exactly was, and how it is related to the more general research axis.

2

Synthesis Chapter: Unconventional Applications of Constraint Programming

2.1 Introduction

This chapter investigates the notion of unconventional applications of Constraint Programming. It is based on a talk that I gave at the Doctoral Program of CP2015, where my goal was to convince young CP researchers to not only work on classically successful CP applications, but also to keep an open eye on weird, atypical, real-life applications. In this chapter, I try to formalize what are those atypical applications, and why, in my opinion, they should be encouraged.

2.1.1 Why is CP Longing for Scheduling Problems?

If we consider the program of the main CP conference, which is called *Principles and Practice of Constraint Programming*, Constraint Programming is applied to a relatively narrow range of problems, or at least, its visible applications at the main conference are restricted to a small family of problems. In the Application Tracks of the last three editions of the CP conference[1], out of 34 articles, 16 were on scheduling problems (including resource allocation and planning). If we make the assumption that these articles are a mirror of what the CP community considers interesting, then we can conclude that approximately half of the CP applications which are considered interesting are about scheduling. To be fair, let us notice that the other applications cover a quite wide range of topics: interactive soccer queries, differential cryptanalysis, management of optical access networks, crisis management... And, to be even more fair, last year edition of the CP conference featured thematic tracks, which were introduced precisely to highlight applications of CP in various domains: Verification, Music, Computational Sustainability, Biology and Social Choice. Yet, it is astonishing that such an important part of CP applications focus on a very specific family of problems.

One can only guess why, and this paragraph expresses a personal, debatable opinion. The family of scheduling problems is well modeled and has a very nice property: applications often require to add constraints to deal with specific needs of the application, which makes the scheduling problem an endless source of extensions. For instance, based on the same set of articles as above, scheduling can be combined with other constraints in order to: deal with different resources such as vehicles and crews [Lorca et al.,

[1] CP 2016 program: `http://cp2016.a4cp.org/program/schedule/`
CP 2015 program: `http://booleconferences.ucc.ie/cp2015schedule`
CP 2014 program: `http://cp2014.a4cp.org/program/schedule`

2016, Lam et al., 2015], deal with uncertainty in the data [Pelleau et al., 2014], manage some energy requirements in the problem [Murphy et al., 2015], etc. Other optimization techniques (such as MIP), which are usually efficient, cannot easily integrate such variations. Thus, scheduling problems provide a rich series of problems on which one can be pretty sure that CP has a good added value. Besides, scheduling problems are usually NP-hard [Lenstra and Kan, 1981]. It seems thus fair to apply a technology which, though it is not necessarily restricted to NP-hard problems, is in practice quite efficient on these. Finally, there is also probably an imitation effect: since many articles about scheduling pass at the CP conference, researchers working on scheduling are inclined to publish at CP while researchers working on other applications are inclined to look for other venues.

Is there anything wrong with the passion of CP researchers for planning, resource allocation or scheduling problems? Obviously no. These problems provide a series of successful applications, which highlight the efficiency of CP and in particular its modeling flexibility. Scheduling can be seen as the admiral ship of CP.

2.1.2 Conventional vs Unconventional Applications

What is more annoying is that this admiral ship has no fleet - or its fleet is not visible enough. Applications of CP to real-life problems, outside of computer science for instance, are not enough developed and valued. More precisely, CP is an extremely powerful technology, offering both a declarative programming framework, and efficient algorithms to solve difficult problems. Probably the most often quoted citations about CP is this one: "Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it.", from an article by Eugene Freuder entitled *In Pursuit of the Holy Grail* published in 1997 [Freuder, 1997].

Obviously, if CP really offered what it often claims to offer (declarative programing, expressive modeling language, efficient solving algorithms), everybody would use it: not only everybody in computer science or combinatorial optimization, but everybody in the world. Artists would use it to imagine new creative languages [Truchet and Assayag, 2011], designers would use it to conceive new conception languages [Bettig and Hoffmann, 2011], computer-aided graphic visualization tools would be based on it, [Kieffer et al., 2013], data miners would find patterns thanks to it [De Raedt et al., 2010]. These applications already exist, but they rely on *ad hoc* collaborations. The Holy Grail has not reached all of its potential applications. On the combinatorial nature of CP too, [O'Sullivan, 2012] identifies a series of potential applications that are still challenging to CP (in data center management, public service, human mobilities and natural resource management). Beside, on computer science applications, compared to SAT and SMT solvers which recently gained a huge interest in the Verification community [Monniaux, 2016], or to the many variants of Linear Programming, CP applications, though numerous, remain relatively restricted and far below the Holy Grail promise.

I believe that CP is in practice under-employed. Its potential use in real-life applications beyond the traditional scope of combinatorial optimization applications is great, in particular to develop solving methods based on modeling languages available to non-computer scientist users. Conventional applications of CP could be defined this way:

- the core of the application is a problem with a clean CP model (variables are all of the same types, there are no weird constraints, all the constraints are given before being sent to the solver),
- this problem is NP-hard,
- the main goal is to improve the solving time to find a solution or prove unsatisfiability.

Conventional applications include all the traditional puzzle problems (n-queens, magic square, etc), scheduling and time-tabling problems, problem from combinatorial mathematics (Langford, Golomb rulers, Social Golfer), etc. In these cases, one can use off-the-shelf CP solvers (because no dirty variables nor weird constraints), and the problem being NP-hard guarantees that no *ad hoc* solutions exist.

Beyond conventional applications, there are also several very interesting unconventional applications of combinatorial optimization in the literature: to medicine [Betts et al., 2015, Dickerson et al., 2016], computer graphics [Tao et al., 2008], disaster management [Hentenryck, 2013], computer music [Truchet and Assayag, 2011]. In fact, historically, CP was born from an unconventional application, drawing graphical scenes with shadows, as Jean-François Puget tells on his blog[2].

In the following, we will call unconventional an application where either one of the previous assumptions does not hold, *i.e.* an application where:

- the problem has a complex model (which sometimes requires to interact with the solver),
- or it cannot be easily proved NP-hard,
- the goal is not to quickly find an optimum.

In the following, I review three unconventional applications I have worked on: in computer music (during my PhD), in urban planning (2010-2015), and in chemistry (2015-now). I will use the Computer Music work to show that unconventional applications often require to take care of specific issues, *e.g.,*: (i) forget about NP-hardness, (ii) interact with the user on flawed models, and (iii) make the solving process as transparent as possible.

2.2 Previous works in Computer Music

During my PhD, I worked on a total of 14 applications of constraint programming to computer music, mainly for computer assisted composition and also for musical analysis. Constraint programming is a natural framework for musical problems for two reasons: first, music exhibits many combinatorial structures [Knobloch, 2002, Hooker, 2016]. Second, writing music is often defined with rules: in classical Western music for instance, there are no constructive methods to write nice music, but instead, harmony treaties provide a declarative framework describing the properties that nice music should respect. This is probably why computer music is a natural application for CP, as the many works on the topic show. For a few recent works, see the Music Track at CP 2016 `http://cp2016.a4cp.org`, and for a more general overview, see the book I co-edited with Gérard Assayag in 2011 [Truchet and Assayag, 2011].

Yet, compared to classical constraint applications, the problems I worked on in computer music were unconventional in many ways. First, some of them were NP-hard, other were not, and in fact: nobody cared. The musician user does not really care if her/his problem is NP-hard or NP-complete, he/she just wants it to be solved. Usually, constraint programming is applied to NP-hard problems because the CP format embeds any problem (whether NP-hard or not) into an exponential search space. If the problem is in P, this search space may be easy to prune (think of a simple problem with only equality constraints between n variables for instance), but still, this embedding does naturally drive CP to be used on NP-hard problems. Most of the well-known applications of CP are indeed based on NP-hard problems (Frequency Assignment, Travelling Salesman, Scheduling, etc). But we must not forget that the declarative framework offered by CP can be very valuable for users who are not expert in combinatorial optimization, even on not-NP-hard problems, or more generally, on problems which may or may not be NP-hard (nobody cares). This typically illustrates rule (i).

Second, the musical applications usually did not come with a clean model. In fact, they did not even come with a model, most of the time. The modeling process is a long way to go for two reasons. First, if the user is not an expert in constraint programming, he/she usually cannot express all of his/her constraints from scratch. For example, nearly all of the musicians who wanted to compute a series of musical objects (chords, notes, intervals...) forgot to mention that these objects had to be different, otherwise a constant series satisfies the constraints while having no musical interest. It is thus important that the user gets an

[2] `https://www.ibm.com/developerworks/community/blogs/jfp/entry/constraint_programming_history?lang=en`

easy tool to manipulate, in order to refine her/his constraints depending on the solutions. Recently, several works on constraint learning or constraint acquisition [Bessiere et al., 2017, Beldiceanu and Simonis, 2016] offer a promising way of helping user in the process of writing down their constraints. The second reason is more profound. In real-life cases (and in particular for applications to artistic activities, or to domains where part of the work can be considered handy-crafted), some of the constraints cannot be expressed. In music, it can happen that composers may face a combinatorial problem (sometimes solved by hand) due to the aesthetic rules she/he has defined, and it can also happen that computer tools are used as a way to enhance creativity. In both cases, a computation step is useful, but in the end, the true goal is to produce nice music, not to solve a problem. Whatever the language, there are no constraints saying: this piece is beautiful, this one is not. In such cases, it is extremely important to let the user get the final cut, and provide him/her with a tool offering a wide set of solutions for her/him to choose. Hence, rule (ii) applies here.

Finally, in musical applications as in applications to other domains where the user has a responsibility in the decision, having the technology accepted by the user could be difficult. It is essential to remember that musical piece are signed by a contemporary composer, *i.e.* a human being for whom this signature is extremely important. It is his/her own work, not something produced by a computer. Even if musicians are more and more accustomed to using computers, they often demanded to understand how the solutions are produced. Similar requirements can be found in other fields where the responsibility of the final user is at stake, for instance in medicine: [Betts et al., 2015] describes a local search algorithm for finding good configurations in Prostate Cancer Focal Brachytherapy. In this case, the doctor is personally responsible of the treatment, hence, he/she must be capable of understanding how it has been produced. Thus, it is extremely important to use a transparent technology. This need for transparency is exactly rule (iii).

Finally, in the musical applications, these three rules guided to choice towards local search algorithms: they are easy to understand, it is reasonably easy to refine the model and also possible to play with the constraints (by weighing the error functions), they naturally solve overconstrained problem (by finding good optima minimizing the number of unsatisfied constraints) and they are quite transparent (the principle of iterated local optimizations is easy to understand).

2.3 Highlight on Urban planning

In collaboration with Marc Christie, from IRISA UMR 6074, and Bruno Belin, PhD student at the University of Nantes, I have been involved in a project called Sustain and funded by the French government (FUI) from 2010 to 2015. The goal of the project was to develop a computer-aided system for the early stage of urban planning. The early stage of urban planning consists in drawing a coarse-grain sketch of a future city map, in order to take some important high level decisions: which type of energy plants will be installed, where the plants should be, where will the commercial and industrial zones be, etc. These decisions have a huge impact on the final map of the city, as well as on its sustainability: assume for instance that the industries are all located in the east and all the workers' houses in the west, then by construction the city generates a useless need for transportation. This early stage is thus of crucial importance. However, according to our experts in urban planning, this step is in general still done by hand by the urban planners. The Sustain project aimed at building a system with a meaningful human-computer interface, where experts (urban planners) or decision makers could interact with the city map in order place or move items. The city map was expected to satisfy a series of constraints about its sustainability.

In this project, Marc, Bruno and I were in charge of the optimization part: modeling the problem, implement a solving method, and define possibilities of interaction with partial re-solving (in collaboration with the HMI laboratory also involved in the project). The most difficult part of the work was to discuss with the urban planners to understand their needs. The same remarks which held for computer music applications also hold here: first, we tried to explain that we needed to build a model with variables, domains and constraints. But in practice, we never obtained one from the urban planners. Instead, we relied on our discussions with the experts to build a model ourselves, based on an attraction-repulsion system because

the notion of attraction appeared very often in the discussions (for instance, rich housing areas are attracted to small hills, but this is not a hard constraint). We then showed some maps computed with this model to the urban planners, and they could identify impossible situations. Relying on these discussions, we added hard constraints to forbid these situations. This is typically an example of rules (i) and (ii).

As for rule (iii), urban planners did not necessarily want to understand the solving process, but they still demanded to be able and re-draw the map themselves. The early stage of urban planning is intended to take high level decisions for the future city, hence, it was important that both the urban planners and the decision makers could make hypothesis, move some places to other places and discuss their options on the map. Thus, we added an interactive mode to the solver. First, the problem is solved for an initial solution (due to the huge size of the problems, this could take a really long time), and a solution is displayed to the user. Then, she/he can modify the solution by moving part of the buildings to difference places, and the solver re-solves part of the problem, only on the modified places (considering that the untouched places are locked, *i.e.* they are no longer variables). Since the user can only move small parts of the city, this re-solving process is rather fast.

The constraint part of this work is described in the next chapter 3, which is in great part taken from an article in the Proceedings of CP-AI-OR 2012, written with Marc Christie and Bruno Belin. The different issues presented in this article give a good overview of the specificities of dealing with unconventional applications.

2.4 Ongoing Work in Chemistry

Since 2015, I have been involved in a regional project called SmartCat, in collaboration with François-Xavier Felpin, professor in chemistry, from the CEISAM laboratory at the University of Nantes. Together with Daniel Cortès-Borda, post-doctoral fellow with François-Xavier and him, we worked on optimization of flow chemistry reactions. The traditional process of chemical reactions is batch chemistry: the reactants are placed into a container, where the reactions happens, and the products are collected from the container. On the contrary, flow chemistry is an online process. A pump sends the reactants inside a tube, where the reactions continuously happens. The products are collected at the end of the tube. As in batch chemistry, several parameters can be controlled: temperature (the tube can go through in a oven), pressure (controlled by the pump introducing the reactants), stoechiometry, and catalysis (catalysts can be flowed in the tube in the same way as the reactants). For a given amount of product, flow chemistry drastically reduces the size of the equipments required to produce it, which makes it a very promising technology for industrial chemistry.

The goal of the SmartCat project is to build tools to ease the use of flow chemistry, including in particular an online optimization of the yield of the reaction depending on the controlled parameters. Two optimization criteria have been defined: optimize the yield of the product, or its cost. In both cases, the process is the following: the reaction is started, and samples are collected at the end of the tube at given time steps. These samples are analyzed either via chromatography or nuclear magnetic resonance (which can be automatized and provides analysis in a few minutes). The analysis gives the value of the optimization criterion, and new reaction conditions are computed.

Hard constraints are added to ensure the safety of the process. These hard constraint define borders inside which the process must remain (for instance, the oven has a maximum temperature, or, due to safety issues, some areas must be left out of the search). In practice, we use a method based on Nelder-Mead, with a specific mechanism to deal with the borders of the search space: when one point of the Nelder-Mead simplex falls outside of the safe space, the simplex is projected on the border and the process continues in a lower dimension. Once a better point has been found, the previous simplex is retrieved and translated to the new position on the concerned axis, from where the search continues. A flowchart of this mechanism is given on Figure 2.1. Since the Nelder-Mead method cannot prove optimality, at every step, the user can decide to stop the process.

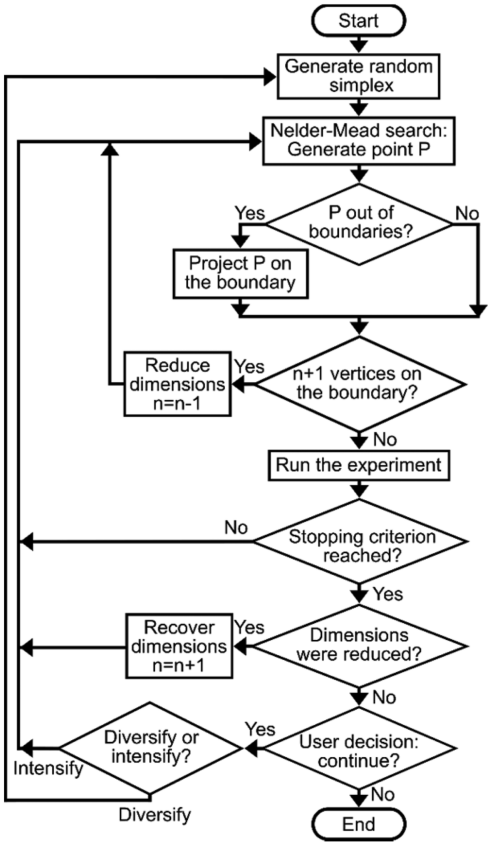


Figure 2.1: Flow chart of the Nelder-Mead method within some borders.

To test the system, experiments were conducted on the palladium-catalyzed Heck-Matsuda reaction. We illustrated the power of our modified algorithm through the optimization of a multivariable reaction involving the arylation of a deactivated olefin with an arenediazonium salt. The great flexibility of our optimization method allowed in practice to fine-tune experimental conditions according to three different objective functions: maximum yield, highest throughput, and lowest production cost. According to our chemists partners: "The beneficial properties of flow reactors associated with the power of intelligent algorithms for the fine-tuning of experimental parameters allowed the reaction to proceed in astonishingly simple conditions".

Our current work on this application is to pursue the experiments, in particular to illustrate the border mechanism, which was not used in practice in the Heck-Matsuda experiment, because it appeared that the optimization criterion was very smooth and "well shaped" (convergence was fast and easy). The chemistry lab now experiments on the synthesis of the carpanone, which is much more complex: it happens in five steps, each one having its own specific parameters and technical issues (for instance, magnetic nuclear resonance may give false positives in some cases). On the first tested reaction of the series, our algorithm, using the border mechanism, gave a very good yield (more that three times the yield obtained with a tuning performed by a human).

Compared to the urban planning application, this is a specific type of interaction (rule (iii)). In addition, the process needs to be done in real time, with the optimization algorithm running while the reaction is happening. Some optimization algorithms (as local search methods used in urban planning, or the Nelder-Mead method here) are easy to adapt to such conditions.

2.5 Conclusion

Real-life, unconventional applications are useful not only to enlarge the spectrum of CP use, but also to push the development of CP tools to new directions. From my past and current experiences, I think that CP offers powerful tools that can be used in applications where the users are not computer-scientists: constraint languages can be the basis of modeling languages in several domains (arts, sustainable development, medicine, etc), which are rather easy to deal with. The solving methods may be costly, but recent advances in both their design and implementation has made them reasonably fast for a practical use. Besides, quite often, unconventional applications do not require an immediate answer: in urban planning for instance, where there were huge instances, the initial resolution could take place overnight, only the interactive re-solving needed to be fast.

Yet, there are several obstacles when dealing with such unconventional applications (some of them may also appear with conventional ones). In the following, we will focus on two of them: the expertise needed to use CP solvers and the restriction of the constraint languages to a single type of variable.

CP solvers are very efficient, yet in practice, the users must have a good knowledge of the algorithms prior to use them in real life. In fact, one has little theoretical knowledge of their inner behaviour, which makes their tuning an expert task. Complete solvers are quite sensitive to different heuristics (variable selection, value selection, propagation loop tuning...) and a bad tuning can severely reduce the performances of the solvers: the most striking example is the heavy-tail phenomena in backtrack procedures that can be broken with randomization [Gomes et al., 2000b]. The same remark holds for local search and other metaheuristics methods, possibly even more accurately since they are randomized by nature: it has been shown decades ago that, when analyzing of their behaviour, the whole distribution of the runtime had to be taken into consideration [Hoos and Stützle, 1999]. Correctly using CP solvers thus requires an expert knowledge in the field. I believe that a better theoretical understanding of the solvers would greatly improve their useability.

Another restriction when using constraints is about the types of the variables. Solving methods and solvers are usually dedicated to only one type of variables (finite domain variables, or real variables), and solving problems on both types requires either *ad hoc* tricks or having different solvers cooperate. Yet, real-life problems often have both integer and real variables. In contemporary music for instance, spectral music is based on chords considered in their spectral dimension (as opposed to their pitch representation as in the classical Western music framework). Some of the problems I have worked on were indeed expressed on chords built as a regular scale in the frequency domain, resulting in chords in a log scale in the pitch domain. By nature, a frequency is a real dimension, yet, the chords frequencies had to be translated into a discrete domain before modeling the problem. Similarly, in the urban planning application, some of the quantities that were used were reals (like the positions and distances of some specific buildings) but were considered as discrete before modeling. More generally, some industrials problems are naturally stated on both integer and real variables, for instance in energy management (as in [Simon et al., 2012] on power restoration after a natural disaster, where boolean variables indicate whether an equipment is operational or not), network configuration (as in [Chi et al., 2008] where some of the network features, like the number of lines, are discrete, and others, like the bandwidth, are continuous). The question of how to elegantly solve mixed problems, involving both finite-domain and real variables is thus still open.

3

Highlight: Urban Planning

This chapter is in great part taken from the following publication: Bruno Belin, Marc Christie, Charlotte Truchet, *Interactive Design of Sustainable Cities with a Distributed Local Search Solver*, CPAIOR 2014: 104-119. It describes work that we did within the Sustain project, where we collaborated with urban planners to model and solve a placement problem for the design of sustainable new cities.

In retrospect, this chapter is interesting as it is a perfect example on how to use constraints into a tool intended for non-computer scientists: modeling has to be made incrementally with the users (in our case, starting from an attraction-repulsion model to which we then added hard constraints), solving needs to be done in such a way that the users keep a hand on the process. In my opinion, the most important contribution of the work is the interactive part of the solver. My contribution in this work, based on previous experience with musicians, was to interact with the urban planners during the modeling phase and to design the interactive solving process in collaboration with Marc and Bruno.

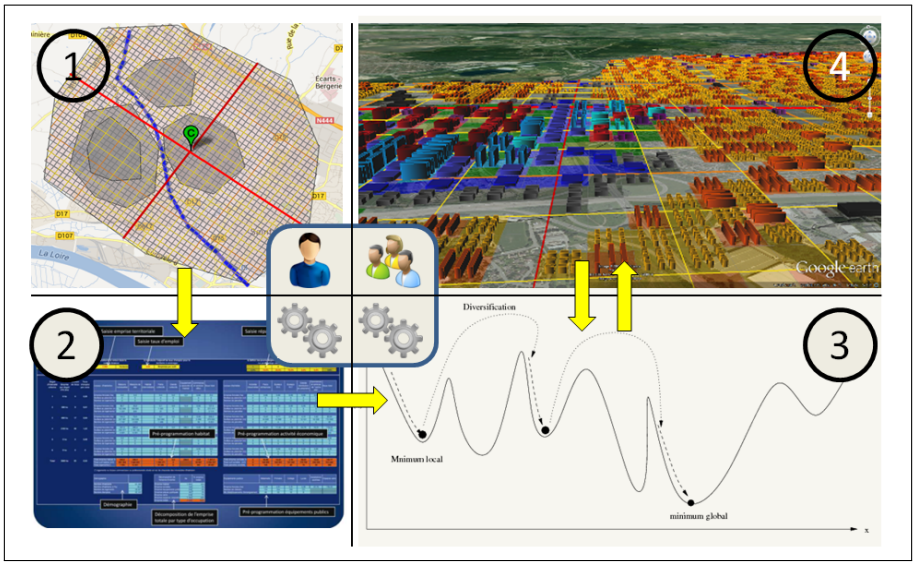
3.1 Introduction

In a world where more than 50% of the population lives in urban areas and where United Nations' projections mention a global urbanization rate of around 70% for 2050 [United-Nations, 2007], crucial questions arise on how to develop conditions for a balance between people, environment and cities. China for example, plans to annually create twenty whole new cities from now to 2020, around one million inhabitants each, to accommodate farmers in urban environments [Mars et al., 2008].

The process of designing whole new cities is by nature a collaborative endeavor gathering urban planners and decision makers around a coarse-grain map of a territory on which to place urban shapes such as centers, industries, housings, commercial units, public equipments, etc. The number of elements as well as their spatial layout needs to be strongly guided by a collection of rules related to social, economic, energy, mobility and sustainability issues.

The urban planning community actually lacks tools to assist this initial design process, and the literature is focused on addressing the problem of predicting the evolution of urban environments. Given a current state, and a set of evolution rules (land price, employment rate, extension,...), tools such as the UrbanSim framework compute the evolution of the city using agent-based representations [Waddell, 2002], or focus on more narrow issues [Dury et al., 1999]. In computer graphics, multiple contributions target the creation

Figure 3.1: A 4-stage design process of a urban environment (1) Setting contours, properties, central areas and intensity footprints (2) Computing the number of urban shapes by level of intensity (3) Automatically positioning urban shapes while enforcing constraints and favouring preferences (4) Interactively manipulating urban shapes while maintaining constraints and preferences. The last two stages are at the core of this contribution.



of new cities [Parish and Müller, 2001, Watson et al., 2008], however primarily focusing on computational models capable of encoding the stylistic appearance of the city (eg. mimicking existing ones), rather than its functional dimensions (some aspects such as land-use are however addressed, see [Lechner et al., 2006]).

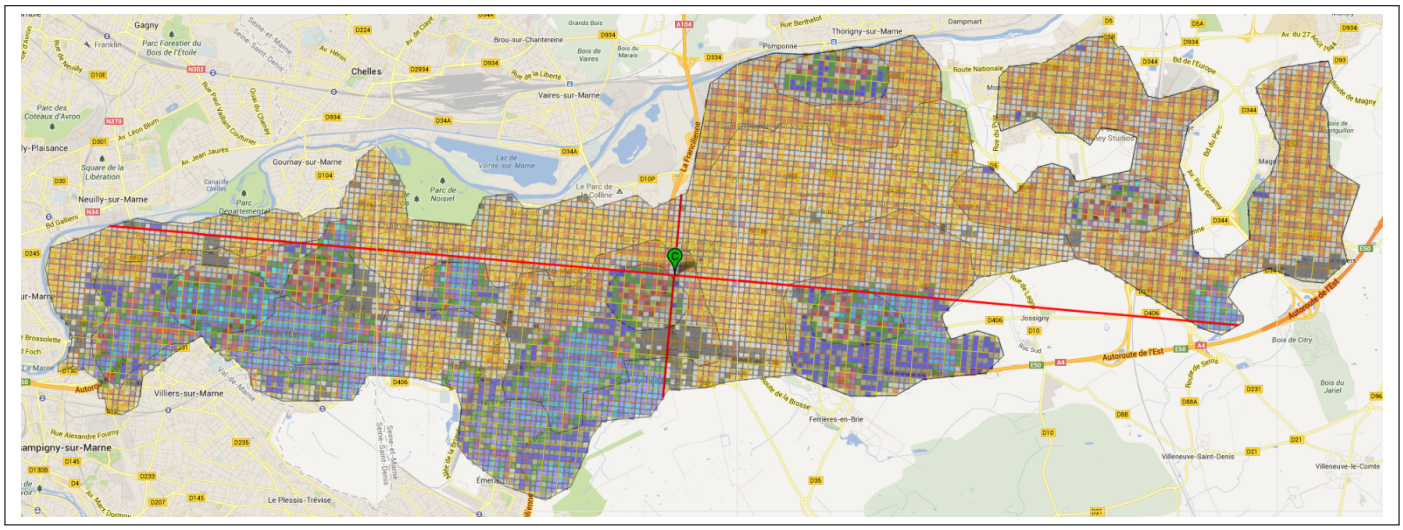
In contrast, the recent work of Vanegas *etal.* [Vanegas et al., 2010], integrate a functional description of the city by relying of UrbanSim's evolution models, yet do not offer any interactive editing tools, and are not designed in mind for urban planners or decision makers. Finally, some design tools are available to urban planners (such as CommunityViz [Kwartler and Bernard, 2001][1]), which offer automated and interactive design tools but address the problem at a very detailed level. The early stage design process of urban environments essentially remains a manual editing process.

In this context, we propose a computer-aided decision tool to assist designers in their task following a 4-stage process described in Figure 3.1: (1) the designer sets city contours, properties, central areas and intensity footprints over a regular grid (intensity footprints are areas with a given population density); (2) a knowledge-driven process computes the number of urban shapes of each kind (housing, industry,...) per level of intensity given an expected employment rate together with country-specific values; (3) urban shapes are then automatically positioned on the regular grid while enforcing constraints and favouring preferences between urban shapes in relation to social, economic and sustainability issues; (4) the designer then manipulates the urban environment while maintaining constraints and preferences. Our tool was part of the SUSTAINS project, a national-funded French research project gathering urban planners and computer scientists. The goal of the project was to deploy a software suite for designing new cities, taking into account the city footprints, and the automated computation of energetic impacts. The suite is made for urban planners and decision makers. It includes an interactive communication tool on large tactile surfaces for public engagement. The work described in this chapter focuses on stages 3 and 4.

More specifically, we addressed the problem of automatically positioning urban shapes on a territory, and proposed a local search method in order to automatically compute realistic coarse-grain maps, which can then be interactively modified while respecting some urban constraints. Compared to the state-of-the-art, our method reduces the amount of information that the user must provide, and optimizes the land use taking into account multiple constraints and preferences. One of its key feature is the solver

[1] http://placeways.com/communityviz/

Figure 3.2: Urban shapes automatically spread over the experimental area of Marne-la-Vallée city, taking into account the constraints of sustainable urban development expressed by urban planners.



interactive mode, where a user can modify the solution by hand while the system maintains the constraints in interactive time. To practically address the issue of scalability, we devised a distributed version of the solving process. A video detailing results and displaying interactive modes is available here `http://vimeo.com/80211470`.

The chapter is organized as follows. Section 3.2 introduces the key concepts expressed by urban planners, focusing on sustainable land-use. Section 3.3 describes our representation of constraints and preferences. Section 3.4 details our solving method, based on a distributed Local Search (LS), for generating a good initial solution and handling interaction. Section 3.5 presents experimental results, with a redesign of the Marne-la-Vallée city (east of Paris) which is spread over an area of 8728 hectares (87.28 km^2) with 234 644 inhabitants.

3.2 A model for early stage design of sustainable cities

The early stage design of a city consists in first selecting the number and the nature of all urban shapes composing the city and then spatially organizing them, by taking into account environmental, social, mobility and energy aspects. Several parameters have to be controlled: population density, landscape constraints, employment rate. The employment rate is calculated by dividing the number of jobs by the working-age population living in a specific area. An employment rate close to 1 corresponds to an ideal situation where each resident can access a job and housing in the area, minimizing commuting. An employment rate far from 1 has severe consequences, inducing incoming or outgoing congestions. This early stage enables decision-makers and stakeholders to agree on the broad guidelines of a preliminary project before initating a costly quantity survey.

3.2.1 Urban model

In urban planning, modeling consists in simplifying the reality of the world in order to better understand how decisions and events interact. It allows to test solutions that may affect political decisions and strategies which may lead to a desirable future [Antoni, 2010]. The urban model proposed by urban planners is based on the core concepts that are *blocks*, *central areas*, *intensities* and *urban shapes* incorporating, in a systemic

approach, major urban constraints linked to urban sustainable development.

Block The city block whose outlines are defined by the roads is the finest level of granularity selected by urban planners. Blocks are represented in grid patterns with a mesh size of 80 meters long. It is assumed that each city block hosts one single urban shape (housing, industry, shop, school, etc).

Central area A very important notion for urban planners is that of central area, a structuring place which gives its name to the neighbourhood. The feeling of belonging to a neighbourhood coincides with the influence area of the place, estimated at a 300 meters radius.

Urban shape In our context, a urban shape represents a dominant type of land use and buildings for a single block. However, the specification of a dominant urban shape can integrate a degree of mixed use (a portion of housing, commerce, etc.). Twenty eight urban shapes have been proposed and organized in four groups: residential (detached house, town house, intermediary housing, collective housing), economic activity (industrial, craft, commercial, tertiary), infrastructure (elementary school, primary school, secondary school, high school, sports equipment), and fixed elements (roads, unbuildable zones, rivers, rough terrains, natural areas, railroads, etc).

Intensity Intensity is a scale metric related to the density of the population in an area. It also translates the notion of activity or liveness of an area. High urban intensity represents a lively neighbourhood, dense, mixed and for which walking is the simplest way to travel for accessing all the essential urban functions. This intensity level is located in the city center, while low urban intensity is related to essentially housing areas. Six intensity levels are defined from 1 for the lowest intensity to 6 for the highest.

3.3 A constraint model of the urban location problem

For a given territory, the urban model provides the number of urban shapes of each type, for each level of intensity. The objective is then to arrange the placement of shapes on the territory. We will refer to this problem as the *urban location* problem. The first task is therefore to provide a formalization of constraints and preferences related to the properties of a sustainable city. Urban planners naturally express the interactions between different urban shapes as location preferences, instead of hard constraints (for instance, manufacturing industries prefer to be next to a river and not far from a highway). We thus express these preferences into cost-functions and express the problem as an optimization problem. Yet, there also are hard constraints which strictly restricts the positions of some urban shapes (one does not place an individual house within an industrial area, for instance).

In the end, the urban location problem encompasses a lot of constraints, some of them very specific. We distinguish in our presentation the most important, core constraints, which apply to all of the variables, and specific constraints that apply only on some areas or on some urban shapes. In the following, only a limited set of cost functions and constraints are detailed, as many of them are similar.

3.3.1 Grid representation of the city

We represent the city with a regular grid where each cell corresponds to a city block. The goal is therefore to assign each cell a urban shape among the possible shapes. Some specific cells are associated with fixed elements (roads, unbuildable zones, etc), and are therefore considered as not free and left out of the problem - yet, they are kept on the map since they might interfere with the other urban forms. Furthermore, each cell is given an intensity level manually specified by the urban planners (in the early design stage). This allows for instance to represent centralities on the map (commercial areas, lively squares, etc).

For the sake of simplicity, the grid is viewed as a rectangular array of cells of size $(x \times y)$, but this representation is more symbolic than geometric in that what is important is the neighborhood and relative placement of the urban shapes, not the rectangular geometry. Each cell is a variable which value is selected among all urban shapes: we note $V_{l,c}$ the variable corresponding to the cell in line l, column c. The urban shapes are encoded into integer values: (1) Detached house, (2) Town house, (3) Intermediary housing,..., (19) Breathing space.

3.3.2 Core constraints

These constraints cover the whole territory and express fundamental aspects of sustainable development.

Urban shape cardinality.

Depending on some features of the final city (number of inhabitants, size, employment rate, etc), the urban planners are able to determine how many instances of each urban shape must appear in the city: for instance, a big city must have a certain surface of parks, a certain surface of industries, etc. In the constraint problem, this gives a hard cardinality constraint for each urban shape.

Intensity requirement.

The urban model provides a given intensity level for each urban shape, and an intensity level for each cell of the grid. Based on these elements, every assignment of urban shape to a cell must comply with the intensity correlation between urban shapes and cells. The different levels of intensity are set by the user on the map. It is a hard unary constraint.

Interaction between urban shapes.

Each urban shape has placement preferences depending on its nature. For instance, shool units are attracted to residential units, and residential units are repelled by industrial units. We model these preferences as a function specifying the attraction (or conversely repulsion) of each urban shape for another urban shape. Between two urban shapes, possible interaction values are 0 (double attraction), 10 (single attraction), 20 (neutral), 50 (single repulsion) and 100 (double repulsion). The value of interaction decreases with the increasing distance between two cells.

The interaction preferences are expressed as a cost function. We note I the interaction matrix, which is an input of the system. $I_{p,q}$ is the interaction value between urban shape p and urban shape q. This matrix is asymmetrical, so it may happen that $I_{p,q} \neq I_{q,p}$. For a urban shape located at cell $V_{l,c}$, the interaction cost only depends on its neighbouring cells [2], namely with a set noted $\mathcal{V}^d_{l,c}$. The neighbouring cells are defined as:

$$\mathcal{V}^d_{l,c} = \{V_{i,j}, i \in [l-d, l+d], j \in [c-d, c+d]\} \setminus \{V_{l,c}\} \tag{3.1}$$

where: d is a parameter controlling the size. Note that this corresponds to a Moore neighborhood without its center, that is, a set of points at a bounded, non-null Chebyshev distance from the cell $V_{l,c}$. To take the distance influence into account, we consider the border of $\mathcal{V}^d_{l,c}$ noted $\bar{\mathcal{V}}^d_{l,c}$ such that:

$$\bar{\mathcal{V}}^d_{l,c} = \{V_{i,j}, (|i-l| = d) \vee (|j-c| = d)\} \tag{3.2}$$

Finally, for a cell $V_{l,c}$, the cost function related to our constraint is:

$$Cost_1(V_{l,c}) = \sum_{d=1}^{D} \left(\frac{\sum_{v \in \bar{\mathcal{V}}^d_{l,c}} \left(I_{(V_{l,c},v)} \right)}{|\bar{\mathcal{V}}^d_{l,c}| * d^2} \right) \tag{3.3}$$

[2] in a geometric neighbouring sense, not the neighbourhood of a LS algorithm

where: D is the maximum interaction distance (in the following, D is set to 3) to consider between two urban shapes. The cost of a cell $V_{l,c}$ includes contributions by all the rings at distance d from $V_{l,c}$, but these contributions are decreasing as d increases. Within a ring, the average contributions of the cells are divided by a correction factor of d^2, in order to obtain a similar effect as the attraction in physics.

3.3.3 High level constraints

In order to improve the sustainable aspects of our model, we add more specific constraints to improve the social equity, the preservation of environment and the economic viability.

Distance.

This constraint specifies that some urban shapes must be located with a minimum distance between them. This distance is expressed as a number of cells. For example, between an individual house and a high tertiary building (R+7), we want a minimum separation distance of 4 cells. For this constraint, we use Euclidean distances that measure the distance of a straight line between two cells. We note D the distance matrix, with $D_{p,q}$ the minimum distance permitted between urban shape p and urban shape q. This matrix is symmetrical. By convention, $D_{p,q} = 0$ when there is no particular distance constraint between p and q. For a urban shape $V_{l,c}$ located at (l, c), if $\mathcal{D}^{Euc}_{p,q}$ is the Euclidean distance in number of cells separating cells p et q, the cost function related to our constraint is:

$$Cost_2\left(V_{l,c}\right) = \sum_{v \in \mathcal{V}^4_{l,c}} \left(max\left(D_{V_{l,c},v} - \mathcal{D}^{Euc}_{V_{l,c},v}, 0 \right) \right) \tag{3.4}$$

Critical size area

Industrial or artisanal areas must have at least a critical size, otherwise, the area will not be created in practice because it will not be cost-effective. The critical size is determined by urban designers for each urban shape that needs grouping (craft activity, industrial activity). There is no maximum size for a grouping. On the other hand, although it is not explicitly expressed by designers, the area must have a sufficient compact structure (a notion used by the urban designers, and intuitively meaning that circles are better than lines or flat rectangles). For these urban shapes, we thus penalize the cells which belong to too small groups, or for which the groups are not compact enough.

Accessibility

This constraint concerns only one urban shape: the breathing spaces. It specifies that, from any inhabited point of the city, a breathing space should be reacheable by walking less than fifteen minutes, which corresponds to distance of about 1.25 km (or a distance of fifteen cells). We propose two complementary versions of this constraint. First, a global version which penalizes uncovered inhabited cells in proportion to their distance to any breathing space. And second, a local version which penalizes uniformly uncovered inhabited cells. The global version is more appropriate than the second one if the number of breathing spaces to spread over the city is insufficient to cover the entire grid, but its computation time then depends on the size of the grid. In contrast, the local version can be computed locally and is relevant if the urban model provides enough breathing spaces to cover the whole area to develop.

Interspace

This is a constraint inherent to buffer spaces or public equipments. For specific urban shapes positioned near each other, it is required that they must not be contiguous (not directly touching), and be separated by a buffer space or a built equipment. The following urban shapes are considered as buffer spaces: sport

equipment, breathing space and green way while urban shapes like nursery school, primary school, secondary school, high school, administrative and technical equipment are considered as built equipments. For example: between an individual house and a high collective housing (R+7), we have to position a buffer space.

Footprint

The principle is: when there is, in immediate vicinity of a secondary or high school, some particular urban shapes (individual houses or town houses, intermediary or collective housing, tertiary buildings), we must provide a place around the school, by placing, close to the building, a given number of green ways. For example, if there is around a secondary school only individual houses, town houses or intermediary housing, then we must allocate at least one green way near the school. However, if there is collective housing or tertiary buildings near the school, we have to build a bigger place around the school with at least two green ways.

Filtering

This constraint is related to central areas. Central areas are special cells marked by the user to identify the center of an urban neighbourhood (see subsection 3.2.1) and they can combine one, two or four cells in the same area. This constraint is used to filter the urban shapes that may occupy a central area and aid the diversity of urban shapes located on a same group. For this special cells, we favor a mix of the following urban shapes: schools, sports equipments, shops, services downtown and green ways.

3.4 A solver for the interactive design of sustainable cities

The problem addressed in this chapter can be viewed as a facility location problem [Moujahed et al., 2009], but in which all the urban elements need to be placed simultaneously. This *urban location* problem, is highly combinatorial, hence difficult to solve for a large number of cells. In addition, we need to deal with two specific requirements of the applicative context. First, the algorithm needs to scale up to the size of real-life cities, with a typical number of cells around 10000 (for a $64km^2$ city). Second, the users (urban designers and decision makers) also require to keep their hands on the system, by interactively modifying the assignment of urban elements to meet their own representations and expectations of locations.

To address both requirements, we designed a system based on two distinct solving stages. In the first step, the system computes and proposes one or several good solutions satisfying the urban constraints. And to address the issue of complexity, we developed a specific solving technique to efficiently handle the computation on multiple processors. The second step is interactive: the user modifies the map by moving single or multiple urban elements simultaneously, and the system adapts the solution by re-solving the constraints in the modified area, at a close-to-interactive frame rate, keeping the constraints satisfied as much as possible.

3.4.1 Initial resolution with Adaptive Search

Our first attempt to solve this problem relied on complete CP techniques. A prototype designed in the Choco solver [Jussien et al., 2008], with only some of the core constraints, failed to scale with a computation time of around half an hour for a small 16×16 map, whilst the typical size of our problems reaches 10000 variables.

Sequential algorithm.

We therefore relied on the *Adaptive Search* (AS) method which has proven its efficiency on large and various instances [Codognet and Diaz, 2001a]. This meta-heuristic takes advantage of the structure of the problem in terms of constraints and variables to guide the search. Our algorithm starts from a random configuration. At this stage, we make sure that the initial assignment respects the intensity level of each cell and the given number of cells for each urban shape (this is analogous to filtering the unary constraint on the intensities and filtering the urban shapes cardinalities).

The algorithm then performs a variant of iterative repair, based on variable and constraint error information, seeking to reduce the error on the worst variable so far. The basic idea is to compute the error function for each constraint, then combine for each variable the errors of all the constraints in which it appears, thereby projecting constraint errors onto the relevant variables. This combination of errors is problem-dependent [Codognet and Diaz, 2001a]. In our case, it is a weighted sum so that the constraints can be given different priorities.

Finally, the variable with the highest error is designated as the "culprit" and chosen for a move. In this second step, we consider the swaps involving the culprit variable and choose the best one according to f. However, there is a restriction to the set of considered swaps: we only swap two cells assigned to the same intensity level in order to satisfy the intensity constraint at any time. This swap policy ensures that the intensity constraint is kept satisfied at every iteration, as is the urban shape cardinality constraint. The algorithm also uses a short-term adaptive memory in the spirit of Tabu Search in order to prevent stagnation in local minima. Note that only the variables which do not improve f are marked tabu. When the algorithm is stuck, with all variables marked tabu, it restarts by resetting a given percentage of the variables, randomly chosen. The AS metaheuristic is described on Algorithm 3.3. In order to improve the speed of computations, we use a data cache for all information that require very frequent access: cost for each variable, groups, etc. We also compute all the costs incrementally, calculating only the delta induced by a swap. To efficiently distribute the AS algorithm, we also introduced two new features: a multi-candidate mode, and a multi-swap mode.

Multi-candidate mode. In the AS algorithm, only one culprit variable can be candidate for a move. We introduce a *multi-candidate* mode, where all the variables with a significant cost are considered. For all of them, we consider the possible swaps as defined above. There are two reasons for this: first, the worst variable may not be the one which will achieve the best swap. Second, in a distributed mode, several candidates can be explored in parallel.

Multi-swap mode. In addition to the multi-candidate management, we add a pool of best swaps. Instead of dealing with a single best swap for the current candidate, we store a small number of best swaps (configurable) related to each considered candidate. There is no significant time overhead because swaps must anyway be computed. This mechanism simply induces an extra memory footprint. Once the pool is filled with the best swap identified for each best candidate, we apply all the swaps of the pool in sequence, under some conditions. Typically, performing a swap may strongly change the impact of a further swap in the list. For a swap to remain valid, it must still produce a profit of a least a ratio (configurable) of its previous profit (before the first swap). In the distributed version of the algorithm, these two mechanisms (multi-candidate, multi-swap) are intended to take benefit from the work made by the different cores: each core performs an important amount of calculations, which are lost in the single-candidate, single-swap version of the algorithm. Although the moves performed in the multi modes may not be the best to apply sequentially, they still improve the cost. Applying them comes at no computational cost in the distributed version of the algorithm.

▷ Parameter: MaxRestart the number of partial resets of the algorithm
▷ f is the global cost function, f_i is the cost function for variable V_i
▷ s is the current configuration
▷ T is the adaptive tabu list
▷ j is the index of the variable with the worst cost

$s \leftarrow$ random configuration
$T \leftarrow \emptyset$
while MaxRestart not reached **do**
  while T is not full **do**
    For all i such that $V_i \notin T$, compute $f_i(s)$
    Select V_j a variable for which $f_j(s)$ is maximum
    Compute the cost f of the configurations obtained from s by swapping V_j with another variable
    Select s' the configuration for which $f(s')$ is minimum
    Update T by removing its oldest variable
    if s' can improve current solution s **then**
      $s \leftarrow s'$
    else
      $T \leftarrow T \cup V_j$
    end if
  end while
end while
return s

Figure 3.3: Base algorithm - initial resolution

Table 3.1: Decomposition of the city map into four uniforms parts and distribution to the different slave processes.

①	②
③	④

Slave	1	2	3	4	5	6	7	8	9	10
Subarea	①	②	③	④	①;②	①;③	①;④	②;③	②;④	③;④

3.4.2 Distributed mode

To tackle the complexity of large-scale problems, we propose to distribute the algorithm on a grid. A multi-walk parallel scheme has already been proposed for AS [Caniou et al., 2011, Diaz et al., 2012], with good speed-ups on classical problems. Instead, we choose a master-slave scheme to distribute the computations. At each iteration, we parallelize the search for the best swaps [Talbi, 2009]. We also take benefit of the geographical / geometrical layout of our problem, to assign different parts of the map to different cores in a coherent way. We detail here how the algorithm works on 10 cores, in which case the map is divided into 4 equal parts (the process is similar for a higher number of cores).

The slaves are in charge of examining the possible swaps (cost evaluation and comparison). Each slave is assigned a couple of subareas, as shown on figure 3.1: for instance, slave 1 investigates the swaps of cells located in subarea 1, with other cells of subarea 1. Slave 5 investigates swaps of cells in subarea 1 with cells in subarea 2. Note that the overall map is shared between each process, so that they can compute the global cost. However, each slave searches for the best possible swaps only on the subareas assigned to it. Given that the search of the best swaps does not change the state of the map, computing processes can operate in parallel without any difficulty and without changing procedures.

The slaves are synchronized with a master process that collects all the best swaps and decides which to apply. In the multi-candidate, multi-swap mode, the master process deals with the multi-candidate variables.

It also collects the pool of best swaps. Finally, the master process sends the swaps that were applied to each slave.Once synchronized, the slaves can search again new swaps in parallel.

3.4.3 Interactive mode

The second stage of our solving process is interactive. The purpose is to let the users have a control on the solution through interactive manipulations. In a way to smartly integrate user interactions, the idea here is to maintain the solving process active (i.e. continue swapping urban shapes) while the user is performing changes to the solution. Our interaction process is founded on the idea of having a *pool of cells* (POC). A POC represents a sub-region of the map in which urban shape assignments have already been made, and on which the solving technique presented in the previous section is applied to locally recompute a good solution. We then rely on this pool of cells to propose multiple interaction modalities, two of which are detailed in the following.

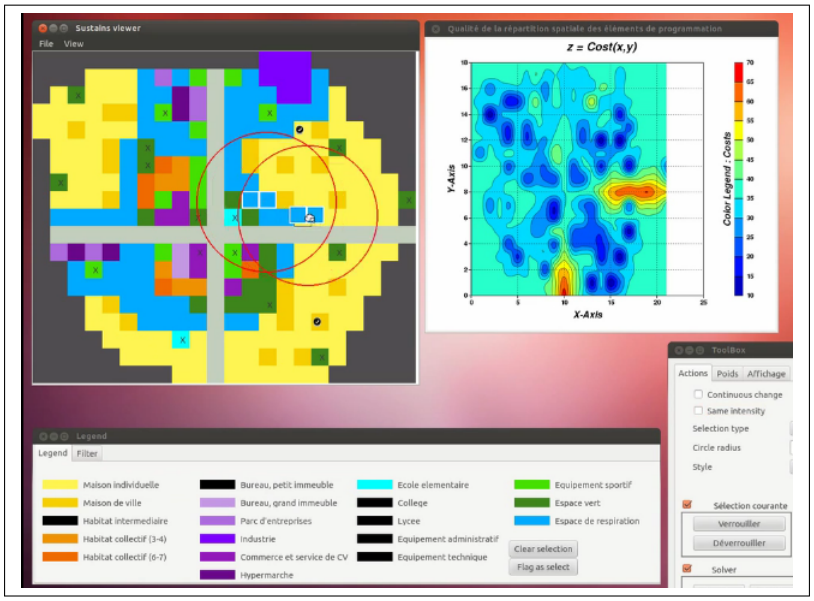
Defining fixed *vs.* free cells. At any moment, the user is able to select a set of cells which he considers temporarily fixed (in the sense that the associated urban shapes are constrained to those cells). The POC is built by computing the difference between all cells of the map and the fixed cells. The solver only performs its optimization on the cells in the POC. In a similar way, the user may define a set of free cells, that will then form the POC to which the solver is applied.

Manually moving a urban shape or a set of urban shapes. To avoid a local change performed by the user from impacting the whole map (i.e. changing the entire solution), the direct manipulation of a set of urban shapes has been defined in a way it only locally impacts the map. To this end, a disc size is first specified by the user (see Figure 3.4) that defines the region in which the computations will be performed around the user's changes. Then, the users selects one or multiple urban shapes and interactively moves them around the map. The POC is then defined by all the cells within the disc, except the cells manipulated by the users (which values are fixed). Two modes of interaction are proposed. Either the computation is continuously performed as the user moves the urban shapes. In such case, the POC is reconstructed and optimized at each move. Or the computation is performed as the user drops the selected urban shapes at a new location, in which case, the POC is composed of the union between the disc area around the initial location of the selected urban shapes and the disc area around the final location of the dropped urban shapes. At the initial location, some cells are no more assigned, while at the final location some urban shapes need to be reassigned to cells.

These manipulation modes are furthermore supported by a visual feedback that assists the users in understanding the impact of their decisions. First, a dynamically recomputed heat map presents the regions in which the costs are the most important (see top right part of Figure 3.4). And second, a dynamically recomputed graphical representation displays the global change in score, together with the change in score for each constraint that is considered (distance, accessibility, critical size...). Some options of the interactive solving process are detailed in the accompanying video (see here `http://vimeo.com/80211470`). In particular, in situations where the user intentionally moves part of an industrial zone (on which the critical size area is specified), the solving process automatically repositions all the industrial urban shapes to reform an industrial zone of specified critical size.

Interestingly, the process is extensible to multiple users performing changes simultaneously on the same map (e.g. multiple collaborators around a large tactile device). Each user is assigned his own pool, and solvers for each pool are distributed on multiple processors. The case of intersecting pools (i.e. two users manipulating close regions) is easily handled by creating a unique pool being the union of the two pools.

Figure 3.4: Interactive mode: on the top left, a radius around the user's selected cell defines the region in which swaps will be performed to best satisfy constraints. On the top right, a heat map displays the regions in which the costs are the highest (red areas) and provides feedback on the impact of the changes w.r.t. sustainability constraints.



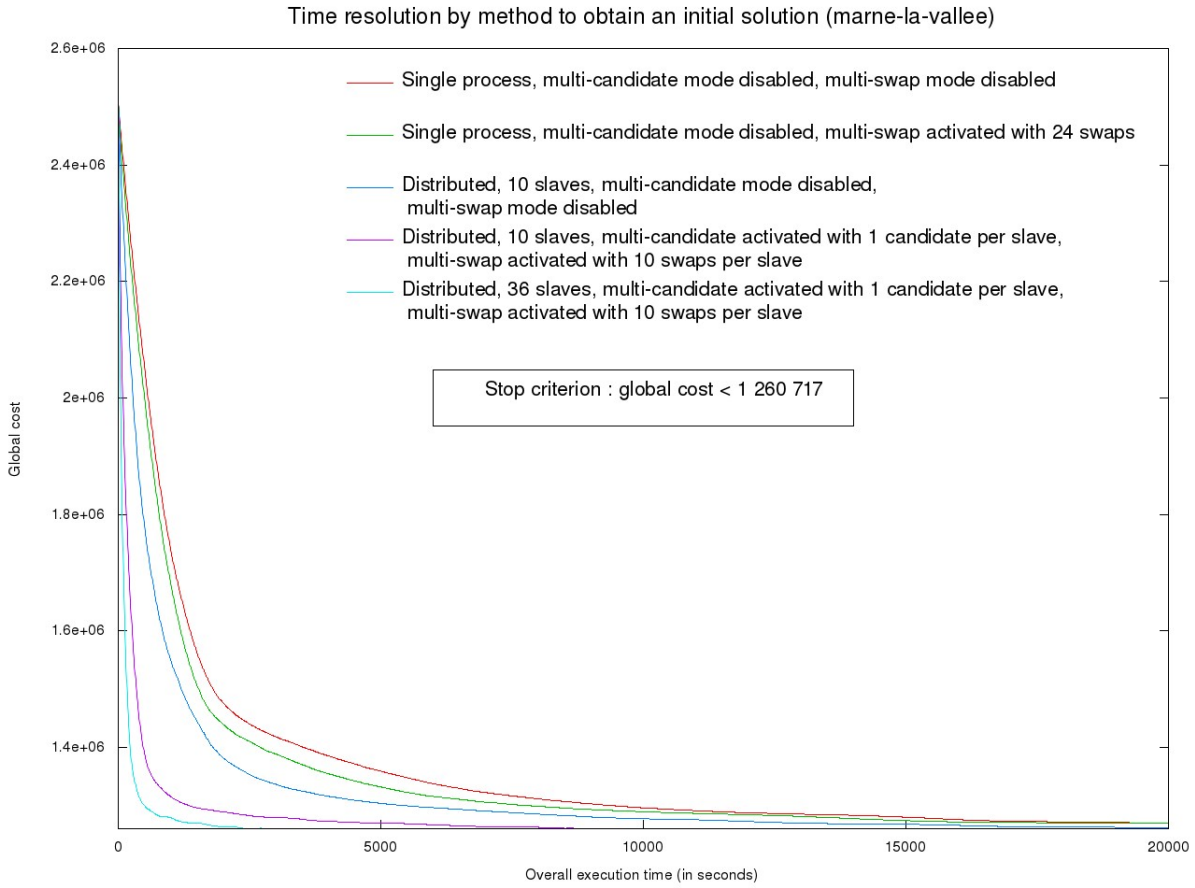
3.5 Experiments

Two different experiments were performed to evaluate our system. The first evaluates the computational costs related to the initial resolution. The second, less formal, evaluates the benefits of the interactive process through an open discussion with three senior urban designers.

The application was developed in *C++* and is based on the *EasyLocal++* framework [Gaspero and Schaerf, 2003]. About the accessibility constraint (section 3.3.3), we activated the local version that offers much better processing times on our real-life benchmark (e.g. reduced the time to find a best swap from 4 minutes to a few milliseconds on a 64x64 map). Synchronization between processes is performed by exchanging messages using the message-oriented middleware *Apache Active MQ* and the C++ library ActiveMQ-CPP.Experiments were conducted on Grid'5000, a french cluster available for research (in practice we used *granduc* in Luxembourg, with 22 nodes, each one having 2 CPU Intel 2.0GHz, 4cores/CPU, 15GB RAM). Each experiment was performed and averaged on 10 runs.

Computationnal costs. Results presented on Fig. 3.5 for the city of Marne-la-Vallée (see Fig. 3.2) address a problem with 9038 variables (i.e. free cells), in multi-candidate and multi-swap modes. Each run was terminated either when it reached a specified timeout, or when a *good* solution was computed We define a good solution as a solution with a score lower or equal to the score of the best solution found during the distributed search with 36 slaves (that is, 1260717). The results show that the sequential process is slightly improved by activating the multi-swap mode, but the best improvements are found in the distribution scheme, which takes full benefit of the multi-candidate and multi-swap mode. In the end, considering the target score of 1260717, the speed-up between the base algorithm (multi-candidate mode disabled, multi-swap mode disabled) and its distributed version (36 slaves, multi-candidate activated with 1 candidate per slave, multi-swap activated with 10 swaps per slave) is $\times 11.51$. This makes it possible to run our algorithm on a real-life city in less than one hour, which appears sufficient for the requirements of urban planners.

Figure 3.5: Tests for different multi-candidate and multi-swap options.



Assessment of the interactive mode. In the interactive mode, the user can select and change parts of the solution map, and during theses interactions, the solver is continuously running. A video accompanying this work (available at `http://vimeo.com/80211470`) shows both the initial resolution and the interactive mode on a small example. As can be seen on the video, the recomputation is performed on a restricted number of cells. The computationnal cost is obviously cheaper than the initial problem and can nearly be performed in real-time. Three senior urban designers who assessed the interactive mode were very interested by the possibility to rearrange and recompute the map, while maintaining the high-level constraints such as the critical size constraint. Another important feature for the urban designers is the maintaining of intensities and centralities while rearranging the map. In practice, the user manipulations are likely to lower the score of the interaction constraint, but it is convenient not to manually deal with those structural constraints.

3.6 Conclusion

In this chapter, we have first presented means to model the *urban location* problem, an important stage of the urban planning process when designing new cities that consists in laying out urban shapes while ensuring the statisfaction of local constraints. We then have presented a solver based on a local search algorithm, the Adaptive Search, which we extended to a parallel version. The proposed system is able to deal with large-scale problems in a two-stage solving process well adapted to the need of urban designers: first, an initial resolution provides the designer with one or several good solutions. The designer can then interact with the proposed solutions, while the solver maintains the constraints active. Further research includes refining the model by considering more evolved constraints required by the designers, and intregrating it

within a complete urban planning, spanning from early design to full 3D model.

Average Case Analysis of Some Constraint Algorithms

4

Synthesis Chapter: Constraint Solving Needs Average-Case Analysis

This Chapter is an introduction to a series of works on average-case analysis of some phenomena which appear in CP solvers. In the following, I first detail the general motivation for this series of works, then I present the approach I have chosen and finally I discuss about the lesson learnt on this axis of research.

4.1 Solvers Can Be Strange

Articles usually present an ideal world, where CP solvers are reliable and efficient. They do not tell the full story. Very often, CP solvers can be unpredictable, with strange behaviors: for instance, modifying the value of a single parameter in a local search solver can drastically improve or deteriorate its performances, as can fine tuning of the constraints propagation. Another example is the behavior of CDCL SAT solvers: changing the way the instance is described in cnf format (exchanging clauses for instance) can have an important impact on their performances. Most of the times, these behaviors are well known in CP but they also are seldom studied in depth. In fact, they are extremely difficult to study formally, since such a study would require average-case analyses of the algorithms. Besides, the natural way of doing average-case analyses is to define random binary CSPs, which is debatable: as for random SAT instances, random binary CSP instances are made up and they do not correspond to real situations.

We cannot be satisfied by the efficiency of our solvers if we cannot answer very basic questions about them: what is the impact of parameter tuning in local search? Of heuristics in complete solvers? Is there a generic optimal way of tuning the propagation loop? What are the good properties of a neighbourhood in local search? Such unanswered questions are countless in CP. Sometimes, high level criteria are introduced to provide an intuitive explanation of the solvers behavior. However, they are often defined in an intuitive way, without a formal, mathematical ground. For instance, local search algorithms, which are randomized by construction, are often described in terms of intensification and diversification: intensification, according to [Blum and Roli, 2003], "refers to the exploitation of the accumulated search experience", making the search concentrate on already visited areas (provided that they are promising) while diversification is the opposite process of having the search widely explore the search space. Then, in a tutorial at AAAI 2004, Hoos and Stützle define intensification as a way to "intensify the search in specific regions of the search space", which is not exactly the same definition: in one case, intensification drives the search to already visited areas while in the other case, intensification drives the search to new areas based on some quality criteria. Nevertheless, in practice, the difference is not essential, but the fact that we cannot even properly

define, in an unambiguous way, the only two high-level notions which are commonly used to describe inner mechanisms of local search algorithms, shows how far we are from a real understanding of their behavior.

I believe that it is absolutely necessary to improve our knowledge of what is happening in solvers, at all levels. In the following of this section, I will review some of the reasons why this is a necessary yet complex exercise.

4.1.1 Worst-case vs Average-case

As it is usually the case in Computer Science, the complexity of CP algorithms is given in the worst-case, which measures the maximum number of operations performed by the algorithm, among all possible inputs. In other words, a worst-case analysis measures the performance of the algorithm on the worst possible input. To the best of our knowledge, all of the complexities given for algorithms filtering global constraints are given in the worst case[1]. Apart from the necessary discussion about the relevance of Big-O approximations, it is worth noticing that, in the case of filtering algorithms, worst-case complexities are practically useless: these algorithms are meant to be used, altogether, in a propagation loop. However, it seems unlikely that the worst-case scenario for one constraint matches the worst-case scenario for another. Thus, in practice, these worst-case analyses provide very little useful information.

A more accurate way of measuring the performances of an algorithm is average-case complexity (see [Vitter and Flajolet, 1990] for an introduction). The principle is to examine the average performance of the analysis on a set of entries. It can be achieved by calculating the performance, as the number of operations of a given form (arithmetic operations, comparisons...), assuming that the input is randomly drawn within a given model. Average-case analysis is in general extremely difficult, even for simple algorithms. Besides, average-case analysis requires a model for the distribution of the possible inputs of the algorithms. Even if this model is uniform on all the potential inputs, this raises the question of the practical application of the results. However even if the analysis is imperfect, knowing the behavior of an algorithm in average would be more informative than their worst-case and potentially interesting when tuning the solvers. CP solvers need heuristics which are necessarily imperfect: a perfect variable-value choice heuristic in a complete solver would make all CSPs solved in linear time, hence, if $P \neq NP$, such a heuristic is NP-hard. Thus the heuristics need to be fed with useful, though necessarily imperfect, information on the state of the solving process. In this spirit, I believe that average-case analysis of some phenomena observed in CP would greatly improve our understanding of the solving process, and in the long term help us improve the solvers.

4.1.2 Life is Not Random

The most interesting results on the analysis of CP solvers have been obtained on random instances. This is particularly true on the SAT problem. We do not discuss here the analysis of the instances themselves (phase transitions, "hard" core, etc), but the analyses of the solvers performances on random instances. In [Barthel et al., 2003], [Semerjian and Monasson, 2003], two teams of physicists perform (independently and simultaneously) an analysis of a local search method, RandomWalkSAT, which looks like WalkSAT [Selman et al., 1994a] with the noise parameter set to 1, on random instances. They consider a run of RandomWalkSAT on a given instance as the dynamical evolution of a spin glass system, where the spin is oriented according to the boolean value of the variable. A local iteration of the algorithm corresponds to a step in the evolution of the spin glass system. By applying tools from statistical physics, they analyze the behavior of RandomWalkSAT and show in particular that, on difficult instances, the algorithm quickly reaches a metastable region. From there, only a fluctuation can make the algorithm reach a solution (the equilibrium of the system), and these fluctuations happen over exponentially large times. This, in practice, corresponds to the behavior of many local search algorithms which, after a fast relaxation, reach a state where the search gets stuck into regions where the cost function is small, but non null. In my opinion, this

[1] http://sofdem.github.io/gccat/

work is one of the most precise analysis of such algorithms, as it not only provides an average-case analysis but also explains in details the behavior of the solver.

In other cases, analysis of algorithms on random instances are made in order to design new, more efficient algorithms. This is for instance the case of [Schöning, 2002], which designs an algorithm for solving k-SAT based on a careful analysis of the probability of making "good" choices at each step (a good choice is a choice reducing the hamming distance to a solution). This algorithm is claimed to be the fastest 3-SAT solving algorithm at the time (in 2002). Another example is belief/survey propagation algorithms [Braunstein et al., 2005], which are message-passing algorithms computing margin probabilities that the variables take one or the other value. These algorithms, when they converge, can be very fast on random instances. Yet, they have been designed specifically for random instances, which is theoretically interesting but useless in practice. In fact, the SAT competition distinguishes three categories of instances: Random, Crafted and Industrial, and it is considered common knowledge that the solvers which perform well on random instances can perform poorly on the other two categories.

The random models have a similar importance in CP, where they are also widely used, in particular to study or design new solving methods. In this case, a CSP is artificially created through its constraint graph, *i.e.* the graph where the variables are nodes and the constraints are edges linking the nodes. A random model[2] allows the user to control the number of variables, values and constraints and also the tightness of the constraints (*i.e.* the number of value combinations authorized by the constraint, w.r.t. the size of the domains). This model has been intensively used in solver design. We will not give an extensive list of such works. Let us simply mention that it is common to test news ideas both on structured and random instances, as it is the case of one of the main solver heuristics, domWdeg [Boussemart et al., 2004], or one of the main propagation algorithm [Bessière et al., 2005].

An interesting approach to bypass the random instances models is based on statistics, or *a posteriori* analysis. This approach is started on parameter tuning for metaheuristics, since the Reactive Tabu Search method [Battiti and Tecchiolli, 1994]. In this refined version of the classical Tabu Search algorithm, the key parameter (called tabu tenure) is constantly refined depending on observations of the algorithm behavior. Recently, methods based either on statistical tools [di Tollo et al., 2015] or machine learning [Battiti and Brunato, 2005] were introduced in order to adapt the parameter tuning to the observed behavior of the algorithm. The design of metaheuristics can even be driven by these observations, adding or removing mechanism [KhudaBukhsh et al., 2009]. This statistical approach seems very promising for metaheuristics, yet it needs to be fed by observations, and besides, it also requires proper mathematical models of the algorithm behavior.

Finally, it is extremely difficult to bridge the gap between theory and practice when studying CP methods in average-case. Theory often provides efficient tools on a given random model, while practice requires to understand what is happening on not-at-all random instances, and the behavior of the algorithms can drastically change in between. The random instances model must be used with caution: it is useful understanding some phenomena happening in the solvers, yet transposition to the real world can be difficult.

4.2 Methodology Issues

Assuming that average-case analyses provide valuable insight on the solving process, the question becomes how to build them. In this section, I discuss two options of which the following Chapters are adaptations.

4.2.1 Physicist Approach

Obviously, trying to estimate average-case behaviors requires applied mathematics tools. In general, when used in the analysis of combinatorial optimization algorithms, the methodology is the following: once a potentially interesting phenomenon is identified, we try and narrow it so that it becomes accessible to

[2] http://www.lirmm.fr/ bessiere/generator.html

analysis. Hence the predominance of studies on random instances. The drawback of this approach is that the narrowing process often looses precious information. In the end, the considered phenomenon is well described in very particular cases (simple, basic algorithms for instance, without many of the key ingredients and fine tuning of the real-life solvers). This is useful, but not sufficient. I think that we could also take a physicist point of view: consider the real solver as a physical phenomenon that needs to be explained. Even though it might not be possible to fully explain precisely the observed behavior, it forces to build a model based on the experience and not the other way around. I think that it is not a coincidence if some of the most advanced analyses in our domain come from physics ([Barthel et al., 2003], [Semerjian and Monasson, 2003], [Braunstein et al., 2005], etc). Their strengths are twofold: they do have the theoretical tools (in statistical physics in particular) to tackle combinatorial analysis and their core methodology consists in observing a phenomenon, neglecting minor effects if need be, and providing controlled approximations of the analysis of the phenomenon. This methodology is for instance the one followed by [Hoos and Stützle, 1998], where they consider Las Vegas algorithms as their physical phenomenon.

This physicist approach is typically the base of Chapter 6, based on a joint work with Philippe Codognet, Alejandro Arbelaez and Florian Richoux, on the performances of a parallel scheme for combinatorial optimization, depending on the problems and instances. The considered phenomenon is the amazing performances obtained by a very simple parallel scheme in some cases, and we developed a framework to explain it. To check the validity of the model, we performed statistical tests, and then compared the results we obtained with reality.

This work was later extended in collaboration with Alejandro Arbelaez and Barry O'Sullivan in a mere predictive way, although this article ([Arbelaez et al., 2016]) is not presented in this document. Considering that, in cloud systems, computation time can be rented by the hour and for a given number of processors, we used our speed-up predictions to estimate the runtime of randomized combinatorial optimization tools. We combined the speed-up estimations with machine learning techniques to predict performance of sequential and parallel local search algorithms. In addition to classical features of the instances used by other machine learning tools, we considered data on the sequential runtime distributions of a local search method. This allowed us to predict with high accuracy the parallel computation time of a large class of instances, by learning the behavior of the sequential version of the algorithm on a small number of instances. Experiments with three solvers on SAT and TSP instances indicate that our method works well, with a correlation coefficient of up to 0.85 for SAT instances and up to 0.95 for TSP instances. This work shows how to use the in-depth understanding of the multiwalk parallel framework in a practical way.

4.2.2 Probabilistic Approach

Another option is more classical and consists in trying to build a probabilistic model of the phenomenon to explain. This approach is particularly well suited for local search methods, which are by nature randomized algorithms. On these algorithms, asymptotical convergence is proven in some cases (as for Tabu Search [Glover and Hanafi, 2002] or WalkSAT [Hoos, 1999]), yet, a full analysis of any real-life algorithm is still out of reach. One of the reasons is that most of local search algorithms are Markov chains (most of the times of orders higher than 1). However, the size of the chain is the size of the search space, hence, it is exponential in the number of variables. Even worse, the transition matrix of the chain has a size which is in square of this quantity. A full analysis would thus come at a cost which is at least the cost of the algorithm (and this is logical, considering that a full analysis does produce the solutions). However, at least one work has attempted to provide such an analysis: [Krishnamachari et al., 2000] studies the tuning of random walk on very small instances, but the methods they use make this work impossible to scale up.

It could seems naive to think that local search algorithms can be analyzed with probabilistic tools. Yet, such analysis are really needed, for instance to adapt the parameter tuning of the algorithms. Thus, we need to aim at providing approximations results, ideally with a control on the approximation. This is the spirit of the work described in Chapter 5, where I study the random restart mechanism for local search methods. In my opinion, the most interesting output of this result is not only the results on the random restart parameter,

but also a better understanding of the restart mechanism (which acts in practice as a way to re-connect un-connected chains).

The same approach is used in Chapter 7, based on a joint work with Jérémie du Boisberranger, Xavier Lorca and Danièle Gardy. In this work, we investigate the consistency of the famous `alldifferent` constraint (probably the most commonly used global constraint[3]). This work comes from a basic observation: this constraint, which states that variables must take different values, is global, but it can also be reformulated as a series of $\neq$ constraints. The difference between the global constraint and its reformulation is thus not semantic, but operational: the global constraint has a more powerful filtering algorithm. The question is: is it possible to measure when this added filtering is useful or not? The answer is somehow surprising: quite often (depending on the tightness of the constraint), it is not. In this work, we have proposed a probabilistic model for the global constraint consistency, and, based on this model, we calculate the asymptotical behavior of the constraints. As before, I think that the value of this work is in fact not really the results themselves (which are rather technical) but their hypothesis: we proved that the constraint could have different asymptotical regimes depending on the ratio of variables/values.

4.3 Other Contributions

Apart from the works described in the next Chapters, I have been working in collaboration with Xavier Lorca and Amin Balafrej (both from my team) on a probabilistic model of random binary CSPs. We introduce a probabilistic-based model for binary CSP that provides a fine grained analysis of its internal structure; *i.e.* the microstructure of the constraint graph taking into account the supported values for each constraint. Assuming that a domain modification could occur in the CSP, we show how to express, in a predictive way, the probability that a domain value becomes inconsistent. Based on this, we compute the expectation of the number of arc-inconsistent values for each domain of the constraint network, and then for the whole constraint network. Next, it provides bounds for each of these three probabilistic indicators. The model is described in the research report [Balafrej et al., 2016]. Our model can be seen as an extension of [Mehta and van Dongen, 2007], where the authors present a new arc-consistency propagation algorithm which, after a domain modification, bases the revision of a value on the probability that it will still be supported.

We are currently working on how to use this information in a constraint solver. We have two test-cases under consideration: first, to design a Probabilistic-Based Search (PBS) heuristic, which selects the variables as well as the domain values. An experimental evaluation shows that PBS improves the state of art search heuristics on a classical benchmark. Second, to derive an original probabilistic-based criterion that allows us to predict which are the domains most likely to contain singleton arc-inconsistent values. This criterion enhances an adaptive singleton based consistency.

4.4 Conclusion

This series of works allowed us to learn several lessons at two levels.

At low level, I think that we now have a better understanding of several phenomena which are commonly observed in CP solvers. We now have a clear understanding of the role of random restart inside local search, where it can be seen as a tool to reconnect the corresponding Markov chain, in case the local search has been designed in a too greedy way. We know precisely why the global `alldifferent` constraint can sometimes be unplugged: when the constraint is not tight enough, it is unlikely to propagate any useful information. We have a clean model of multiwalk parallel local search and of the microstructure constraint graphs. Each time, we were able to formally prove what was common knowledge in the community.

[3] http://web.emn.fr/x-info/sdemasse/gccat/C`alldifferent`.html

At a higher level, this series of works also provides insights on how to analyse CP solvers in general. The main obstacle when working on these problems is the modeling issue, as discussed in section 4.2. In each case, we had to make hypotheses on the inputs of the algorithms and in each case, we chose a uniform random model. To what extent does this model correspond to reality? It probably does not. However, in my opinion, the main reason why these results are still valuable is not exactly the results themselves, but the hypothesis under which the results hold.

In Chapter 5, we did approximate the runtime with and without restart but the most interesting output of this work is that we understood that restart acted as a way to reconnect the local search Markov chain. In Chapter 6, we did manage to model the parallel speed-up, but the most interesting output of this work was to understand precisely why some instances had linear speed-ups and others did not, and also identify the role of the different parameters of the sequential distributions, which gives tools to then approximate them in a predictive way. In Chapter 7, we give the asymptotical behavior of the global `alldifferent` propagation, but the most interesting part is the hypothesis of the theorem, which identify to regimes depending on the tightness of the constraints. I believe that such theoretical works, which are usually hard to publish and sometimes seen as useless in practice, enforce us to gain a deep and mathematically sound understanding on the studied objects (here, CP solving processes). By doing so, we often have to refine the model or add hypotheses which convey a very practical meaning.

5

A Study of the Restart Mechanism for Local Search Algorithms

This chapter is an unpublished article, of which I am the sole author. It answers a question that arose, during my PhD, when I implemented a local search algorithm as a library within a computer-music language (OpenMusic). I had to set the parameters of the local search, in particular the Random Restart parameters, by guesswork. It was unreasonable to leave such a choice to musician users, and I still think it is unreasonable to be satisfied with local search performance while not being capable to explain these performances.

This chapter presents an analysis of random restart in local search and in particular the influence of its main parameter, the number of iterations before restart, on the performance of the algorithm. This result explains both in theory and in practice the mechanism of random restart.

5.1 Introduction

Local Search (LS) algorithms have become a state-of-the-art tool for solving combinatorial problems. They are very efficient in practice, although they are incomplete and cannot prove unsatisfiability or optimality. Basically, a LS algorithm explores a huge search space, guided by an error function to minimize or nullify. It starts from a random configuration of the search space, and then iteratively explores a neighborhood of the current configuration, on which it chooses one of the best neighbors with respect to the error function. Different metaheuristics ensure that the algorithms do not cycle, yet they are limited to a certain number of iterations, hence the incomplete property.

A LS algorithm is not, in theory, guaranteed to terminate. In practice, it does if the problem is satisfiable: the probability of cycling is made very low by the metaheuristics. Asymptotical convergence is proven in some cases, for instance for Tabu Search in [Glover and Hanafi, 2002] or for WalkSAT in [Hoos, 1999]. Some works even provide an average case study for the resolution time of some theoretical LS algorithms. For a variant of the WalkSAT algorithm, called RandomWalkSAT, both [Semerjian and Monasson, 2003] and [Barthel et al., 2003] use tools from statistical physics to analyze an average behavior on the algorithm on random SAT instances. With only probabilistic tools, works such as [Schöning, 2007] or [Balint and Schöning, 2012] provide average-case studies of specific algorithms, mostly on SAT. Yet, compared to their importance in combinatorial optimization, the knowledge on LS algorithms is still far from complete. In

this chapter, we investigate the restart mechanism for a local search algorithm. This question has also been studied on complete methods, for instance in CDCL [Biere, 2008] or on constraint solvers [Gomes et al., 2000a]. On local search, [Schöning, 2002] shows (among other results) that adding restart to a greedy algorithm does improve its average-case behavior. Our result is complementary theirs, since we show that restart is useless when the neighborhood makes the Markov chain corresponding to the local search algorithm connected.

5.1.1 Randomization

LS algorithms use randomization as a way to explore the search space widely, and avoid being limited to some of its regions. For instance, the WalkSAT algorithm [Selman et al., 1994b] adds to the GSAT algorithm [Selman et al., 1992] a step of random walk on the neighborhood, and this partial randomization improves the solving time. It has even been observed in [Hoos and Stutzle, 2000] that the best proportion of random walks iterations is quite high, around $1/2$: one out of two iterations is a random walk. A controlled, well tuned randomization improves the algorithm.

A very common randomization mechanism is the random restart. For a parameter m, given, the algorithm is stopped every m iterations and restarted from scratch. This is used as a diversification process, and forces the algorithm to visit different regions of the search space, with the intuitive explanation that there are so few solutions to the problem that a wide exploration is necessary.

5.1.2 Markov Chains

LS algorithms explore the search space partly randomly, with a limited memory of the visited configurations. In probability theory, this corresponds to Markov Chains, random processes that evolve in time with a limited memory. The natural link between LS algorithms and Markov Chains has already been exhibited in the literature ([Schöning, 2007], [Semerjian and Monasson, 2003] or [Barthel et al., 2003] for instance). [Krishnamachari et al., 2000] provides an exhaustive study of two SAT algorithms (Random Walk and Random Noise), and their probabilistic model is very close to ours. Unfortunately, due to the huge quantity of calculations required, they have to limit the applications of their study to SAT instances with 7 variables.

Although Markov Chains is a very common and useful model in probability theory, they are still complicated to analyze in general. A classical restriction is to limit the memory to 1 step, and consider only homogeneous chains (performing the same operations for each iteration, as it is often the case for LS algorithms). In this case, the behavior of the chain is known. For irreductible chains, which do not have dead-end states and are connected, the asymptotical stationary distribution of the process can be computed in many cases (although it might be costly), as well as the time to reach the stationary distribution. We will consider in the following absorbing chains, which have dead-ends and get asymptotically trapped into these particular states. Again, the time for the chain to be trapped can be computed thanks to a fixpoint equation, but this is also costly and intractable in our case.

5.1.3 Outline

This chapter is organized as follows. Section 5.2 present Markov Chains and the probabilistic notions that are needed afterwards. Section 5.3 details how to integrate the random restart mechanism in the model, and gives the theoretical result on the expected solving time. Section 5.4 illustrates the result with approximated simulations, and experiments on the WalkSAT algorithm. In Section 5.5, we leave the mathematical framework and discuss the LS case in practice.

5.2 Probabilistic Model for LS Algorithms

We present here the general probabilistic framework that we will use below. For a full introduction to probability and Markov Chains, see [Grimmett and Stirzaker, 1992]. Chapter 11 of [Charles M. Grinstead, 1997] (available online) completes the former with a focus on absorbing Markov Chains.

5.2.1 Markov Chains

Consider a finite set of states $S = \{s_1...s_p\}$. Let $\mathcal{X} = (X_t)_{t\in\mathbb{N}}$ a discrete random process on S, where t represents a discrete time. The random variable X_t is equal to s_j iff at time $t \in \mathbb{N}$, the random process is in state s_j. The process $\mathcal{X}$ is a Markov Chain of order 1 if it has no memory of the states that it has visited: at time t, the move chosen by X_{t+1} only depends on the state where X_t is in, and not on the path it has previously followed $(X_0...X_{t-1})$. In addition, $\mathcal{X}$ is said homogeneous if the transition probabilities from X_t to X_{t+1} do not depend on t. In the following, we will limit all the Markov Chains (MC) to be homogeneous and of order 1, and omit the details for sake of simplicity.

To define this, we use the notation for conditional probability: for two states s and s', $\mathbb{P}(X_t = s|X_{t-1} = s')$ is the probability that $X_t = s$, under the assumption that $X_{t-1} = s'$.

Definition Let $S = (s_1...s_p)$ a finite set and $\mathcal{X}$ a random process on S. $\mathcal{X}$ is a homogeneous order 1 Markov Chain iff $\forall t \in \mathbb{N}$, $\forall j_t, j_{t-1}...j_1, j_0 \in [1..p]$

$$\mathbb{P}(X_t = s_{j_t}|X_{t-1} = s_{j_{t-1}} \text{ and } ...X_1 = s_1 \text{ and } X_0 = s_0)$$
$$= \mathbb{P}(X_1 = s_{j_t}|X_0 = s_{j_{t-1}})$$

The transition probabilities from a state s_j to a state $s_{j'}$ do not depend on time, thus they can be stored in a transition matrix which fully represents the process.

Definition Let S a finite set and $\mathcal{X}$ a MC on S. The transition matrix of $\mathcal{X}$, written $M^{\mathcal{X}}$, is the $p \times p$ matrix with $M^{\mathcal{X}}_{j,k} = \mathbb{P}(X_1 = s_k|X_0 = s_j)$.

Notice that $M^{\mathcal{X}}$ is a probability matrix: the sum of the elements on a line which represents the overall probability of leaving some state, is equal to 1. In addition, the transition matrix for k successive iterations of the Markov process is $(M^{\mathcal{X}})^k$.

5.2.2 Absorbing Chains

MC feature different asymptotical behaviors depending on some of their inner characteristics: wether the graph of possible moves is connected, wether they admit dead-end states, etc. We focus here on the particular case of absorbing Markov Chains, because they adequately model LS algorithms.

Definition Let $\mathcal{X}$ a MC on S a finite set. A state $s \in S$ is absorbing iff $\forall s' \in S, s' \neq s \implies \mathbb{P}(X_1 = s'|X_0 = s) = 0$.

A state s is absorbing if the probability of getting out of s is null, in other words, s a dead-end for $\mathcal{X}$. It implies that, if $\mathcal{X}$ eventually gets in s, it stays there forever.

Definition Let $S = \{s_1...s_p\}$ a finite set and $\mathcal{X}$ a MC on S. Let $i_0, i_k \in [1...m]$. The state s_{i_k} is reachable from s_{i_0} iff there exists a path $s_{i_0}, s_{i_1}...s_{i_{k-1}}, s_{i_k}$ such that $\forall j \in [0...k-1], \mathbb{P}(X_1 = s_{i_{j+1}}|X_0 = s_{i_k}) > 0$.

Simply said, a state is reachable from another if there is a path with non-null probability between them: starting from the first state, the chain will eventually reach the second one.

Definition Let S a finite set and $\mathcal{X}$ a MC on S. $\mathcal{X}$ is an absorbing chain iff $\mathcal{X}$ has at least one absorbing state s, and for every state $s' \in S$, there exists an absorbing state $s \in S$ reachable from s'.

Absorbing chains have dead-ends that may be reached from everywhere. In this case, the asymptotical behavior of the chain can be intuitively described. The only stable states are the absorbing states. From any other state, the process eventually reaches an absorbing state, and it cannot be trapped elsewhere. The process will thus asymptotically end in the absorbing states.

Without loss of generality, the absorbing states can be chosen as the last ones in S, which simplifies the notations. Given an absorbing MC with a absorbing states, we always assume in the following that the absorbing states are $s_p...s_{p-a}$. We write $S_{ab} = s_p...s_{p-a}$ the set of absorbing states. The transition matrix is then

$$M^{\mathcal{X}} = \left(\begin{array}{c|c} Q^{\mathcal{X}} & R^{\mathcal{X}} \\ \hline 0 & I \end{array} \right)$$

where I is the $a \times a$ identity matrix and 0 the null matrix.

The behavior of $\mathcal{X}$ can also be read on the transition matrix. Because the absorbing states must be connected to the transient states, $R^{\mathcal{X}}$ is non null. From a non-absorbing state $s_j \in S \backslash S_{ab}$, the process may go out either in $Q^{\mathcal{X}}$, or in $R^{\mathcal{X}}$ in which case it drops into the I matrix beneath and stays there, in S_{ab}. In the case where the process stays in $Q^{\mathcal{X}}$, then the same thing happens again from another state of $Q^{\mathcal{X}}$.

5.2.3 Absorption Time

For an absorbing chain $\mathcal{X}$, the absorption time is defined as the expectation of the time needed for the process to be absorbed in S_{ab}. It is generally computed considering an initial uniform distribution on S, which also corresponds to the common random initialization of LS algorithms. In all the following, we thus assume that X_0 is uniformly randomly chosen on S.

We will use the following notation: for a state $s \in S$, and a particular time $t \in \mathbb{N}$, $\mathbb{P}(\mathcal{X}, s, t)$ is the probability that $\mathcal{X}$ visits s for the first time at the t-th iteration, formally: $\mathbb{P}(\mathcal{X}, s, t) = \mathbb{P}(X_t = s \text{ and } \forall t' < t, X_{t'} \neq s)$. For a subset of states $S' \subset S$, $\mathbb{P}(\mathcal{X}, S', t)$ is the probability that $\mathcal{X}$ visits one of the states in S' for the first time at iteration t.

Definition Let S a finite set and $\mathcal{X}$ a MC on S. The absorption time for $\mathcal{X}$, written $\mathbb{E}(\mathcal{X}, S_{ab})$, is $\sum_{t \in \mathbb{N}} t * \mathbb{P}(\mathcal{X}, S_{ab}, t)$.

Again, the absorption time can be intuitively and approximately quantified with the transition matrix. Because $M^{\mathcal{X}}$ is a probability matrix and $R^{\mathcal{X}}$ is non null, one has $||Q^{\mathcal{X}}||_1 < 1$ (where $||Q^{\mathcal{X}}||_1$ is the sum of all the elements of $Q^{\mathcal{X}}$). We write $\gamma = ||Q^{\mathcal{X}}||_1$. Approximately, at every iteration the process has a probability of staying into $Q^{\mathcal{X}}$ equal to γ, and a probability of going into $R^{\mathcal{X}}$ equal to $1 - \gamma$. Thus, the probability that the process visits S_{ab} for the first time at time t is the probability that it spends $t - 1$ iterations in $Q^{\mathcal{X}}$ and goes to $R^{\mathcal{X}}$ once, that is, $\gamma^{t-1} * (1 - \gamma)$. Thus the absorption time is approximately $\sum_{t=1}^{\infty} t * \gamma^{t-1} * (1 - \gamma) = 1/(1 - \gamma)$.

All this calculation is approximate, because for each state in $S \backslash S_{ab}$, we replaced the exact probability of going into $R^{\mathcal{X}}$ by an average, and considered that the successive iterations were not correlated while they obviously are. Yet, it provides a good intuition of the absorption phenomenon. In particular, in the case of a LS algorithm on a NP-hard problem with parameter n, the set of states S evolves exponentially fast with n, say, in $O(2^n)$. Assuming that there are only a small, $O(1)$ number of absorbing states (the solutions), so $\gamma \approx 1 - 1/2^n$. Then the absorption time roughly evolves as $1/(1 - \gamma) = 2^n$ as expected.

5.3 Probabilistic Model for Random Restart

We add a random restart mechanism to a MC, keeping the same notations.

5.3.1 Restarted Random Process

Our goal is now to add to a Markov process a random restart mechanism, which, every m iterations for some $m \in \mathbb{N}$, cuts the process, realizes a uniform random draw on S and then lets the process continue again. This breaks the Markov property, adding a long term memory to the process (the number of iterations before restart). Notice that the new process still corresponds to a Markov process of order m, for which a full analysis is too costly for a practical use.

Instead, we model the restarted process in a very simple way, by defining, from a Markov process $\mathcal{X}$, a new random process $\mathcal{Y}$ integrating the restart steps. The number of iterations before restart m is fixed, and we write U a random draw under the uniform random distribution on S.

Definition Let S a finite set and $\mathcal{X}$ a MC on S with the same notations as above. The corresponding restarted process, $\mathcal{Y}$, is defined by:

$$\forall t \in \mathbb{N}, Y_t = \begin{cases} U & \text{if } t \bmod m = 0, \text{ and } X_{t-1} \notin S_{ab}, \\ M^{\mathcal{X}} Y_{t-1} & \text{otherwise,} \end{cases}$$

where $M^{\mathcal{X}} Y_t$ stands for: an iteration of $\mathcal{X}$ on the random variable Y_t.

Cutting the chain every m iterations modifies its characteristics, but not the absorbing property.

Proposition 5.3.1 *Let $\mathcal{X}$ an absorbing Markov process on S, and S_{ab} its absorbing states. Let $\mathcal{Y}$ the corresponding restarted process with the same notations as above. If $\mathcal{X}$ is absorbing, then so is $\mathcal{Y}$.*

Proof By definition of $\mathcal{Y}$, an absorbing state for $\mathcal{X}$ is also absorbing for $\mathcal{Y}$. So we only need to check the reachability property. Let $s \in S$, we need to find a path from s to an absorbing state. Since $\mathcal{X}$ is absorbing, there is an absorbing state s_k and path $s, s_1, ...s_k$ with non null probability from s to s_k in $\mathcal{X}$. Let $t \in \mathbb{N}$ a time and assume that $Y_t = s$. In the case where there has been no restart between t and $t + k + 1$, then the same path links s to s_k in $\mathcal{Y}$ and s is connected to an absorbing state. In the other case, let t' be the time of the restart. The restart is made uniformly on S so there is a non-null probability of going from $s_{t'-1}$ to s_k, and $s, s_1, ...s_{t'-1}, s_k$ is a path with non null probability from s to s_k in $\mathcal{Y}$. So every state is connected to an absorbing state in $\mathcal{Y}$ and $\mathcal{Y}$ is absorbing.

5.3.2 Absorption Time

With this model of random restart, it is possible to compute the absorbing time for $\mathcal{Y}$, depending on the absorbing time for $\mathcal{X}$ and m. We will call run an execution of the $m-1$ successive iterations of the process between two restarts.

Theorem 5.3.2 *Let $\mathcal{X}$ an absorbing Markov process on S, and S_{ab} its absorbing states. Let $\mathcal{Y}$ the corresponding restarted process. Let $a(m) = \sum_{t=m}^{\infty} \mathbb{P}(\mathcal{X}, S_{ab}, t)$ the probability that $\mathcal{X}$ has not been in S_{ab} before time m. Then:*

$$\mathbb{E}(\mathcal{Y}, S_{ab}) = \frac{\sum_{t=0}^{m-1} t * \mathbb{P}(\mathcal{X}, S_{ab}, t) + m * a(m)}{1 - a(m)}$$

Proof Let $t \in \mathbb{N}$. We first determine exactly when $\mathcal{Y}$ visits an absorbing state for the first time at iteration t. For this to happen, $\mathcal{Y}$ must not have visited an absorbing state before and it must arrive in S_{ab} exactly at time t, which means that:

(i) none of the t div m first runs visited S_{ab},

(ii) the first $((t-1) \bmod m)$ iterations of the current run (since the last restart) did not visit S_{ab},

(iii) the $(t \bmod m)$-th iteration of the current run visits S_{ab}.

The probability that $\mathcal{Y}$ does not visit S_{ab} in one run is equal to $a(m)$ by definition of $a(m)$, and the different runs are independent, so $\mathbb{P}(i) = a(m)^{t \text{ div } m}$. The probability of (i) and (ii) is $\mathbb{P}(\mathcal{X}, S_{ab}, t \bmod m)$ since the $\mathcal{Y}$ process is, in the current run, the same as the $\mathcal{X}$ process with a time shift since the last restart. These events are independent, so in the end: $\mathbb{P}(\mathcal{Y}, S_{ab}, t) = a(m)^{t \text{ div } m} * \mathbb{P}(\mathcal{X}, S_{ab}, t \bmod m)$. Then:

$$\begin{aligned}\mathbb{E}(\mathcal{Y}, S_{ab}) &= \sum_{t=0}^{\infty} t\mathbb{P}(\mathcal{Y}, S_{ab}, t) \\ &= \sum_{t=0}^{\infty} t a(m)^{t \text{ div } m} \mathbb{P}(\mathcal{X}, S_{ab}, t \bmod m) \\ &= \sum_{k=0}^{\infty} \sum_{i=0}^{m-1} (km+i) a(m)^k \mathbb{P}(\mathcal{X}, S_{ab}, i)\end{aligned}$$

Since both sums converge we can inverse them, and we obtain $\mathbb{E}(\mathcal{Y}, S_{ab}) = \sum_{i=0}^{m-1} \mathbb{P}(\mathcal{X}, S_{ab}, i) \sum_{k=0}^{\infty} (km+i)a(m)^k$. The new inner sum is made of two known geometric series

$$\begin{aligned}\sum_{k=0}^{\infty} (km+i)a(m)^k &= m \sum_{k=0}^{\infty} k a(m)^k + i \sum_{k=0}^{\infty} a(m)^k \\ &= \frac{a(m) * m}{(1-a(m))^2} + \frac{i}{1-a(m)}\end{aligned}$$

Thus

$$\begin{aligned}\mathbb{E}(\mathcal{Y}, S_{ab}) &= \sum_{i=0}^{m-1} \mathbb{P}(\mathcal{X}, S_{ab}, i) \left(\frac{i}{1-a(m)} + \frac{a(m) * m}{(1-a(m))^2} \right) \\ &= \frac{1}{1-a(m)} \sum_{i=0}^{m-1} i * \mathbb{P}(\mathcal{X}, S_{ab}, i) + \frac{m * a(m)}{1-a(m)}\end{aligned}$$

which gives the result.

5.3.3 With or Without Restart

The Theorem 5.3.2 is very generic and valid for any absorbing MC. It can be used to compare the expectation of the solving time for the same LS algorithm, with and without random restart. For this we use the equality binding the solving time expectation of $\mathcal{Y}$ and the first terms of the sum of the expectation for $\mathcal{X}$.

First we study a lower bound for $\mathbb{E}(\mathcal{Y}, S_{ab})$. To do the calculation, we need to know the convergence rate of $\mathbb{P}(\mathcal{X}, S_{ab}, t)$. It is reasonable to assume that $\mathbb{P}(\mathcal{X}, S_{ab}, t)$ behaves like a geometric serie ρ^t for some $\rho < 1$, because the probabilities of being absorbed multiplies at each iteration.

Corollary 5.3.3 *Let $\mathcal{X}$ an absorbing Markov process on S, and $\mathcal{Y}$ the corresponding restarted process with the same notations as above. Assume that $\rho^t \leq \mathbb{P}(\mathcal{X}, S_{ab}, t) \leq \rho'^t$ for some $\rho, \rho' \in]0, 1[$. Then*

$$\mathbb{E}(\mathcal{X}, S_{ab}) \leq \mathbb{E}(\mathcal{Y}, S_{ab})$$

Proof Since $(1 - a(m))$ is positive, $\mathbb{E}(\mathcal{Y}, S_{ab}) - \mathbb{E}(\mathcal{X}, S_{ab})$ has the same sign than:

$(1 - a(m)) * (\mathbb{E}(\mathcal{Y}, S_{ab}) - \mathbb{E}(\mathcal{X}, S_{ab}))$
$= \sum_{t=0}^{m-1} t\mathbb{P}(\mathcal{X}, S_{ab}, t) + ma(m) - (1 - a(m))\mathbb{E}(\mathcal{X}, S_{ab})$
$= m * a(m) - \sum_{t=m}^{\infty} t\mathbb{P}(\mathcal{X}, S_{ab}, t) + a(m)\mathbb{E}(\mathcal{X}, t)$
$= a(m)(m + \mathbb{E}(\mathcal{X}, t)) - \sum_{t=m}^{\infty} t\mathbb{P}(\mathcal{X}, S_{ab}, t)$
$= \sum_{t=m}^{\infty}(m + \mathbb{E}(\mathcal{X}, S_{ab}) - t)\mathbb{P}(\mathcal{X}, S_{ab}, t)$

The end of the proof is very technical and we only give the sketch of it, with a simplified version of the computation: we only use an approximation for $\mathbb{P}(\mathcal{X}, S_{ab}, t)$, instead of the detailed upper and lower bounds.

$(1 - a(m)) * (\mathbb{E}(\mathcal{Y}, S_{ab}) - \mathbb{E}(\mathcal{X}, S_{ab}))$
$\approx \sum_{t=m}^{\infty}(m + \mathbb{E}(\mathcal{X}, S_{ab}) - t)\rho^t$
$\approx \frac{\rho^m * (\mathbb{E}(\mathcal{X}, S_{ab}) - \rho - \rho * \mathbb{E}(\mathcal{X}, S_{ab})))}{(\rho-1)^2}$

The hypothesis on $\mathbb{P}(\mathcal{X}, S_{ab}, t)$ also allows us to compute $\mathbb{E}(\mathcal{X}, S_{ab}) = \rho^m/(1 - \rho)$. By replacing in the above formula, we finally obtain

$\approx \frac{\rho - \rho(1-\rho)^2 - \rho^2}{(1-\rho)^2}$

which is positive for $\rho \in]0, 1[$.

This result is important because it proves that adding random restart to an absorbing Markov chain is strictly useless. Theorem 5.3.2 also yields an upper bound for $\mathbb{E}(\mathcal{Y})$:

Corollary 5.3.4 *Let $\mathcal{X}$ an absorbing Markov process on S, and $\mathcal{Y}$ the corresponding restarted process with the same notations as above. Then*

$$\mathbb{E}(Y, S_{ab}) \leq \frac{\mathbb{E}(\mathcal{X}, S_{ab})}{1 - a(m)}$$

Proof Again, the proof is mostly technical. Cutting the sum of the absorption time for $\mathcal{X}$ at m, we have:

$$\begin{aligned} \sum_{t=m}^{\infty} t\mathbb{P}(\mathcal{X}, S_{ab}, t) &\geq \sum_{t=0}^{\infty} m\mathbb{P}(\mathcal{X}, S_{ab}, t) \\ &\geq ma(m) \end{aligned}$$

Thus

$$\begin{aligned} \sum_{t=0}^{m-1} t\mathbb{P}(\mathcal{X}, S_{ab}, t) + ma(m) &\leq \mathbb{E}(\mathcal{X}) \\ (1 - a(m))\mathbb{E}(\mathcal{Y}) &\leq \mathbb{E}(\mathcal{X}) \end{aligned}$$

If $\mathbb{E}(\mathcal{X}, S_{ab})$ is finite, then $a(m) \to 0$ when $m \to \infty$, so this upper bound tends toward $\mathbb{E}(\mathcal{X}, S_{ab})$. In the end, we obtain a frame for $\mathbb{E}(\mathcal{Y}, S_{ab})$ with an upper bound converging toward the lower bound $\mathbb{E}(\mathcal{X}, S_{ab})$. This also shows that the limit for $\mathbb{E}(\mathcal{Y}, S_{ab})$ when $m \to \infty$ is $\mathbb{E}(\mathcal{X}, S_{ab})$, as expected: the restarted process tends to the not-restarted process when the number iterations before restart tends toward infinity.

5.4 Practical Study

This section provides both an approximated version of Theorem 5.3.2, and experiments on a classical LS algorithm.

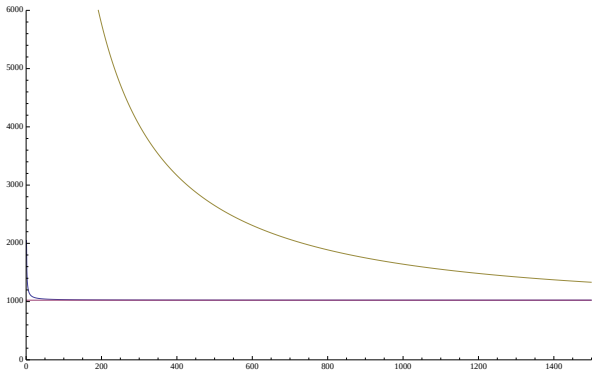


Figure 5.1: Approximations for the average of $\mathbb{E}(\mathcal{Y}, S_{ab})$ (blue curve), its upper bound (green curve) and its limit $\mathbb{E}(\mathcal{X}, S_{ab})$ (pink curve), depending on the restart parameter m.

5.4.1 Simulations

We can illustrate Theorem 5.3.2 in different ways. First, we can draw $\mathbb{E}(\mathcal{Y}, S_{ab})$ using the same approximations as in 5.2.3, that is: we consider a constant average probability of being absorbed in one step, $1 - \gamma$. Under this assumption, one has $\mathbb{P}(\mathcal{X}, S_{ab}, t) = \gamma^{t-1} * \gamma$ and $a(m) = 1/(1-\gamma)^{m-1}$.

We can then draw for instance the expectation of the absorption time for $\mathcal{Y}$ depending on m. Fig. 5.1 shows the approximations of $\mathbb{E}(\mathcal{Y}, S_{ab})$ (blue), its lower bound which is also its limit $\mathbb{E}(\mathcal{X}, S_{ab})$ (pink) and its upper bound, for the following values: γ is chosen close to one, here $\gamma = 1 - 1/2^{10}$, and m goes from 2 to 1500.

The simulations show a very fast decrease in the beginning, and an asymptotical convergence to $\mathbb{E}(\mathcal{X}, S_{ab})$. This confirms that the random restart on an absorbing Markov process does not improve the absorption time, although, very quickly, for a big enough m parameter, it does not cost too much either: the restarted curves become very close to that of the process without restart.

This figure are also interesting to see the orders of magnitude involved in the problem. The chosen value for γ, $1 - 1/2^{10}$, typically corresponds to a SAT problem of size 10 with only one solution. The order of magnitude of the absorption time for $\mathbb{E}(\mathcal{X}, S_{ab})$ is thus $2^{10} = 1024$. We can see that in average, the restarted process reaches the base process very quickly: around $m = 100$, the difference between the two curves is around 10^{-30}. This confirms that the random restart, although useless, is also harmless as soon as m is big enough.

5.4.2 Experiments

We have performed experiments with the WalkSAT algorithms, which corresponds to a homogeneous, order 1 MC. The implementation we use is walksat-v47 maintained by H. Kautz. We have chosen two problems from the SATLib Benchmark[1]. The first problem is a random 3SAT instance with 175 variables and 753 clauses. Because LS algorithm may behave very differently on random or structured instances, we also experiment on a SAT instance encoding the All-Interval Series (AIS) problem for $n = 10$. Experiments have been run on a cluster with Intel Xeon E5462 processors, each one with 4 cores at 2.80GHz. Runtimes reported here are the CPU times. Each dot of Fig. 5.2,5.3,5.4,5.5 is a mean on 200 runs. These figures show the solving time of the algorithm when the restart parameter m goes from 100 to 20000 with step 100.

For each instance, we test two parameter values for the noise parameter of WalkSAT. The first value is 50, as suggested by [Hoos and Stutzle, 2000]. It means that the algorithm has a probability $1/2$ of doing a random walk instead of the greedy step. With this value, the probability that WalkSAT reaches an absorbing state from every other state is non null. Thus WalkSAT is absorbing and respect our hypothesis. The second

[1] `http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html`

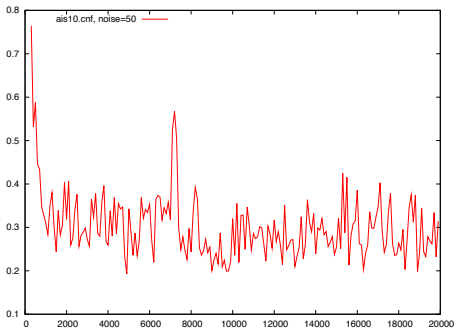


Figure 5.2: Runtime of the WalkSAT algorithm on a SAT instance encoding the AIS-10 problem, with noise=50, depending on the restart parameter m.

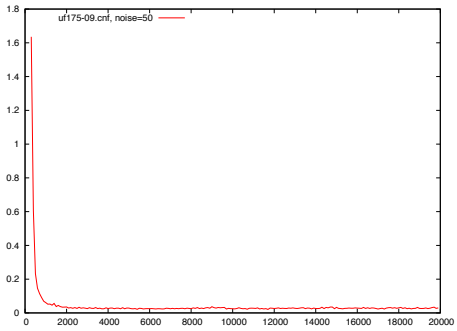


Figure 5.3: Runtime of the WalkSAT algorithm on a random SAT instance with 175 variables (uf175-09), with noise=50, depending on the restart parameter m.

value we try is noise=0. In this case, WalkSAT does not perform random moves and is equivalent to GSAT, which is greedy and expected not to be absorbing.

Fig. 5.2,5.3 show the results for a noise parameter set to 50. The curve behaves exactly as expected, with a very fast convergence to a finite limit, the solving time without restart. The curve is nearly perfect for the random instance, and far less smooth for the AIS instance. We believe that this is due to the fact that random instances, which have a uniform distribution of the clauses on the variables, give a smooth transition matrix for the associated LS algorithm. Their behavior is thus expected to be near the average case. On the contrary, structured instances cause a more peaked distribution of values in the transition matrix, thus a more variable solving time with a higher probability of variations. Yet, the very fast decrease can still be observed at the beginning of the curve.

Fig. 5.4,5.5 show the results for a noise parameter set to 0. The behavior is obviously different: the solving time is better for small values of random restart, contrarily to the preceding experiments. In this case, the Markov chain is (in general, depending on the instance) not absorbant, hence the hypothesis of Theorem 5.3.2 does not apply. This suggests that the restart mechanism is useful when the corresponding Markov chain is not connected, which corresponds to a too greedy local search (here, without escape mechanism).

In conclusion, the experiments confirm that the absorbing chain criterion discriminates the effects of the random restart on LS algorithms, and confirm that our hypothesis applies to absorbing LS algorithms.

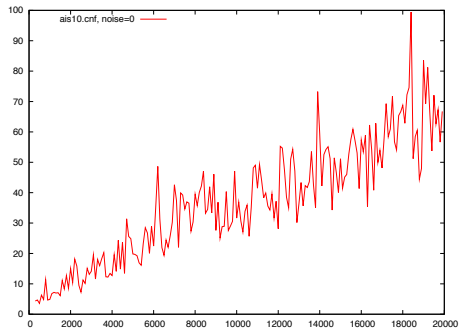


Figure 5.4: Runtime of the WalkSAT algorithm on a SAT instance encoding the AIS-10 problem, with noise=0, depending on the restart parameter m.

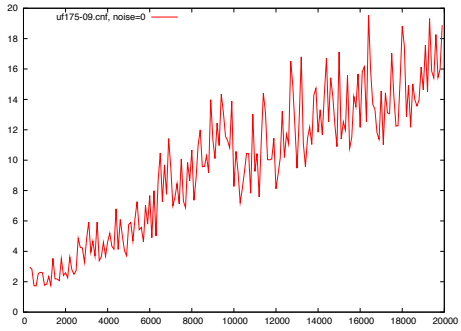


Figure 5.5: Runtime of the WalkSAT algorithm on a random SAT instance with 175 variables (uf175-09), with noise=0, depending on the restart parameter m.

5.5 Discussion

We now provide a more practical interpretation of the result, linking it the the behavioral study of real-life LS algorithms that motivates this work. For this, we investigate the validity of our hypotheses in the particular case of LS algorithms.

5.5.1 First Hypothesis: Homogeneous, Order 1

The MC we consider are very restrictive, because they need to be of order 1. It means that they have no memory of the visited space. This is thus a strong hypothesis. It is obviously valid for basic LS algorithms such as the GSAT family, but no longer valid for metaheuristics such as Tabu Search [Glover and Laguna, 1997]. In addition, the MC needs to be homogeneous. For LS algorithms, this means that the iterative behavior of the algorithm needs to be the same during all the solving process. This is true for most of the LS algorithms, except recent ones such as Sparrow/Captain Jack [Tompkins et al., 2011] which adapt their behavior along the resolution process.

However, a MC of order $t > 1$ can be reformulated into an equivalent MC of order 1, by adding states to the chain (corresponding to the possible paths of length t). This reformulation makes the number of state explode, thus the new chain is even less tractable than the original one. In this case, a study such as ours seems out of reach without approximations. However, there is no reason why the general result presented here should not apply, as the key property is only the Markov property. The only difficulty induced by a chain of order t is that all the possible paths of length t should be detailed. Yet, the key ingredient of our demonstration (detailed computation of $\mathbb{P}(\mathcal{X}, S_{ab}, t)$) should not change significantly.

5.5.2 Second Hypothesis: Absorbing Chains

The second hypothesis is that the MC is absorbing. In the context of LS algorithms, it can be false for two reasons. The first possibility is not to have absorbing states: this happens when the problem to solve has no solution. In this case, the base LS algorithm will run forever, as does the restarted algorithm. This is coherent with Theorem 5.3.2 where both sides of the equality would be infinite.

The second possibility happens when the LS algorithm has absorbing states, but there may be some states from where it is not possible to reach them. This case is more interesting. It happens when the algorithm is too greedy or has *plateau*s where it can get stuck. This phenomenon creates different connex components in the associated transition graph. Each component may, or may not, have absorbing states or solutions. In this case, randomization may have a role to play. For instance, random restart may be useful for such algorithms because it reconnects the associated graph.

5.6 Conclusion

We have proposed a probabilistic model for the random restart mechanism in Local Search algorithms. It allows us to characterize the effect of the restart on the solving time of an absorbing algorithm, with a fast convergence toward the solving time without restart. In brief, the restart is useless but also harmless: it does not cost too much as soon as the restart parameter is big enough. Our experiments show that the theoretical result applies well to LS algorithms, and that the absorbing property indeed discriminates different behaviors of the restarted algorithms.

6

Average Speed-up of Multiwalk Parallel Las Vegas Algorithms

This chapter is mostly taken from Charlotte Truchet, Alejandro Arbelaez, Florian Richoux, Philippe Codognet, *Estimating parallel runtimes for randomized algorithms in constraint solving*, J. Heuristics 22(4): 613-648 (2016), which itself is an extended version of an article at ICPP with a similar study on only local search algorithms, and an article on SAT solvers at ICLP. In this version, we also extend these two studies to complete solvers.

We designed this model in order to explain the behavior of multiwalk parallel randomized algorithms. Our main motivation was initially simply to understand a phenomenon observed by the JFLI team: multiwalk local search did perform astonishingly well on one problem (Costas arrays) and more normally on others. Once we had the probabilistic model to explain this, we also realized it could be generalized, in particular to learn speed-ups from a set of instances and use our model for predictions. We explored this possibility in a recent article with Alejandro Arbelaez and Barry O'Sullivan (ICTAI'16). My contribution to this work was to build the model and calculate the approximated speed-ups.

The original article has been lifted here. In particular, the experiments, which were extremely detailed in the article, have been greatly summarized here. I chose to only keep the tables with the actual and predicted speed-ups. The original experiments can be found in the original article.

6.1 Introduction

In the last years, parallel algorithms for solving hard combinatorial problems, such as Constraint Satisfaction Problems (CSP), have been of increasing interest in the scientific community. The combinatorial nature of the problem makes it difficult to parallelize existing solvers, without costly communication schemes. Several parallel schemes have been proposed for incomplete or complete solvers, one of the most popular (as observed in the latest SAT competitions www.satcompetition.org) being to run several competing instances of the algorithm on different processes with different initial conditions or parameters, and let the fastest process win over others. The resulting parallel algorithm thus terminates with the minimal runtime among the launched processes. The framework of independent multi-walk parallelism, seems to be a promising

way to deal with large-scale parallelism. Cooperative algorithms might perform well on shared-memory machines with a few tens of processors, but are difficult to extend efficiently on distributed hardware. This leads to so-called independent multi-walk algorithms in the CSP community [Verhoeven and Aarts, 1995] and portfolio algorithms in the SAT community (satisfiability of Boolean formula) [Gomes and Selman, 2001].

However, although it is easy to obtain good speed-up on a small-scale parallel machine (*viz.* with a few tens of processes), it is not easy to know how a parallel variant of a given algorithm would perform on a massively parallel machine (*viz.* with thousands of processes). Parallel performance models are thus particularly important for parallel constraint solvers, and any indication on how a given algorithm (or, more precisely, a pair formed by the algorithm and the problem instance) would scale on massively parallel hardware is valuable. If it becomes possible to estimate the maximum number of processes until which parallelization is efficient, then the actual parallel computing power needed to solve a problem could be deduced. This piece of information might be quite relevant, since supercomputers or systems such as Google Cloud and Amazon EC2 can be rented by processor-hour with a limit on the maximum number of processors to be used. In this context, modelling tools for the behavior of parallel algorithms are expected to be very valuable in the future.

The goal of this chapter is to study the parallel performance of randomized constraint solving algorithms under the independent multi-walk scheme, and to model the performance of the parallel execution from the runtime distribution of sequential runs of a given algorithm. Randomized constraint solvers considered in this chapter include Local Search algorithms for Constraint Satisfaction Problems, Local Search techniques for SAT, and complete algorithms with random components *e.g.,* a propagation-based backtrack search with random heuristics. An important application of this work relates to the increasing computational power being available in cloud systems (e.g., Amazon Cloud EC2, Google Cloud and Microsoft Azure), a good estimate on how the algorithm scales might allow users to rent just the right number of cores. Most papers on the performance of stochastic Local Search algorithms focus on the average runtime in order to measure the performance of both sequential and parallel executions. However, a more detailed analysis of the runtime behavior could be done by looking at the runtime of the algorithm (*e.g.,* CPU-time or number of iterations) as a random variable and performing a statistical analysis of its probability distribution. More precisely, we first approximate the empirical sequential runtime distribution by a well-known statistical distribution (*e.g.,* exponential or lognormal) and then derive the runtime distribution of the parallel version of the solver. Our model is related to *order statistics*, a rather recent domain of statistics [David and Nagaraja, 2003], which is the statistics of sorted random draws. Our method encompasses any algorithm whose solving time is random and makes it possible to formally determine the average parallel runtime of such algorithms for any number of processors.

For Local Search, we will consider algorithms in the framework of *Las Vegas algorithms* [Babai, 1979], a class of algorithms related to Monte-Carlo algorithms introduced a few decades ago, whose runtime may vary from one execution to another, even on the same input. The classical parallelization scheme of multi-walks for Local Search methods can easily be generalized to any Las Vegas algorithm. We will study two different sets of algorithms and problems: first, a Constraint-Based Local Search solver on CSP instances, and, secondly, two SAT Local Search solvers on random and crafted instances. Interestingly, this general framework encompasses other types of Las Vegas algorithms, and we will also apply it to a propagation-based constraint solver with a random labeling procedure on CSP instances.

We will confront the performance predicted by the statistical model with actual speed-ups obtained for parallel implementations of the above-mentioned algorithms and show that the prediction can be quite accurate, matching the actual speed-up up to a large number of processors. More interestingly, we can also model both the initial and the asymptotic behavior of the parallel algorithm.

This chapter extends [Truchet et al., 2013] and [Arbelaez et al., 2013] by giving a detailed presentation of the runtime estimation model, based on order statistics, and by validating the model on randomized propagation-based constraint solvers, extensive experimental results for stochastic local search algorithms on well-known CSP instances from CSPLib and SAT instances obtained from the international SAT com-

petition. Additionally, we provide a more detailed theoretical analysis of the reference distributions used for predicting the parallel performance.

The chapter is organized as follows. Section 6.2 presents the existing approaches in parallel constraint solving, and formulates the question we address in the following. Section 6.3 details our probabilistic model for the class of parallel algorithms we tackle in this article, based on Las Vegas algorithms. Several such algorithms can be used in constraint solving. The last section describes experimental results, on three different algorithms: Constraint-Based Local Search, SAT Local Search and Propagation-based methods with randomization. For each method, we applied the model to compute the parallel speed-ups, run the parallel algorithm in practice and compare and analyze the results. A conclusion, in Section 11.5, ends the chapter.

6.2 Parallel Constraint Solving

This section presents existing approaches for parallel constraint solving. The base methods are Constraint-Based Local Search, Local Search for SAT, and complete solvers based on propagation techniques and backtrack search. We then present the state of the art on estimating parallel speed-up for randomized solvers.

6.2.1 Parallel Local Search

Parallel implementation of Local Search metaheuristics [Gonzalez, 2007, Ibaraki et al., 2005] has been studied since the early 1990s, when parallel machines started to become widely available [Pardalos et al., 1995, Verhoeven and Aarts, 1995]. With the increasing availability of PC clusters in the early 2000s, this domain became active again [Alba, 2004, Crainic and Toulouse, 2002]. [Moisan et al., 2013] recently proposed a scalable parallel algorithm for backtracking algorithms based in load-balacing to distribute the work across multiple processes with low communication overhead. Apart from domain-decomposition methods and population-based method (such as genetic algorithms), [Verhoeven and Aarts, 1995] distinguishes between single-walk and multi-walk methods for Local Search. Single-walk methods consist in using parallelism inside a single search process, *e.g.,* for parallelizing the exploration of the neighborhood (see for instance [Van Luong et al., 2010] for such a method making use of GPUs for the parallel phase). Multi-walk methods, *i.e.* parallel execution of multi-start methods, consist in developing concurrent explorations of the search space, either independently or cooperatively with some communication between concurrent processes. Sophisticated cooperative strategies for multi-walk methods can be devised by using solution pools [Crainic et al., 2004], but require shared-memory or emulation of central memory in distributed clusters, thus impacting on performance. A key point is that a multi-walk scheme is easier to implement on parallel computers without shared memory and can lead, in theory at least, to linear speed-ups [Verhoeven and Aarts, 1995]. However, this is only true under certain assumptions and we will see that we need to develop a more realistic model in order to cope with the performance actually observed in parallel executions.

6.2.2 Parallel Local Search for SAT

Like for CSP, it is now currently admitted that an easy and effective manner to parallelize Local Search solvers consists in executing in parallel multiple copies of the solver with or without cooperation. We remark that nearly all parallel solvers in the parallel SAT competitions are based on the multi-walk framework[1]. The non-cooperative approach has been used in the past to solve SAT and MaxSAT instances. gNovelty+ [Pham and Gretton, 2007] executes multiple copies of gNovelty without cooperation until a solution is obtained or a given timeout is reached; and [Pardalos et al., 1996] executes multiple copies of

[1]The SAT community usually refer to the multi-walk framework as portfolio algorithms.

GRASP until an assignment which satisfies a given number of clauses is obtained. Strategies to exploit cooperation between parallel SAT Local Search solvers have been studied in [Arbelaez and Hamadi, 2011] in the context of multi-core architectures with shared memory and in [Arbelaez and Codognet, 2012] in massively parallel systems with distributed memory.

6.2.3 Parallel Complete Solvers

Early experiments on the parallelization of propagation-based complete solvers date back to the beginning of Constraint Logic Programming, cf. [Van Hentenryck, 1989], and used the search parallelism of the host logic language [de Kergommeaux and Codognet, 1994]. Most of the proposed implementations have been based on some kind of OR-parallelism, splitting the search space between different processors and relying on the shared-memory multicore architecture as the different processors work on shared data-structures representing a global environment in which the subcomputations take place. However only very few implementations of efficient constraint solvers on such machines have ever been reported, for instance [Perron, 1999] or [Chu et al., 2009] for a shared-memory architecture with 8 CPU processors. The Comet system [Van Hentenryck and Michel, 2005] has been parallelized for small clusters of PCs, both for its Local Search solver [Michel et al., 2006] and its propagation-based constraint solver [Michel et al., 2007]. More recent experiments have been done up to 12 processors [Michel et al., 2009].

Search-space splitting techniques such as domain decomposition have also been implemented in the domain of Constraint Programming, but initial experiments [Bordeaux et al., 2009] show that the speed-up goes to flatten after a few tens of processors, thus away from linear speed-up. A recent approach based on a smaller granularity domain decomposition [Régin et al., 2013] shows better performance. The results for all-solution search on classical CSPLib benchmarks are quite encouraging and show an average speed-up of 14 to 20 up with 40 processors w.r.t. the base sequential solver.

In the domain of combinatorial optimization, the most popular complete method that has been parallelized at a large scale is the classical branch and bound method [Gendron and Crainic, 1994], because it does not require much information to be communicated between parallel processes: basically only the current upper (or lower) bound of the solution. It has thus been a method of choice for experimenting the solving of optimization problems using grid computing, see for instance [Aida and Osumi, 2005] and also [Caromel et al., 2007], which use several hundreds of nodes of the Grid'5000 platform. Good speed-ups are achieved up to a few hundreds of processors but interestingly, their conclusion is that runtimes tend to stabilize afterward. A simple solving method for project scheduling problems has also been implemented on an IBM Bluegene/P supercomputer [Xie and Davenport, 2010] up to 1,024 processors, but with mixed results since they reach linear speed-ups until 512 processors only, and then no improvements beyond this limit.

6.2.4 How to Estimate Parallel Speed-Up ?

The multi-walk parallel scheme is rather simple, yet it provides an interesting test-case to study how Las Vegas algorithms can scale-up in parallel. Indeed, runtime will vary among the processes launched in parallel and the overall runtime will be equal to the minimal runtime (*i.e.* "long" runs are killed by "shorter" ones). The question is thus to quantify the relative notion of short and long runs and their probability distribution. This might give us a key to quantify the expected parallel speed-up. This can be deduced from the sequential behavior of the algorithm, and more precisely from the proportion of long and short runs in the sequential runtime distribution.

In the following, we propose a probabilistic model to quantify the expected speed-up of multi-walk Las Vegas algorithms. This makes it possible to give a general formula for the speed-up, depending on the sequential behavior of the algorithm. Our model is related to *order statistics*, which is the statistics of sorted random draws, a rather recent domain of statistics [David and Nagaraja, 2003]. Indeed, explicit formulas have been given for several well-known distributions. Relying on an approximation of the sequential

distribution, we compute the average speed-up for the multi-walk extension. Experiments show that the prediction is quite sharp and opens the way for defining more accurate models and apply them to larger classes of algorithms.

Previous works [Verhoeven and Aarts, 1995] studied the case of a particular distribution for the sequential algorithm: the exponential distribution. This case is ideal and it yields a linear speed-up. Our model enable us to approximate Las Vegas algorithms by other types of distribution, such as a shifted exponential distribution or a lognormal distribution. In the last two cases the speed-up is no longer linear, but admits a finite limit when the number of processors goes toward infinity. We will see that these distributions fit experimental data for some problems.

The literature provides other probabilistic models for parallel Las Vegas algorithms. For parallelization schemes based on restart strategies, Luby [Luby et al., 1993] proposes an optimal universal strategy, which achieves the best speed-ups amongst the restart-based universal strategies. Although our work, which is not based on restart, does not fit within this framework, it is related to it since it uses similar probabilistic ideas. In our case, a probabilistic model is used to model the efficiency of a parallel scheme, without modifying the algorithm. In their case, probabilistic tools are used to find an optimal restart scheme for a base algorithm. With the same idea of restart-based parallelization of Las Vegas algorithms, [Shylo et al., 2011] goes a step further and provides a more detailed model. It clarifies in particular the cases of super-linear speed-ups, which are due, in this framework, to inefficient (or not well-tuned) sequential algorithms. Interestingly, this article also experiments with log-normally distributed sequential algorithms, which confirms our hypothesis that not only exponential distributions have to be investigated (as assumed by [Aiex et al., 2002, Aiex et al., 2007]).

6.3 Probabilistic Model

Randomized algorithms are stochastic processes. Their behavior (output, running time...) is non-deterministic and varies according to a probabilistic distribution, which may or may not be known. For instance, Local Search algorithms include several random components: choice of an initial configuration, choice of a move among several candidates, plateau mechanism, random restart, etc. A complete algorithm may also have a stochastic behavior when they include random components, such as random heuristics, restarts with randomization, etc.

In the following, we first define the class of algorithms that our model encompasses. We then present our probabilistic model, considering the *computation time* of an algorithm (whatever it is) as a random variable, and using elements of probability theory to study its multi-walk parallel version.

6.3.1 Parallel Las Vegas Algorithms

The notion of Las Vegas algorithm encompasses a wide range of combinatorial solvers. We borrow the following definition from [Hoos and Stütze, 2005], Chapter 4.

Las Vegas Algorithm An algorithm A for a problem class Π is a (generalized) Las Vegas algorithm if and only if it has the following properties:

1. If for a given problem instance $\pi \in \Pi$, algorithm A terminates returning a solution s, s is guaranteed to be a correct solution of π.

2. For any given instance $\pi \in \Pi$, the runtime of A applied to π is a random variable.

This definition includes algorithms which are not guaranteed to return a solution. However in practice, we will only consider terminating Las Vegas algorithms, such as Local Search algorithms which always terminate if run for an unbounded time.

Let us now formally define a parallel multi-walk Las Vegas algorithm.

Multi-walk Las Vegas Algorithm An algorithm A' for a problem class Π is a (parallel) multi-walk Las Vegas algorithm if and only if it has the following properties:

1. It consists of n instances of a sequential Las Vegas algorithm A for Π, say $A_1, ..., A_n$.
2. If, for a given problem instance $\pi \in \Pi$, there exists at least one $i \in [1, n]$ such that A_i terminates, then let $A_m, m \in [1, n]$, be the instance of A terminating with the minimal runtime and let s be the solution returned by A_m. Then algorithm A' terminates in the same time as A_m and returns solution s.
3. If, for a given problem instance $\pi \in \Pi$, all $A_i, i \in [1, n]$, do not terminate then A' does not terminate.

6.3.2 Min Distribution

Consider the problem of solving a given problem instance using a Las vegas algorithm, say, tabu search on the MAGIC-SQUARE 10×10. Depending on the result of some random components inside the algorithm, it may find a solution after 0 iterations, 10 iterations, or 10^6 iterations. The number of iterations of the algorithm is thus a discrete random variable, let us call it Y, with values in $\mathbb{N}$. Y can be studied through its cumulative distribution, which is by definition the function $\mathcal{F}_Y$ s.t. $\mathcal{F}_Y(x) = \mathbb{P}[Y \leq x]$, that is, the function which, for a x in the scope of the random variables, gives the probability that a random draw is smaller than x. Another possible tool to model a random variable is its distribution, which is by definition the derivative of $\mathcal{F}_Y$: $f_Y = \mathcal{F}'_Y$. Notice that the computation time is not necessarily the CPU-time; it can also be the number of iterations performed during the execution of the algorithm.

It is often more convenient to consider distributions with values in $\mathbb{R}$ because it makes calculations easier. For the same reason, although f_Y is defined in $\mathbb{N}$, we will use its natural extension to $\mathbb{R}$. This step is merely technical, and the probability distribution in $\mathbb{N}$ can be retrieved from the distribution in $\mathbb{R}$.

The expectation of the computation is then defined by the standard formula for real-valued distributions: $\mathbb{E}[Y] = \int_0^\infty t f_Y(t) dt$. This formula is the extension to $\mathbb{R}$of the classical expectation formula in the case of integer distributions ($\sum_0^\infty t f_Y(t)$).

Assume that the base algorithm is concurrently run in parallel on n processors. In other words, over each processor the running process is a copy of the algorithm with different initial random seed. The first process that finds a solution then kills all others and the algorithm terminates. The i-th process corresponds to a draw of a random variable X_i, following distribution f_Y. The variables X_i are thus independently and identically distributed (i.i.d.). The computation time of the whole parallel process is also a random variable, let's call it $Z^{(n)}$, with a distribution $f_{Z^{(n)}}$ that depends on both n and f_Y. Since all the X_i are i.i.d., the cumulative distribution $\mathcal{F}_{Z^{(n)}}$ can be computed as follows:

$$\begin{array}{rcll}
\mathcal{F}_{Z^{(n)}} = & \mathbb{P}[Z^{(n)} \leq x] & \text{by definition} \\
= & \mathbb{P}[\exists i \in \{1...n\}, X_i \leq x] & \text{multiwalk rule} \\
= & 1 - \mathbb{P}[\forall i \in \{1...n\}, X_i > x] & \text{probability formula for the negation} \\
= & 1 - \prod_{i=1}^n \mathbb{P}[X_i > x] & \text{because the random variables are i.i.d.} \\
= & 1 - (1 - \mathcal{F}_Y(x))^n & \text{by definition}
\end{array}$$

which leads to:

$$\begin{array}{rcl}
f_{Z^{(n)}} & = & (1 - (1 - \mathcal{F}_Y)^n)' \\
& = & n f_Y (1 - \mathcal{F}_Y)^{n-1}
\end{array}$$

Thus, knowing the distribution for the base algorithm Y, one can calculate the distribution for $Z^{(n)}$. In the general case, the formula shows that the parallel algorithm favors short runs, by killing the slower

processes. Thus, we can expect that the distribution of $Z^{(n)}$ moves toward the origin, and is more peaked. As an example, Figure 6.1 shows this phenomenon when the base algorithm admits a Gaussian distribution.

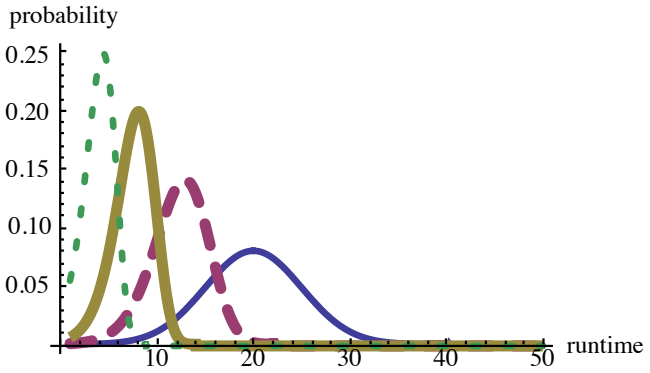


Figure 6.1: Distribution of $Z^{(n)}$, in the case where Y admits a Gaussian distribution (cut on $\mathbb{R}^-$ and renormalized). The blue curve is Y. The distributions of $Z^{(n)}$ are in pink for $n = 10$, in yellow for $n = 100$ and in green for $n = 1000$.

6.3.3 Expectation and Speed-up

The model described above gives the probability distribution of a parallelized version of any randomized algorithm. We can now calculate the expectation for the parallel process with the following relation:

$$\begin{aligned}\mathbb{E}\left[Z^{(n)}\right] &= \int_0^\infty t f_{Z^{(n)}}(t)\, dt \\ &= n\int_0^\infty t f_Y(t)(1-\mathcal{F}_Y(t))^{n-1}\, dt\end{aligned}$$

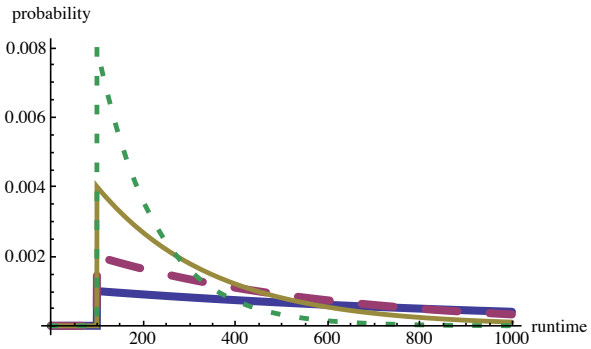
Unfortunately, this does not lead to a general formula for $\mathbb{E}\left[Z^{(n)}\right]$. In the following, we will study it for different specific distributions.

To measure the gain obtained by parallelizing the algorithm on n processors, we will study the speed-up $\mathcal{G}_n$ defined as:

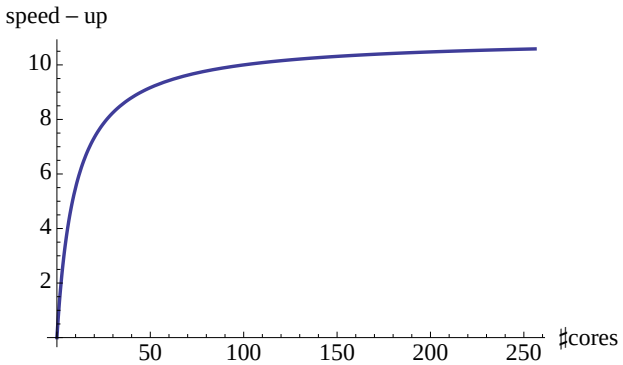
$$\mathcal{G}_n = \mathbb{E}\left[Y\right] / \mathbb{E}\left[Z^{(n)}\right]$$

Again, no general formula can be computed and the expression of the speed-up depends on the distribution of Y.

However, it is worth noting that our computation of the speed-up is related to order statistics, see [David and Nagaraja, 2003] for a detailed presentation. Order statistics are the statistics of sorted random draws. For instance, the first order statistics of a distribution is its minimal value, and the k^{th} order statistic is its k^{th}-smallest value. For predicting the speed-up of a multi-walk Las Vegas algorithm on n processors, we are indeed interested in computing the expectation of the distribution of the minimum among n draws. As the above formula suggests, this may lead to heavy calculations, but recent studies such as [Nadarajah, 2008] give explicit formulas for this quantity for several classical probability distributions. Except in the case of the exponential distribution, detailed below, the formulas given for the minimum order statistics are rather complicated. In some cases, a symbolic computation may not even success in a reasonable amount of time. When this happens, we will perform a first step of symbolic computation, then another step of numeric integration to obtain the numerical value for the speed-up.



(a) For an exponential distribution, here in blue with $x_0 = 100$ and $\lambda = 1/1000$, simulations of the distribution of w.r.t. the number of processors. $Z^{(n)}$ for $n = 2$ (pink), $n = 4$ (yellow) and $n = 8$ (green).



(b) Predicted speed-up with $x_0 = 100$ and $\lambda = 1/1000$, w.r.t. the number of processors.

Figure 6.2: Case of an exponential distribution

6.3.4 Case of an Exponential Distribution

Assume that Y has a shifted exponential distribution, as it has been suggested by [Aiex et al., 2002, Aiex et al., 2007].

$$f_Y(t) = \begin{cases} 0 & \text{if } t \leq x_0 \\ \lambda e^{-\lambda(t-x_0)} & \text{if } t > x_0 \end{cases}$$

$$\mathcal{F}_Y(t) = \begin{cases} 0 & \text{if } t \leq x_0 \\ 1 - e^{-\lambda(t-x_0)} & \text{if } t > x_0 \end{cases}$$

$$\mathbb{E}[Y] = x_0 + 1/\lambda$$

Then the formula of Section 6.3.2 can be symbolically computed by hand:

$$f_{Z^{(n)}}(t) = \begin{cases} 0 & \text{if } t \leq x_0 \\ n\lambda e^{-n\lambda(t-x_0)} & \text{if } t > x_0 \end{cases}$$

$$\mathcal{F}_{Z^{(n)}}(t) = \begin{cases} 0 & \text{if } t \leq x_0 \\ 1 - e^{-n\lambda(t-x_0)} & \text{if } t > x_0 \end{cases}$$

The intuitive observation of section 6.3.2 is easily seen on the expression of the parallel distribution, which has an initial value multiplied by n but an exponential factor decreasing n-times faster, as shown on the curves of Figure 6.2(a).

And in this case, one can symbolically compute both the expectation and speed-up for $Z^{(n)}$:

$$\begin{aligned} \mathbb{E}\left[Z^{(n)}\right] &= n\lambda \int_{x_0}^{\infty} t e^{-n\lambda(t-x_0)}\, dt \\ &= x_0 + \frac{1}{n\lambda} \\ \mathcal{G}_n &= \frac{x_0 + \frac{1}{\lambda}}{x_0 + \frac{1}{n\lambda}} \end{aligned}$$

Figure 6.2(b) shows the evolution of the speed-up when the number of processors increases. With such a rather simple formula for the speed-up, it is worth studying what happens when the number of processors n goes to infinity. Depending on the chosen algorithm, If $x_0 = 0$, then the expectation goes to 0 and the

speed-up is equal to n. This case has already been studied by [Verhoeven and Aarts, 1995]. For $x_0 > 0$, the speed-up admits a finite limit which is $\frac{x_0+\frac{1}{\lambda}}{x_0} = 1 + \frac{1}{x_0\lambda}$. Yet, this limit may be reached slowly, and depends on the values of x_0 and λ. From the previous formula we observe that the closer x_0 is to zero, the better is the speedup. Another interesting value is the coefficient of the tangent at the origin, which approximates the speed-up for a small number of processors. In case of an exponential, it is $(x_0 * \lambda + 1)$. The higher x_0 and λ, the bigger is the speed-up at the beginning. In the following, we will see that, depending on the combinations of x_0 and λ, different behaviors can be observed.

6.3.5 Case of a Lognormal Distribution

Other distributions can be considered, depending on the behavior of the base algorithm. We will study the case of a lognormal distribution, which is the log of a Gaussian distribution. The lognormal distribution appears in several of our experiments in section 6.4.1 and 6.4.2. The lognormal distribution has two parameters, the mean μ and the standard deviation σ. In the same way as the shifted exponential, we shift the distribution so that it starts at a given parameter x_0. Formally, a (shifted) lognormal distribution is defined as:

$$f_Y(t) = \begin{cases} 0 & \text{if } t < x_0 \\ \frac{\Phi(log(t-x_0))}{t-x_0} & \text{if } t > x_0 \end{cases}$$

where $\Phi(t) = \frac{1}{\sqrt{2*\pi}} e^{-t^2/2}$.

The mean, variance and median are known and equal to $e^{\mu+\frac{\sigma^2}{2}}$, $e^{2\mu+\sigma^2}(e^{\sigma^2}-1)$ and e^{μ} respectively.

Figure 6.3(a) depicts lognormal distributions of $Z^{(n)}$, for several n. The computations for the distribution of $Z^{(n)}$and the theoretical speed-up are the same as given in section 6.3.3. The computation of the expectation for the moments of order statistics of a lognormal distribution can be found in [Nadarajah, 2008], which gives an explicit formula with only a numerical integration step. We only recall from [Nadarajah, 2008] this formula for the first moment (expectation) of the first order statistics (minimum distribution), shifted by x_0, which we are interested in.

We will need Lauricella functions defined by:

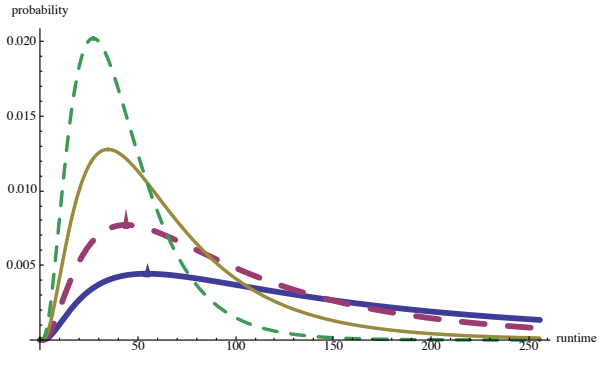
$$F_A^{(n)}(a; b_1, ..., b_n; c_1, ..., c_n; x_1, ..., x_n)) = \sum_{m_1=0}^{\infty} ... \sum_{m_n=0}^{\infty} \frac{(a)_{m_1+...+m_n}(b_1)_{m_1}...(b_n)_{m_n}}{(c_1)_{m_1}...(c_n)_{m_n}} \frac{x_1^{m_1}...x_n^{m_n}}{m_1!...m_n!}$$

Then one has:

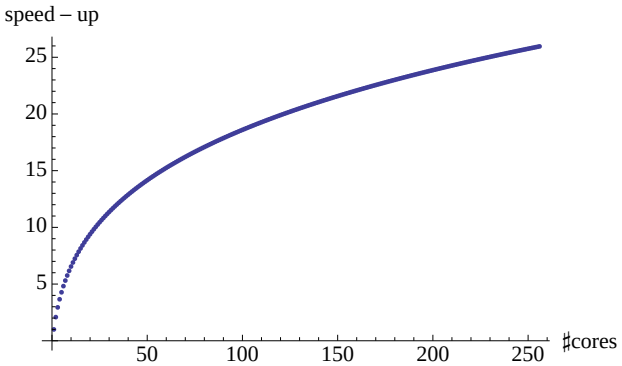
$$\mathbb{E}\left[Z^{(n)}\right] = x_0 + ne^{1/2} \sum_{l=0}^{n-1} \binom{n-1}{l} \left(-\frac{1}{2}\right)^l \sum_{p=0}^{r} \binom{l}{p} \left(-\frac{2}{\sqrt{\pi}}\right)^p * \mathbb{E}\left[\left(\frac{\mathcal{N}\sqrt{2}-1}{2\sqrt{2}}\right)^p F_A^{(p)}(\frac{1}{2}, ..., \frac{1}{2}; \frac{3}{2}, ..., \frac{3}{2}; -\frac{(\mathcal{N}\sqrt{2}-1)^2}{8}, ..., -\frac{(\mathcal{N}\sqrt{2}-1)^2}{8})\right]$$

where $\mathcal{N}$ is a standard normal random variable. In addition, [Nadarajah, 2008] provides pointers to routines in Mathematica to compute the terms of this formula with only one step of numerical integration. In practice, we will get our numerical results with an earlier step of numerical integration in Mathematica, with a good accuracy.

This allows us to draw the general shape of the speed-up, an example being given on Figure 6.3(b). Due to the numerical integration step, which requires numerical values for the number of processors n, we restrict the computation to integer values of n.



(a) Histogram of sequential behaviour



(b) Predicted speed-up with $x_0 = 0$, $\mu = 5$ and $\sigma = 1$, w.r.t. the number of processors.

Figure 6.3: Case of a lognormal distribution

6.3.6 Methodology

In this section we detail how to perform the prediction of the parallel runtime and speed-up with respect to sequential execution from the analysis of the sequential runtime distribution. On each problem, the sequential benchmark gives observations of the distribution of the algorithm runtime f_Y. Yet, the exact distribution is still unknown. It can be any real distribution, not even a classical one. In the following, we will rely on the assumption that Y is distributed with a known parametric distribution. We perform a statistical test, called Kolmogorov-Smirnov test, on the hypothesis $\mathcal{H}_0$ that the collected observations correspond to a theoretical distribution. Assuming $\mathcal{H}_0$, the test first computes the probability that the distance between the collected data and the theoretical distribution does not significantly differ from its theoretical value. This probability is called the p-value.

Then, the p-value is compared to a fixed threshold (usually 0.05). If it is smaller, one rejects $\mathcal{H}_0$. For us, it means that the observations do not correspond to the theoretical distribution. If the p-value is high, we will consider that the distribution of Y is approximated by the theoretical one. Note that the Kolmogorov-Smirnov test is a statistical test, which in no way proves that Y follows the distribution. Beside, it is based on a metric which measures the maximum interval between the two tested curves, which in our case can happen far from the zone of interest (the beginning of the curve). However, as it will be seen in the following, it is accurate enough for our purpose.

Our benchmarks appear to fit with two distributions: the exponential distribution, as suggested by [Eadie, 1971], and the lognormal distribution. We have also performed the Kolmogorov-Smirnov test on other distributions (*e.g.,* Gaussian and Lévy), but obtained negative results w.r.t. the experimental benchmarks, thus we do not include them in the sequel. For each problem, we need to estimate the value of the parameters of the distribution, which is done on a case by case basis. Once we have an estimated distribution for the runtimes of Y, it becomes possible to compute the expectation of the parallel runtimes and the speed-up using the formulas of Section 6.3.3.

In the following, all the analyses are done on the number of iterations, because they are more likely to be unbiased, and all the mathematical computations are done with Mathematica [Wolfram, 2003].

6.4 Experiments

In this section, we briefly describe the results obtained on three algorithms in combinatorial optimization. Only the main result, *e.g.,* the comparison between the predicted curve and the actual one, is given. We refer the interested reader to the original publication for more details.

The methodology is always the same: for a given method and a given problem, we run our benchmarks in a sequential manner a certain number of times to evaluate the quality of the proposed model.

Instance		speed-up on k processors				
		16	32	64	128	256
MS200	experimental	16.6	22.2	29.9	34.3	45.0
	predicted	15.94	22.04	28.28	34.26	39.7
AI700	experimental	12.8	20.2	29.3	37.3	48.0
	predicted	13.7	23.8	37.8	53.3	67.2
Costas21	experimental	15.8	26.4	60.0	159.2	290.5
	predicted	16.0	32.0	64.0	128.0	256.0

Table 6.1: Comparison: experimental and predicted speed-ups

6.4.1 Application to Constraint-based Local Search

We performed experiments based on a generic, domain-independent Constraint-Based Local Search method, named Adaptive Search, has been proposed by [Codognet and Diaz, 2001b, Codognet and Diaz, 2003]. We used for our experiments the reference implementation of Adaptive Search (AS) which has been developed as a framework library in C is and available as a freeware at the URL:
`http://cri-dist.univ-paris1.fr/diaz/adaptive/`

We detail here the performance and speed-ups obtained with both sequential and parallel multi-walk Adaptive Search implementations. We have chosen to test this method on a hard combinatorial problem abstracted from radar and sonar applications (COSTAS ARRAY) and two problems from the CSPLib benchmark library[2]: ALL-INTERVAL Series (prob007 in CSPLib), and the MAGIC-SQUARE problem (prob019 in CSPLib). In the three cases, runtimes and numbers of iterations are spread over a large interval, illustrating the stochasticity of the algorithm. Depending on the benchmark, there is a ratio of a few thousands times between the minimum and the maximum runtimes.

Table 6.1 presents the comparison between the predicted and the experimental speed-ups. We can see that the accuracy of the prediction is very good up to 64 parallel processors and then the divergence is limited even for 256 parallel processors.

For the MS 200 problem, the experimental speed-up and the predicted one are almost identical up to 128 processors and diverging by 10% for 256 processors. For the AI 700 problem, the experimental speed-up is below the predicted one by a maximum of 30% for 128 and 256 processors. For the Costas 21 problem, the experimental speed-up is above the predicted one by 15% for 128 and 256 processors.

6.4.2 Application to SAT Local Search

Let us now look at a different problem domain (SAT - the Satisfiability Problem for Boolean formulas) and different local search solvers. We focus our attention on two well-known problem families: random and crafted instances. Moreover, we consider the two best Local Search solvers from the 2012 SAT competition[3]: CCASAT [Cai et al., 2012] and Sparrow [Balint and Fröhlich, 2010]. Both solvers were used with their default parameters and with a timeout of 3 hours for each experiment.

We performed experiments with two well-known problem families of instances coming from the SAT'11 competition: random and crafted. In particular, we used 10 random instances (6 around the phase transition and 4 outside the phase transition) and 10 crafted instances. Complete details of the selected instances for the evaluation is available at [Arbelaez et al., 2013], hereafter we denote random instances as Rand-[1 to 10] and crafted instances as crafted-[1 to 9]. In order to obtain the empirical data for the theoretical distribution (predicted by our model from the sequential runtime distribution), we performed 500 runs of the sequential algorithm.

[2] `http://www.csplib.org`
[3] http://baldur.iti.kit.edu/SAT-Challenge-2012/results.html

Instance		Sparrow - Runtime on k proc.				CCASAT - Runtime on k proc.			
		48	96	192	384	48	96	192	384
Rand-1	Actual	163.8	140.4	125.2	113.7	160.0	143.0	122.8	112.0
	Predicted	133.8	110.5	92.7	78.8	137.7	120.6	106.7	95.3
Rand-2	Actual	213.2	191.4	166.2	142.5	186.8	169.3	159.3	142.8
	Predicted	183.5	152.8	129.2	110.6	153.4	134.7	119.6	107.1
Rand-3	Actual	175.9	151.2	135.8	123.5	166.7	155.6	143.5	132.2
	Predicted	183.5	152.8	129.2	110.6	153.4	134.7	119.6	107.1
Rand-4	Actual	202.3	179.2	159.5	141.8	193.1	176.0	169.4	158.7
	Predicted	175.7	149.5	128.9	112.4	170.6	155.9	143.5	132.8
Rand-5	Actual	219.6	201.0	182.5	161.9	212.2	191.3	176.8	165.8
	Predicted	185.0	155.3	132.3	114.0	179.8	164.3	151.2	140.0
Rand-6	Actual	185.5	167.1	150.3	137.5	190.9	179.3	168.4	153.4
	Predicted	158.3	133.6	114.4	99.1	160.6	147.0	135.4	125.6
Rand-7	Actual	151.2	102.7	63.8	51.1	22.9	33.7	54.3	67.8
	Predicted	195.8	143.0	107.3	82.3	182.8	142.6	113.7	92.2
Rand-8	Actual	126.6	81.9	51.1	30.9	131.8	83.9	64.8	39.7
	Predicted	93.8	58.5	40.8	32.0	76.9	56.4	46.1	41.0
Rand-9	Actual	33.9	18.4	13.1	9.0	45.0	31.0	22.7	16.3
	Predicted	28.1	17.4	12.1	9.4	38.5	29.4	23.0	18.3
Rand-10	Actual	63.4	48.9	40.7	30.9	113.8	94.7	72.9	54.2
	Predicted	54.6	36.8	27.9	23.4	105.6	85.3	70.2	58.6

Table 6.2: Runtimes for random instances up to 384 proc. (processors)

Random instances

Table 6.2 shows the empirical and predicted runtime for both Sparrow and CCASAT on all instances using 48, 96, 192, and 384 processors. Detailed tables with the speed-up for Sparrow and CCASAT are available in [Arbelaez et al., 2013]. Summing up both solver report the same tendency as the empirical data. Furthermore, the speed-up factor of the references far from linear (ideal), a phenomenon well described by the predicted model.

It can also be observed that random instances around the phase transition exhibit a lower speed-up factor than the remaining random instances. For instance, the best empirical speed-up factor obtained for instances in the phase transition is 7.0 for Sparrow and 3.4 for CCASAT; and the best speed-up factor obtained for instances outside the phase transition is 114.2 for Sparrow and 50.5 for CCASAT.

Crafted instances

Let us switch our attention now to crafted instances, for which we have to treat CCASAT and Sparrow differently. For CCASAT, we were unable to find a theoretical distribution which fits the empirical data. It should be also noticed that CCASAT has mainly been designed and tuned to handle random instances. For Sparrow on all crafted instances, the KS test shows a much better p-value for the exponential distribution than for the lognormal one, see Table **??**. The confidence level is quite high for the instances Crafted-2,-3,-4,-5,-8,-9, with p-value up to 0.97, while the p-value is between 0.01 and 0.02 for Crafted-1,-6,-7. Also, as the minimum runtime is much smaller than the mean (at least 300 times smaller), we can approximate the empirical data by a non-shifted exponential distribution [Truchet et al., 2013].

As can be seen in Table 6.3 the multi-walk parallel approach scales well for Sparrow on crafted instances as the number of processors increases. Indeed a nearly linear speed-up is obtained for nearly all the instances. As expected, the speed-up predicted by our model is optimal, and this result is consistent with those obtained in [Hoos and Stützle, 1999].

Instance		Runtime on k proc.				Speed-Up ok k proc.			
		48	96	192	384	48	96	192	384
Crafted-1	Actual	97.7	43.7	19.1	9.8	35.1	78.6	179.6	349.8
	Predicted	71.6	35.8	17.9	8.9	48.0	96.0	192.0	384.0
Crafted-2	Actual	67.8	36.4	17.5	7.2	39.9	74.4	154.7	375.2
	Predicted	56.4	28.2	14.1	7.0	48.0	96.0	192.0	384.0
Crafted-3	Actual	94.8	49.3	23.2	11.9	36.1	69.6	147.6	286.1
	Predicted	71.5	35.7	17.8	8.9	48.0	96.0	192.0	384.0
Crafted-4	Actual	87.5	42.0	17.3	9.7	30.8	64.2	155.4	277.8
	Predicted	56.2	28.1	14.0	7.0	48.0	96.0	192.0	384.0
Crafted-5	Actual	33.7	15.1	7.6	4.2	46.3	103.2	204.1	371.6
	Predicted	32.5	16.2	8.1	4.0	48.0	96.0	192.0	384.0
Crafted-6	Actual	130.0	69.8	25.6	12.8	27.6	51.5	140.5	279.5
	Predicted	74.9	37.4	18.7	9.3	48.0	96.0	192.0	384.0
Crafted-7	Actual	95.0	51.3	28.4	11.6	37.8	70.0	126.3	308.0
	Predicted	74.9	37.4	18.7	9.3	48.0	96.0	192.0	384.0
Crafted-8	Actual	17.2	10.8	5.3	2.6	56.4	89.6	181.1	363.6
	Predicted	20.2	10.1	5.0	2.5	48.0	96.0	192.0	384.0
Crafted-9	Actual	27.2	12.1	5.9	3.6	47.5	106.6	217.3	358.0
	Predicted	27.0	13.5	6.7	3.3	48.0	96.0	192.0	384.0

Table 6.3: Parallel performance of Sparrow on crafted instances (proc. stands for processors)

6.4.3 Application to Propagation-based Constraint Solving

In order to perform the following experimentation we equipped Gecode[4] with the multi-walk framework. We also used a randomized version of the solver. At each node, we use *wdeg* and *min-dom* to select the most appropriate variable. The former selects the variable with the largest *weighted degree*[5] value is selected (breaking ties at random), the latter selects the variables with the smallest valid domain at a given state of the search. The value for the chosen variable is selected uniformly at random.

It is highly recognized that the runtime distribution for tree-based search methods for a large number of problems is heavy-tailed [Gomes et al., 2000a] and adding restarts (*i.e.* stopping the search after a given number of backtracks, and re-initializing it) helps to alleviate the heavy-tailed phenomenon. In this chapter, we use a fixed restart strategy where the restart cutoff is 250 backtracks. For each problem instance we compute the sequential performance using *wdeg* and *min-dom*, and use the heuristic with better performance (in sequential settings) to evaluate the model described above to estimate the performance of the solver in parallel.

Tables 6.4 reports the estimated runtime (top) and speed-up (bottom) of the multi-walk version of the propagation-based constraint solving algorithm. Interestingly, this simple parallel scheme exhibits very good speedup factors for the three benchmarks: super-linear for MAGIC-SQUARE with 48, 96, cores, linear for nearly all experiments for ALL-INTERVAL, and a factor 122 (w.r.t. the sequential solver) for COSTAS ARRAY with 384 cores.

The runtime prediction is very close to the empirical data for the three problems with gap between the prediction and the actual execution time of up 13.7% for MAGIC-SQUARE with 384 cores, 14% for ALL-INTERVAL with 92 cores, and 8% for COSTAS ARRAY with 384 cores. Furthermore, the predictions of the speedup of the solvers are very encouraging, for the MAGIC-SQUARE problem it can be observed that the gap between the actual and estimated speedup is up to 13% with 24 cores, for the ALL-INTERVAL problem is up to 12% with 96 cores, and for COSTAS ARRAY is up to 4% with 192 cores. Finally, we

[4] www.gecode.org

[5] We use AFC, the Gecode implementation of *wdeg*

Problem	1 proc.		performance (time/speed-up) on k proc. 24	48	96	192	384
MS 15	38.20	Actual	2.06	0.66	0.47	0.31	0.25
			18.546	57.87	81.27	123.22	152.8
		Predicted	1.78	0.99	0.59	0.39	0.29
			21.46	38.58	64.74	97.94	131.72
AI 18	58.08	Actual	3.54	1.56	0.69	0.42	0.28
			16.41	37.23	84.17	138.29	207.43
		Predicted	2.42	1.21	0.60	0.30	0.15
			24.00	48.00	96.00	192.00	384.00
Costas 17	40.39	Actual	3.41	1.64	0.93	0.49	0.33
			11.83	24.62	43.43	82.42	122.39
		Predicted	1.93	1.09	0.68	0.47	0.36
			20.92	37.05	59.39	85.93	112.191

Table 6.4: Parallel runtimes in seconds (proc. stands for processors). Each cell in the performance indicates the runtime (top) and speedup (bottom) for MS with *wdeg*, AI and Costas with *min-dom*

would like to point out that this methodology requires independently and identically distributed executions of the sequential algorithm, so that, parallel algorithms based on search-splitting techniques [Bordeaux et al., 2009, Chu et al., 2009, Moisan et al., 2013] and cooperative techniques [Arbelaez and Codognet, 2012] do not fit within the proposed framework.

6.5 Conclusion and Future Work

We have proposed a theoretical model for predicting and analyzing the speed-ups of Las Vegas algorithms and applied it for Local Search methods in two different domains, Constraint-Based Local Search and SAT, and also for propagation-based constraint solving with random labeling heuristics. Interestingly, we have observed that, for the different algorithms and the variety of instances considered in this study, the runtime distribution can be characterized using two types of distributions: exponential (shifted and non-shifted) and lognormal.

It is worth noting that our model mimics the behaviors of the experimental results very closely, as shown by the predicted speed-ups matching closely the real ones. We showed that the parallel speed-ups predicted by our statistical model are accurate, matching the actual speed-ups very well up to several hundreds of processors.

However, a limitation of our approach is that, in practice, we need to be able to approximate the sequential distribution. In addition, this distribution must be one of the distributions for which the first order statistics is known, symbolically (as the exponential) or numerically (as the lognormal). Nevertheless, recent results in the field of order statistics give explicit formulas for a number of useful distributions: Gaussian, lognormal, gamma, beta. This provides a wide range of tools to analyze different behaviors.

Another interesting extension of this work would be to devise a method for predicting the speed-up from scratch, that is, without any knowledge on the algorithm distribution. Our observations suggest that the sequential runtime of both LS and complete search are well approximated by a small number of distributions. This could be intensively tested on a wider range of problems. In particular, it is important to know wether the sequential distribution of different instances of a given problem belong to the same family. Then we can devise a method for estimating the sequential distribution based on a limited number of observations, possibly on small instances, and then estimate the parallel speed-up for larger instances. This would allow us to predict if a simple multi-walk parallelization scheme, which does not imply to modify the algorithms, is likely to be efficient, or not.

7

Average Case Study of the `all-different` Constraint Propagation

This Chapter is taken from Jérémie du Boisberranger, Danièle Gardy, Xavier Lorca, Charlotte Truchet, *When is it worthwhile to propagate a constraint? A probabilistic analysis of AllDifferent*, ANALCO 2013: 80-90. The main result is a theorem describing the asymptotical behavior of the `AllDifferent` propagator in terms of removed values, based on the bound consistency of the global constraint.

I think that the most important part of this theorem is not the approximations themselves, but the hypotheses, which define a notion of tightness of the constraint, depending of which the propagator may be useful or not. My contribution to this work was to build the probabilistic model.

7.1 Constraint Solvers and Complexity Analysis.

7.1.1 Constraint Programming.

This chapter explores a mathematical way to predict the behavior of some algorithms, called *propagators*, used in *Constraint Programming* [Rossi et al., 2006a].

Constraint Programming (CP) aims at solving hard combinatorial problems expressed as *Constraint Satisfaction Problems* (CSP). A CSP is made of three parts: the first one is a set of *variables* that represents the unknowns of the problem; the second part is a set of finite *domains* describing the possible (usually integer) values of each variable; the third part contains the *constraints* which express a combinatorial relation between the variables. A constraint can be built on classical logical predicates, for instance $x = y + 2$, or on specific relations called *global constraints* [Bessière and van Hentenryck, 2003]. Some global constraints extend the language expressivity by adding predicates that could not be formulated with basic predicates. Others could be reformulated as classical constraints, but they encapsulate a global relationship that can be used to improve the algorithms implementing the constraints and the resolution process. For instance, the `AllDifferent` constraint forces a list of variables to take different values; it is semantically identical to $n(n-1)/2$ constraints of the type $x_i \neq x_j$ on all the possible couples of the variables. A *solution* for a CSP

is an assignment of each variable to a value in its domain such that all the constraints are simultaneously satisfied.

CP operational nature is based on the *propagation-search* paradigm. The *propagation* mechanism detects and suppresses inconsistent parts of the domains, *i.e.* values that cannot appear in a solution. For instance, let us consider the constraint $x = y + 2$: assuming that the domain for x is $D_x = [1...20]$ and the domain for y is $D_y = [3...30]$, it is obvious that only the values $[5...20]$ for x and $[3...18]$ for y must be considered; the other values in D_x and D_y cannot appear in a solution, and are thus inconsistent. [1] For each particular constraint, a specific algorithm, called *propagator*, removes inconsistent values from the domains: it takes as an input the domains of the variables involved in the constraint, and outputs the corresponding consistent domains, which are called *globally arc-consistent* domains (GAC). In practice and depending on the constraint, a propagator does not always remove all the inconsistent values. When reaching GAC is too costly, propagators may only consider the bounds of the domains, and only those bounds are ensured to be consistent. This weaker property is called *bound-consistency* (BC). More generally, the efficiency of propagators is decomposed in different classes [Debruyne and Bessière, 2001]. Because there are several constraints in the problem, all the constraints are iteratively propagated until a *fixed point* is reached, *i.e.* all the domains are stable for all the propagators; see for example [Apt, 1999] for an overview of constraint propagation and questions relative to confluence.

It is obvious that, most of the time, propagation is not sufficient to solve the CSP. The domains can be stable, but still contain more than one value. A *search engine* then iteratively instantiates values to the variables, until either a solution or a failure (constraint always false, empty domains) is found. In case of a failure, a special mechanism, called *backtracking*, allows to return to the last choice point and try another value for the variable under consideration. Every time a choice is made (e.g. a particular value has been instantiated to a variable) propagation is run in order to remove the inconsistent values of the domains as soon as possible and avoid useless computation. Due to the combinatorial nature of the problems, this makes the worst-case number of calls to the propagators exponential in the number of variables.

7.1.2 Motivating Example.

We illustrate the way a constraint solver works on the following toy scheduling problem. Consider a construction site where six different tasks, each one lasting one day, have to be scheduled (eventually in parallel) within a given duration of 5 days. They are represented by their starting time, which is an integer value between 1 and 5. We have six variables $T_1...T_6$, with domains $D_1...D_6$ initially equal to $[1...5]$. The construction starts at time $S = 0$ and ends at $E = 6$.

Some tasks need to be finished before some other tasks begin (*e.g.* the walls need to be built before the roof). This is modeled by precedence constraints, which are inequalities between the two involved tasks. In addition, some tasks may be done in parallel, but others require the same equipments, so they cannot be executed at the same time (*e.g.* there is only one concrete mixer, required to build both the walls and the floor). This is modeled by `AllDifferent` constraints on the involved tasks.

This leads to a so-called scheduling problem with precedence constraints. We will detail the example shown on Figure 7.1 where the variable are the nodes, the precedence constraints the edges in red (C_1 to C_{10}) and the `AllDifferent` constraint the green ellipse around the variables it involves (C_{11}).

Let us work out in detail the initial propagation. First, D_4, D_5 and D_6 are reduced to $[2...5]$ by propagation of C_4, C_5, C_6 and C_7. Conversely, propagating these precedence constraints from the end yields $D_1 = D_2 = D_3 = [1...4]$. At this stage, the `AllDifferent` constraint C_{11} should be propagated as well but it cannot reduce the domains in practice.

Then the search begins, and a variable, say T_4, is assigned a value, say 3. The precedence constraint C_4 is propagated to reduce D_1 to $[1...2]$, but no propagation of C_{11} can be done: it can be checked that the remaining problem is still consistent. Thus a second choice is made, say T_5 is assigned to 4. Again,

[1] A problem may be consistent without having a solution. Consider for example the constraints $x \neq y$, $y \neq z$ and $z \neq x$ with the domains $D_x = D_y = D_z = \{1, 2\}$: we cannot suppress values in the domains; yet no solution exists.

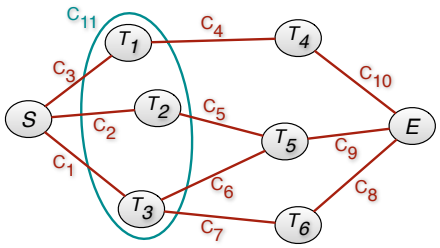


Figure 7.1: Example of a scheduling problem with precedence constraints and an `AllDifferent`constraint.

the remaining problem is already consistent and C_{11} is not propagated – although propagation of C_5 and C_6 reduces both D_2 and D_3 to $[1...3]$. So a third choice is made, say T_6 is assigned to 3. Now C_7 can be propagated and $D_3 = [1...2]$. No other propagation of precedence constraints can be done. Now the C_{11} constraint is not bound-consistent: $[1...2]$ is equal to the union of the two domains D_1 and D_3, and we have two values to allocate to two variables T_1 and T_3; this leads to reducing D_2 to $[4]$, hence assigning T_2.

This very simple example shows that the propagation of `AllDifferent` constraints may have an effect on the variables' domains only quite late in the search (here, after the third assignment on a problem with six variables), and is unlikely to have any effect on the domains when they are large enough, as is the case at the beginning of the search.

7.1.3 Cost of Propagation and Complexity Trade-offs.

The propagators have an algorithmic cost which, most of the time, is a sizeable part of the propagation engine computing time. Practical experiments highlight that the algorithmic effect (*i.e.*, reduction of the variables' domains) of the propagators is not uniformly distributed during the resolution process. In other words, these algorithms are frequently called when solving a hard combinatorial problem, but often do nothing. This phenomenon is rarely explored by the CP community, where most of the research efforts on propagation algorithms are focused on their worst-case time complexity; we refer the reader to [Rossi et al., 2006a] for a global reference and further studies.

In [Katriel, 2006], Katriel identifies one of the major issues in studying propagation: the pursuit of a fair balance between efficiency (time complexity) and effective performance (number of inconsistent values detected by the propagators). The author proposes a particular mechanism to decrease the number of calls to the propagators in the case of the global cardinality constraint during the search process. She shows that only subsets of the values in the variable domains are important for the propagation engine, and proposes to delay the propagation algorithm calls until a certain number of values are removed from the domains. The main limit of this work remains the algorithmic cost related to the detection of these important values, which is never amortized during the search process.

Obviously, observing the worst-case complexity of a propagator does not give enough information on its usefulness. Average time complexity is certainly very difficult to obtain and has never been studied. As shown by Katriel, the problem of freezing calls to useless propagators (a weak, yet interesting way of measuring the algorithm efficiency) is tricky for two reasons. Firstly, missing an inconsistent value leads to important needless computations afterwards. Secondly, in order to know whether a propagator is useful or not, we need to have an indicator that can be computed much faster than the evaluation of the propagator itself.

We propose here a theoretical study of the behavior for a specific propagator, the `AllDifferent` constraint. Given a set of variable domains, we provide a probabilistic indicator that allows us to predict if the `AllDifferent` propagator, for bound-consistency, will detect and remove some inconsistent part of these variable domains. We then show that such a prediction can be asymptotically estimated in constant

time, depending on some macroscopic quantities related to the variables and the domains. Experiments show that the precision is good enough for a practical use in constraint programming. Compared to [Katriel, 2006], we tackle on the same question but provide, within another model, a computable approximation for the efficiency of the algorithm, measured as the probability it does remove at least one value.

In the next Section we first give a formal definition of the bound consistency for the constraint `AllDifferent`, then consider how we can characterize situations where the constraint `AllDifferent` will not restrict any of the variables' domains and propagation should be avoided, and finally present a probabilistic model for variables and their domains. Section 7.3 presents exact and asymptotic formulae for the probability that the propagation of the constraint `AllDifferent` will have no effect on the domains. Finally, Section 9.5 considers applying our results to an actual solver, what we can hope to gain, and the problems that we face. Several propositions and theorems presented here require long and tedious proofs, in particular the two main theorems 7.3.2 and 7.3.3. The proofs are not detailed here, but they can be found in [du Boisberranger et al., 2011].

7.2 A Probabilistic Model for `AllDifferent`.

The `AllDifferent` constraint is a well-known global constraint for which many consistency algorithms have been proposed. The reader can refer to the surveys of Van Hoeve [van Hoeve, 2001] or Gent et al. [Gent et al., 2008] for a state of the art.

The property of being bound consistent (BC) applies to all global constraints. Intuitively, *a constraint on n variables is bound consistent if, assuming the domains of the n variables to be intervals, whenever we choose to affect any variable to either its minimal or maximal value, it is possible to find a global affectation of the remaining $n-1$ variables that satisfies the global constraint.* This intuition can be formalized, and we give below a mathematical characterization of bound consistency for `AllDifferent`. We next introduce a probabilistic model for Bound Consistency of `AllDifferent` and consider how we can check whether an `AllDifferent` constraint, initially BC, remains so after an instantiation.

7.2.1 Definitions and Notations.

Consider an `AllDifferent` constraint on n variables $V_1...V_n$, with respective domains $D_1...D_n$ of sizes d_i, $1 \leq i \leq n$. We assume the size of each domain to be greater than or equal to 2, otherwise the corresponding variable is already instanciated. We focus here on bound consistency, as proposed by [Puget, 1998], and we assume that *all the domains D_i are integer intervals*.

We now introduce some notations.

- The union of all the domains is $E = \bigcup_{1 \leq i \leq n} D_i$, of size denoted by m. Notice that, up to a relabelling of the values of the D_i, their union E can also be assumed to be an integer *interval* w.l.o.g.
- For a set I, we write $I \subset E$ as a shortcut for : $I \subset E$ and I is an integer interval.
- For an interval $I \subset E$, we write $\underline{I}$ its minimum bound and $\overline{I}$ its maximum bound; hence $I = [\underline{I}...\overline{I}]$.
- For a value $x \in E$, $pos(x)$ is the position of x within E. By convention $pos(\underline{E}) = 1$.

With these notations, we can now give the classical definition of bound-consistency for `AllDifferent`.

Definition Let `AllDifferent`$(V_1...V_n)$ be a constraint on n variables. It is *bound-consistent* iff for all $i \in [1..n]$, the two following statements hold:

- $\forall j \in [1..n], j \neq i, \exists v_j \in [\underline{D_j}...\overline{D_j}]$ s.t. `AllDifferent`$(v_1...v_{i-1}, \overline{D_i}, v_{i+1}...v_n)$;
- $\forall j \in [1..n], j \neq i, \exists v'_j \in [\underline{D_j}...\overline{D_j}]$ s.t. `AllDifferent`$(v'_1...v'_{i-1}, \underline{D_i}, v'_{i+1}...v'_n)$.

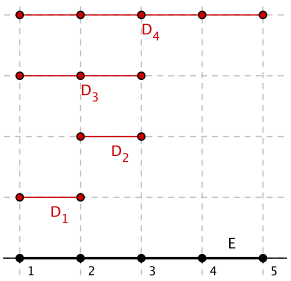


Figure 7.2: Unconsistent domain configuration for an `AllDifferent` constraint.

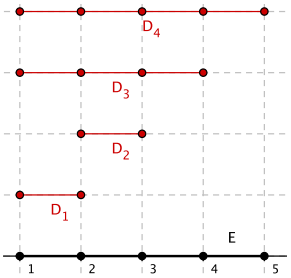


Figure 7.3: Consistent domain configuration for an `AllDifferent` constraint.

An example of non bound-consistent domains for an `AllDifferent` constraint with four variables is shown on Figure 2: the domains for $V_1...V_4$ are respectively $[1...2], [2...3], [1...3]$ and $[1...5]$, and it can be checked that the value 1, the lowest bound of D_4, cannot appear in a solution.

For the example of Figure 3, the domains $[1...2], [2...3], [1...4]$ and $[1...5]$ are bound-consistent, since all the extremal values of the domains can be extended to a solution.

Definition Let $I \subset E$. We define K_I as the set of variables for which the domains are subintervals of I: $K_I = \{i \text{ such that } D_i \subset I\}$.

For instance, on Figure 2 we have $K_{[1...3]} = \{1, 2, 3\}$, $K_{[3...5]} = \emptyset$ and $K_{[1...2]} = \{1\}$.

Some subintervals of E play a special rôle: they contain just enough values to ensure that every variable of K_I can be assigned a value. [2] Consequently the variables that do not belong to K_I cannot take their values in I. This leads to the following proposition, from [van Hoeve, 2001].

Proposition 7.2.1 *An `AllDifferent` constraint on a set of variables $V_1...V_n$ with a set of domains $D_1...D_n$ is* bound-consistent *if and only if the two following conditions are true:*

1. *for all $I \subset E$, $|K_I| \leq |I|$,*
2. *and for all $I \subset E$, $|K_I| = |I|$ implies $\forall i \notin K_I, I \cap \{\underline{D_i}, \overline{D_i}\} = \emptyset$.*

[2] The subintervals I of E s.t. $|K_I| = |I|$ are called *Hall intervals;* they frequently appear in characterizations of bound cosnsistency for `AllDifferent`.

For example, on Figure 2 the domain $D_3 = [1...3]$ is of size 3, and so is $K_{[1...3]}$; hence this interval satisfies the first condition of Prop. 7.2.1. But it does not satisfies the second condition, since the lowest bound of D_4, which is 1, has a non-empty intersection with $[1...3]$. On the contrary, on Figure 3 all the subintervals of E satisfy the proposition, since every $I \subset E$ is strictly bigger than the associated set K_I.

For technical reasons, we reformulate Proposition 7.2.1 into the equivalent

Proposition 7.2.2 *An* `AllDifferent` *constraint on a set of variables $V_1...V_n$ with a set of domains $D_1...D_n$ is* bound-consistent *if and only if for all $I \subset E$, one of the following condition is true:*

1. $|K_I| < |I|$,
2. $|K_I| = |I|$ *and* $\forall i \notin K_I, I \cap \{\underline{D_i}, \overline{D_i}\} = \emptyset$.

This property is useful to determine BC of an `AllDifferent` constraint as a whole, at the beginning of the resolution for instance. In practice, the problem (or rather the data) is constantly modified during the solving process. Thus we are also interested in answering the following question:

> *Knowing that an* `AllDifferent` *constraint is initially BC, under which conditions does it remain BC after the instanciation of a variable?*

7.2.2 Bound Consistency After an Instanciation.

We consider here the effect of an instantiation on the domains and on bound consistency. Up to a renaming of the variables, we can assume w.l.o.g. that the instantiation is done for the variable V_n, which is assigned a value $x \in D_n$. We also assume that the binary constraints `Neq` have been propagated (that is, the x value has been removed from the other domains when applicable).

After the instanciation, the situation is thus the following: D_n has disappeared (or is reduced to $\{x\}$), and for $i \neq n$, two cases can occur. If $x \notin D_i$, then the domain remains unchanged, and if $x \in D_i$, the domain D_i is now the union of the two disjoint intervals $[\underline{D_i}...x-1]$ and $[x+1...\overline{D_i}]$. The question of bound consistency thus seems no longer relevant, because the domains are no longer intervals. However, we can define new domains $D_i^{'}$=$D_i \setminus \{x\}$, which are not subintervals of E, but of

$$E^{'} := [\underline{E}...x-1] \cup [x+1...\overline{E}]$$

because all the values of $E^{'}$ between $\underline{D_i}$ and $\overline{D_i}$ are in $D_i^{'}$. This leads to the following

Definition The `AllDifferent` constraint *remains BC* after the instanciation of V_n iff `AllDifferent` is BC with respect to the new domains $D_1^{'}...D_{n-1}^{'}$.

Moreover, for every subinterval $I^{'} \subset E^{'}$, we define an associated interval $I \subset E$ as

$$I = \begin{cases} I^{'} \cup \{x\} \text{ if it is a subinterval of } E; \\ I \text{ otherwise.} \end{cases}$$

The following Proposition details in which cases the constraint, being BC on $V_1...V_n$, *remains* BC (as defined above) after the instantiation of V_n. This result, although not usually explicitly stated as such, belongs to the folklore of CP; we present it for the sake of completeness.

Proposition 7.2.3 *With the above notations, the* `AllDifferent` *constraint remains BC after the instanciation of V_n to a value x iff for all $I^{'} \subset E^{'}$, such that $I = I^{'} \cup \{x\}$ and $D_n \not\subset I$, none of the two following statement holds:*

(i) $|K_I| = |I|$,

(ii) $|K_I| = |I| - 1$ *and there exists $i \notin K_I$ such that $\underline{D_i} \in I$ or $\overline{D_i} \in I$.*

7.2.3 A Probabilistic Model for the Bound Consistency.

The key ingredients that determine the consistency of an `AllDifferent` constraint are the domain sizes and their relative positions. But we do not always have to know precisely the domains, to decide whether the constraint is consistent or not. For instance, an `AllDifferent` constraint on three variables with domains of size 3 is always BC, whatever the exact positions of the domains. If the domains are of size 2, then the constraint may be BC or not, depending on the relative positions of the domains. But if their union E is also of size 2, the constraint is always inconsistent.

Such basic remarks show that a partial knowledge on the domains sometimes suffices to determine consistency properties. This is the basis of our probabilistic model: we assume that we know the union E of the domains (which can indeed be observed in usual cases), but the domains themselves become discrete random variables with a uniform distribution on the set $\mathcal{I}(E)$ of subintervals of E, of size ≥ 2: they are not fully determined. Their exact sizes and positions are unknown; only some macroscopic quantities are known.

We consider first the union $E = \bigcup_{i=1}^{n} D_i$ of the domains, which is assumed to satisfy the following assumption:

A1. *E is a known integer interval of size m; w.l.o.g. we take $E = [1...m]$.*

Assume from now on that the domains $D_1, ..., D_n$ are replaced by random variables $\mathcal{D}_1, ..., \mathcal{D}_n$, as follows.

A2. *The variables $\mathcal{D}_i$ are independent and uniformly distributed on $\mathcal{I}(E) = \{[a...b], 1 \leq a < b \leq m\}$.*

As a consequence of Assumption **A2**, the sample space $\mathcal{I}(E)$ has size $m(m-1)/2$ (we recall that we forbid domains of size 1). For $J \subset E$ and $1 \leq i \leq n$, we have $\mathbb{P}[D_i = J] = 2/m(m-1)$. Indeed, there are $m-1$ possible subintervals of size 2, $m-2$ of size 3, ..., 1 of size m.

7.3 The Probability of Remaining BC.

This Section details the evaluation of the probability that an `AllDifferent` constraint remains BC after an instantiation, under the assumptions **A1** and **A2** of Section 7.2.3. We first establish a general formula for this probability, then compute its asymptotic value in the case where the observable variables are large; we also show that this asymptotic approximation can be computed in constant time.

7.3.1 Exact Results.

We first consider some intermediate probabilities that we need in order to write down the probability of remaining BC.

Proposition 7.3.1 *For a given interval $I \subset E$ and a domain $\mathcal{D}$ drawn with a uniform distribution on $\mathcal{I}(E)$, with $m = |E|$ and $l = |I|$, let p_l and q_l be the respective probabilities that $\mathcal{D} \subset I$, and that either $\mathcal{D} \cap I = \emptyset$, or $\underline{\mathcal{D}} < \underline{I} < \overline{I} < \overline{\mathcal{D}}$. Then*

$$p_l = \frac{l(l-1)}{m(m-1)}; \qquad q_l = \frac{(m-l)(m-l-1)}{m(m-1)}.$$

In order to compute the probability that the constraint remains BC, we can now inject into Proposition 7.2.3 the quantities we have just obtained, which leads to the following result.

Theorem 7.3.2 *Consider an* `AllDifferent` *constraint on variables $V_1, ..., V_n$, initially BC. Let E and the domains $\mathcal{D}_i$, $1 \leq i < n$, satisfy the assumptions* **A1** *and* **A2**. *Furthermore, assume that we know the domain $D_n = [a...b]$. Then the probability $P_{m,n,x,a,b}$ that the constraint remains BC after the instanciation of the variable V_n to a value x is*

$$P_{m,n,x,a,b} = \prod_{l=1}^{n-2} (1 - P^{(1)}_{m,n,l} - P^{(2)}_{m,n,l})^{\Phi(m,l,x,a,b)}$$

where $\Phi(m, l, x, a, b)$ is defined as

$$\min(x, m-l) - \max(1, x-l) + 1$$

if $l < b - a$ and as

$$\min(x, m-l) - \max(1, x-l) - \min(a, m-l) + \max(1, b-l)$$

otherwise, and where

$$\begin{aligned} P^{(1)}_{m,n,l} &= \binom{n-1}{l+1} p_{l+1}^{l+1} (1 - p_{l+1})^{n-l-2}, \\ P^{(2)}_{m,n,l} &= \binom{n-1}{l} p_{l+1}^{l} \left((1 - p_{l+1})^{n-l-1} - q_{l+1}^{n-l-1} \right), \end{aligned}$$

with p_l and q_l given by Proposition 7.3.1.

7.3.2 Asymptotical Approximation.

We recall that n is the number of variables and that the union of their domains has size m. From the expression of $P_{m,n,x,a,b}$ given in Theorem 7.3.2, we can compute the probability $P_{m,n,x,a,b}$ in time $O(n)$ (with a large constant). However, if we are to use a probabilistic indicator for the bound consistency as part of a solver, this indicator will be computed repeatedly, and we must be able to do this in a reasonably short time even when n and m are both large. Thus the formula of Theorem 7.3.2 cannot be used as such, and we need an approximation of it, both precise and that can be computed "quickly enough". The following proposition gives such an asymptotic approximation in a scale of powers of $1/m$. Two different behaviors arise, depending on how n compares to m. In the first case n is proportional to m, which corresponds to an `AllDifferent` constraint with many values and few variables. In the second case $m - n = o(m)$, which corresponds to a sharp `AllDifferent` constraint where there are nearly as many values as variables.

Theorem 7.3.3 *Consider an* `AllDifferent` *constraint on n variables $V_1, ..., V_n$. Assume that the domains $\mathcal{D}_1, ..., \mathcal{D}_{n-1}$ follow a uniform distribution on E. Let $D_n = [a...b]$. Define a function $\Psi(m, x, a, b)$ for $1 \leq a \leq x \leq b \leq m$ and $a \neq b$ by*

- $\Psi(m, x, a, a+1) = 1$, *except* $\Psi(m, 1, 1, 2) = \Psi(m, m, m-1, m) = 0$;
- *If $b > a + 1$ then $\Psi(m, x, a, b) = 2$, except $\Psi(m, 1, 1, b) = \Psi(m, m, a, m) = 1$.*

Then the probability $P_{m,n,x,a,b}$ that the constraint remains BC after the instanciation of V_n to the value x has asymptotic value

- *if $n = \rho m,\ \rho < 1$,*

$$1 - \Psi(m, x, a, b) \frac{2\rho(1 - e^{-4\rho})}{m} + O\left(\frac{1}{m^2}\right);$$

- *if* $n = m - i,\ i = o(m)$,

$$e^{C_i}\left(1 - \Psi(m, x, a, b)\frac{2(1-e^{-4}) + D_i}{m} + O\left(\frac{1}{m^2}\right)\right).$$

When $n = m - i$ *with* $i = o(m)$*, the constants* C_i *and* D_i *(which also depend on* x *and* a*) can be expressed as (* C_i **négatif?)**

$$\sum_{a-i \le j < x-i} (j + i + 1 - a)\varepsilon_{i,j} + (x - a)\sum_{j \ge x-i} \varepsilon_{i,j},$$

with $\varepsilon_{i,j}$ *equal to* $\log(1 - f_{i,j})$ *for* C_i *and to* $g_{i,j}/(1 - f_{i,j})$ *for* D_i*, where we set*

$$f_{i,j} = \lambda_{i,j}\left(1 + \frac{j}{2(i+j)}\right)$$

and

$$g_{i,j} = \lambda_{i,j}\left(i(i+1) + \frac{j}{4}(3i+5) + \frac{j(i^2-1)}{4(i+j)}\right)$$

with

$$\lambda(i, j) = \frac{(i+j)^j 2^j e^{-2(i+j)}}{j!}.$$

Theorem 7.3.3 is important for two reasons. In the first place, it gives the quickly-computable probabilistic indicator that we expected (see the discussion that follows in Section 7.3.3). Then, it also exhibits two different behaviors for an `AllDifferent` constraint. It thus formalizes and gives a rigorous proof of what is folklore knowledge in CP: the sharpness of the `AllDifferent` constraint (the ratio of n over m) is a key ingredient for the efficiency of its propagation. It can be seen from the expression of Theorem 7.3.3 that the probability of remaining BC has an asymptotic limit equal to 1 in the first case, and to a constant strictly smaller than 1 in the second case. Thus, for large m, propagation of `AllDifferent` is almost surely useless unless $m - n = o(m)$, that is, unless m and n are very close.

7.3.3 Practical Computation of the Indicator.

We have just seen that Theorem 7.3.3 gives an asymptotic approximation (when m becomes large) for the probability of remaining BC.

When $n = \rho\, m$, we have a closed-form expression for the approximation, that can be computed in constant (small) time. In the case $n = m - i$ the constants C_i and D_i, although not depending on m and n (they do depend on a and x), have no closed-form expressions but are given as limits of infinite sums. Nevertheless, a good approximation can be obtained with a finite number of terms. Indeed, the terms $f_{i,j}$ and $g_{i,j}$ are exponentially small for fixed i and $j \to +\infty$. E.g., the value $\log(1 - f_{i,j})$ is roughly of exponential order $(2/e)^j$, and so is the general term of the series: the convergence towards the limit is quick enough for fast computation.

Another practical question is: how do we choose between the two cases of the formula? that is, how do we decide when n is "close enough" to m? Theorem 7.3.3 is valid for $m \to \infty$, but splits into two cases according to whether n is such that m/n remains roughly constant, or $m - n = o(m)$.

A numerical example is shown on Fig. 7.4, where the theoretical probability (plain) and its approximation (dashed) are plotted for $m = 25$ and $m = 50$, and for a varying ratio n/m. (In both figures, the values x, a and b were arbitrarily fixed at 15, 3 and 9 respectively.) The dashed curve actually has a discontinuity at some point: we applied Case 1 for $n \le m - 2\sqrt{m}$ and Case 2 for $n > m - 2\sqrt{m}$, which appears from our computations to be the best compromise. Even though $m = 50$ is not a large value, the approximation already fits closely the actual values.

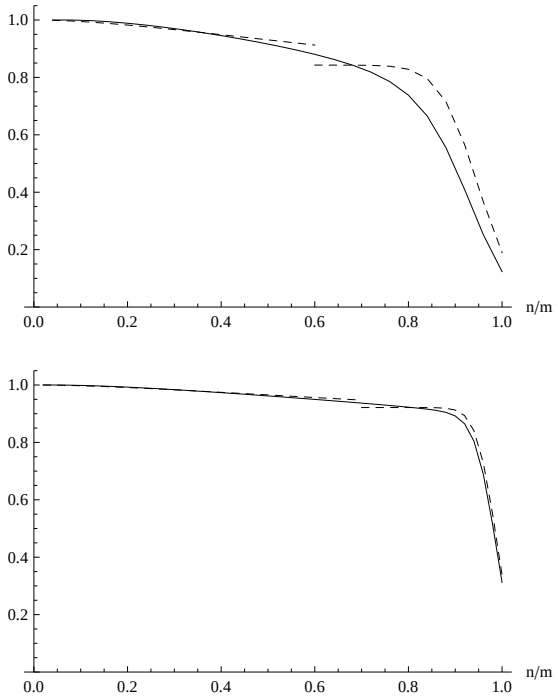


Figure 7.4: Numerical evaluation of the theoretical (plain) and approached (dashed) probability $P_{25,n,15,3,19}$ for $m = 25$ (top) or 50 (bottom), and for n varying from 1 to m.

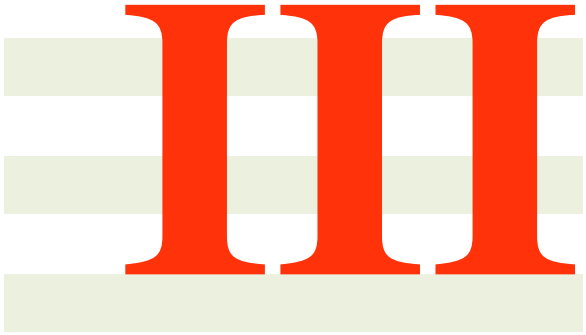
In practice, numerical evaluations for varying values of m and n do indicate that the best compromise is indeed to set a threshold at $n = m - 2\sqrt{m}$ and to use it to distinguish between the two cases of Theorem 7.3.3. Notice that this threshold can be used as a quantitative definition for the sharpness of the constraint, which leads us to propose the following

Definition An `AllDifferent` constraint on n variables and m values is *sharp* iff $m - n < 2\sqrt{m}$.

7.4 Conclusion

We have presented here a probabilistic framework for modeling the `AllDifferent` constraint and for checking whether it is worthwhile to propagate it. We have obtained the probability that the constraint does reduce the sizes of domains, and given an asymptotical formula that can be computed in constant time. Like all models, ours relies on a simplistic view of the reality, and this simplification is expressed by the mathematical hypotheses that have been used. A fundamental point is whether these hypotheses (independance of the domains, uniform distribution) are valid in real-life situations, and what is the robustness of our results if not.

This Chapter also concludes Part II. We have presented a series of work which all consist in finding mathematically founded explanations to some phenomena observed in CP solving processes (both complete and incomplete). I believe that the value of these works is not only the results obtained, but also the way we obtained it: in each case, the hypothesis or modeling choices that we had to take give interesting insights on the phenomenon under observation.



Abstract Domains for Constraint Programming

8

Synthesis Chapter: From Domains to Abstract Domains

This Chapter is a general introduction to the series of work at the crossing between Constraint Programming and Abstract Interpretation. First, I introduce the motivation for these works, then I put into perspective the notions of domains and consistencies and finally, I explain without any formalism how Abstract Domains, as introduced in Abstract Interpretation, can be used within the CP framework.

8.1 Domains in CP

In CP, the notion of domains is rarely given much thought, in particular when they are discrete. Domains are defined as the set of possible values for the variables, and classically, each variable has its own domain independently of the other variables. The attention given to the notion of domains in CP is mostly focussed on their implementation. On discrete variables, domains are finite subsets of the integers, which can either be represented as such (finite subsets of integers) or as finite integer intervals. It is worth mentioning that CP features two different consistencies which naturally arise in each case: if the domains are finite integer intervals, then the natural consistency property is called bound consistency. In this case, only the bounds of the domains are checked for consistency, and it is not allowed to create holes in the domains: this makes them compliant with the interval representation. For finite subsets of integers, the natural consistency is arc-consistency or generalized arc-consistency, which checks the consistency of every value of the domains.

On real variables, the domain representation is by nature an issue, since the true values are not computer-representable in general. The classical domain representation is real intervals with floating-point bounds, which are both computer representable and (approximately) computable thanks to interval arithmetic [Moore, 1966]. On such domains, the natural notion of consistency is hull-consistency, which is very much like bound consistency extended to the reals, except that it is commonly defined on several variables at a time: domains are hull-consistent for a constraint iff their cartesian product is the smallest box over-approximating the constraint solutions.

These examples show that domain representations are not only a matter of conveniently choosing how to implement the domains: they also condition the future computations that will be made on them.

8.2 Domains and consistencies from different perspectives

In this section, we focus on the relations between classical consistencies, and domain representations. We will give the definitions that are common in the litterature, taken from [Bessière, 2006], Chapter 3. Our goal is to show that the usual definitions may look quite different, but they all capture a very generic property that we will develop later. This property was already seen by [Apt, 1999], in the case of independent domains for the variables.

8.2.1 Integer domains

Let us begin with generalized arc-consistency on integer domains (also called arc-consistency when the constraints are binary). The following definition is that of [Bessière, 2006].

Definition Let $D_1, ..., D_n$ be domains of variables appearing in a constraint C. They are said ***generalized arc-consistent*** (GAC for short) iff $\forall i \in [1..n], \forall v \in D_i, \exists w_1 \in D_1, w_2 \in D_2, ...w_{i-1} \in D_{i-1}, w_{i+1} \in D_{i+1}, ...w_n \in D_n$ s.t. $C(w_1, w_2, ...w_{i-1}, v, w_{i+1}, ...w_n)$.

The w_js are call supports of value v. This definition is often presented like this: for each value in one of the domains, there exist values in the other domains such that the constraint holds. It is easier to understand the definition by expressing its negation: a domain where there are values with no supports is not consistent. In fact, this definition exactly states that each domain D_i is equal to the i-th projection of the solution set in $\mathbb{N}^n$.

Figure 8.1(b) shows the generalized arc-consistent domains for two variables, in the integer plane. The solutions of the constraint are in green, and the consistent domains are in dark blue. One can see on this figure that the arc-consistent domains are the smallest cartesian product of finite subsets of the integers, containing the solutions: this property is actually the core of consistency, as we will see later.

Another classical consistency on the integers is bound consistency, which is very close to GAC except that it only checks the consistency of the bounds of the domains.

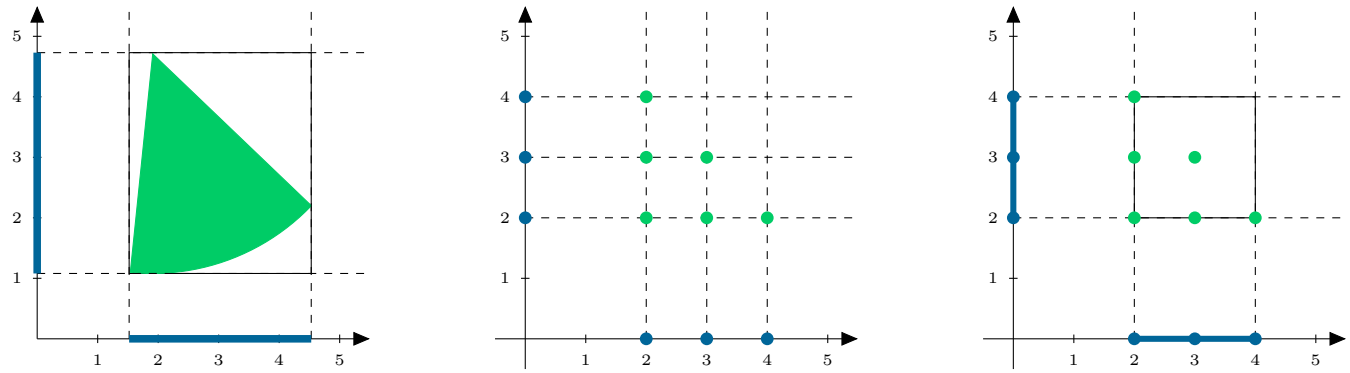
Definition Let $D_1, ..., D_n$ be domains of variables appearing in a constraint C. For each $i \in [1..n]$, let $\underline{D_i}$ the lower bound of D_i and $\overline{D_i}$ is upper bound. The domains are said ***bound consistent*** iff $\forall i \in [1..n], \forall v \in \{\overline{D_i}, \underline{D_i}\}, \exists w_1 \in D_1, w_2 \in D_2, ...w_{i-1} \in D_{i-1}, w_{i+1} \in D_{i+1}, ...w_n \in D_n$ s.t. $C(w_1, w_2, ...w_{i-1}, v, w_{i+1}, ...w_n)$.

This definition is usually given explicitely on $\underline{D_i}$ and $\overline{D_i}$, but we'd rather present it this way to show its similarities with the previous definition 8.2.1. Bound consistency is very close to GAC, except that it applies only to the bounds of the domains.

Figure 8.1(c) shows the bound-consistent domains, in dark blue, for the same constraint as previously. Here, the bound consistent domains are an integer box. Again, it is the smallest integer box containing the solutions.

To sum up, each of the following assertions expresses the fact that domains $D_1...D_n$ are arc-consistent (resp. bound consistent):

- for each $i \in [1..n]$, D_i does not contain inconsistent values (resp. have inconsistent bounds), *e.g.* values that cannot be extended to a solution,
- for each $i \in [1..n]$, the domain D_i is equal to the projection (resp. its bounds are the bounds of the projection) of the solutions on the i-th axis,
- the domain is the smallest cartesian product of integer sets (resp. integer box) containing the solutions.



(a) Hull consistency (b) Generalized arc-consistency (c) Integer bound consistency

Figure 8.1: Some classical consistencies over the integers or the reals

The first definition is the most commonly used to the best of our knowledge. For instance, it is the definition chosen in the *Handboof of Constraint Programming* [Rossi et al., 2006a], in Roman Barták's online tutorial which is a useful reference for beginners[1] and even on Wikipedia in the article about arc-consistency [2]. In the following, we will build another, more generic definition for domains, based on the third definition.

Remark The key remark for the following of this part is the following: for the two main consistencies, the consistent domains is the smallest *something* containing the solutions, with *something* being a given shape (cartesian products of finite integer subsets, or integer boxes). In both case, the set of all *something*s, ordered by inclusion, is a lattice, and closed by intersection. This is very convenient when we need to intersect consistent domains, which is exactly what propagation does. This remark will be developed later as it is crucial to understand what domains exactly are. Apart from the nature of the *something*s, both definitions are similar.

8.2.2 Real domains

The case of real domains is a little more complicated, because the real numbers are not computer representable. Traditionally, in CP, real numbers domains are encoded as real intervals with floating-point bounds. This makes in fact two abstractions in one step: first, a set of real number is represented as an interval containing it, possibly with real bounds, and then, this interval itself is approximated by a floating point interval containing the real interval. Note that the second abstraction (from an interval with real bounds to an interval with floating-point bounds) could be replaced by other abstractions, for instance in intervals with rational bounds.

When defining consistency, we will only consider the first abstraction, from real values to intervals with real bounds. We refer the reader to [Benhamou and Granvilliers, 2006] for a more detailed presentation. The *Handbook of Constraint Programming* that we take as a reference does not provide a definition for hull-consistency, but it is described this way: "Hull consistency is a complete approximation of arc consistency obtained by replacing the set defined in equation (16.1)[3] with the smallest enclosing interval, which is called the interval hull. The domain reduction rules combine interval arithmetic and constraint inversion steps" (from [Benhamou and Granvilliers, 2006]). From there, we can give the following definition.

Definition Let $D_1, ..., D_n$ be domains of variables appearing in a constraint C, with $D_i \subset \mathbb{R}$. They are said ***hull consistent*** iff $D_1 \times ... \times D_n$ is the smallest box containing the solutions.

[1] `http://ktiml.mff.cuni.cz/~bartak/constraints/consistent.html`
[2] `https://en.wikipedia.org/wiki/Local_consistency`
[3] This set is defined as the arc-consistent domain would be in the reals.

Figure 8.1(a) shows the hull-consistent domains for a real constraint, of which the solutions are in green. The consistent domains (in the reals) are in dark blue. They are the smallest real box containing the solutions.

Again, this definition is equivalent to state that each D_i is the i-th projection of the solution set in $\mathbb{R}^n$, or that the domains considered as a box are the smallest box containing the solutions. Since real boxes, ordered by inclusion, are also a lattice and closed by intersection, the remark above is also true, with *something* a cartesian product of real intervals.

8.2.3 Propagation

In the previous subsections, we have presented different definitions for the same notion, consistency. This presentation is not exhaustive, but focused on basic consistencies for one constraint. Other consistencies exist and are not described here, for instance path-consistency or strong consistencies. We refer the reader to [Rossi et al., 2006a] for a more detailed presentation.

As seen above and on Figure 8.1, these definitions may seem rather different, but they all capture the same property: the domains do not contain values which cannot appear in a solution, *i.e.* values which are obviously useless. In fact, these definitions are quite complicated because they are not really focused on what consistency is, but rather on what it is not. One can only guess why: in constraint solvers, consistency is used to propagate the constraints, this process being a key element in constraint solvers, and propagation consists, in practice, in removing inconsistent values.

In the following, for a constraint C on variables $x_1...x_n$, with domains $D_1...D_n$, we will call **the** consistent domains the domains $D_1^C...D_n^C$ which are consistent for C and such that $\text{Sol}_C(D_1...D_n) = \text{Sol}_C(D_1^C...D_n^C)$. A propagator for a constraint computes the consistent domains for this constraint.

Definition Let E a set, and $D_1...D_n$ subsets of E. Let C a constraint on variables $x_1...x_n$ with domains $D_1...D_n$. A propagator for C is a function $\rho_C : E^n \to E^n$ such that the sets $D_i^C = \pi_i(\rho_C(D_1...D_n))$ for $i \in [1..n]$ are consistent for C (based on one of the above definitions), and $\text{Sol}_C(D_1...D_n) = \text{Sol}_C(D_1^C...D_n^C)$.

Many definitions for propagators exist in the literature. On integer domains, a definition which is very close to ours can be found in [Schulte and Tack, 2009], which also exhibits properties that the propagator must satisfy: contraction, *i.e.* $\rho_C(d) \subset d$, idempotence, *i.e.* $\rho_C(\rho_C(d)) = \rho_C(d)$, and weak monotonicity, an *ad hoc* property to ensure correctness. On real domains, [Chabert and Jaulin, 2009] introduces a similar definition on real intervals, except that they require the propagators to be continuous in some sense. Our definition applies in the same way for integer consistencies and for real consistencies.

We will refine this definition later, however, it needs to be commented. Firstly, note that E can be the set of integers or of reals. Secondly, this definition conveniently presents a propagator as a single function in E^n, but in practice, it is often more a n-uplet of functions from E to E, since only the projections of the consistent domains are actually relevant in the existing consistencies (case of cartesian domains). The propagators in existing constraint solvers are often cartesian, *i.e.*, they compute the consistent domain for each variable independently, but in some sense, generating a cutting plane can be seen as a non-cartesian propagation. We will see later that consistency and propagation can also be generically defined in a non-cartesian way. Thirdly, this definition assumes that a propagator always compute the consistent domains. This is often true in practice, but for some constraints, propagation can be NP-hard (this is for instance the case for the `n-value` constraint[4]). It happens that they do not reach consistency, *i.e.*, they remove some useless values from the domains, but not all. Similarly, the HC4 algorithms which propagates continuous constraints also over-approximates the consistent domains [Benhamou et al., 1999]. Finally, propagators are what implement constraints in solvers. In fact, a constraint can be represented by a checker (checking solutions in the case of discrete domains), and a propagator. For instance, the IBEX solver even replaces the notion of constraint by the notion of propagators [Chabert and Jaulin, 2009].

[4] http://web.emn.fr/x-info/sdemasse/gccat/Cnvalue.html

Propagators are meant to eliminate inconsistent values from the domains. Obviously, it is not sufficient to call them independently for each constraint, since each domain modification due to one constraint may influence the consistency of the other constraints. Intuitively, there is a fixpoint notion in the propagation. The propagators are called iteratively until the domains cannot be modified, *e.g.,* they are consistent for the constraints as a whole. The consistent domains for all the constraints are the fixpoint of all the propagators. In particular cases, such as binary constraint networks, many algorithms have been proposed to efficiently reach this fixpoint: AC3 maintains a list of domains where values have been removed and applies constraint propagation until this list is empty. AC6 keeps track of the supported values in the domains to avoid recomputing them, and AC2001 does the same, only more efficiently: it uses an order on the values and keeps the minimal support [Bessière and Régin, 2001]. But in general, tuning the propagation loop is a difficult task. For instance, the Choco solver features a domain specific language to tune propagation engines [Prud'Homme et al., 2014].

Theoretically, the article [Apt, 1999] presents a formal definition of propagation as chaotic iterations, as defined in Abstract Interpretation [Cousot and Cousot, 1977b].

8.2.4 Limitations of the Existing Consistencies

Is the notion of consistency, as defined in the literature and recalled above, satisfying? Obviously, it is, at least for writing efficient constraint solvers, since efficient constraint solvers do exist. But maybe, the fact that these definitions are operational and easy to translate into propagators is not enough. In particular, they loose the strong connection which often exist between the domain representation, and the notion of consistency that this representation naturally derives.

The first limitation is obvious: each consistency is made for its own variable type and domain representation. Consider a constraint on variables in both integer and real domains. What would the consistency look like? This is not obvious, as the different definitions on discrete and continuous domain are not unified. In practice, mixing integer and real variables is often done by linking a real solver with an integer one. This is for instance the case with the Choco solver, which can deal with real constraints by calling the Ibex solver [Fages et al., 2014]. On the other hand, it is also possible to add integrity constraints into a real solver [Berger, 2010]. None of these solutions are really satisfying: in practice, one of the variable types is dismissed from the search process. For instance, adding integrity constraints into a real solver makes it impossible to call propagators which are specific to the integers, such as many global constraint propagators.

The second limitation is maybe less obvious, yet equally important. In CP, the domains are always considered independently for each variable. This naturally makes the abstract domains cartesian. But in other research areas, such as Linear Programming, relational representations, such as polyhedra, naturally appear. Hence, a question worth being asked is: is it possible to use relational domains in CP? The traditional consistency definitions make this question difficult to answer.

8.3 From Domains to Abstract Domains

As seen above, a domain in Constraint Programming is always attached to one variable, and has a restricted number of possible representations. One of my contributions was to define a more generic notion of domains for CP, very much inspired by Abstract Domains in Abstract Interpretation. Abstract Interpretation is a domain of semantic where program properties are proven by examining over-approximations of the program traces. Those over-approximations, called abstract domains, are defined within lattice theory, and fixpoint theorems are at the core of the soundness of Abstract Interpretation. This makes abstract domains a natural candidate for a generic expression of CP domains.

8.3.1 Avoiding Confusions

The rest of this chapter consists in introducing a notion of abstract domain, inspired by abstract domains as defined in Abstract Interpretation. Before doing this, to avoid confusions, the vocabulary should be clarified, as some words have different meanings in CP and in Abstract Interpretation:

- in the CP community, AI usually means Artificial Intelligence. To avoid confusions, we write AbsInt for Abstract Interpretation (although this is the name of a tool - but not in the CP community).
- in Abstract Interpretation, an abstract domain is a set of domains, while in CP, a domain is just a set of values (which would be called an abstract element, *i.e.* a member of an abstract domain in AbsInt). In the following, we will as much as possible keep the AbsInt vocabulary.
- in CP, a sound solver under-estimates the solution set, and a complete solver over-estimates it (*e.g.,* it does not lose solutions). In AbsInt, a sound interpreter over-estimates a set of traces. As much as possible, we will use the words "over/under-approximation" instead of "sound/complete" to avoid confusions.

8.3.2 Why Stealing Abstract Domains from Abstract Interpretation?

As shown in the previous section, for several consistencies (which are probably the most classical ones) the definitions can be reformulated to be very similar. In fact, the only difference is the type of *something* that we use to over-approximate the solutions: integer boxes, real boxes or cartesian products of finite integer subsets. This *something* is a shape in which we try to enclose the solutions, as shown in the unified definitions of consistency given above. This shape can be more or less expressive and more or less costly to compute. In the following, this *something* will be called *Abstract Domain*.

Obviously, the set of *something*/shapes, or abstract domain, cannot be anything. It needs to have an easy to manipulate computer representation. In addition, we would rather have abstract domains that are closed by intersection, so that propagating the different constraints reaches a unique fixpoint (though this may not be mandatory, this is true for the existing domains). From [Apt, 1999], if the abstract domain is a complete lattice, then the propagation loop will behaves well (convergence toward a unique fixpoint). And finally, abstract domains must feed the propagators.

In fact, abstract domains already exist in another research area, Abstract Interpretation, where they are used to represent traces of programs. Static analyzers require the abstract domains to over-approximate the traces at every program point, thus, they come with several operators such as meet and join which mimic intersection and union, transfer functions which mimic the effect of the program instructions, etc. Many abstract domains have already been introduced, such as signs, boxes, octagons, polyhedra, BDDs, etc.

Although the theory in CP and the theory in AbsInt may be quite different, in practice, they have several points in common. In both cases, the problem is to compute some set of interest: solutions in CP, traces in AbsInt. This set is impossible to compute exactly: too costly to be computed directly in CP (often NP-hard), and even undecidable in the general case in AbsInt. In both cases, this set is over-approximated in a given shape, with an easy computer representation (boxes for instance), and for which the computations are easy enough. Finally, in both cases, there is a threshold between the precision obtained with the abstract domain and its computational cost.

8.3.3 Abstract Domains for CP

Hence, a natural question arises: given a CP problem, is it possible to write a computer program, so that the traces of the program are the solutions of the problem, and a static analyzer using AbsInt techniques would compute the solutions? In fact, the answer is no, to the best of our knowledge. The notion of precision that exists in CP does not exist in AbsInt, where abstract domains are usually designed for fast computations. When the analysis of some programs requires to improve the precision of the domains, the AbsInt point

of view is, in general, that the behavior of these programs cannot be captured by the domain (for instance, loops containing angle computations cannot be captured by boxes) and a new abstract domain is designed (in the previous example, the ellipsoids).

In order to use AbsInt abstract domains in CP, one must define at least two new operators: a choice operator, in order to cut the abstract elements in several parts as the solvers do at each node, and a precision operator, to check for the termination of the solving process ; it is straightforward on the integers (singleton check) but not on the reals. Once this is done, we can define an abstract solving process which is exactly a constraint solver, from which the domains have been abstracted in the same way as in AbsInt. This process is described in the next Chapter 9.

This abstract solving methods comes with great advantages: first, it is possible to use it with any abstract domain, not only cartesian ones. In fact, several relational domains, offering a trade-off between precision and computation cost, have been introduced in AbsInt. In some cases, the domains directly express a sublanguage of constraints and propagation is optimal in one pass: for instance, the polyhedra domain is well suited to propagate linear constraints. In other cases, propagation and choice must be redefined ; then, propagation can been seen as constraint inference (inferring the constraints of the abstract domains). In Chapter 10, we will detail an example of such a relational domain, the octagons, and show how they can be used in CP.

Second, the abstract solver can be used with any abstract domains, in particular with Reduced Product domains which combine two (or more) existing domains. Reduced Products can communicate informations (such as domain reductions) from one to another of their base domains. In practice, this means that we can solve, without losing any property, in both real and integer domains, each one transmitting the informations collected by propagation to the other one. Two examples of Reduced Products are given in Chapter 11: integer and real boxes, and boxes and polyhedra.

9

Abstract Domains for Constraint Programing

This Chapter is based on Marie Pelleau, Antoine Miné, Charlotte Truchet, Frédéric Benhamou, *A Constraint Solver Based on Abstract Domains*, VMCAI 2013: 434-454. This article is the root of the development of the *AbSolute* solver, which uses an abstract domain library to solve constraints, although it has been published after our article on Octagons - in fact, this article extends a previous article published in the Proceedings of the Synasc 2010 conference, but with a more formal presentation in an Abstract Interpretation style.

I strongly believe that abstract domains offers a perfect framework for elegantly doing several operations which are very useful in CP solving, but usually left to hacking-level techniques: mixing integers and reals, using different consistencies or solvers for different constraints, and probably more. Having a well-defined framework allows us to mix different solving techniques while keeping formal properties on each. My contribution in this work was to define, with Marie Pelleau, the very first model of abstract domains, which we developed later in a more Abstract Interpretation style with Antoine Miné.

9.1 Stealing Abstract Domains from Abstract Interpretation

Abstract Interpretation (AI) is a research area in Semantic which, by many aspects, is really far from CP. It is used to design static program analyzers that are sound and always terminate (such as Astrée [Bertrane et al., 2010]) by developing computable approximations of essentially undecidable problems. The (uncomputable) concrete collecting semantics expresses in fixpoint form the set of observable behaviors of the program. It is approximated in an abstract domain that restricts the expressiveness to a set of properties of interest, provides data-structure representations, efficient algorithms to compute abstract versions of concrete operators, and acceleration operators to approximate fixpoints in finite time. Soundness guarantees that the analyzer observes a super-set of the program behaviors. Numeric domains, focusing on numeric variables and properties, are particularly well developed; major ones include intervals [Cousot and Cousot, 1977a] and polyhedra [Cousot and Halbwachs, 1978], and recent years have seen the development of new domains, such as octagons [Miné, 2006a], and libraries, such as Apron [Jeannet and Miné, 2009]. They can handle all kinds of numeric variables, including mathematical integers, rationals, and reals, machine integers and floating-point numbers, and even express relationships between variables of different types [Miné, 2004, Bertrane et al., 2010]. Each domain corresponds to some trade-off between cost and preci-

sion. Finally, domains can be modified and combined by generic operators, such as disjunctive completions and reduced products.

9.1.1 Contribution.

In the rest of this chapter, we seek to use AI techniques to build an abstract, generic CP solver. Our contributions are as follows: we show the links between AI and CP and recast the later as a fixpoint computation similar to local iterations in a disjunctive completion of non-relational domains; we design a generic abstract solver parametrized by abstract domains and prove its termination; we show that, by using relational and mixed integer-real abstract domains, we can go beyond some limitations of existing solvers. We do not study in this paper the dual problem, *i.e.,* exploiting CP techniques in AI; it is one of the perspectives of this work.

9.1.2 Related works.

Some interactions between CP and verification techniques have been explored in previous works. For instance, CP has been used to automatically generate test configurations [Hervieu et al., 2011], or to verify CP models [Lazaar et al., 2012]. In another direction, several recent works, such as [D'Silva et al., 2012, Thakur and Reps, 2012], establish connections between AI and SAT solving algorithms, holding promise for cross-pollination between these fields. Our aim is similar, but linking AI to CP. While related, CP and SAT differ significantly enough in the chosen models (numeric versus boolean) and solving algorithms that previous results do not apply. Our work is in the continuity of [Truchet et al., 2010] that extends CP solving methods to use richer domain representations, such as octagons. However, embedding a new domain required *ad hoc* techniques to express its operations in the native language of the solver: boxes. In this chapter, we reverse that process: we redesign from the ground up the solver in an abstract way so that it is not tied to boxes but can reuse as-is existing abstract operators and domains from AI.

The work which is probably the closer to ours is the PhD of Joseph Scott [Scott, 2016], or his recent article [Scott, 2017]. He also builds a framework to clean and generalize the notions of consistencies and propagations, and this framework is also built upon lattice theory. He also introduces the notion of concrete domain (the set of values within the domains) and the notion of abstract domain (a lattice representing an abstraction of the concrete domains). But instead of defining abstract domains as based on a given lattice, he defines propagation based on a Galois connection between the concrete and abstract domains. I believe that it is a strong restriction, as it eliminates potentially interesting abstract domains, such as polyedra, for which there is no Galois connection with the boxes. However, this very elegant definition coincides with ours in case there is a Galois connection.

9.2 Preliminaries

In this section we present some notions of Abstract Interpretation and Constraint Programming that will be needed later.

9.2.1 Elements of Abstract Interpretation

We first present some elements of Abstract Interpretation that will prove useful in the design of our solver (see [Cousot and Cousot, 1992, Cousot and Cousot, 1977a] for a more detailed presentation).

Fix-point abstractions.

The concrete semantics of a program is given as the least fixpoint $\mathrm{lfp}_{\bot} F$ of an operator $F : \mathcal{D} \to \mathcal{D}$ in some partially ordered structure $(\mathcal{D}, \sqsubseteq, \bot, \sqcup)$, such as a complete partial order or a lattice. With suitable

hypotheses [Cousot and Cousot, 1992] on F and $\mathcal{D}$, the fixpoint can be expressed as the limit of a (possibly transfinite) increasing iteration $\mathrm{lfp}_\bot\, F = \bigsqcup_{i\in\mathrm{Ord}} F^i(\bot)$ on ordinals.

Similarly, we denote by $(\mathcal{D}^\sharp, \sqsubseteq^\sharp, \bot^\sharp, \sqcup^\sharp)$ the abstract domain. A monotonic concretization $\gamma : \mathcal{D}^\sharp \to \mathcal{D}$ associates a concrete meaning to each abstract element. An abstract operator $F^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ is a sound abstraction of F if $F \circ \gamma \sqsubseteq \gamma \circ F^\sharp$. Sometimes, but not always, there exists an abstraction function $\alpha : \mathcal{D} \to \mathcal{D}^\sharp$ such that (α, γ) forms a Galois connection, which ensures that each concrete element X has a best abstraction $\alpha(X)$, and the optimal abstract operator $F^\sharp$ can be uniquely defined as $F^\sharp = \alpha \circ F \circ \gamma$. In all cases, $\mathrm{lfp}_\bot\, F$ can be approximated as $\bigsqcup^\sharp_{i\in\mathrm{Ord}} F^{\sharp i}(\bot^\sharp)$. This limit may not be computable, even if $F^\sharp$ is, or may require many iterations. It is thus often replaced with the limit of an increasing sequence: $X^\sharp_0 = \bot^\sharp$, $X^\sharp_{i+1} = X^\sharp_i \triangledown F^\sharp(X^\sharp_i)$ using a widening operator $\triangledown$ to accelerate convergence. The widening is designed to over-approximate $\sqcup$ and converge in finite time δ to a post-fixpoint $X^\sharp_\delta$ of $F^\sharp$. Then, $\gamma(X^\sharp_\delta) \sqsupseteq \mathrm{lfp}_\bot\, F$. The limit is often refined by a decreasing iteration: $Y^\sharp_0 = X^\sharp_\delta$, $Y^\sharp_{i+1} = Y^\sharp_i \vartriangle F^\sharp(Y^\sharp_i)$, using a narrowing operator $\vartriangle$ designed to stay above any fixpoint of F and converge in finite time. As all the $Y^\sharp_i$ are abstractions of $\mathrm{lfp}_\bot\, F$, we can stop the iteration at any time.

Local iterations.

In addition to refining the results of least fixpoint computations, decreasing iterations have been used by Granger [Granger, 1992] locally, *i.e.,* within the computation of $F^\sharp$. Granger observes that the concrete operator F often involves lower closure operators, *i.e.,* operators ρ that are monotonic, idempotent ($\rho \circ \rho = \rho$) and reductive ($\rho(X) \sqsubseteq X$). Given any sound abstraction $\rho^\sharp$ of ρ, the limit $Y^\sharp_\delta$ of the sequence $Y^\sharp_0 = X^\sharp$, $Y^\sharp_{i+1} = Y^\sharp_i \vartriangle \rho^\sharp(Y^\sharp_i)$ is an abstraction of $\rho(\gamma(X^\sharp))$. Whenever $\rho^\sharp$ is not an optimal abstraction of ρ, $Y^\sharp_\delta$ may be significantly more precise than $\rho^\sharp(X^\sharp)$. A relevant application is the analysis of complex test conjunctions $C_1 \wedge \cdots \wedge C_p$ where each atomic test C_i is modeled in the abstract as $\rho^\sharp_i$. Generally, $\rho^\sharp = \rho^\sharp_1 \circ \cdots \circ \rho^\sharp_p$ is not optimal, even when each $\rho^\sharp_i$ is. A complementary application is the analysis of a single test C_i using a sequence of relaxed, non-optimal test abstractions. For instance, non-linear expression parts may be replaced with intervals computed based on variable bounds [Miné, 2004]. As applying the relaxed test refines these bounds, the relaxation is not idempotent and benefits from local iterations. The link between local iterations and least fixpoint refinements lies in the observation that $\rho(X)$ computes a trivial fixpoint: the greatest fixpoint of ρ smaller than X: $\mathrm{gfp}_X\, \rho$. In both cases, a decreasing iteration starts from an abstraction of a fixpoint ($\mathrm{lfp}_\bot\, F$ in one case, $\mathrm{gfp}_X\, \rho$ in the other) and computes a smaller abstraction of that fixpoint.

On narrowings.

While a lot of work has been devoted to designing smart widenings, narrowings have gathered far less attention. Some major domains, such as polyhedra [Cousot and Halbwachs, 1978], do not feature any. This may be explained by three facts: firstly, narrowings (unlike widenings) are not necessary to achieve soundness; secondly, performing a bounded number of decreasing iterations without narrowing is sometimes sufficient to recover enough precision after widening [Bertrane et al., 2010]; thirdly, when this simple technique is not sufficient, narrowings do not actually help further in practice and solutions beyond decreasing iterations must be considered [Halbwachs and Henry, 2012]. In the following, we argue that Constraint Programming can be seen as a form of decreasing iteration, but uses different techniques that are, in some respects, more advanced than the corresponding ones used in Abstract Interpretation.

9.2.2 Constraint Programming

We now recall the basic definitions of Constraint Programming (see [Rossi et al., 2006b] for a more detailed presentation). In this section, we employ CP terminology, and take special care to point out terms with a different meaning in AI and CP.

Problems are modeled in a specific format, called Constraint Satisfaction Problem (CSP), and defined as follows:

Constraint Satisfaction Problem A CSP is defined by a set of variables $(v_1, \ldots, v_n)$ taking their value in domains $(\hat{D}_1, \ldots, \hat{D}_n)$ and a set of constraints $(C_1, \ldots, C_p)$ that are relations on the variables.

A *domain* D_i in CP denotes the set of possible values for a variable v_i and $D = D_1 \times \cdots \times D_n$ is called the *search space*. As the search space evolves during the solving process, we distinguish the initial search space of the CSP and note it $\hat{D} = \hat{D}_1 \times \cdots \times \hat{D}_n$ as in Def. 9.2.2. Problems may be discrete ($\hat{D} \subseteq \mathbb{Z}^n$) or continuous ($\hat{D} \subseteq \mathbb{R}^n$). Domains are, however, always bounded.

Given a constraint C on variables $v_1, \ldots, v_n$ in domains $D_1, \ldots, D_n$, and given values $x_i \in D_i$, we denote by $C(x_1, \ldots, x_n)$ the fact that the constraint is satified when each variable v_i takes the value x_i. The set of solutions is $S = \{(s_1, \ldots, s_n) \in \hat{D} \,|\, \forall i \in [\![1, p]\!],\, C_i(s_1, \ldots, s_n)\}$, with p the number of constraints and where $[\![a, b]\!] = \{x \in \mathbb{Z} \,|\, a \leq x \leq b\}$ denotes the interval of integers between a and b.

For discrete problems, two domain representations are traditionally used: subsets and intervals.

Integer Cartesian Product Let $v_1, \ldots, v_n$ be variables over finite discrete domains $\hat{D}_1, \ldots, \hat{D}_n$. We call integer Cartesian product any Cartesian product of integer sets in $\hat{D}$. Integer Cartesian products form a finite lattice:

$$\mathcal{S}^\sharp = \{ \prod_i X_i \,|\, \forall i,\, X_i \subseteq \hat{D}_i \}$$

Integer Box Let $v_1, \ldots, v_n$ be variables over finite discrete domains $\hat{D}_1, \ldots, \hat{D}_n$. We call integer box a Cartesian product of integer intervals in $\hat{D}$. Integer boxes form a finite lattice:

$$\mathcal{I}^\sharp = \{ \prod_i [\![a_i, b_i]\!] \,|\, \forall i,\, [\![a_i, b_i]\!] \subseteq \hat{D}_i,\, a_i \leq b_i \} \cup \{\emptyset\}$$

For continuous problems, domains are represented as intervals with floating-point bounds. Let $\mathbb{F}$ be the set of floating-point machine numbers. Given $a, b \in \mathbb{F}$, we note $[a, b] = \{x \in \mathbb{R} \,|\, a \leq x \leq b\}$ the interval of reals bounded by a and b, and $\mathbb{I} = \{[a, b] \,|\, a, b \in \mathbb{F}\}$ the set of such intervals.

Box Let $v_1, \ldots, v_n$ be variables over bounded continuous domains $\hat{D}_1, \ldots, \hat{D}_n \in \mathbb{I}$. A box is a Cartesian product of intervals in $\hat{D}$. Boxes form a finite lattice:

$$\mathcal{B}^\sharp = \{\prod_i I_i \,|\, \forall i,\, I_i \in \mathbb{I}, I_i \subseteq \hat{D}_i\} \cup \{\emptyset\}$$

Solving a CSP means computing exactly or approximating its solution set S.

Approximation A *complete* (resp. *sound*) approximation of the solution S is a collection $\mathcal{A}$ of domain sequences such that $\forall (D_1, \ldots, D_n) \in \mathcal{A}$, $\forall i,\, D_i \subseteq \hat{D}_i$ and $S \subseteq \bigcup_{(D_1,\ldots,D_n)\in\mathcal{A}} D_1 \times \cdots \times D_n$ (resp. $\bigcup_{(D_1,\ldots,D_n)\in\mathcal{A}} D_1 \times \cdots \times D_n \subseteq S$).

Soundness guarantees that we find only solutions, while completeness guarantees that no solution is lost. On discrete domains, constraint solvers are expected to be sound and complete, *i.e.,* compute the exact set of solutions. This is generally impossible on continuous domains, and we usually withdraw either soundness (most of the time) or completeness. Note that the terms *sound* and *complete* have opposing definitions in AI and CP so, to avoid confusion, we will use the term *over-approximations* (resp. *under-approximations*) to denote CP-complete AI-sound (resp. CP-sound AI-complete) approximations.

In this article, we consider solving methods that over-approximate the solutions of continuous problems and compute the exact solutions of discrete ones. These methods alternate two steps: propagation and search.

Propagation.

The goal of a propagation algorithm is to use the constraints to reduce the domains. Intuitively, we remove inconsistent values from domains, *i.e.,* values that cannot appear in any solution. Several definitions of consistency have been proposed in the literature. We present the most common ones.

Generalized Arc-Consistency Given variables $v_1, \ldots, v_n$ over finite discrete domains $D_1, \ldots, D_n$, $D_i \subseteq \hat{D}_i$, the domains are said *generalized arc-consistent* (GAC) for a constraint C iff $\forall i \in [\![1, n]\!], \forall x_i \in D_i, \forall j \neq i, \exists x_j \in D_j$ such that $C(x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n)$ holds.

Bound-Consistency Given variables $v_1, \ldots, v_n$ over finite discrete domains $D_1, \ldots, D_n$, $D_i \subseteq \hat{D}_i$, the domains are said *bound-consistent* (BC) for a constraint C iff $\forall i \in [\![1, n]\!]$, D_i is an integer interval $[\![a_i, b_i]\!]$, and the condition of Def. 9.2.2 holds for $x_i = a_i$ and $x_i = b_i$ (but not necessarily other values of x_i in $[\![a_i, b_i]\!]$).

Hull-Consistency Given variables $v_1, \ldots, v_n$ over continuous interval domains $D_1, \ldots, D_n \in \mathbb{I}$, $D_i \subseteq \hat{D}_i$, the domains are said *Hull-consistent* for a constraint C iff $D_1 \times \cdots \times D_n$ is the smallest floating-point box containing all the solutions for C in $D_1 \times \cdots \times D_n$.

Each constraint kind and consistency comes with an algorithm, called *propagator*, that tries to achieve consistency. When considering several constraints, a *propagation loop* iterates the constraint propagators until a fixpoint is reached. As shown in [Benhamou, 1996], the order of propagator applications does not matter, as the set of domains lives in a finite lattice ($\mathcal{B}^\sharp$, $\mathcal{I}^\sharp$ or $\mathcal{S}^\sharp$) and the consistent fixpoint is its unique least element. When consistency is too costly to achieve, the propagators and propagator loops settle instead for an over-approximation (*e.g.,* removing only some inconsistent values). In addition to providing a tighter search space, the propagation is sometimes able to discover that it contains no solution at all, or that all its points are solutions.

Search.

Generally, propagation alone cannot compute the exact solution (in the discrete case) or a precise enough over-approximation (in the continuous case). Thus, in a second step, a *search engine* is employed to try various assumptions on variable values. In the discrete case, a chosen variable is instantiated to each value in its domain. In the continuous case, its domain is split into two smaller subdomains. The solving algorithm continues by selecting a new search space and applying a propagation step (as it may no longer be consistent), and possibly making further choices. This interleaving of propagations and choices terminates when the search space can be proved to contain no solution, only solutions or, in the continuous case, when its size is below a user-specified threshold. In the discrete case, at worst, all the variables are instantiated. After exploring a branch, in case of failure or if all the solutions should be computed, the algorithm returns to a choice point (instantiation or split) by *backtracking* and tries another assumption.

We illustrate the search algorithm by an example solver in Fig. 9.1 corresponding to a continuous solver based on Hull-Consistency (Def. 9.2.2) computing an over-approximation of all the solutions. As explained above, a discrete solver would differ significantly. Existing solutions to embed discrete variables in continuous solvers consist in adding constraints expressing integerness and their propagators [Chabert et al., 2009, Berger and Granvilliers, 2009], while keeping a search engine based on continuous domains.

9.2.3 Comparing Abstract Interpretation and Constraint Programming

We now present informally some connections between Abstract Interpretation and Constraint Programming. The next section will make these connections formal by expression CP in the AI framework.

Both techniques are grounded in the theory of fixpoints in lattices. They pursue similar goals and means: computing or over-approximating solutions to complex equations by manipulating abstracted views

```
list of boxes sols ← ∅                                  ▷ stores the solutions
queue of boxes toExplore ← ∅                            ▷ stores the boxes to explore
push D̂ in toExplore                                     ▷ initialization with CSP search space

while toExplore ≠ ∅ do
    b ← pop(toExplore)
    b ← Hull-consistency(b)
    if b ≠ ∅ then
        if b contains only solutions or b is small enough then
            sols ← sols ∪ b
        else
            split b into b1 and b2 by cutting in half along the largest box dimension
            push b1 and b2 in toExplore
        end if
    end if
end while
```

Figure 9.1: A classic continuous solver.

of potential solution sets, such as boxes (called domains in CP, and abstract domain elements in AI). Their goals, however, do not coincide. Solvers aim at completeness and thus always allow refinement up to an arbitrary precision. On the contrary, the precision of abstract interpreters is fixed by their choice of abstract domains; they can seldom represent arbitrary precise over-approximations. AI embraces incompleteness. The choice of abstract domains sets the cost and precision of an interpreter, while the choice of domains sets the cost of a solver to reach a given precision.

Although they aim at completeness, solvers nevertheless employ simple, non-relational domains. They rely on collections of simple domains (similar to disjunctive completions) to reach the desired precision. The domains are homogeneous and cannot mix variables of different type. On the contrary, AI enjoys a rich collection of abstract domains, including relational and heterogeneous ones.

On the algorithmic side, AI and CP share common ideas. Iterated propagations in CP are similar to local iterations in AI. In fact, approximating consistency in CP is similar to approximating the effect of a complex test in AI. However, search engines in CP use features, such as choice points and backtracking, that have no equivalent in AI. Dually, the widening from AI has no equivalent in CP, as CP does not employ increasing iterations but only decreasing ones.

Finally, while abstract interpreters are usually defined in a very generic way and parametrized by arbitrary abstract domains, solvers are far less flexible and embed choices of abstractions (such as domains and consistencies) as well as concrete semantics (the type of variables) in their design. In the following, we will design an abstract solver that avoids these pitfalls and can benefit from the large library of abstract domains designed for AI.

9.3 An Abstract Constraint Solver

We now present our main contribution: expressing constraint solving as an abstract interpreter, which involves defining concrete and abstract domains, abstract operators for split and consistency, and an iteration scheme.

9.3.1 Concrete Solving

A CSP is similar to the analysis of a conjunction of tests and can be formalized in terms of local iterations. We consider as concrete domain $\mathcal{D}$ the subsets of the CSP search space $\hat{D} = \hat{D}_1 \times \cdots \times \hat{D}_n$ (Def. 9.2.2), *i.e.,*

$(\mathcal{P}(\hat{D}), \subseteq, \emptyset, \cup)$. Each constraint C_i corresponds to a concrete lower closure operator $\rho_i : \mathcal{P}(\hat{D}) \to \mathcal{P}(\hat{D})$, such that $\rho_i(X)$ keeps only the points in X satisfying C_i. The concrete solution of the problem is simply $S = \rho(\hat{D})$, where $\rho = \rho_1 \circ \cdots \circ \rho_p$. It is expressed in fixpoint form as $\mathrm{gfp}_{\hat{D}}\, \rho$.

9.3.2 Abstract Domains

Solvers do not manipulate individual points in $\hat{D}$, but rather collections of points of certain forms, such as boxes, called domains in CP. We now show that CP-domains are elements of an abstract domain $(\mathcal{D}^\sharp, \sqsubseteq^\sharp, \bot^\sharp, \sqcup^\sharp)$ in AI, which depends on the chosen consistency. In addition to standard AI operators, we require a monotonic *size function* $\tau : \mathcal{D}^\sharp \to \mathbb{R}^+$ that we will use later as a termination criterion (Def. 9.3.4).

Example Generalized arc-consistency (Def. 9.2.2) corresponds to the abstract domain of integer Cartesian products $\mathcal{S}^\sharp$ (Def. 9.2.2), ordered by element-wise set inclusion. It is linked with the concrete domain $\mathcal{D}$ by the standard Cartesian Galois connection:

$$\begin{aligned}
&\mathcal{D} \underset{\alpha_a}{\overset{\gamma_a}{\leftrightarrows}} \mathcal{S}^\sharp \\
&\gamma_a(S_1, \ldots, S_n) = S_1 \times \cdots \times S_n \\
&\alpha_a(X) = \lambda i.\{x \,|\, \exists (x_1, \ldots, x_n) \in X,\, x_i = x\}
\end{aligned}$$

The size function τ_a uses the size of the largest component, minus one, so that singletons have size 0:

$$\tau_a(S_1, \ldots, S_n) = \max_i(|S_i| - 1)$$

Example Bound consistency (Def. 9.2.2) corresponds to the domain of integer boxes $\mathcal{I}^\sharp$ (Def. 9.2.2), ordered by element-wise interval inclusion. We have a Galois connection, and use as size function the length of the largest dimension:

$$\begin{aligned}
&\mathcal{D} \underset{\alpha_b}{\overset{\gamma_b}{\leftrightarrows}} \mathcal{I}^\sharp \\
&\gamma_b([\![a_1, b_1]\!], \ldots, [\![a_n, b_n]\!]) = [\![a_1, b_1]\!] \times \cdots \times [\![a_n, b_n]\!] \\
&\alpha_b(X) = \lambda i.[\![\ \min\{x \in \mathbb{Z} \,|\, \exists (x_1, \ldots, x_n) \in X,\, x_i = x\}, \\
&\qquad\qquad \max\{x \in \mathbb{Z} \,|\, \exists (x_1, \ldots, x_n) \in X,\, x_i = x\}]\!] \\
&\tau_b([\![a_1, b_1]\!], \ldots, [\![a_n, b_n]\!]) = \max_i(b_i - a_i)
\end{aligned}$$

Example Hull consistency (Def. 9.2.2) corresponds to the domain of boxes with floating-point bounds $\mathcal{B}^\sharp$ (Def. 9.2.2). We use the following Galois connection and size function:

$$\begin{aligned}
&\mathcal{D} \underset{\alpha_h}{\overset{\gamma_h}{\leftrightarrows}} \mathcal{B}^\sharp \\
&\gamma_h([a_1, b_1], \ldots, [a_n, b_n]) = [a_1, b_1] \times \cdots \times [a_n, b_n] \\
&\alpha_h(X) = \lambda i.[\ \max\{x \in \mathbb{F} \,|\, \forall (x_1, \ldots, x_n) \in X,\, x_i \geq x\}, \\
&\qquad\qquad \min\{x \in \mathbb{F} \,|\, \forall (x_1, \ldots, x_n) \in X,\, x_i \leq x\}] \\
&\tau_h([a_1, b_1], \ldots, [a_n, b_n]) = \max_i(b_i - a_i)
\end{aligned}$$

We observe that to each choice corresponds a classic non-relational abstract domain, which is an homogeneous Cartesian product of identical single-variable domains. However, this needs not be the case: new solvers can be designed beyond the ones considered in traditional CP by varying the abstract domains further. A first idea is to apply different consistencies to different variables which permits, in particular, mixing variables with discrete domains and variables with continuous domains. A second idea is to parametrize the solver with other abstract domains from the AI literature, in particular relational domains, which we illustrate below.

Example The octagon domain $\mathcal{O}^\sharp$ [Miné, 2006a] assigns a (floating-point) upper bound to each binary unit expression $\pm v_i \pm v_j$ on the variables $v_1, \ldots, v_n$. It enjoys a Galois connection, and we use the size function from [Pelleau et al., 2011]:

$$\mathcal{D} \underset{\alpha_o}{\overset{\gamma_o}{\leftrightarrows}} \mathcal{O}^\sharp$$
$$\mathcal{O}^\sharp = \{\alpha v_i + \beta v_j \,|\, i, j \in [\![1, n]\!],\, \alpha, \beta \in \{-1, 1\}\} \to \mathbb{F}$$
$$\gamma_o(X^\sharp) = \{(x_1, \ldots, x_n) \in \mathbb{R}^n \,|\, \forall i, j, \alpha, \beta,\, \alpha x_i + \beta x_j \leq X^\sharp(\alpha v_i + \beta v_j)\}$$
$$\alpha_o(X) = \lambda(\alpha v_i + \beta v_j).\, \min\{x \in \mathbb{F} \,|\, \forall (x_1, \ldots, x_n) \in X,\, \alpha x_i + \beta x_j \leq x\}$$
$$\tau_o(X^\sharp) = \min(\ \max_{i,j,\beta}(X^\sharp(v_i + \beta v_j) + X^\sharp(-v_i - \beta v_j)),\ \max_i(X^\sharp(v_i + v_i) + X^\sharp(-v_i - v_i))/2)$$

Example The polyhedron domain $\mathcal{P}^\sharp$ [Cousot and Halbwachs, 1978] abstract sets as convex, closed polyhedra. Modern implementations [Jeannet and Miné, 2009] generally follow the "double description approach" and maintain two dual representations for each polyhedron: a set of linear constraints and a set of generators (vertices and rays, although our polyhedra never feature rays as they are bounded). There is no abstraction function α for polyhedra, and so, no Galois connection. Operators are generally easier on one representation. In particular, we define the size function on generators as the maximal Euclidian distance between pairs of vertices:

$$\tau_p(X^\sharp) = \max_{g_i, g_j \in X^\sharp} ||g_i - g_j||$$

9.3.3 Constraints and Consistency

We now assume that an abstract domain $\mathcal{D}^\sharp$ underlying the solver is fixed. Given the concrete semantics of the constraints $\rho = \rho_1 \circ \cdots \circ \rho_p$, and if $\mathcal{D}^\sharp$ enjoys a Galois connection $\mathcal{D} \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathcal{D}^\sharp$, then the semantics of the perfect propagator achieving the consistency for all the constraints is simply: $\alpha \circ \rho \circ \gamma$. Solvers achieve this algorithmically by applying the propagator for each constraint in turn until a fixpoint is reached or, when this process is deemed too costly, return before a fixpoint is reached. By observing that each propagator corresponds to an abstract test transfer function $\rho_i^\sharp$ in $\mathcal{D}^\sharp$, we retrieve the local iterations proposed by Granger to analyze conjunctions of tests [Granger, 1992]. A trivial narrowing is used here: stop refining after an iteration limit is reached.

Additionally, each $\rho_i^\sharp$ can be internally implemented by local iterations [Granger, 1992], a technique which is used in both the AI and CP communities. A striking connection is the analysis in non-relation domains using forward-backward iterations on expression trees [Miné, 2004, §2.4.4], which is extremely similar to the HC4-revise algorithm [Benhamou et al., 1999] developed independently for CP.

When there is no Galois connections (as for polyhedra), there is no equivalent to consistency. Nevertheless, we can still use local iterations on approximate test transfer functions $\rho_i^\sharp$, which serve the same purpose: to remove some points that do not satisfy the constraints.

9.3.4 Disjunctive Completion and Split

In order to approximate the solution to an arbitrary precision, solvers use a coverage of finitely many abstract elements from $\mathcal{D}^\sharp$. This corresponds in AI to the notion of disjunctive completion. We now consider the abstract domain $\mathcal{E}^\sharp = \mathcal{P}_{\text{finite}}(\mathcal{D}^\sharp)$, and equip it with the Smyth order $\sqsubseteq_{\mathcal{E}}^\sharp$, a classic order for disjunctive completions defined as:

$$X^\sharp \sqsubseteq_{\mathcal{E}}^\sharp Y^\sharp \iff \forall B^\sharp \in X^\sharp,\, \exists C^\sharp \in Y^\sharp,\, B^\sharp \sqsubseteq^\sharp C^\sharp$$

The creation of new disjunctions is achieved by a split operation $\oplus$, that splits an abstract element into two or more elements:

Split Operator A *split operator* $\oplus : \mathcal{D}^\sharp \to \mathcal{E}^\sharp$ satisfies:

1. $\forall e \in \mathcal{D}^\sharp$, $|\oplus(e)|$ is finite,
2. $\forall e \in \mathcal{D}^\sharp$, $\forall e_i \in \oplus(e)$, $e_i \sqsubseteq^\sharp e$, and
3. $\forall e \in \mathcal{D}^\sharp$, $\gamma(e) = \bigcup\{\gamma(e_i) \,|\, e_i \in \oplus(e)\}$.

Condition 2 implies $\oplus(e) \sqsubseteq^\sharp_{\mathcal{E}} \{e\}$. Condition 3 implies that $\oplus$ is an abstraction of the identity; thus, $\oplus$ can be freely applied at any place during the solving process without altering the AI-soundness (over-approximation). We now present a few example splits.

Split in $\mathcal{S}^\sharp$ The instantiation of a variable v_i in a discrete domain $X^\sharp = (S_1, \ldots, S_n) \in \mathcal{S}^\sharp$ is a split operator:

$$\oplus_a(X^\sharp) = \{(S_1, \ldots, S_{i-1}, \{x\}, S_{i+1}, \ldots, S_n) \,|\, x \in S_i\}$$

Split in $\mathcal{B}^\sharp$ Cutting a box in two along a variable v_i in a continuous domain $X^\sharp = (I_1, \ldots, I_n) \in \mathcal{B}^\sharp$ is a split operator:

$$\oplus_h(X^\sharp) = \{(I_1, \ldots, I_{i-1}, [a, h], I_{i+1}, \ldots, I_n),\ (I_1, \ldots, I_{i-1}, [h, b], I_{i+1}, \ldots, I_n)\}$$

where $I_i = [a, b]$ and $h = (a + b)/2$ rounded in $\mathbb{F}$ in any direction.

Split in $\mathcal{O}^\sharp$ Given a binary unit expression $\alpha v_i + \beta v_j$, we define the split on an octagon $X^\sharp \in \mathcal{O}^\sharp$ along this expression as:

$$\oplus_o(X^\sharp) = \{X^\sharp[(\alpha v_i + \beta v_j) \mapsto h],\ X^\sharp[(-\alpha v_i - \beta v_j) \mapsto -h]\}$$

where $h = (X^\sharp(\alpha v_i + \beta v_j) - X^\sharp(-\alpha v_i - \beta v_j))/2$, rounded in $\mathbb{F}$ in any direction.

Split in $\mathcal{P}^\sharp$ Given a polyhedron $X^\sharp \in \mathcal{P}^\sharp$ represented as a set of linear constraints, and a linear expression $\sum_i \beta_i v_i$, we define the split:

$$\oplus_p(X^\sharp) = \{X^\sharp \cup \{\textstyle\sum_i \beta_i v_i \leq h\}, X^\sharp \cup \{\sum_i \beta_i v_i \geq h\}\}$$

where $h = (\min_{\gamma(X^\sharp)} \sum_i \beta_i v_i + \max_{\gamma(X^\sharp)} \sum_i \beta_i v_i)/2$ can be computed by the Simplex algorithm.

These splits are parametrized by the choice of a direction of cut (some variable or expression). For non-relational domains we can use two classic strategies from CP: split each variable in turn, or split along a variable with maximal size (*i.e.,* $|S_i|$ or $b_i - a_i$). These strategies lift naturally to octagons by replacing the set of variables with the (finite) set of unit binary expressions (see also [Pelleau et al., 2011]). For polyhedra, one can bisect the segment between two vertices that are the farthest apart, in order to minimize τ_p. However, even for relational domains, we can use a faster and simpler non-relational split, *e.g.,* cut along the variable with the largest range.

To ensure the termination of the solver, we impose that any series of reductions, splits, and choices eventually outputs a small enough element for τ:

Definition The operators $\tau : \mathcal{D}^\sharp \to \mathcal{R}^+$ and $\oplus : \mathcal{D}^\sharp \to \mathcal{E}^\sharp$ are compatible if, for any reductive operator $\rho^\sharp : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ (*i.e.,* $\forall X^\sharp \in \mathcal{D}^\sharp, \rho^\sharp(X^\sharp) \sqsubseteq^\sharp X^\sharp$) and any family of choice operators $\pi_i : \mathcal{E}^\sharp \to \mathcal{D}^\sharp$ (*i.e.,* $\forall Y^\sharp \in \mathcal{E}^\sharp, \pi_i(Y^\sharp) \in Y^\sharp$), we have:

$$\forall e \in \mathcal{D}^\sharp,\ \forall r \in \mathbb{R}^{>0}, \exists K \text{ s.t. } \forall j \geq K,\ (\tau \circ \pi_j \circ \oplus \circ \rho \circ \cdots \circ \pi_1 \circ \oplus \circ \rho)(e) \leq r$$

Each of the split function we presented above, $\oplus_a$, $\oplus_h$, $\oplus_o$, and $\oplus_p$, is compatible with the size function τ_a, τ_h, τ_o, and τ_p we proposed on the corresponding domain.

The search procedure can be represented as a search tree where each node corresponds to a search space and the children of a node are constructed by applying the split operator on the parent and then applying a reduction. With this representation, the set of nodes at a given depth corresponds to a disjunction over-approximating the solution. Moreover, a series of reduction (ρ), selection (π), and split ($\oplus$) operators corresponds to a tree branch. Definition 9.3.4 states that each branch of the search tree is finite.

9.3.5 Abstract Solving

We are now ready to present our solving algorithm, in Fig. 9.2. It maintains in toExplore and sols two disjunctions in $\mathcal{E}^\sharp$, and iterates the following steps: choose an abstract element e from toExplore (**pop**), apply the consistency ($\rho^\sharp$), and either discard the result, add it to the set of solutions sols, or split it ($\oplus$). The solver starts with the maximal element $\top^\sharp$ of $\mathcal{D}^\sharp$, which represents $\gamma(\top^\sharp) = \hat{D}$.

Correctness.

At each step, $\bigcup\{\gamma(x) \mid x \in \text{toExplore} \cup \text{sols}\}$ is an over-approximation of the set of solutions, because the consistency $\rho^\sharp$ is an abstraction of the concrete semantics ρ of the constraints and the split $\oplus$ is an abstraction of the identity. We note that abstract elements in sols are consistent and either contain only solutions or are smaller than r. The algorithm terminates when toExplore is empty, at which point sols over-approximates the set of solutions with consistent elements that contain only solutions or are smaller than r. To compute the exact set of solutions in the discrete case, it is sufficient to choose $r < 1$.

The termination is ensured by the following proposition:

Proposition 9.3.1 *If τ and $\oplus$ are compatible, the algorithm in Fig. 9.2 terminates.*

Proof The search tree is finite. Otherwise, as its width is finite by Def. 9.3.4, there would exist an infinite branch (König's lemma), which would contradict Def. 9.3.4.

The solver in Fig. 9.2 uses a queue data-structure, and splits the oldest abstract element first. More clever choosing strategies are possible (*e.g.,* split the largest element for τ). The algorithm remains correct and terminates for any strategy.

Comparison with Abstract Interpretation.

Similarly to local iterations in AI, our solver performs decreasing abstract iterations. Indeed, $\text{toExplore} \cup \text{sols}$ is decreasing for $\sqsubseteq^\sharp_{\mathcal{E}}$ in the disjunctive completion domain $\mathcal{E}^\sharp$ at each iteration of the loop (indeed, $\oplus(e) \sqsubseteq^\sharp_{\mathcal{E}} \{e\}$ and we can assume that $\rho^\sharp$ is reductive in $\mathcal{D}^\sharp$ without loss of generality). However, it differs from classic AI in two ways. Firstly, there is no split operator in AI: new components in a disjunctive completion are generally added only at control-flow joins (by delaying the abstract join $\sqcup^\sharp$ in $\mathcal{D}^\sharp$). Secondly, the solving iteration strategy is far more elaborated than in AI. The use of a narrowing is replaced with a data-structure that maintains an ordered list of abstract elements and a splitting strategy that performs a refinement process and ensures its termination. Actually, more complex strategies than the simple one we presented here exist in the CP literature. One example is the AC-5 algorithm [van Hentenryck et al., 1992] where, each time the domain of a variable changes, the variable decides which constraints need to be propagated. The design of efficient propagation algorithms is an active research area [Schulte and Tack, 2001].

9.4 AbSolute

We have implemented a solver called AbSolute and available on github[1]. We describe its main features and present experimental results.

[1] `https://github.com/mpelleau/AbSolute`

```
list of abstract domains sols ← ∅                    ▷ stores the abstract solutions
queue of abstract domains toExplore ← ∅              ▷ stores the abstract elements to explore
push ⊤♯ in toExplore                                 ▷ initialization with the abstract search space: γ(⊤♯) = D̂

while toExplore ≠ ∅ do
    e ← pop(toExplore)
    e ← ρ♯(e)
    if e ≠ ∅ then
        if τ(e) ≤ r or isSol(e)                      ▷ isSol(e) returns true if e contains only solutions then
            sols ← sols ∪ e
        else
            push ⊕(e) in toExplore
        end if
    end if
end while
```

Figure 9.2: Our generic abstract solver.

9.4.1 Implementation

Absolute, is implemented in OCaml. It uses Apron, a library of numeric abstract domains intended primarily for static analysis [Jeannet and Miné, 2009]. We benefit from Apron domains (intervals, octagons, and polyhedra), its ability to hide their internal algorithms under a uniform API, and its handling of integer and real variables and of non-linear constraints.

Consistency.

Apron provides a language of constraints sufficient to express many CSPs: equalities and inequalities over numeric expressions (including operators such as $+$, $-$, $\times$, $/$, $\sqrt{}$, power, modulo, and rounding to integers). The test transfer function naturally provides propagators for these constraints. Internally, each domain implements its own algorithm to handle tests, including sophisticated methods to handle non-linear constraints (such as HC4 and linearization [Miné, 2004]). Our solver then performs local iterations until either a fixpoint or a maximum number of iterations is reached (which is set to 3 to ensure a fast solving). In CP solvers, only the constraints containing at least one variable that has been modified during the previous step are propagated. However, for simplicity, our solver propagates all the constraints at each step of the local iteration.

Split.

Currently, our solver only splits along a single variable at a time, cutting its range in two, even for relational domains and integer variables. It chooses the variable with the largest range. It uses a queue to maintain the set of abstract elements to explore (as in Fig. 9.2). Compared to most CP solvers, this splitting strategy is very basic. It will be improved in the future by integrating more clever strategies from the CP literature.

9.4.2 Exemple of AI-solving with Absolute

In order to make the abstract solving process clear, we detail here an example with a very simple problem. Consider a CSP on two continuous variables v_1 and v_2 taking their values in $D_1 = D_2 = [-5, 5]$, and the constraints $C_1 : x^2 + y^2 \leq 4$ and $C_2 : (x-2)^2 + (y+1)^2 \leq 1$.

Figure 9.3 shows the first iterations of the AI-solving method for this CSP. The root corresponds to the initial search space after applying the reduction based on HC4 as explained above ($D_1 = [1, 2], D_2 =$

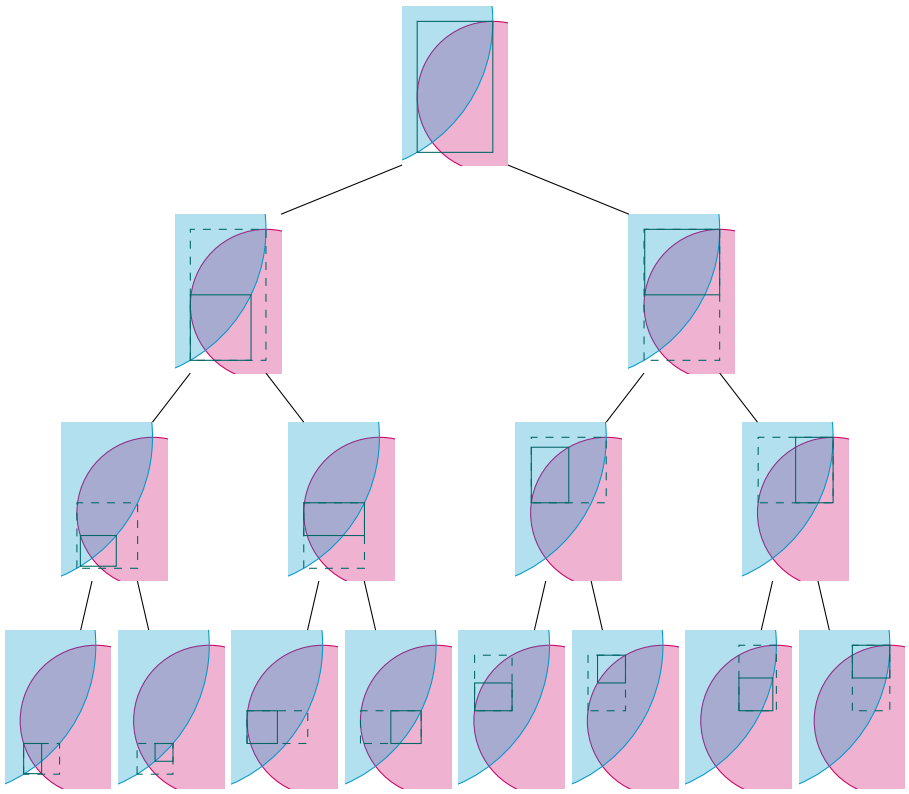


Figure 9.3: First iterations of the AI-solving method.

$[-1.73, 0]$). Its successor nodes correspond to the search spaces obtained after splitting the domain D_2 in half and applying the reduction to the new states. These two steps (split and reduction) are repeatedly applied until all the solutions have been found, but Figure 9.3 only shows the first three steps.

At a given depth in the search tree, the current approximation of the solution space is made of the disjunctive completion of the abstract elements currently investigated. Figure 9.4 shows these disjunctive completions for the search tree depicted in Fig. 9.3.

9.4.3 Experimental Results

We have run Absolute on two classes of problems: firstly, on continuous problems to compare its efficiency with state-of-the-art CP solvers; secondly, on mixed problems, that these CP solvers cannot handle while our abstract solver can.

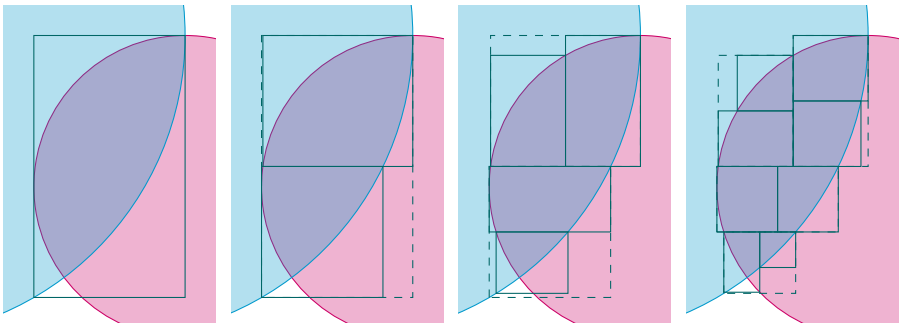


Figure 9.4: Disjunctive completions of the first iterations of the AI-solving method search tree given in Fig. 9.3.

			$\mathcal{B}^\sharp$		$\mathcal{O}^\sharp$	
name	# vars	ctr type	Ibex	Absolute	Ibex	Absolute
b	4	=	0.009	0.018	0.053	0.048
nbody5.1	6	=	32.85	708.47	0.027	$\geq$ 1h
ipp	8	=	0.66	9.64	19.28	1.46
brent-10	10	=	7.96	4.57	0.617	$\geq$ 1h
KinematicPair	2	$\leq$	0.013	0.018	0.016	0.011
biggsc4	4	$\leq$	0.011	0.022	0.096	0.029
o32	5	$\leq$	0.045	0.156	0.021	0.263

Table 9.1: CPU time in seconds to find the first solution with Ibex and Absolute.

			$\mathcal{B}^\sharp$		$\mathcal{O}^\sharp$	
name	# vars	ctr type	Ibex	Absolute	Ibex	Absolute
b	4	=	0.02	0.10	0.26	0.14
nbody5.1	6	=	95.99	1538.25	27.08	$\geq$ 1h
ipp	8	=	38.83	39.24	279.36	817.86
brent-10	10	=	21.58	263.86	330.73	$\geq$ 1h
KinematicPair	2	$\leq$	59.04	23.14	60.78	31.11
biggsc4	4	$\leq$	800.91	414.94	1772.52	688.56
o32	5	$\leq$	27.36	22.66	40.74	33.17

Table 9.2: CPU time in seconds to find all solutions with Ibex and Absolute.

Continuous solving.

We use problems from the COCONUT benchmark[2], a standard CP benchmark with only real variables. We compare Absolute with the standard (interval-based) Ibex CP continuous solver[3]. Notice that the COCONUT problems have a relatively small number of variables, compared for instance to the number of variables that can be analyzed for a single program in AI. The difficulty of the benchmark is here due to both the expressions of the constraints (non linear with multiple variable occurences) and the high precision that is required.

Additionally, we compare Absolute to our extension of Ibex to octagons from previous work [Pelleau et al., 2011], which allows comparing the choice of domain (intervals versus octagons) independently from the choice of solver algorithm (classic CP solver versus our AI-based solver). Tables 9.1 and 9.2 show the run time in seconds to find all the solutions or only the first solution of each problem. Tables 9.3 and 9.4 show the number of nodes created to find all the solutions or only the first solution of each problem.

On average, Absolute is competitive with the traditional CP approach. More precisely, it is globally slower on problems with equalities, and faster on problems with inequalities. This difference of performance seems to be related to the following ratio: the number of constraints in which a variable appears over the total number of constraints. As said previously, at each iteration, all the constraints are propagated even those for which none of their variables have changed. This increases the computation time at each step and thus increases the overall time. For instance, in the problem `brent-10`, there are ten variables, ten constraints, and each variable appears in at most three constraints. If only one variable has been modified, we will nevertheless propagate all ten constraints, instead of three at most. This may explain the timeouts observed on problems `brent-10` and `nbody5.1` with Absolute.

Moreover, in our solver, the consistency loop is stopped after three iterations while, in the classic CP

[2]Available at `http://www.mat.univie.ac.at/~neum/glopt/coconut/`.
[3]Available at `http://www.emn.fr/z-info/ibex/`.

			$\mathcal{B}^\sharp$		$\mathcal{O}^\sharp$	
name	# vars	ctr type	Ibex	Absolute	Ibex	Absolute
b	4	$=$	145	28	45	207
nbody5.1	6	$=$	262 659	2 765 630	105	-
ipp	8	$=$	4 039	25 389	899	3 421
brent-10	10	$=$	101 701	12 744	2 113	-
KinematicPair	2	$\leq$	43	55	39	55
biggsc4	4	$\leq$	98	96	94	84
o32	5	$\leq$	87	344	85	942

Table 9.3: Number of nodes created to find the first solution with Ibex and Absolute.

			$\mathcal{B}^\sharp$		$\mathcal{O}^\sharp$	
name	# vars	ctr type	Ibex	Absolute	Ibex	Absolute
b	4	$=$	551	577	147	1057
nbody5.1	6	$=$	598 521	5 536 283	7 925	-
ipp	8	$=$	237 445	99 179	39 135	2 884 925
brent-10	10	$=$	211 885	926 587	5 527	-
KinematicPair	2	$\leq$	847 643	215 465	520 847	215 465
biggsc4	4	$\leq$	3 824 249	6 038 844	2 411 741	6 037 260
o32	5	$\leq$	161 549	120 842	84 549	111 194

Table 9.4: Number of nodes created to find all solutions with Ibex and Absolute.

approach, the fixpoint is reached. The consistency in Absolute may be less precise than the one used in Ibex, which reduces the time spent during the propagation step but may increase the search phase. This probably explains why in tables 9.3 and 9.4 the number of nodes created during the solving process with Absolute is most of the times larger than the one with Ibex. Less reductions are performed thus more splitting operations are needed, hence more nodes are created during the solving process.

These experimentations show that our prototype, which only features quite naïve CP strategies, behaves reasonably well on a classic benchmark. Further studies will include a deeper analysis of the performances and improvements of Absolute on its identified weaknesses (splitting strategy, propagation loop).

9.5 Conclusion

In this chapter, we have exposed some links between AI and CP, and used them to design a CP solving scheme built entirely on abstract domains. The preliminary results obtained with our solver are encouraging and open the way to the development of hybrid CP–AI solvers able to naturally handle mixed constraint problems.

This framework is used in next Chapter with an instanciation of the abstract domain to the Octagon domains, which provides a first extension of classical CP domains: Octagons are relational, while common CP domains, such as integer or real boxes, are cartesian.

10

Octagons

This chapter is based on Marie Pelleau, Charlotte Truchet, Frédéric Benhamou, Octagonal Domains for Continuous Constraints. CP 2011: 706-720. This article is chronologically earlier than the previous one, although it describes a particular case of abstract domain, instanciating the framework described previously. The most important result is the definition of octagonal consistency and heuristic. Octagons are a good alternative to heavy preconditioning techniques which adapt the axes to the specific geometry of the problem.

In practice, the Octagons described here are derived from the works of Antoine Miné, who applied the Floyd-Warshall algorithm to compute the canonical representation of an octagon. We extended this algorithm by adding Hull consistency steps, interleaving them with the Floyd-Warshall propagation.

10.1 Introduction

Continuous Constraint Programming (CP) relies on interval representation of the variables domains. Filtering and solution set approximations are based on Cartesian products of intervals, called boxes. In this chapter, we propose to improve the Cartesian representation precision by introducing an n-ary octagonal representation of domains in order to improve filtering accuracy.

By introducing non-Cartesian representations for domains, we do not modify the basic principles of constraint solving. The main idea remains to reduce domains by applying constraint propagators that locally approximate constraint and domains intersections (filtering), by computing fixpoints of these operators (propagation) and by splitting the domains to search the solution space. Nevertheless, each of these steps has to be redesigned in depth to take the new domains into account, since we lose the convenient correspondence between approximate intersections and domain projections.

While shifting from a Cartesian to a relational approach, the resolution process is very similar. In the interval case, one starts with the Cartesian product of the initial domains and propagators reduce this global box until reaching a fixpoint. In the octagonal case, the Cartesian product of the initial domains is itself a particular case of octagon and each constraint propagator computes in turn the smallest octagon containing the intersection of the global octagon and the constraint itself, until reaching an octagonal fixpoint. In both cases, splitting operators drive the search space exploration, alternating with global domain reduction.

The octagon are chosen for different reasons: they represent a reasonable tradeoff between boxes and

more complex approximation shapes (e.g. polyhedron, ellipsoids) and they have been studied in another context to approximate numerical computations in static analysis of programs. More importantly, we show that octagons allows us to translate the corresponding constraint systems in order to incorporate classical continuous constraint tools in the resolution.

The contributions of this chapter concern the different aspects of octagon-based solving. First, we show how to transform the initial constraint problem to take the octagonal domains into account. The main idea here is to combine classical constraint matrix representations and rotated boxes, which are boxes defined in different $\pi/4$ rotated bases. Second, we define a specific local consistency, oct-consistency, and propose an appropriate algorithm, built on top of any continuous filtering method. Third, we propose a split algorithm and a notion of precision adapted to the octagonal case.

After some preliminary notions on continuous CSPs and octagons (Section 10.2), we present in Section 10.3 the octagon representation and the notion of octagonal CSP. Section 10.4 addresses octagonal consistency and propagation. The general solver, including discussions on split and precision is presented in Section 10.5. Finally, experimental results are presented in Section 10.6, related work in Section 10.7, while conclusion and future work end the presentation of this work.

10.2 Preliminaries

This section recalls basics notions of CP and provides material on octagons from [Miné, 2006a].

10.2.1 Notations and Definitions

We consider a Constraint Satisfaction Problem (CSP) on variables $\mathcal{V} = (v_1...v_n)$, taking their values in domains $\mathcal{D} = (D_1...D_n)$, with constraints $(C_1...C_p)$. The set of tuples representing the possible assignments for the variables is $D = D_1 \times ... \times D_n$. The solutions of the CSP are the elements of D satisfying the constraints. We denote by $\mathcal{S}$ the set of solutions, $\mathcal{S} = \{(s_1...s_n) \in \mathcal{D}, \forall i \in 1..n, C_i(s_1...s_n)\}$.

In the CP framework, variables can either be discrete or continuous. In this chapter, we focus on real variables. Domains are subintervals of $\mathbb{R}$ whose bounds are floating points, according to the norm IEEE 754. Let $\mathbb{F}$ be the set of floating points. For $a, b \in \mathbb{F}$, we can define $[a, b] = \{x \in \mathbb{R}, a \leq x \leq b\}$ the real interval delimited by a and b, and $\mathbb{I} = \{[a, b], a, b \in \mathbb{F}\}$ the set of all such intervals. Given an interval $I \in \mathbb{I}$, we write $\underline{I}$ (resp. $\overline{I}$) its lower (resp. upper) bound, and, for any real point x, $\underline{x}$ its floating-point lower approximation (resp. $\overline{x}$, upper). A cartesian product of intervals is called a box. For CSPs with domains in $\mathbb{I}$, constraint solver usually return a box containing the solutions, that is, an overapproximation for $\mathcal{S}$.

The notion of *local consistency* is central in CP. We recall here the definition of Hull-consistency, one of the most usual local consistency for continuous constraints.

Hull-Consistency Let $v_1...v_n$ be variables over continuous domains represented by intervals $D_1...D_n \in \mathbb{I}$, and C a constraint. The domains $D_1...D_n$ are said *Hull-consistent* for C iff $D_1 \times ... \times D_n$ is the smallest floating-point box containing the solutions for C.

Given a constraint C over domains $D_1...D_n$, an algorithm that computes the local consistent domains $D'_1...D'_n$, such that $\forall i \in 1...n, D'_i \subset D_i$ and $D'_1...D'_n$ are locally consistent for C, is called a *propagator* for C. The domains which are locally consistent for all constraints are the largest common fixpoints of all the constraints propagators [Benhamou, 1996, Schulte and Stuckey, 2008]. Practically, propagators often compute overapproximations of the locally consistent domains. For instance, the algorithm HC4 [Benhamou et al., 1999] efficiently propagates continuous constraints, relying on the syntax of the constraint and interval arithmetic [Moore, 1966]. It generally does not reach Hull consistency, in particular in case of multiple occurrences of the variables.

Local consistency computations can be seen as deductions, performed on domains by analyzing the constraints. If the propagators return the empty set, the domains are inconsistent and the problem has

no solution. Otherwise, non-empty local consistent domains are computed. This is often not sufficient to accurately approximate the solution set. In that case choices are made on the variables values. For continuous constraints, a domain D is chosen and split into two (or more) parts, which are in turn narrowed by the propagators. The solver recursively alternates propagations and splits until a given precision is reached. In the end, the collection of returned boxes covers $\mathcal{S}$, under some hypotheses on the propagators and splits [Benhamou, 1996].

10.2.2 Octagons

In geometry, an octagon[1] is a polygon having eight faces in $\mathbb{R}^2$. In this chapter, we use a more general definition given in [Miné, 2006a].

Octagonal constraints Let v_1, v_2 be two real variables. An octagonal constraint is a constraint of the form $\pm v_1 \pm v_2 \leq c$ with $c \in \mathbb{R}$.

For instance in $\mathbb{R}^2$, octagonal constraints define straight lines which are parallel to the axis if $i = j$, and diagonal if $i \neq j$. This remains true in $\mathbb{R}^n$, where the octagonal constraints define hyperplanes.

Octagon Given a set of octagonal constraints $\mathcal{O}$, the subset of $\mathbb{R}^n$ points satisfying all the constraints in $\mathcal{O}$ is called an octagon.

Remark Here follows some general remarks on octagons:

- the geometric shape defined above includes the geometric octagons, but also other polygons (e.g. in $\mathbb{R}^2$, an octagon can have less than eight faces);
- an octagon can be defined with redundant constraints (for example $v_1 - v_2 \leq c$ and $v_1 - v_2 \leq c'$), but only one of them defines a face of the octagon (the one with the lowest constant in this example),
- in $\mathbb{R}^n$, an octagon has at most $2n^2$ faces, which is the maximum number of possible non-redundant octagonal constraints on n variables,
- the intersection of two octagons is also an octagon, satisfying the conjunction of the constraints of the original octagons,
- an octagon is a set of *real* points, but, like the intervals, they can be restricted to have floating-points bounds ($c \in \mathbb{F}$).

In the following, octagons are restricted to floating-point octagons. Without loss of generality, we assume octagons to be defined with no redundancies.

10.2.3 Matrix Representation of Octagons

An octagon can be represented with a *difference bound matrix* (DBM) as described in [Menasche and Berthomieu, 1983, Miné, 2006a]. This representation is based on a normalization of the octagonal constraints as follows.

Difference constraints Let w, w' be two variables. A difference constraint is a constraint of the form $w - w' \leq c$ for c a constant.

By introducing new variables, it is possible to rewrite an octagonal constraint as an equivalent difference constraint: let $C \equiv (\pm v_i \pm v_j \leq c)$ an octagonal constraint. Define the new variables $w_{2i-1} = v_i, w_{2i} = -v_i, w_{2j-1} = v_j, w_{2j} = -v_j$. Then

[1] http://mathworld.wolfram.com/Octagon.html

- for $i \neq j$
  - if $C \equiv (v_i - v_j \leq c)$, then C is equivalent to the difference constraints $(w_{2i-1} - w_{2j-1} \leq c)$ and $(w_{2j} - w_{2i} \leq c)$,
  - if $C \equiv (v_i + v_j \leq c)$, then C is equivalent to the difference constraints $(w_{2i-1} - w_{2j} \leq c)$ and $(w_{2j-1} - w_{2i} \leq c)$,
  - the two other cases are similar,
- for $i = j$
  - if $C \equiv (v_i - v_i \leq c)$, then C is pointless, and can be removed,
  - if $C \equiv (v_i + v_i) \leq c)$, then C is equivalent to the difference constraint $(w_{2i-1} - w_{2i} \leq c)$,
  - the two other cases are similar.

In what follows, regular variables are always written $(v_1...v_n)$, and the corresponding new variables are written $(w_1, w_2, ...w_{2n})$ with: $w_{2i-1} = v_i$, and $w_{2i} = -v_i$. As shown in [Miné, 2006a], the rewritten difference constraints represent the same octagon as the original set of octagonal constraints, by replacing the positive and negative occurrences of the v_i variables by their w_i counterparts. Storing difference constraints is thus a suitable representation for octagons.

DBM Let $\mathcal{O}$ be an octagon in $\mathbb{R}^n$, and its sequence of potential constraints as defined above. The octagon DBM is a $2n \times 2n$ square matrix, such that the element at row i, column j is the constant c of the potential constraint $w_j - w_i \leq c$.

An example is shown on Figure 10.1(c): the element on row 1 and column 3 corresponds to the constraint $v_2 - v_1 \leq 2$ for instance.

At this stage, different DBMs can represent the same octagon. For example on Figure 10.1(c), the element row 2 and column 3 can be replaced with 100, for instance, without changing the corresponding octagon. In [Miné, 2006a], an algorithm is defined so as to optimally compute the smallest values for the elements of the DBM. This algorithm is adapted from the Floyd-Warshall shortest path algorithm [Floyd, 1962], modified in order to take advantage of the DBM structure. It exploits the fact that w_{2i-1} and w_{2i} correspond to the same variable. It is fully presented in [Miné, 2006a].

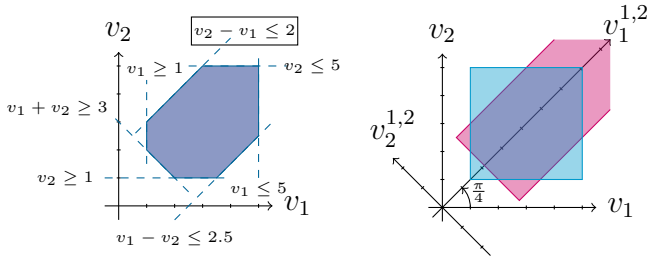
10.3 Boxes Representation

In the following section we introduce a box representation for octagons. This representation, combined with the DBM, will be used to define, from an initial set of continuous constraints, an equivalent system taking the octagonal domains into account.

10.3.1 Intersection of boxes

In two dimensions, an octagon can be represented by the intersection of one box in the canonical basis for $\mathbb{R}^2$, and one box in the basis obtained from the canonical basis by a rotation of angle $\pi/4$ (see figure 10.1(b)). We generalize this remark to n dimensions.

Rotated basis Let $B = (u_1, ..., u_n)$ be the canonical basis of $\mathbb{R}^n$. Let $\alpha = \pi/4$. The *(i,j)-rotated basis* $B_\alpha^{i,j}$ is the basis obtained after a rotation of α in the subplane defined by (u_i, u_j), the other vectors remaining unchanged:
$B_\alpha^{i,j} = (u_1, ..., u_{i-1}, (cos(\alpha)u_i + sin(\alpha)u_j), ...u_{j-1}, (-sin(\alpha)u_i + cos(\alpha)u_j), ...u_n)$.



$$\begin{pmatrix} 0 & -2 & 2 & -3 \\ 10 & 0 & +\infty & 2.5 \\ 2.5 & -3 & 0 & -2 \\ +\infty & 2 & 10 & 0 \end{pmatrix}$$

(a) (b) (c)

Figure 10.1: Equivalent representations for the same octagon: the octagonal constraints 10.1(a), the intersection of boxes 10.1(b), and the DBM 10.1(c).

By convention, for any $i \in \{1...n\}$, $B_\alpha^{i,i}$ represents the canonical basis. In what follows, α is always $\pi/4$ and will be omitted. Then $cos(\alpha) = sin(\alpha) = 1/\sqrt{2}$. Finally, every variable v living in the $B^{i,j}$ rotated basis and whose domain is D will be denoted by $v^{i,j}$ and its domain by $D^{i,j}$.

The DBM can also be interpreted as the representation of the intersection of one box in the canonical basis and $n(n-1)/2$ other boxes, each one living in a rotated basis. Let $\mathcal{O}$ be an octagon in $\mathbb{R}^n$ and its DBM M, with the same notations as above (M is a $2n \times 2n$ matrix). For $i, j \in \{1...n\}$, with $i \neq j$, let $\mathcal{B}_\mathcal{O}^{i,j}$ be the box $I_1 \times ... \times I_i^{i,j} \times ... \times I_j^{i,j} ... \times I_n$, in $B^{i,j}$, such that $\forall k \in \{1...n\}$

$$\begin{array}{rclcrcl} \underline{I_k} & = & \underline{-\frac{1}{2}M[2k-1, 2k]} & & \overline{I_k} & = & \overline{\frac{1}{2}M[2k, 2k-1]} \\ \underline{I_i^{i,j}} & = & \underline{-\frac{1}{\sqrt{2}}M[2j-1, 2i]} & & \overline{I_i^{i,j}} & = & \overline{\frac{1}{\sqrt{2}}M[2j, 2i-1]} \\ \underline{I_j^{i,j}} & = & \underline{-\frac{1}{\sqrt{2}}M[2j-1, 2i-1]} & & \overline{I_j^{i,j}} & = & \overline{\frac{1}{\sqrt{2}}M[2j, 2i]} \end{array}$$

Proposition 10.3.1 *Let $\mathcal{O}$ be an octagon in $\mathbb{R}^n$, and $\mathcal{B}_\mathcal{O}^{i,j}$ the boxes as defined above. Then $\mathcal{O} = \bigcap_{1 \leq i,j \leq n} \mathcal{B}_\mathcal{O}^{i,j}$.*

Proof Let $i, j \in \{1..n\}$. We have $v_i^{i,j} = 1/\sqrt{2}(v_i + v_j)$ and $v_j^{i,j} = 1/\sqrt{2}(v_i - v_j)$ by definition 10.3.1. Thus $(v_1...v_i^{i,j}...v_j^{i,j}...v_n) \in \mathcal{B}_\mathcal{O}^{i,j}$ iff it satisfies the octagonal constraints on v_i and v_j, and the unary constraints for the other coordinates, in the DBM. The box $\mathcal{B}_\mathcal{O}^{i,j}$ is thus the solution set for these particular octagonal constraints. The points in $\bigcap_{1 \leq i,j \leq n} \mathcal{B}_\mathcal{O}^{i,j}$ are exactly the points which satisfy all the octagonal constraints.

Example Considering the DBM Figure 10.1(c), the boxes are $I_1 \times I_2 = [1, 5] \times [1, 5]$, and $I_1^{1,2} \times I_2^{1,2} = [3/\sqrt{2}, +\infty] \times [-2.5/\sqrt{2}, \sqrt{2}]$.

To summarize, an octagon with its DBM representation can also be interpreted as a set of octagonal constraints (definition in intension) or equivalently as an intersection of rotated boxes (definition in extension), at the cost of a multiplication / division with the appropriate rounding mode. We show below that the octagonal constraints (or the bounds in the case of boxes) can be inferred from the CSP.

10.3.2 Octagonal CSP

Consider a CSP on variables $(v_1...v_n)$ in $\mathbb{R}^n$. The goal is now to equip this CSP with an octagonal domain. We detail here how to build an octagonal CSP from a regular one, and show that the two systems are equivalent.

First, the CSP is associated to an octagon, by stating all the possible octagonal constraints $\pm v_i \pm v_j \leq c_k$ for $i, j \in \{1...n\}$. The constants c_k represent the bounds of the octagon boxes and are dynamically modified. They are initialized to $+\infty$.

The rotations defined in the previous section introduce new axes, that is, new variables $v_i^{i,j}$. Because these variables are redundant with the regular ones, they are also linked through the CSP constraints $(C_1...C_p)$, and these constraints have to be rotated as well. This can be done by symbolically applying the rotation to the constraint variables.

Rotated constraint Given a constraint C_k holding on variables $(v_1...v_n)$, the (i,j)-rotated constraint $C_k^{i,j}$ is the constraint obtained by replacing each occurrence of v_i by $\cos(\alpha)v_i^{i,j} - \sin(\alpha)v_j^{i,j}$ and each occurrence of v_j by $\sin(\alpha)v_i^{i,j} + \cos(\alpha)v_j^{i,j}$.

Given a continuous CSP $< v_1...v_n, D_1...D_n, C_1...C_p >$, we define an octagonal CSP by adding the rotated variables, the rotated constraints, and the rotated domains stored as a DBM. To sum up and fix the notations, the octagonal CSP thus contains:

- the regular variables $(v_1...v_n)$;
- the rotated variables $(v_1^{1,2}, v_2^{1,2}, v_1^{1,3}, v_3^{1,3}...v_n^{n-1,n})$, where $v_i^{i,j}$ is the i-th variable in the (i,j) rotated basis $B_\alpha^{i,j}$;
- the regular constraints $(C_1...C_p)$;
- the rotated constraints $(C_1^{1,2}, C_1^{1,3}...C_1^{n-1,n}...C_p^{n-1,n})$.
- the regular domains $(D_1...D_n)$;
- a DBM which represents the rotated domains. It it initialized with the bounds of the regular domains for the cells at position $2i, 2i-1$ and $2i-1, 2i$ for $i \in \{1...2n\}$, and $+\infty$ everywhere else.

In these conditions, the initial CSP is equivalent to this transformed CSP, restricted to the variables $v_1...v_n$, as shown in the following proposition.

Proposition 10.3.2 *Consider a CSP $< v_1...v_n, D_1...D_n, C_1...C_p >$, and the corresponding octagonal CSP as defined above. The solution set of the original CSP $\mathcal{S}$ is equal to the solution set of the $(v_1...v_n)$ variables of the octagonal CSP.*

Proof Let $s \in \mathbb{R}^n$ a solution of the octogonal CSP for $(v_1...v_n)$. Then $s \in D_1 \times ... \times D_n$ and $C_1(s)...C_p(s)$ are all true. Hence s is a solution for the original CSP. Reciprocally, let $s' \in \mathbb{R}^n$ a solution of the original CSP. The regular constraints $(C_1...C_p)$ are true for s'. Let us show that there exist values for the rotated variables such that the rotated constraints are true for s'. Let $i, j \in \{1...n\}$, $i \neq j$, and $k \in \{1...p\}$ and $C_k^{i,j}$ the corresponding rotated constraint. By definition 10.3.2, $C_k^{i,j}(v_1...v_{i-1}, \cos(\alpha)v_i^{i,j} - \sin(\alpha)v_j^{i,j}, v_{i+1}... \sin(\alpha)v_i^{i,j} + \cos(\alpha)v_j^{i,j}...v_n) \equiv C_k(v_1...v_n)$. Let us define the two reals $s_i^{i,j} = cos(\alpha)s_i + sin(\alpha)s_j$ and $s_j^{i,j} = -sin(\alpha)s_i + cos(\alpha)s_j$, the image of s_i and s_j by the rotation of angle α. By reversing the rotation, $cos(\alpha)s_i^{i,j} - sin(\alpha)s_j^{i,j} = s_i$ and $sin(\alpha)s_i^{i,j} + cos(\alpha)s_j^{i,j} = s_j$, thus $C_k^{i,j}(s_1...s_i^{i,j}, ...s_j^{i,j}...s_n) = C_k(s_1...s_n)$ is true. It remains to check that $(s_1...s_i^{i,j}, ...s_j^{i,j}...s_n)$ is in the rotated domain, which is true because the DBM is initialized at $+\infty$. ∎

For a CSP on n variables, this representation has an order of magnitude n^2, with n^2 variables and domains, and $p\frac{n(n-1)}{2} + p$ constraints. Of course, many of these objects are redundant. We explain in the next sections how to use this redundancy to speed up the solving process.

10.4 Octagonal Consistency and Propagation

We first generalize the Hull-consistency definition to the octagonal domains, and define propagators for the rotated constraints. Then, we use the modified version of Floyd-Warshall (briefly described in section 10.2.3) to define an efficient propagation scheme for both octagonal and rotated constraints.

10.4.1 Octagonal Consistency for a Constraint

We generalize to octagons the definition of Hull-consistency on intervals for any continuous constraint. With the box representation, we show that any propagator for Hull-consistency on boxes can be extended to a propagator on the octagons. For a given n-ary relation on $\mathbb{R}^n$, there is a unique smallest octagon (wrt inclusion) which contains the solutions of this relation, as shown in the following proposition.

Proposition 10.4.1 *Consider a constraint C (resp. a constraint sequence $(C_1...C_p)$), and $\mathcal{S}_C$ its set of solutions (resp. $\mathcal{S}$). Then there exist a unique octagon $\mathcal{O}$ such that: $\mathcal{S}_C \subset \mathcal{O}$ (resp. $\mathcal{S} \subset \mathcal{O}$), and for all octagons $\mathcal{O}'$, $\mathcal{S}_C \subset \mathcal{O}'$ implies $\mathcal{O} \subset \mathcal{O}'$. $\mathcal{O}$ is the unique smallest octagon containing the solutions, wrt inclusion.*

Proof Let $\mathcal{O} = \bigcap_{\mathcal{O}' \text{s.t.} \mathcal{S}_C \subset \mathcal{O}'} \mathcal{O}'$. The set of all octagons is closed by intersection (see remark 10.2.2), thus $\mathcal{O}$ is an octagon. Let us show that it is the smallest. Let $\mathcal{O}_1$ be an octagon containing the solutions. By construction: $\mathcal{O} = \mathcal{O}_1 \cap \{\text{other octagons}\} \subset \mathcal{O}_1$. Hence the existence. Let us show the unicity. Let $\mathcal{O}_1$ be another smallest octagon containing the solutions. By reversing the preceding argument, $\mathcal{O}_1 \subset \mathcal{O}$ thus $\mathcal{O}_1 = \mathcal{O}$. ∎

Oct-Consistency Consider a constraint C (resp. a constraint sequence $(C_1...C_p)$), and $\mathcal{S}_C$ its set of solutions (resp. $\mathcal{S}$). An octagon is said Oct-consistent for this constraint iff it is the smallest octagon containing $\mathcal{S}_C$ (resp. $\mathcal{S}$), wrt inclusion.

From proposition 10.4.1, this definition is well founded. With the expression of an (i, j)-rotated constraint $C^{i,j}$, any propagator defined on the boxes can be used to compute the Hull-consistent boxes for $C^{i,j}$ (although such a propagator, as HC4, may not reach consistency). This gives a consistent box in basis $B^{i,j}$, and can be done for all the bases. The intersection of the Hull-consistent boxes is the Hull-consistent octagon.

Proposition 10.4.2 *Let C be a constraint, and $i, j \in \{1...n\}$. Let $\mathcal{B}^{i,j}$ be the Hull-consistent box for the rotated constraint $C^{i,j}$, and $\mathcal{B}$ the Hull-consistent box for C. The Oct-consistent octagon for C is the intersection of all the $\mathcal{B}^{i,j}$ and $\mathcal{B}$.*

Proof Let $\mathcal{O}$ be the Oct-consistent octagon. By definition 10.2.2, a box is an octagon. Since $\mathcal{O}$ is the smallest octagon containing the solutions, and all the $\mathcal{B}^{i,j}$ contain the solutions, for all $i, j \in \{1...n\}$, $i \neq j$ $\mathcal{O} \subset \mathcal{B}^{i,j}$ (the same holds for $\mathcal{B}$). Thus $\mathcal{O} \subset \bigcap_{1 \leq i,j \leq n} \mathcal{B}^{i,j}$. Reciprocally, we use the box representation for the Oct-consistent octagon: $\mathcal{O} = \bigcap_{1 \leq i,j \leq n} \mathcal{B}_o^{i,j}$, where $\mathcal{B}_o^{i,j}$ is the box defining the octagon in $B^{i,j}$. Because $\mathcal{O}$ contains all the solutions and $\mathcal{B}_o^{i,j}$ contains $\mathcal{O}$, $\mathcal{B}_o^{i,j}$ also contains all the solutions. Since $\mathcal{B}^{i,j}$ is the Hull-consistent box in $B_\alpha^{i,j}$, it is the smallest box in $B_\alpha^{i,j}$ which contains all the solutions. Thus $\mathcal{B}^{i,j} \subset \mathcal{B}_o^{i,j}$. From there, $\bigcap_{1 \leq i,j \leq n} \mathcal{B}^{i,j} \subset \bigcap_{1 \leq i,j \leq n} \mathcal{B}_o^{i,j} = \mathcal{O}$. Again, the same holds for $\mathcal{B}$. The two inclusions being proven, $\mathcal{O} = \bigcap_{1 \leq i,j \leq n} \mathcal{B}^{i,j}$. ∎

float dbm[2n, 2n] /*the dbm containing the octagonal constraints*/
list propagList $\leftarrow (\rho_{C_1}, ...\rho_{C_p}, \rho_{C_1^{1,2}}...\rho_{C_p^{n-1,n}})$ /*list of the propagators to apply*/
while propagList $\neq \emptyset$ **do**
  apply all the propagators of propagList **to** dbm /*initial propagation*/
  propagList $\leftarrow \emptyset$
  for i,j from 1 to n **do**
    m $\leftarrow$ minimum of (dbm$[2i-1, k]$+dbm$[k, 2j-1]$) for k from 1 to $2n$
    m $\leftarrow$ minimum(m, dbm$[2i-1, 2i]$+dbm$[2j, 2j-1]$)
    if m $<$ dbm$[2i-1, 2j-1]$ **then**
      dbm$[2i-1, 2j-1] \leftarrow$ m /*update of the DBM*/
      add $\rho_{C_1^{i,j}}...\rho_{C_p^{i,j}}$ **to** propagList /*get the propagators to apply*/
    end if
    repeat the 5 previous steps **for** dbm$[2i-1, 2j]$, dbm$[2i, 2j-1]$, **and** dbm$[2i, 2j]$
  end for
end while

Figure 10.2: Pseudo code for the propagation loop mixing the Floyd Warshall algorithm and the regular and rotated propagators $\rho_{C_1}...\rho_{C_p}, \rho_{C_1^{1,2}}...\rho_{C_p^{n-1,n}}$, for an octagonal CSP as defined in section 10.3.2.

10.4.2 Propagation Scheme

The propagation scheme presented in subsection 10.2.3 for the octagonal constraints is optimal. We thus rely on this propagation scheme, and integrate the non-octagonal constraints propagators in this loop. The point is to use the octagonal constraints to benefit from the relational properties of the octagon. This can be done thanks to the following remark: all the propagators defined in the previous subsections are monotonic and complete (as is the HC4 algorithm). It results that they can be combined in any order in order to achieve consistency, as shown for instance in [Benhamou, 1996].

The key idea for the propagation scheme is to interleave the refined Floyd-Warshall algorithm and the constraint propagators. A pseudocode is given on figure 10.2. At the first level, the DBM is recursively visited so that the minimal bounds for the rotated domains are computed. Each time a DBM cell is modified, the corresponding propagators are added to the propagation list. The propagation list is applied before each new round in the DBM (so that a cell that would be modified twice is propagated only once). The propagation is thus guided by the additional information of the relational domain.

We show here that the propagation as defined on figure 10.2 computes the consistent octagon for a sequence of constraints.

Proposition 10.4.3 (Correctness) *Let $< v_1...v_n, D_1...D_n, C_1...C_p >$ a CSP. Assume that, for all $i, j \in \{1...n\}$, there exists a propagator ρ_C for the constraint C, such that ρ_C reaches Hull consistency, that is, $\rho_C(D_1 \times ... \times D_n)$ is the Hull consistent box for C. Then the propagation scheme as defined on figure 10.2 computes the Oct-consistent octagon for $C_1...C_p$.*

Proof This derives from proposition 10.4.2, and the propagation scheme of figure 10.2. The propagation scheme is defined so as to stop when propagList is empty. This happens when $\forall i, j \in \{1...n\}, k \in \{1...2n\}$, dbm$[2i-1, k]$+dbm$[k, 2j-1]$, dbm$[2i-1, 2i]$+dbm$[2j, 2j-1] \geq$ dbm$[2i-1, 2j-1]$, the same holds for dbm$[2i-1, 2j]$, dbm$[2i, 2j-1]$, and dbm$[2i, 2j]$. The octagonal constraints are thus consistent. In addition, each time a rotated box is modified in the DBM, its propagators are added to propagList. Hence, the final octagon is stable by the application of all $\rho_{C_k^{i,j}}$, for all $k \in \{1...p\}$ and $i, j \in \{1...n\}$. By hypothesis, the propagators reach consistency, the boxes are thus Hull-consistent for all the (rotated and regular) constraints. By proposition 10.4.2, the returned octagon is Oct-consistent. ∎

The refined Floyd-Warshall has a time complexity of $O(n^3)$. For each round in its loop, in the worst case we add p propagators in the propagation list. Thus the time complexity for the propagation scheme

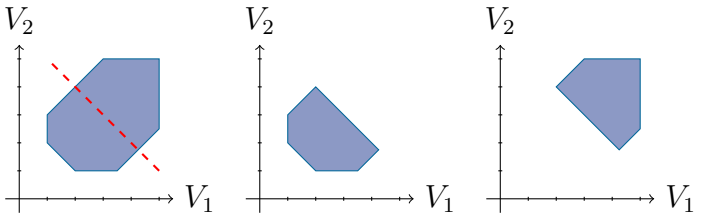


Figure 10.3: Example of a split: the octagon on the left is cut in the $B^{1,2}$ basis.

of figure 10.2 is $O(n^3p^3)$. In the end, the octagonal propagation uses both representations of octagons. It takes advantage of the relational property of the octagonal constraints (Floyd-Warshall), and of the usual constraint propagation on boxes (propagators). This comes to the cost of computing the octagon, but is expected to give a better precision in the end.

10.5 Solving

Besides the expected gain in precision obtained with octagon consistency, the box representation of octagons allows us to go a step further and define a fully octagonal solver. We thus define an octagonal split, in order to be able to cut the domains in any octagonal direction, and an octagonal precision, and end up with a fully octagonal solver.

10.5.1 Octagonal Split

The octagonal split extends the usual split operator to octagons. Splits can be performed in the canonical basis, thus being equivalent to the usual splits, or in the rotated basis. It can be defined as follow:

Definition Given an octagonal domain defined with the box representation $D_1...D_n, D_1^{1,2}...D_1^{n-1,n}...D_n^{n-1,n}$, such that $D_k^{i,j} = [a,b]$, a splitting operator for variable $v_k^{i,j}$, computes the two octagonal subdomains $D_1...[a,\overline{(a+b)/2}]...D_n^{n-1,n}$ and $D_1...[\underline{(a+b)/2},b]...D_n^{n-1,n}$.

As for the usual split, the union of the two octagonal subdomains is the original octagon, thus the split does not lose solutions. This definition does not take into account the correlation between the variables of the different basis. We take advantage again of the octagonal representation to communicate the domain reduction to the other basis. A split is thus immediately followed by a Floyd-Warshall propagation. Figure 10.3 shows an example of the split.

10.5.2 Precision

In most continuous solvers, the precision is defined as the size of the largest domain. For octagons, this definition leads to a loss of information because it does not take into account the correlation between the variables and domains.

Definition Let $\mathcal{O}$ be an octagon, and $I_1...I_n, I_1^{1,2}...I_n^{n-1,n}$ its box representation. The octagonal precision is $\tau(\mathcal{O}) = min_{1\leq i,j\leq n}(max_{1\leq k\leq n}(\overline{I_k^{i,j}} - \underline{I_k^{i,j}}))$.

For a single regular box, τ would be the same precision as usual. On an octagon, we take the minimum precision of the boxes in all the bases because it is more accurate, and it allows us to retrieve the operational semantics of the precision, as shown by the following proposition: in an octagon of precision r overapproximating a solution set $\mathcal{S}$, every point is at a distance at most r from $\mathcal{S}$.

```
Octogone oct
queue splittingList ← oct                          /*queue of the octagons*/
list acceptedOct ← ∅                               /*list of the accepted octagons*/
while splittingList ≠ ∅ do
    Octogone octAux ← splittingList.top()
    splittingList.pop()
    octAux ← Oct-consistence(octAux)
    if τ_Oct(octAux) < r or octAux contains only solutions then
        add octAux to acceptedOct
    else
        Octogone leftOct ← left(octAux)            /*left and right are the split operators*/
        Octogone rightOct ← right(octAux)
        add leftOct to splittingList
        add rightOct to splittingList
    end if
end while
return acceptedOct
```

Figure 10.4: Solving with octagons.

Proposition 10.5.1 *Let $< v_1...v_n, D_1...D_n, C_1...C_p >$ be a CSP, and $\mathcal{O}$ an octagon overapproximating $\mathcal{S}$. Let $r = \tau(\mathcal{O})$. Let $(v_1, ...v_n) \in \mathbb{R}^n$ be a point of $D_1 \times ... \times D_n$. Then $\forall 1 \leq i \leq n$, $min_{s \in \mathcal{S}} |v_i - s_i| \leq r$, where $s = (s_1...s_n)$. Each coordinate of all the points in $\mathcal{O}$ are at a distance at most r of a solution.*

Proof By definition 10.5.2, the precision r is the minimum of some quantities in all the rotated basis. Let $B^{i,j}$ be the basis that realizes this minimum. Because the box $\mathcal{B}^{i,j} = D_1 \times ... \times D_i^{i,j} \times ...D_j^{i,j}... \times D_n$ is Hull-consistent by proposition 10.4.2, it contains $\mathcal{S}$. Let $s \in \mathcal{S}$. Because $r = max_k(\overline{D_k} - \underline{D_k})$, $\forall 1 \leq k \leq n$, $|s_k - v_k| \leq \overline{D_k} - \underline{D_k} \leq r$. ∎

10.5.3 Octagonal Solver

Figure 10.4 describes the octagonal solving process. By proposition 10.4.3, and the split property, it returns a sequence of octagons whose union overapproximate the solution space. Precisely, it returns either octagons for which all points are solutions, or octagons overapproximating solution sets with a precision r.

An important feature of a constraint solver is the variable heuristic. For continuous constraints, one usually chooses to split the variable that has the largest domain. This would be very bad for the octagons, as the variable which has the largest domain is probably in a basis that is of little interest for the problem (it probably has a wide range because the constraints are poorly propagated in this basis). We thus define a default octagonal strategy which relies on the same remark as for definition 10.5.2: the variable to split is the variable $V_k^{i,j}$ which realizes the minimum of $min_{1 \leq i,j \leq n}(max_{1 \leq k \leq n}(\overline{D_k^{i,j}} - \underline{D_k^{i,j}}))$. The strategy is the following: choose first a promising basis, that is, a basis in which the boxes are tight (choose i, j). Then take the worst variable in this basis as usual (choose k). Figure 10.5 shows the result of boxes and octagons boxes on a toy problem within the same time limit. Octagons are costlier to compute, thus octagon solving provides less nodes, but they are more precise.

10.6 Experiments

Because the experiments are out of date (performed in 2011) and due to space reasons, the detailed experiments are not shown in this chapter. They can be found in the original article and we only provide a summary of the results.

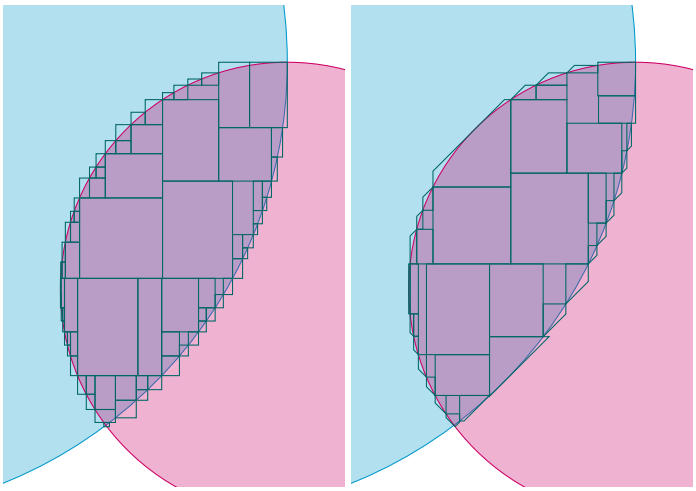


Figure 10.5: Within a given time limit (7ms), results on a simple problem. On the left, default IBEX solving in $\mathbb{I}^n$. On the right, Octagon solving.

We have tested the prototype octagonal solver on problems from the Coconut benchmarks[2]. Our problem selection involves different types of constraints (inequations, equations, or both). We compared our method with Ibex 1.18 Apart from the variable heuristic presented in subsection 10.5.3, the experimental comparisons between box and octogonal resolutions were run with the same configuration in Ibex. The octagonal solver is faster and creates less boxes to find the first solution of our preliminary experimental set of problems. We obtain better results on problems containing inequalities.

An important point is the constraints rotation. The HC4 algorithm is sensitive to multiple occurrences of the variables, and the basic symbolic rewriting defined in section 10.3.2 noticeably increases the number of multiple occurrences. The HC4 propagation performance on the rotated constraints is then not satisfactory. In order to simplify the rotated constraints with respect to the number of multiple occurrences for the variables, we use the Simplify function of Mathematica. The computation time indicated below does not include the time for this treatment, but it is generally negligible compared to the running time for solving (less than a second). Further work includes adding propagators such as [Araya et al., 2010], which better takes into account the symbolic expression of the constraint to improve the propagation.

10.7 Related Works

Our work is related to [Miné, 2006a], in static analysis of programs. Their goal is to compute overapproximations for the traces of a program. The octagons are shown to provide a good trade off between the precision of the approximation and the computation cost. We use their matrix representation and their version of the Floyd-Warshall algorithm.

Propagation algorithms for the difference constraints, also called *temporal*, have already presented in [Dechter et al., 1989, Régin and Rueher, 2005]. They have a better complexity than the one we use, but are not suited to the DBM case, because they do not take into account the doubled variables.

The idea of rotating variables and constraints has already been proposed in [Goldsztejn and Granvilliers, 2008], in order to better approximate the solution set. Their method is dedicated to under-constrained systems of equations.

[2]These benchmarks can be found at `http://www.mat.univie.ac.at/~neum/glopt/coconut/`

10.8 Conclusion

In this chapter, we have proposed a solving algorithm for continuous constraints based on octagonal approximations. Starting from the remark that domains in Constraint Programming can be interpreted as components of a global multidimensional parallelepipedic domain, we have constructed octagonal approximations on the same model and provided algorithms for octagonal CSP transformations, filtering, propagation, precision and splitting. Experimental results on classical benchmarks are encouraging, particularly in the case of systems containing inequalities.

More generally, moving from a Cartesian to a relational representation of domains paves the way to a number of new propagation and solving alternatives, while keeping the main CP principles. In next Chapter, we investigate another new form of propagation and solving, but without introducing new domains: instead, we use the notion of Reduced Products in Abstract Interpretation to combine different existing domains into a new abstract domain, in order to benefit from the strengths of each of them, in particular for propagation.

11

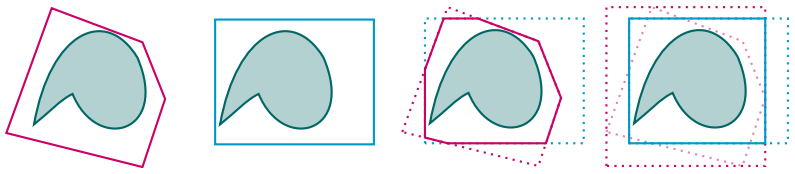
Reduced Products

This chapter is based on a preprint based on a joint work with Emmanuel Rauzy, Ghiles Ziat, Marie Pelleau and Antoine Miné. It shows how to use abstract domains to mix different domains, in our example Boxes and Polyhedra. This work is promising since boxes are at the core of constraints on real variables, and polyhedra is the natural domain of linear programming. My contribution was to investigate and experiment with boxes and polyhedra with my (then) intern Emmanuel Rauzy. We also introduce the integer-real reduced product which is formally defined but not experimented yet.

The reduced product, introduced in [Cousot and Cousot, 1979], is a construction to derive a new, more expressive abstract domain, by combining two or more existing ones. Its principle is to use all of the components of the product, to abstract the desired space by the intersection of these representations. Additionally, operations in the reduced product apply a reduction operator to communicate information between the base domains and thus improving the precision. This feature makes the reduced product a very powerful construction: it is not necessary to redefine all operations on the product, the only required definition being the reduction operation.

Intuitively, Reduced Product can combine the specific consistencies of each of the domains. Figure 11.1 gives an example: it shows the results of a reduced product applied between the polyhedra and boxes. The solution space (in green) is approximated using the polyhedra abstract domain on one side 11.1(a) and using the boxes abstract domain on the other 11.1(b). The informations are then shared using the reduced product. The reduced product first transforms the polyhedron into a box by computing its bounding box, (this operation is made very easy by the simplex representation) and then the box into a polyhedron (this step is straightforward as boxes are particular cases of polyhedra). Then the reduction is performed for each abstract element: We propagate constraints from the box into the polyhedron (this step induces no loss of precision and gives a polyhedron) and symmetrically from the polyhedron to the box which gives an over-approximation of their intersection. 11.1(c). Note that in this example, both abstract elements are reduced, but applying the reduced product does not necessarily change both or even either one of the abstract elements.

In the following, we first introduce the polyhedra abstract domain in section 11.1, and combine it with the boxes domain introduce in Chapter 8 to form a Reduced Product where the linear constraints are solved in the polyhedra and the non-linear ones in the boxes. Then, we also present the general scheme of a Reduced Product combining integer boxes and real boxes, where mixed constraint problems can be solved.



(a) Polyhedra (b) Boxes (c) Reduced Products

Figure 11.1: A reduced product for the Box-Polyhedra abstract domains.

11.1 A New Relational Abstract Domain: Polyhedra

The polyhedra domain $\mathbb{P}$ [Cousot and Halbwachs, 1978] abstracts sets as convex closed polyhedra.

Polyhedron Given a set of linear constraints $\mathcal{P}$, the convex set of $\mathbb{R}^n$ points satisfying all the constraints in $\mathcal{P}$ is called a convex polyhedron.

Modern implementations [Jeannet and Miné, 2009] generally follow the "double description approach" and maintain two dual representations for each polyhedron: a set of linear constraints and a set of generators. A generator is either a vertex or a ray of the polyhedron. A ray corresponds to a vector along which, starting from any vertex of the polyhedron, any point is part of the polyhedron. However, in practice, the polyhedra used in a constraint solver do not have rays, given that they are bounded.

Figure 11.2 illustrates the different representations for a same polyhedron. The graphical representation 11.2(a), the set of linear constraints 11.2(b) and, the generators and the maximal distance between two generators 11.2(c).

Operators are generally easier to define in one representation or the other. We define the initialization, and the consistency of a polyhedron on the set of linear constraints. The size function is defined on generators and the splitting operator relies on both representations.

The abstract consistency in polyhedra is an important matter. In the following, we will consider a weak form of consistency for polyhedra: the non-linear constraints are ignored (not propagated), only the linear constraints are considered.

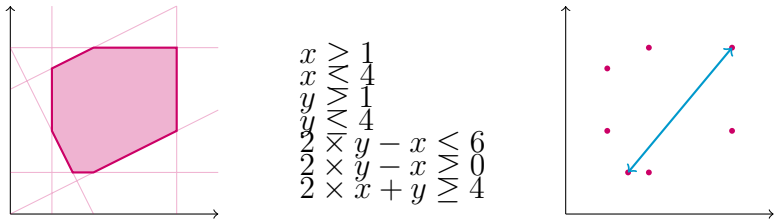
Polyhedral consistency Let $\mathcal{C}_l$ be a set of linear constraints, $\mathcal{C}_{nl}$ a set of non-linear constraints, and $\mathcal{C} = \mathcal{C}_l \cup \mathcal{C}_{nl}$. The consistent polyhedron for $\mathcal{C}$ is the smallest polyhedron including the solutions of $\mathcal{C}_l$.

With this weak definition, the consistent polyhedron given a set of constraints always exists and is unique. This simple consistency definition is sufficient for using the polyhedron in the Box-Polyhedra Reduced Product. Note that higher level consistencies could be defined to propagate non-linear constraints, using for instance quasi-linearization [Miné, 2006b], linearization for the polynomial constraints [Maréchal et al., 2016], generating cutting planes, or computing the hull box, to name a few.

The consistency can be directly computed by adding the constraints to the polyhedron representation: the algorithm which builds the polyhedron computes the generators in one shot, hence the consistent polyhedron for the considered constraints.

Proposition 11.1.1 (Polyhedral consistency) *The polyhedral consistency is complete.*

Proof Assume that it is not complete, then there exists a polyhedron $P \in \mathbb{P}$, a set of constraints $\mathcal{C}$, the corresponding consistent polyhedron P_C, and a solution $s \in P$ of the problem which has been lost, *i.e.,* $s \notin P_C$. Necessarily, one has $C(s)$ for all non-linear constraints C, because non-linear constraints are not considered. Hence there exists a linear constraint C such that $C(s)$ (because s is a solution of the problem) and $\neg C(s)$ because $s \notin P_C$, which gives a contradiction.



(a) A polyhedron. (b) Set of linear constraints. (c) Generators and Precision.

Figure 11.2: Different representations for the polyhedra.

The size function is defined as the maximal Euclidean distance between pairs of vertices. Let $P \in \mathbb{P}$,

$$\tau_{\mathbb{P}}(P) = \max_{g_i, g_j \in P} ||g_i - g_j||$$

Finally, the splitting operator duplicates the polyhedron into two polyhedra. It then adds a linear constraint in each of the resulting sets representing the polyhedra. Let $(v_1, \ldots, v_n)$ be variable over continuous domains, let $P \in \mathbb{P}$ be a polyhedron, $\sum_{i \in [\![1,n]\!]} \alpha_i v_i$ be the linear constraint corresponding to the split we want to perform and h the value where to split. The splitting operator is:

$$\begin{aligned} \oplus_{\mathbb{P}}(P) = \{ \; & P \cup \{\textstyle\sum_{i \in [\![1,n]\!]} \alpha_i v_i \leq h\}, \\ & P \cup \{\textstyle\sum_{i \in [\![1,n]\!]} \alpha_i v_i > h\}\} \end{aligned}$$

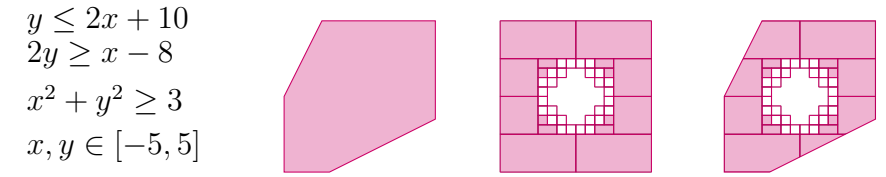
Obviously, the resulting polyhedra cover entirely the initial one. In practice, we choose the constraint to add to be such that both of the resulting polyhedra are, to a certain extent, equally sized: we compute the segment between the two most distant generators and add the constraint corresponding to its perpendicular bisector.

11.2 Reduced Products

We use only two abstract domains and augment it by introducing a hierarchy between the two components of the product. Instead of duplicating the propagation in both domains for each constraint, we make one of the component a specialized domain, dedicated to one type of constraints only, and the second, a default domain which will apply to the other constraints. We will refer afterwards to these as the default and the specialized domain. This configuration avoids unnecessary computations on the constraints that are not precise nor cheap to represent on some domains (*e.g.*, $x = cos(y)$ with the domain of polyhedra or $x = y$ with the domain of intervals). Nevertheless, we still keep the modular aspect of the reduced product: we can still add a new domain over the top of a reduced product by defining a reduction operator with each existing component, and an attribution operator which specifies for a constraint c if the new specialized domain is able to handle it.

Definition Let A_1, A_2 be two abstract domains ordered with inclusion and $\mathcal{C}$ a list of constraints, we define the product $A_1 \times A_2$ where A_2 is the specialized domain and A_1 the default one.

- The product $A_1 \times A_2$ is an abstract domain ordered by component-wise comparison. Let x_1, y_1 be two elements of A_1 and x_2, y_2 be two elements of A_2, then $(x_1, x_2) \subseteq (y_1, y_2) \iff x_1 \subseteq y_1 \wedge x_2 \subseteq y_2$.
- A reduction operator is a function $\rho : A_1 \times A_2 \to A_1 \times A_2$ such that $\rho(x_1, x_2) = (y_1, y_2) \implies (y_1, y_2) \subseteq (x_1, x_2)$.



(a) Linear and non-linear constraints (b) Consistent polyhedron (c) Solving the non-linear part (d) Intersection of the domains

Figure 11.3: Example of the Reduced product of Box-Polyhedra

- Let c be a constraint, an attribution operator κ is a predicate $\kappa : C \rightarrow \{true, false\}$ such that $\kappa(c) = true$ if the domain A_2 is well suited for the constraint c.

Using the reduced product, the propagation loop slightly differs from the usual one in CP: the solving process first uses only the the specialized domain and the constraints that are tied to it according to the attribution operator. For each obtained solutions, we then solve the remaining part of the problem using the default domain (if the default domain is itself a reduced product, we repeat the same process). Each time we propagate the constraints, then the reduced product with the specialized element is applied on the resulting element.

Consider A_1, A_2 two abstract domains ordered with inclusion, and $A = A_1 \times A_2$ the product abstract domain with A_1 the default domain and A_2 the specialized one. The consistency in A is defined as follow:

Product-consistency Let $\mathcal{C}$ be a set of constraints such that $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ with $\mathcal{C}_1 = \{C \in \mathcal{C} \mid \neg\kappa(C)\}$, the constraints for the default domain A_1, and $\mathcal{C}_2 = \{C \in \mathcal{C} \mid \kappa(C)\}$, the constraints for the specialized domain A_2. The product consistent element for $\mathcal{C}$ is the product of the smallest A_1 element including the solutions of $\mathcal{C}_1$ with the smallest A_2 element including the solutions of $\mathcal{C}_2$.

Proposition 11.2.1 (Product-consistency) *The Product-consistency is complete.*

Proof Assume that it is not complete, then there exists a product $P \in A$, a set of constraints $\mathcal{C}$, the corresponding consistent product P_C, and a solution $s \in P$ of the problem which has been lost, *i.e.,* $s \notin P_C$. Then, there exists a constraint C such that $C(s)$ (because s is a solution of the problem) and $\neg C(s)$ because $s \notin P_C$, which gives a contradiction.

11.3 The Box-Polyedra Reduced Product

The Box-Polyhedra abstract domain $\mathbb{BP}$ is particularly useful when solving problems which involve both linear and non-linear constraints. Here, the Polyhedra domain is used as a specialized domain working only on the linear subset of the problem. We use the Box domain as the default domain to solve the non-linear part of the problem. More precisely, let $\mathcal{C}_l$ bet the set of linear constraints and V_l the set of variables appearing in $\mathcal{C}_l$, and let $\mathcal{C}_{nl}$ be the set of non-linear constraints and V_{nl} the set of variables appearing in $\mathcal{C}_{nl}$. We build an over-approximation of the space defined by $\mathcal{C}_l$ with the Polyhedra domain. By construction, this polyhedron is consistent with respect to $\mathcal{C}_l$ once it is created (conjunctions of linear constraints can be expressed with a convex polyhedron with no loss of precision). In short terms, the linear constraints are propagated once and for all at the initialization of the polyhedron. The variables V_{nl} appearing in at least one non-linear constraint are then represented with the box domain and the sub-problem containing only the constraints in $\mathcal{C}_{nl}$ is solved accordingly.

Figure 11.3 gives an example of the Box-Polyhedra abstract domain applied on a problem with both linear and non-linear constraints. Figure 11.3(a) gives the set of constraints, Figure 11.3(b) the consistent polyhedron (for the linear constraints), Figure 11.3(c) the union of boxes solving the non-linear constraints, and Figure 11.3(d) the intersection of both domains elements obtained with the reduced product.

As, by construction, the initial polyhedron is consistent for all the linear constraints of the problem, the operators in the reduced abstract domain $\mathbb{BP}$ are defined only on the box part.

Box-Polyhedra Consistency Let $\mathcal{C} = \mathcal{C}_l \cup \mathcal{C}_{nl}$ with $\mathcal{C}_{nl}$ (resp. $\mathcal{C}_l$) the set of non-linear (resp. linear) constraints. The box-polyhedra consistent element is the product of the smallest consistent box including the solutions of $\mathcal{C}_{nl}$ with the initial polyhedron.

This definition of the consistency is not a weak version of the Product-consistency, as by construction, the initial polyhedron is consistent for the linear constraints ($\mathcal{C}_l$). This consistency is thus complete by Proposition 11.2.1.

Let $X = X_b \times X_p \in \mathbb{BP}$ with X_b the box and X_p the polyhedron. The splitting operator splits on a variable in $V_{nl} = (v_1, \ldots, v_k)$ (in a dimension in X_b):

$$\oplus_{\mathbb{BP}}(X) = \{\oplus_{\mathbb{B}}(X_b) \times X_p\}$$

Finally, the size function is:

$$\tau_{\mathbb{BP}}(X) = \tau_{\mathbb{B}}(X_b)$$

Thus, we take advantage of both the precision of the polyhedra and the generic aspect of the boxes. Moreover, we bypass the disadvantages bound to the use of the polyhedra. We do not need any kind of constraint linearization and we reduce the propagation/split phase to one step.

Proposition 11.3.1 (Completness of solving with $\mathbb{BP}$) *The solving method in Algorithm 9.2 with the $\mathbb{BP}$ abstract domain is complete.*

Proof The Box-polyhedra consistency is complete then by Definition 10 in [Pelleau et al., 2013] the abstract solving method using the $\mathbb{BP}$ abstract domain is complete.

11.3.1 Experimental Results

We have implemented the method presented above in the AbSolute constraint solver. It implements the solving method presented in [Pelleau et al., 2013], which is based on abstract domains. This solver is written in OCaml, and uses the APRON numerical abstract domain library [Jeannet and Miné, 2009]. The implementation can be found on GitHub `https://github.com/mpelleau/AbSolute`. Its current version features a generic implementation of the propagation loop with reduced products, the heuristic for the mixed box-polyhedra abstract domain, and a visualization tool (not described in this paper). Also, note that AbSolute does not enjoy classical classical CP techniques and only rely on the method presented above.

11.3.2 Experiments

Experiments have been compared with the `defaultsolver` of Ibex 2.3.1 [Chabert and Jaulin, 2009], on a computer equipped with an Intel Core i7-6820HQ CPU at 2.70GHz 16GB Ram running the GNU/Linux operating system.

Problems have been selected from the Coconut benchmark[1].

We have selected problems with only linear constraints, only non-linear constraints or both as this is the main focus of our work. Also, we have fixed the precision (the maximum size of the solutions, w.r.t. to the size metric of the employed domain) to $1e^{-3}$ for all problems for both solvers, which is a reasonable precision for this kind of problems.

The first three columns in Table 1 describe the problem: name, number of variables and number of constraints. The next columns indicate the time and number of solutions (*i.e.,* abstract elements) obtained with AbSolute(col. 4 & 6) and Ibex (col. 5 & 7).

[1] Available at `http://www.mat.univie.ac.at/~neum/glopt/coconut/`

Table 11.1: Comparing Ibex and AbSolute with the interval domain

problem	#var	#ctrs	time, AbS	time, Ibex	#sols AbS	#sols, Ibex
booth	2	2	**3.026s**	26.36s	**19183**	1143554
exnewton	2	3	**0.158s**	26.452s	**14415**	1021152
supersim	2	3	0.7s	**0.008s**	**1**	**1**
aljazzaf	3	2	**0.008s**	0.02s	**42**	43
bronstein	3	3	0.01s	**0.004s**	8	**4**
eqlin	3	3	0.07s	**0.008s**	**1**	**1**
cubic	2	2	**0.007s**	0.009	9	**3**
hs23	2	6	2.667s	**2.608s**	**27268**	74678
powell	4	4	**0.007s**	0.02	4	**1**
combustion	10	10	**0.007s**	0.012s	**1**	**1**

11.3.3 Analysis

We must define here the concept of solution for both solvers. Ibex and AbSolute try to entirely cover a space defining by a set of constraints with a set of element. In Ibex, these elements are only boxes. In AbSolute, these are both polyhedra and boxes. Thus the performance metric we adopt is, given a minimum size for the output elements, the number of elements required to cover the solution space[2]. Hence, the less elements we have, the faster the computation will be. Furthermore, having a few elements makes the reuse of the ouput easier.

According to this metric, on most of these problems, AbSolute outperforms or at least compete with Ibex in terms of time and solution number. We justify the good results obtained by our method by two main facts, firstly, the linear constraints solving is almost immediate with our method. For example, the `booth` problem is composed by one linear constraint and one non-linear constraint. the linear constraint is directly representable with a polyhedron and thus, the solving process immediately finds the corresponding solution, while working only with boxes makes the search goes through many splits before obtaining a set of boxes smaller than the required precision. Secondly, after each split operation, AbSolute checks for each resulting abstract elements if it satisfies the set of constraints. If it is the case, the propagation/split phase stops for this element. This makes possible to stop the computation as soon as possible. The `defaultsolver` of Ibex does not do this verification and thereby goes much slower. This makes our implementation competitive on problems with only non-linear constraints. For the `exnewton` problem which only involve non-linear constraints, (the resolution thus only uses boxes), we also obtain very good performances. Note that disabling the satisfaction verification in AbSolute leads to results with the same number of solutions as for Ibex, but still with a gain in time. For instance with this configuration, on `exnewton` without the satisfaction check, we obtain 1130212 elements in 9.032 seconds.

11.4 The Integer-Real Reduced Product

To deal with mixed problems, *e.g.,* problems with both integer and real variables, we define a mixed integer-real abstract domain $\mathbb{M}$. It is ordered by inclusion. Let $(v_1, \ldots, v_n)$ the variables. Among them, we distinguich $(v_1, \ldots, v_m), m \in [\![1, n]\!]$ the integer variables and $(v_{m+1}, \ldots, v_n)$ the real variables. The abstract domain of mixed boxes assigns to integer variables an integer interval and to real ones a real interval with floating-point bounds.

[2] Note that AbSolute discriminates the elements in two categories: the ones such that all of the points in them satisfy the constraints, and the one where it is not the case. We have not showed this information in the experiments as Ibex does not do any kind of discrimination on the resulting elements.

$$\mathbb{M} = \{\prod_{i=1}^{m} [\![a_i, b_i]\!] \mid \forall i,\ [\![a_i, b_i]\!] \subseteq D_i,\ a_i \leq b_i\} \times \{\prod_{i=m+1}^{n} I_i \mid I_i \in \mathbb{I}, I_i \subseteq D_i\} \cup \{\emptyset\}$$

The consistency for this abstract domain is the Bound-consistency for the integer box, the Hull-consistency for the floating-bound box. If the considered constraint only contains integer (resp. real) variables then the Bound-consistency (resp. Hull-consistency) is applied. If the constraint contains both type of variables, then the quasi-linearization [Miné, 2006b] can be used, the real (resp. integer) variables are replaced by their intervals and the constraint is propagated in the integer (resp. floating-bound) box. Note that there exists no reduced product for this abstract domain, as the two abstract domains do not share any variable.

For sake of simplicity, the splitting operator splits the interval corresponding to the chosen variable. Let $M \in \mathbb{M}$, v_i be the variable we want to split and D_i its domain. If v_i is an integer variable then $D_i = [\![a_i, b_i]\!]$, otherwise $D_i = [a_i, b_i]$. Let $h = \frac{a_i + b_i}{2}$. The splitting operator is:

$$\oplus_m(M) = \begin{cases} \{\ D_1 \times \cdots \times [\![a_i, \lfloor h \rfloor]\!] \times \cdots \times D_n, & \text{if } 1 \leq i \leq m \\ \quad D_1 \times \cdots \times [\![\lceil h \rceil, b_i]\!] \times \cdots \times D_n\} & \\ \{\ D_1 \times \cdots \times [a_i, h] \times \cdots \times D_n, & \text{if } m < i \leq n \\ \quad D_1 \times \cdots \times [h, b_i] \times \cdots \times D_n\} & \end{cases}$$

The precision function corresponds to the size of the largest dimension:

$$\tau_m([\![a_1, b_1]\!] \times \ldots \times [\![a_m, b_m]\!] \times [a_{m+1}, b_{m+1}] \times \ldots \times [a_n, b_n]) = \max_i(b_i - a_i)$$

Choosing a $r < 1$ guarantees that all the integer variables are instantiated when the solvers terminates.

11.5 Conclusion

In this Chapter, we have shown that it is possible to define a constraint solving method based entirely on abstract domains, and introduced a well-defined way of solving problems with several domains together.

The framework we have presented is based on the idea of using an expressive domain able to encode exactly a certain kind of constraints, and a low-cost domain to abstract the constraints that can not be represented exactly (with no loss of precision) in the specialized domain. This allows us to get the best of both domains, while keeping the solver properties. The Box-Polyhedra product, for instance, is particularly adapted to problems with linear and non-linear constraints.

This Chapter also concludes Part III. We have presented a generic framework for constraint solving by abstracting the notion of domains in CP solvers in a similar way as Abstract Interpretation does for concrete domains of programs. We have shown that this framework allowed to define relational domains, such as Octagons, which come with an *ad hoc* consistency, or Polyhedra, and Reduced Product domains which combine abstract domains together. I believe that the notion of abstract domain enhances the expressivity of CP solvers.

IV

Conclusion

12

Conclusion

In this thesis, I have presented a series of contributions in Constraint Programming, organized around two main axes: average-case analysis of constraint algorithms and abstract domains in CP. In both cases, my contribution has been to open CP to other disciplines, specifically Abstract Interpretation, and Applied Mathematics. Relying on my background as a theoretical computer scientist, I developed long-term collaborations with experts in both fields. In Abstract Interpretation, a question which was initially only a matter of curiosity (why can static analyzer mix domains, while CP cannot?) finally became a rich work inspiring many other questions and extensions. I started the collaboration with Antoine Miné in 2012, who is an expert in semantic, and this collaboration was later developed in the Coverif ANR project (2015-2020), which provides a good basis to attack constraint and verification questions.

In average-case analysis, the natural questions arising from CP were already investigated, but mostly by theoretical compute scientists, mathematicians and physicists. I believe that practical computer science needs to get involved in those researches, in order to bring the theoretical results back to practice. On this axis, I was invited by Brigitte Vallée, to participate to the Alea working group, which allowed me to join the Boole ANR project (2009-2013). Based on this project, we managed to find several bridges between theoretical questions and practical issues, and, in collaboration with Xavier Lorca (expert in CP), Danièle Gardy (expert in analytic combinatorics) and Vlady Ravelomanana (expert in graph theory), we now have a perfect spectrum of expertise to attack these questions.

I strongly believe that CP needs to develop its links to other fields, beyond its natural collaboration with Operations Research. For each axis, I present in the following my current work and possible future extensions which are, in my opinion, feasible in the short/mid-term, and then a more prospective research plan. The latter made under the assumption of unlimited resources in time, money and manpower.

12.1 Average-case analysis

On the first axis, three contributions are detailed, two of them on randomized algorithms, and one on a global constrain. Introducing probabilities into local search solvers and complete solvers produces new insights on their behavior, and provide new methods to improve the solving process.

12.1.1 Current and mid-term future works

The analysis of global constraints is currently continued with Xavier Lorca and Giovanni Lo Bianco on a generalization of our work to the global cardinality constraint. This is part of a CNRS PICS project,

where we collaborate in particular with Danièle Gardy (Université de Versaille Saint Quentin) and Vlady Ravelomanana (Université de Paris 7). Danièle Gardy has developed an accurate urn model for the `nvalue` constraint (which restricts the number of values that a sequence of variables can take), based on analytic combinatorics tools. This model allows us to compute the probability that the constraint is consistent.

However, it requires calculations with an exponential number of operations, and we will work on approximations. In the near future, we will also investigate methods to efficiently approximate this result, and, as well as on the `alldifferent` constraint, use it to compute thresholds for freezing propagation. In addition, we turned our efforts to solution counting for global constraints, with is the main topic of Giovanni Lo Bianco's PhD, supervised by Xavier Local and me. We believe that some of the cardinality constraints have a phase transition that can be observed on their variable-value graph models. In collaboration with Vlady Ravelomanana, we are currently working on identifying and characterizing this phase transition.

12.1.2 Further reseach

Working on these topics, I learnt that we should have more of a physicist approach. Physicists do not care whether their equations are solvable or not, they care about explaining observed phenomena. Hence, they make observations, try and find a mathematical model and have to solve it whatsoever. If solving it requires approximations, they approximate. If solving is mathematically impossible, they either work on the mathematics or they simplify their model so that it becomes solvable. In computer science in general, we deeply lack such a methodology and I think it is necessary to develop approximation tools for the combinatorics problems appearing in constraint solving, that we can consider as our "physical phenomenon".

Estimation of speed-ups

This physical approach can be used for another interesting extension of the work on parallel multiwalk speed-up estimations. In collaboration with Philippe Codognet, we plan to devise a method for predicting the speed-up from scratch, that is, without any knowledge on the algorithm distribution. Our observations suggest that the sequential runtime of both local search and complete search are well approximated by a small number of distributions. This could be intensively tested on a wider range of problems. In particular, it is important to know wether the sequential distribution of different instances of a given problem belong to the same family. Then we can devise a method for estimating the sequential distribution based on a limited number of observations, possibly on small instances, and then estimate the parallel speed-up for larger instances. Another question is the development of estimators for the smallest observed duration of the considered local search algorithm, in a given number of runs (the quantity denoted by x_0 in Chapter 6.

Simple machine learning approximations based on the instances features allowed us to infer the most probable distribution for the sequential runtime on a set of similar instances, which in turn allowed us to predict the expected behavior of the solver on a whole benchmark. By combining these approximations with an estimator for x_0 and an hypothesis on the distribution shape, we could greatly improve their accuracy. Since the multiwalk parallel scheme is extremely easy to implement, I think that developing tools predicting their efficiency can spare useless effort in the development of constraint parallel solvers. This could prove very useful when trying to solve multiple instances in a cloud system, where the computation time is rented by the hour.

Global constraint analysis

This question will be declined on two axes: consistency probability and solution counting. On consistency probability, which consists in studying in average the propagation of a constraint, I would like to develop our collaborations with the mathematics-computer science community, in particular on real problems from real solvers, as we did for the `all-different` constraint. This is difficult for many reasons, one of them being the difficulty in finding the good models to study our constraints: such models need both to be reasonably close to the CP reality, but also to be mathematically tractable. A promising object for studying

cardinality constraints are urn models, on which combinatorial analytics provide powerful tools. Another promising idea is to consider the cardinality constraints representation in bipartite variables-values graphs, which can be used both for consistency probability and solution counting. We will use our PICS CNRS project to work in this with Danièle Gardy (expert in analytic combinatorics and urn models) and Vlady Ravelomanana (expert in graph analysis and phase transition in graphs), based on the PhD of Giovanni Lo Bianco, whose supervisors are Xavier Lorca and me. Computing the probability that a constraint is consistent, or approximating its number of solutions, can be useful when tuning the propagation loop (for instance to freeze useless propagations) and also when defining heuristics, as a prediction of promising search areas.

12.2 Constraints and Verification

On this second axis, we have build a new framework for constraint programming which offers many possibilities both to define new abstract domains (*e.g.,* relational) and to mix abstract domains *via* reduced products, which in fact means mixing different solving methods while keeping the solvers properties (soundness or completeness). The new operators we had to introduce (choice and precision) can also be interesting to the verification community, for instance to refine some analysis without defining new domains, as [Miné et al., 2016] did for inductive invariants.

12.2.1 Current and mid-term future works

Current research include, of course, continuing the exploration of new abstract domains and their use in CP, in particular for reduced products. We are currently working, in collaboration with Marie Pelleau (Université de Nice), Ghiles Ziat and Antoine Miné (both Université de Paris 6), on the implementation of the Reduced Product of integer and real boxes. In future work, we also wish to technically improve our solver. The areas of improvement include: split operators for abstract domains, specialized propagators, and improvements to the propagation loop. We built our solver on abstractions in a modular way, so that existing and new methods can be combined together, as is the case for reduced products in AI.

Besides, we are also investigating using abstract domains as constraints, which makes it possible to improve the domain precision without defining an entire new abstract domain (since constraints are easy to add, while abstract domains are complex to refine). I am currently working in collaboration with David Cachera (ENS Rennes - IRISA) on a refinement of a method by [Bygde et al., 2011] for worst-case execution time analysis: we manage to improve loop counters over-approximations by adding constraints corresponding to the guards of the loops. With an expressive constraint language, we can analyze any loop guards provided that it is in the constraint language, and not only linear guards as it was the case before.

12.2.2 Further research

Constraint and Verification is a very active research topic, with many recent contributions and events (the "Constraint in Testing, Verification and Analysis" is now a regular workshop at the CP conference, there has been a masterclass on constraint and verification at the CPAIOR conference in 2015, etc). One of the reasons for this interest is the successful application of SAT/SMT solver to verification. Quite often, the works in this area consist in applying CP methods to verification issues. We tried the reverse approach with a motivation which was initially quite far from what we finally developed: our goal was simply to check how static analyzers could deal with program with float and integer variables, while keeping their high level property (soundness). In the end, the links between AI and CP revealed much richer than we initially thought, and there are many research questions which remain open.

Abstract domains

My research project is based on the development of the Absolute solver. This is financially supported by the ANR project Coverif, allowing me to pursue the collaboration with Ghiles Ziat, Antoine Miné in Paris 6, as well as the other partners, Eric Goubault and Sylvie Putot from Ecole Polytechnique and Michel Rueher, Claude Michel and Marie Pelleau from the University in Nice. A keypoint of abstract domains is that they allow the user to reintroduce a geometrical intuition on the problems, and separate the constraints so that each constraint is solved in the domain best suited for it. Integer domains in particular could greatly be improved, and for now, Absolute does not feature relational integer abstract domains. Yet, the octagon domains could be useful also for the integers, for instance on precedence constraints which are octagonal by nature. We could thus solve scheduling problems by having the precedence constraints filtered in the octagons (in the same spirit as [Dechter et al., 1989] did for temporal constraint networks) and other constraints in classical domains. More generally, abstract domains for global constraints need further investigation. Each time a global constraint comes with an intern data structure (which is true for pretty much all of them), having an *ad hoc* abstract domain makes sense. For instance, cardinality constraints consistencies are expressed on the variable-value graph, which can be seen as a relational abstract domains where the relations are not linear constraint (as for octagons or polyhedra), but the fact that the variables share values. Ultimately, each problem or even each constraint should be automatically solved in the abstract domains which best fit it, as it is the case in AI. A natural future work is thus the development of new abstract domains adapted to specific constraint kinds.

Another axis of research on abstract domains is the elimination technique, one of the main contribution of the PhD of Ghiles Ziat. On continuous constraints, Ghiles has introduced and implemented a new step on constraint abstract domains, called *elimination*, which is complementary to the propagation. Elimination divides the search space into two sub-spaces: the undetermined one and the satisfied one, where the constraints are always true. It is based on a key observation: the satisfied part of a problem is equivalent to the inconsistent part of the complementary problem. Combined with propagation, elimination allows the solver to better exploit the constraints and focus the search at the frontier of the problem, where the constraints are neither always false nor always true. This elimination technique is specific to continuous constraints, and needs to be extended to integer domains. For bound consistency, the adaptation is nearly straightforward. For GAC (or, equivalently, for the finite subset of integers domain), expressing it raises interesting questions. For instance, a Hall subset for an `AllDifferent`constraint is, in some sense, a subset of the search space where there are always solutions (any combination of values are solution). Such "satisfiable cores" are not easily dealt with by solvers - in fact, all the consistency algorithms consist in computing them. Yet, it seems possible to express satisfiable cores with the elimination and differences operator, in some kind of "non-nogood" acquisition. More generally, propagation in relational domains such as the Octagons consist in constraint acquisition (the octagonal constraints being inferred by the propagator). Considering abstract domains as a constraint acquisition tool seems a promising axis of research.

Counting with abstract domains

On another topic, I believe that abstract domains could provide a useful tool for solutions counting (see [Pesant, 2005] for a general framework), which joins this research axis with the previous one. This is at the core of our current work on loop counters analysis, but we could probably generalize this work to combination of constraints. There are two challenges on this topic: first, being able to efficiently count solutions in complex domains (polyhedra for instance, where computing the volume is costly). Second, being able to estimate the contribution of several constraints within the same problem, assuming that we can estimate the number of solutions for each single constraint (as presented above). Then, we could build an abstract domain for each constraint, so that their size approximate the number of solutions, and combine them to have an estimatior of the number of solutions of the whole problems. Solution counting is still a challenge in CP, and is in $\sharp$P is the initial CSP is NP-hard. The abstract domain approach seems quite promising as is offers a toolbox to combine approximations in a sound way. We will investigate these

questions in the future PhD of Simon Durand, whose PhD advisors are David Cachera, Eric Monfroy and me. He has started working on an abstract domain based on integer parametrized intervals (where the variables domain can be $[1..n]$ and not only $[1..10]$, with n a parameter and not a macro as in MiniZinc), which is the first step toward such developments.

In the longer term, another exciting development would be to use some methods form CP in an AI-based static analyzer. Considering abstract domains as a conjunction of constraints is possible for many relational domains (boxes, a possibility that we already exploit in the work with David Cachera on WCET, but also octagons, polyhedra, ellipsoids, etc), which reverses the link between CP and AI that we already investigated. Areas of interest include: decreasing iteration methods, which are more advanced in CP than in AI, based on the great practical knowledge that CP has on the propagation loop tuning, the use of a split operator in disjunctive completion domains, and the ability of CP to refine an abstract element to achieve completeness.

12.3 Conclusion

Constraint Programming has put a huge research effort in the development of efficient solvers. Indeed, CP solvers are now very efficient, and the more than 400 existing global constraints make it possible to tackle difficult, NP-hard problems in many application areas. I think that the core CP methods are now mature enough to be studied in collaboration with other fields. I will rely on my well established collaborations and my background in mathematics and semantic to develop the links presented in this thesis. Abstract Interpretation can help us design more expressive solvers, and gain a geometrical interpretation on the problems. Applied Mathematics will provide tools to understand the combinatorics inside the solvers and, in the long term, make them more efficient and make their tuning easier.

Vitae

Bibliography

[Aida and Osumi, 2005] Aida, K. and Osumi, T. (2005). A Case Study in Running a Parallel Branch and Bound Application on the Grid. In *SAINT '05: Proceedings of the The 2005 Symposium on Applications and the Internet*, pages 164–173, Washington, DC, USA. IEEE Computer Society. 66

[Aiex et al., 2007] Aiex, R., Resende, M., and Ribeiro, C. (2007). TTT Plots: A Perl Program to Create Time-To-Target Plots. *Optimization Letters*, 1:355–366. 67, 70

[Aiex et al., 2002] Aiex, R. M., Resende, M. G. C., and Ribeiro, C. C. (2002). Probability distribution of solution time in grasp: An experimental investigation. *Journal of Heuristics*, 8(3):343–373. 67, 70

[Alba, 2004] Alba, E. (2004). Special Issue on New Advances on Parallel Meta-Heuristics for Complex Problems. *Journal of Heuristics*, 10(3):239–380. 65

[Antoni, 2010] Antoni, J. (2010). *Modéliser la ville: Formes urbaines et politiques de transport*. Collection "Méthodes et approches". Economica. 31

[Apt, 1999] Apt, K. R. (1999). The essence of constraint propagation. *Theoretical Computer Science*, 221. 78, 90, 93, 94

[Araya et al., 2010] Araya, I., Trombettoni, G., and Neveu, B. (2010). Exploiting monotonicity in interval constraint propagation. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI 2010*. 121

[Arbelaez and Codognet, 2012] Arbelaez, A. and Codognet, P. (2012). Massivelly Parallel Local Search for SAT. In *ICTAI'12*, pages 57–64, Athens, Greece. IEEE Computer Society. 66, 76

[Arbelaez and Hamadi, 2011] Arbelaez, A. and Hamadi, Y. (2011). Improving Parallel Local Search for SAT. In Coello, C. A. C., editor, *Learning and Intelligent Optimization, 5th International Conference, LION'11*, volume 6683 of *LNCS*, pages 46–60. Springer. 66

[Arbelaez et al., 2013] Arbelaez, A., Truchet, C., and Codognet, P. (2013). Using Sequential Runtime Distributions for the Parallel Speedup Prediction of SAT Local Search. *Journal Theory and Practice of Logic Programming (TPLP)*. special issue, proceedings of ICLP13, to Appear. 64, 73, 74

[Arbelaez et al., 2016] Arbelaez, A., Truchet, C., and O'Sullivan, B. (2016). Learning sequential and parallel runtime distributions for randomized algorithms. In *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 655–662. 48

[Babai, 1979] Babai, L. (1979). Monte-Carlo Algorithms in Graph Isomorphism Testing. Research Report D.M.S. No. 79-10, Université de Montréal. 64

[Balafrej et al., 2016] Balafrej, A., Lorca, X., and Truchet, C. (2016). A probabilistic-based model for binary CSP. *CoRR*, abs/1606.03894. 49

[Balint and Fröhlich, 2010] Balint, A. and Fröhlich, A. (2010). Improving Stochastic Local Search for SAT with a New Probability Distribution. In Strichman, O. and Szeider, S., editors, *SAT'10*, volume 6175 of *LNCS*, pages 10–15, Edinburgh, UK. Springer. 73

[Balint and Schöning, 2012] Balint, A. and Schöning, U. (2012). Choosing probability distributions for stochastic local search and the role of make versus break. In *SAT*, pages 16–29. 51

[Barthel et al., 2003] Barthel, W., Hartmann, A. K., and Weigt, M. (2003). Solving satisfiability problems by fluctuations: The dynamics of stochastic local search algorithms. *CoRR*, cond-mat/0301271. 46, 48, 51, 52

[Battiti and Brunato, 2005] Battiti, R. and Brunato, M. (2005). Reactive search: machine learning for memory-based heuristics. Technical report, Teofilo F. Gonzalez (Ed.), Approximation Algorithms and Metaheuristics, Taylor & Francis Books (CRC Press). 47

[Battiti and Tecchiolli, 1994] Battiti, R. and Tecchiolli, G. (1994). The reactive tabu search. *INFORMS Journal on Computing*, 6(2):126–140. 47

[Beldiceanu and Simonis, 2016] Beldiceanu, N. and Simonis, H. (2016). Modelseeker: Extracting global constraint models from positive examples. In *Data Mining and Constraint Programming - Foundations of a Cross-Disciplinary Approach*, pages 77–95. 24

[Benhamou, 1996] Benhamou, F. (1996). Heterogeneous constraint solvings. In *Proceedings of the 5th International Conference on Algebraic and Logic Programming*, pages 62–76. 101, 112, 113, 118

[Benhamou et al., 1999] Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J.-F. (1999). Revisiting hull and box consistency. In *Proceedings of the 16th International Conference on Logic Programming*, pages 230–244. 92, 104, 112

[Benhamou and Granvilliers, 2006] Benhamou, F. and Granvilliers, L. (2006). Continuous and interval constraints. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 16. Elsevier. 91

[Berger, 2010] Berger, N. (2010). *Modélisation et résolution en programmation par contraintes de problèmes mixtes continu/discret de satisfaction de contraintes et d'optimisation. (Modeling and Solving Mixed Continuous/Discrete Constraint Satisfaction and Optimisation Problem)*. PhD thesis, University of Nantes, France. 93

[Berger and Granvilliers, 2009] Berger, N. and Granvilliers, L. (2009). Some interval approximation techniques for MINLP. In *Eighth Symposium on Abstraction, Reformulation, and Approximation, SARA 2009, Lake Arrowhead, California, USA, 8-10 August 2009*. 101

[Bertrane et al., 2010] Bertrane, J., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., and Rival, X. (2010). Static analysis and verification of aerospace software by abstract interpretation. In *AIAA Infotech@Aerospace 2010*, Atlanta, Georgia. American Institute of Aeronautics and Astronautics. 97, 99

[Bessière, 2006] Bessière, C. (2006). Constraint propagation. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 3. Elsevier. 90

[Bessiere et al., 2017] Bessiere, C., Koriche, F., Lazaar, N., and O'Sullivan, B. (2017). Constraint acquisition. *Artif. Intell.*, 244:315–342. 24

[Bessière and Régin, 2001] Bessière, C. and Régin, J.-C. (2001). Refining the basic constraint propagation algorithm. In *IJCAI*, volume 1, pages 309–315. 93

[Bessière et al., 2005] Bessière, C., Régin, J.-C., Yap, R. H., and Zhang, Y. (2005). An optimal coarse-grained arc consistency algorithm. *Artificial Intelligence*, 165(2):165–185. 47

[Bessière and van Hentenryck, 2003] Bessière, C. and van Hentenryck, P. (2003). To be or not to be... a global constraint. In *Principles and Practice of Constraint Programming*, pages 789–794. 77

[Bettig and Hoffmann, 2011] Bettig, B. and Hoffmann, C. M. (2011). Geometric constraint solving in parametric computer-aided design. *Journal of computing and information science in engineering*, 11(2):021001. 22

[Betts et al., 2015] Betts, J. M., Mears, C., Reynolds, H. M., Tack, G., Leo, K., Ebert, M. A., and Haworth, A. (2015). Optimised robust treatment plans for prostate cancer focal brachytherapy. In *Proceedings of the International Conference on Computational Science, ICCS 2015, Computational Science at the Gates of Nature, Reykjavík, Iceland, 1-3 June, 2015, 2014*, pages 914–923. 23, 24

[Biere, 2008] Biere, A. (2008). Adaptive restart strategies for conflict driven SAT solvers. In *Theory and Applications of Satisfiability Testing - SAT 2008, 11th International Conference, SAT 2008, Guangzhou, China, May 12-15, 2008. Proceedings*, pages 28–33. 52

[Blum and Roli, 2003] Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308. 45

[Bordeaux et al., 2009] Bordeaux, L., Hamadi, Y., and Samulowitz, H. (2009). Experiments with Massively Parallel Constraint Solving. In Boutilier, C., editor, *Proceedings of IJCAI 2009,21st International Joint Conference on Artificial Intelligence*, pages 443–448. 66, 76

[Boussemart et al., 2004] Boussemart, F., Hemery, F., Lecoutre, C., and Sais, L. (2004). Boosting Systematic Search by Weighting Constraints. In *ECAI'2004*, pages 146–150. 47

[Braunstein et al., 2005] Braunstein, A., Mézard, M., and Zecchina, R. (2005). Survey propagation: An algorithm for satisfiability. *Random Struct. Algorithms*, 27(2):201–226. 47, 48

[Bygde et al., 2011] Bygde, S., Ermedahl, A., and Lisper, B. (2011). An efficient algorithm for parametric WCET calculation. *Journal of Systems Architecture - Embedded Systems Design*, 57(6):614–624. 135

[Cai et al., 2012] Cai, S., Luo, C., and Su, K. (2012). CCASAT: Solver Description. In *SAT Challenge 2012: Solver and Benchmark Descriptions*, volume B-2012-2 of Department of Computer Science Series of Publications B, pages 13–14, University of Helsinki. 73

[Caniou et al., 2011] Caniou, Y., Codognet, P., Diaz, D., and Abreu, S. (2011). Experiments in parallel constraint-based local search. In *EvoCOP'11, 11th European Conference on Evolutionary Computation in Combinatorial Optimisation*, Lecture Notes in Computer Science, Torino, Italy. Springer Verlag. 37

[Caromel et al., 2007] Caromel, D., di Costanzo, A., Baduel, L., and Matsuoka, S. (2007). Grid'BnB: A Parallel Branch and Bound Framework for Grids. In *proceedings of HiPC'07, 14th international conference on High performance computing*, pages 566–579. Springer Verlag. 66

[Chabert and Jaulin, 2009] Chabert, G. and Jaulin, L. (2009). Contractor programming. *Artificial Intelligence*, 173:1079–1100. 92, 127

[Chabert et al., 2009] Chabert, G., Jaulin, L., and Lorca, X. (2009). A constraint on the number of distinct vectors with application to localization. In *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, pages 196–210. 101

[Charles M. Grinstead, 1997] Charles M. Grinstead, J. L. S. (1997). *Introduction to Probability: Second Revised Edition*. AMS. 53

[Chi et al., 2008] Chi, K., Jiang, X., Horiguchi, S., and Guo, M. (2008). Topology design of network-coding-based multicast networks. *IEEE Transactions on Parallel and Distributed Systems*, 19(5):627–640. 27

[Chu et al., 2009] Chu, G., Schulte, C., and Stuckey, P. J. (2009). Confidence-Based Work Stealing in Parallel Constraint Programming. In Gent, I. P., editor, *CP 2009, 15th International Conference on Principles and Practice of Constraint Programming*, volume 5732 of *Lecture Notes in Computer Science*, pages 226–241. Springer. 66, 76

[Codognet and Diaz, 2001a] Codognet, P. and Diaz, D. (2001a). Yet another local search method for constraint solving. In *SAGA*, volume 2264 of *Lecture Notes in Computer Science*, pages 73–90. Springer. 36

[Codognet and Diaz, 2001b] Codognet, P. and Diaz, D. (2001b). Yet Another Local Search Method for Constraint Solving. In *proceedings of SAGA'01*, pages 73–90. Springer Verlag. 73

[Codognet and Diaz, 2003] Codognet, P. and Diaz, D. (2003). An Efficient Library for Solving CSP with Local Search. In Ibaraki, T., editor, *MIC'03, 5th International Conference on Metaheuristics*. 73

[Cousot and Cousot, 1977a] Cousot, P. and Cousot, R. (1977a). Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252. 97, 98

[Cousot and Cousot, 1977b] Cousot, P. and Cousot, R. (1977b). Static determination of dynamic properties of generalized type unions. In *Proceedings of an ACM conference on Language design for reliable software*, pages 77–94, New York, NY, USA. ACM. 93

[Cousot and Cousot, 1979] Cousot, P. and Cousot, R. (1979). Systematic design of program analysis frameworks. In *Proceedings of the 6th ACM SIGACT-SIGPLAN symposium of Principles of Programming Languages*, pages 269–282. 123

[Cousot and Cousot, 1992] Cousot, P. and Cousot, R. (1992). Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547. 98, 99

[Cousot and Halbwachs, 1978] Cousot, P. and Halbwachs, N. (1978). Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 84–96. 97, 99, 104, 124

[Crainic and Toulouse, 2002] Crainic, T. and Toulouse, M. (2002). Special Issue on Parallel Meta-Heuristics. *Journal of Heuristics*, 8(3):247–388. 65

[Crainic et al., 2004] Crainic, T. G., Gendreau, M., Hansen, P., and Mladenovic, N. (2004). Cooperative Parallel Variable Neighborhood Search for the -Median. *Journal of Heuristics*, 10(3):293–314. 65

[David and Nagaraja, 2003] David, H. and Nagaraja, H. (2003). *Order Statistics*. Wiley series in probability and mathematical statistics. Probability and mathematical statistics. John Wiley. 64, 66, 69

[de Kergommeaux and Codognet, 1994] de Kergommeaux, J. C. and Codognet, P. (1994). Parallel Logic Programming Systems. *ACM Computing Surveys*, 26(3):295–336. 66

[De Raedt et al., 2010] De Raedt, L., Guns, T., and Nijssen, S. (2010). Constraint programming for data mining and machine learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)*, pages 1671–1675. 22

[Debruyne and Bessière, 2001] Debruyne, R. and Bessière, C. (2001). Domain filtering consistencies. *J. Artificial Intelligence Research*, 14:205–230. 78

[Dechter et al., 1989] Dechter, R., Meiri, I., and Pearl, J. (1989). Temporal constraint networks. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning*. 121, 136

[di Tollo et al., 2015] di Tollo, G., Lardeux, F., Maturana, J., and Saubion, F. (2015). An experimental study of adaptive control for evolutionary algorithms. *Applied Soft Computing*, 35:359 – 372. 47

[Diaz et al., 2012] Diaz, D., Abreu, S., and Codognet, P. (2012). Targeting the cell/be for constraint-based local search. *Concurrency and Computation : Practice and Experience*, 24(6):647–660. 37

[Dickerson et al., 2016] Dickerson, J. P., Manlove, D. F., Plaut, B., Sandholm, T., and Trimble, J. (2016). Position-indexed formulations for kidney exchange. In *Proceedings of the 2016 ACM Conference on Economics and Computation, EC '16, Maastricht, The Netherlands, July 24-28, 2016*, pages 25–42. 23

[D'Silva et al., 2012] D'Silva, V., Haller, L., and Kroening, D. (2012). Satisfiability solvers are static analysers. In *Proceedings of the 19th International Static Analysis Symposium (SAS'12)*, volume 7460 of *Lecture Notes in Computer Science*. Springer. 98

[du Boisberranger et al., 2011] du Boisberranger, J., Gardy, D., Lorca, X., and Truchet, C. (2011). A probabilistic study of bound consistency for the alldifferent constraint. Research report hal-00588888. 80

[Dury et al., 1999] Dury, A., Le Ber, F., and Chevrier, V. (1999). A reactive approach for solving constraint satisfaction problems. In *Intelligent Agents V: Agents Theories, Architectures, and Languages*, pages 397–411. Springer. 29

[Eadie, 1971] Eadie, W. (1971). *Statistical Methods in Experimental Physics*. North-Holland Pub. Co. 72

[Fages et al., 2014] Fages, J.-G., Chabert, G., and Prud'homme, C. (2014). Combining finite and continuous solvers. *Computing Research Repository (CoRR)*, abs/1402.1361. 93

[Floyd, 1962] Floyd, R. (1962). Algorithm 97: Shortest path. *Communications of the ACM*, 5(6). 114

[Freuder, 1997] Freuder, E. C. (1997). In pursuit of the holy grail. *Constraints*, 2(1):57–61. 22

[Gaspero and Schaerf, 2003] Gaspero, L. D. and Schaerf, A. (2003). Easylocal++: an object-oriented framework for the flexible design of local-search algorithms. *Softw., Pract. Exper.*, 33(8):733–765. 39

[Gendron and Crainic, 1994] Gendron, B. and Crainic, T. (1994). Parallel Branch-And-Bound Algorithms: Survey and Synthesis. *Operations Research*, 42(6):1042–1066. 66

[Gent et al., 2008] Gent, I. P., Miguel, I., and Nightingale, P. (2008). Generalised arc consistency for the `AllDifferent`constraint: An empirical survey. *Artificial Intelligence*, 172(18):1973–2000. 80

[Glover and Hanafi, 2002] Glover, F. and Hanafi, S. (2002). Tabu search and finite convergence. *Discrete Appl. Math.*, 119(1-2):3–36. 48, 51

[Glover and Laguna, 1997] Glover, F. and Laguna, M. (1997). *Tabu Search*. Kluwer Academic Publishers. 60

[Goldsztejn and Granvilliers, 2008] Goldsztejn, A. and Granvilliers, L. (2008). A new framework for sharp and efficient resolution of ncsp with manifolds of solutions. In *Proceedings of the 14th international conference on Principles and Practice of Constraint Programming (CP '08)*, pages 190–204, Berlin, Heidelberg. Springer-Verlag. 121

[Gomes and Selman, 2001] Gomes, C. P. and Selman, B. (2001). Algorithm Portfolios. *Artificial Intelligence*, 126(1-2):43–62. 64

[Gomes et al., 2000a] Gomes, C. P., Selman, B., Crato, N., and Kautz, H. (2000a). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reason.*, 24(1-2):67–100. 52, 75

[Gomes et al., 2000b] Gomes, C. P., Selman, B., Crato, N., and Kautz, H. A. (2000b). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *J. Autom. Reasoning*, 24(1/2):67–100. 27

[Gonzalez, 2007] Gonzalez, T., editor (2007). *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall / CRC. 65

[Granger, 1992] Granger, P. (1992). Improving the results of static analyses of programs by local decreasing iterations. In *Proceedings of the 12th Conference on Foundations of Software Technology and Theoretical Computer Science*. 99, 104

[Grimmett and Stirzaker, 1992] Grimmett, G. and Stirzaker, D. (1992). *Probability and Random Processes (second edition)*. Oxford Science Publications. 53

[Halbwachs and Henry, 2012] Halbwachs, N. and Henry, J. (2012). When the decreasing sequence fails. In *Proceedings of the 19th International Static Analysis Symposium (SAS'12)*, volume 7460 of *Lecture Notes in Computer Science*. Springer. 99

[Hentenryck, 2013] Hentenryck, P. V. (2013). Computational disaster management. In *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 12–19. 23

[Hervieu et al., 2011] Hervieu, A., Baudry, B., and Gotlieb, A. (2011). Pacogen: Automatic generation of pairwise test configurations from feature models. In *Proceedings of the 22nd International Symposium on Software Reliability Engineering*, pages 120–129. 98

[Hijazi et al., 2015] Hijazi, H. L., Mak, T. W. K., and Hentenryck, P. V. (2015). Power system restoration with transient stability. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 658–664. 17

[Hooker, 2016] Hooker, J. N. (2016). Finding alternative musical scales. In *CP 2016 Proceedings*, pages 753–768. 23

[Hoos and Stütze, 2005] Hoos, H. and Stütze, T. (2005). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann. 67

[Hoos, 1999] Hoos, H. H. (1999). On the run-time behaviour of stochastic local search algorithms for SAT. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI'99)*, pages 661–666, Orlando, Florida. 48, 51

[Hoos and Stützle, 1998] Hoos, H. H. and Stützle, T. (1998). Evaluating Las Vegas Algorithms: Pitfalls and Remedies. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, UAI'98*, pages 238–245. Morgan Kaufmann. 48

[Hoos and Stützle, 1999] Hoos, H. H. and Stützle, T. (1999). Towards a characterisation of the behaviour of stochastic local search algorithms for SAT. *Artif. Intell.*, 112(1-2):213–232. 27

[Hoos and Stützle, 1999] Hoos, H. H. and Stützle, T. (1999). Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artif. Intell.*, 112(1-2):213–232. 74

[Hoos and Stutzle, 2000] Hoos, H. H. and Stutzle, T. (2000). Local search algorithms for SAT: An empirical evaluation. *Journal of Automated Reasoning*, 24(4):421–481. 52, 58

[Ibaraki et al., 2005] Ibaraki, T., Nonobe, K., and Yagiura, M., editors (2005). *Metaheuristics: Progress as Real Problem Solvers*. Springer Verlag. 65

[Jeannet and Miné, 2009] Jeannet, B. and Miné, A. (2009). Apron: A library of numerical abstract domains for static analysis. In *Proceedings of the 21th International Conference Computer Aided Verification (CAV 2009)*. 97, 104, 107, 124, 127

[Jussien et al., 2008] Jussien, N., Rochart, G., and Lorca, X. (2008). Choco: an Open Source Java Constraint Programming Library. In *CPAIOR'08 Workshop on Open-Source Software for Integer and Contraint Programming (OSSICP'08)*, pages 1–10, Paris, France, France. 35

[Katriel, 2006] Katriel, I. (2006). Expected-case analysis for delayed filtering. In *CPAIOR*, Lecture Notes in Computer Science, pages 119–125. Springer. 79, 80

[KhudaBukhsh et al., 2009] KhudaBukhsh, A. R., Xu, L., Hoos, H. H., and Leyton-Brown, K. (2009). SATenstein: Automatically building local search sat solvers from components. In Boutilier, C., editor, *IJCAI*, pages 517–524, Pasadena, California, USA. 47

[Kieffer et al., 2013] Kieffer, S., Dwyer, T., Marriott, K., and Wybrow, M. (2013). Incremental grid-like layout using soft and hard constraints. In *Graph Drawing - 21st International Symposium, GD 2013, Bordeaux, France, September 23-25, 2013, Revised Selected Papers*, pages 448–459. 22

[Knobloch, 2002] Knobloch, E. (2002). *The Sounding Algebra: Relations Between Combinatorics and Music from Mersenne to Euler*, pages 27–48. Springer Berlin Heidelberg, Berlin, Heidelberg. 23

[Krishnamachari et al., 2000] Krishnamachari, B., Xie, X., Selman, B., and Wicker, S. (2000). Analysis of random walk and random noise algorithms for satisfiability testing. *Proceedings of 6th Intl. Conference on the Principles and Practice of Constraint Programming (CP-2000).*, 1894. 48, 52

[Kwartler and Bernard, 2001] Kwartler, M. and Bernard, R. (2001). Communityviz: an integrated planning support system. *Planning Support Systems*. 30

[Lam et al., 2015] Lam, E., Hentenryck, P. V., and Kilby, P. (2015). Joint vehicle and crew routing and scheduling. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, pages 654–670. 22

[Lazaar et al., 2012] Lazaar, N., Gotlieb, A., and Lebbah, Y. (2012). A cp framework for testing cp. *Constraints*, 17(2):123–147. 98

[Lechner et al., 2006] Lechner, T., Ren, P., Watson, B., Brozefski, C., and Wilenski, U. (2006). Procedural modeling of urban land use. In *ACM SIGGRAPH 2006 Research posters*, SIGGRAPH '06, New York, NY, USA. ACM. 30

[Lenstra and Kan, 1981] Lenstra, J. K. and Kan, A. H. G. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227. 22

[Lorca et al., 2016] Lorca, X., Prud'homme, C., Questel, A., and Rottembourg, B. (2016). 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*, pages 636–649. 22

[Luby et al., 1993] Luby, M., Sinclair, A., and Zuckerman, D. (1993). Optimal Speedup of Las Vegas Algorithms. *Information Processing Letters*, 47:173–180. 67

[Maréchal et al., 2016] Maréchal, A., Fouilhé, A., King, T., Monniaux, D., and Périn, M. (2016). Polyhedral approximation of multivariate polynomials using handelman's theorem. In *17th International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 166–184. 124

[Mars et al., 2008] Mars, N., Hornsby, A., and Foundation, D. C. (2008). *The Chinese Dream: A Society Under Construction*. 010 Publishers. 29

[Mehta and van Dongen, 2007] Mehta, D. and van Dongen, M. R. C. (2007). Probabilistic consistency boosts MAC and SAC. In *Proceedings of IJCAI'07*, pages 143–148. 49

[Menasche and Berthomieu, 1983] Menasche, M. and Berthomieu, B. (1983). Time petri nets for analyzing and verifying time dependent communication protocols. In *Protocol Specification, Testing, and Verification*. 113

[Michel et al., 2006] Michel, L., See, A., and Van Hentenryck, P. (2006). Distributed Constraint-Based Local Search. In Benhamou, F., editor, *CP'06, 12th Int. Conf. on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 344–358. Springer Verlag. 66

[Michel et al., 2007] Michel, L., See, A., and Van Hentenryck, P. (2007). Parallelizing Constraint Programs Transparently. In Bessiere, C., editor, *CP'07, 13th Int. Conf. on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 514–528. Springer Verlag. 66

[Michel et al., 2009] Michel, L., See, A., and Van Hentenryck, P. (2009). Parallel and Distributed Local Search in Comet. *Computers and Operations Research*, 36:2357–2375. 66

[Miné, 2004] Miné, A. (2004). *Weakly Relational Numerical Abstract Domains. (Domaines numériques abstraits faiblement relationnels)*. PhD thesis, École Polytechnique, Palaiseau, France. 97, 99, 104, 107

[Miné, 2006a] Miné, A. (2006a). The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100. 97, 104, 112, 113, 114, 121

[Miné, 2006b] Miné, A. (2006b). Symbolic methods to enhance the precision of numerical abstract domains. In *7th International Conference on Verification, Model Checking, and Abstract Interpretation*. 124, 129

[Miné et al., 2016] Miné, A., Breck, J., and Reps, T. W. (2016). An algorithm inspired by constraint solvers to infer inductive invariants in numeric programs. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, pages 560–588. 135

[Moisan et al., 2013] Moisan, T., Gaudreault, J., and Quimper, C. (2013). Parallel Discrepancy-Based Search. In *Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings*, pages 30–46. 65, 76

[Monniaux, 2016] Monniaux, D. (2016). A survey of satisfiability modulo theory. In *Computer Algebra in Scientific Computing - 18th International Workshop, CASC 2016, Bucharest, Romania, September 19-23, 2016, Proceedings*, pages 401–425. 22

[Moore, 1966] Moore, R. E. (1966). *Interval Analysis*. Prentice-Hall, Englewood Cliffs N. J. 89, 112

[Moujahed et al., 2009] Moujahed, S., Simonin, O., and Koukam, A. (2009). Location problems optimization by a self-organizing multiagent approach. *Multiagent and Grid Systems*, 5(1):59–74. 35

[Murphy et al., 2015] Murphy, S. Ó., Manzano, O., and Brown, K. N. (2015). Design and evaluation of a constraint-based energy saving and scheduling recommender system. In *Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings*, pages 687–703. 22

[Nadarajah, 2008] Nadarajah, S. (2008). Explicit Expressions for Moments of Order Statistics. *Statistics & Probability Letters*, 78(2):196–205. 69, 71

[O'Sullivan, 2012] O'Sullivan, B. (2012). Opportunities and challenges for constraint programming. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 2148–2152. AAAI Press. 22

[Pachet and Roy, 2001] Pachet, F. and Roy, P. (2001). Musical harmonization with constraints : a survey. *Constraints*. 17

[Pardalos et al., 1995] Pardalos, P. M., Pitsoulis, L. S., Mavridou, T. D., and Resende, M. G. C. (1995). Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP. In *Parallel Algorithms for Irregularly Structured Problems (IRREGULAR)*, pages 317–331. 65

[Pardalos et al., 1996] Pardalos, P. M., Pitsoulis, L. S., and Resende, M. G. C. (1996). A Parallel Grasp for MAX-SAT Problems. In Wasniewski, J., Dongarra, J., Madsen, K., and Olesen, D., editors, *3rd International Workshop on Applied Parallel Computing, Industrial Computation and Optimization*, LNCS, Lyngby, Denmark. Springer. 65

[Parish and Müller, 2001] Parish, Y. I. and Müller, P. (2001). Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM. 30

[Pelleau et al., 2013] Pelleau, M., Miné, A., Truchet, C., and Benhamou, F. (2013). A constraint solver based on abstract domains. In *Proceedings of the 14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2013)*. 127

[Pelleau et al., 2014] Pelleau, M., Rousseau, L., L'Ecuyer, P., Zegal, W., and Delorme, L. (2014). Scheduling agents using forecast call arrivals at hydro-québec's call centers. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, pages 862–869. 22

[Pelleau et al., 2011] Pelleau, M., Truchet, C., and Benhamou, F. (2011). Octagonal domains for continuous constraints. In *Proceedings of the 17th International Conference on Principles and Practice of Constraint Programming (CP'11)*, volume 6876 of *Lecture Notes in Computer Science*, pages 706–720. Springer-Verlag. 104, 105, 109

[Perron, 1999] Perron, L. (1999). Search Procedures and Parallelism in Constraint Programming. In *CP'99, 5th Int. Conf. on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 346–360. Springer Verlag. 66

[Pesant, 2005] Pesant, G. (2005). Counting solutions of csps: A structural approach. In *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*, pages 260–265. 136

[Pham and Gretton, 2007] Pham, D. N. and Gretton, C. (2007). gNovelty+. In *Solver description, SAT competition 2007*. 65

[Prud'Homme et al., 2014] Prud'Homme, C., Lorca, X., Douence, R., and Jussien, N. (2014). Propagation engine prototyping with a domain specific language. *Constraints*, 19(1):57–76. 93

[Puget, 1998] Puget, J.-F. (1998). A fast algorithm for the bound consistency of `AllDifferent` constraints. In *AAAI/IAAI*, pages 359–366. 80

[Régin et al., 2013] Régin, J.-C., Rezgui, M., and Malapert, A. (2013). Embarrassingly Parallel Search. In Schulte, C., editor, *Proceedings of CP'2013, 19th International Conference on Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 596–610. Springer Verlag. 66

[Régin and Rueher, 2005] Régin, J.-C. and Rueher, M. (2005). *Inequality-sum : a global constraint capturing the objective function*. RAIRO Operations Research. 121

[Rossi et al., 2006a] Rossi, F., van Beek, P., and Walsh, T., editors (2006a). *Handbook of Constraint Programming*. Elsevier. 77, 79, 91, 92

[Rossi et al., 2006b] Rossi, F., van Beek, P., and Walsh, T. (2006b). *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier. 99

[Schöning, 2002] Schöning (2002). A probabilistic algorithm for k -sat based on limited local search and restart. *Algorithmica*, 32(4):615–623. 47, 52

[Schöning, 2007] Schöning, U. (2007). Principles of stochastic local search. In *Unconventional Computation*, pages 178–187. 51, 52

[Schulte and Stuckey, 2008] Schulte, C. and Stuckey, P. J. (2008). Efficient constraint propagation engines. *Transactions on Programming Languages and Systems*, 31(1):2:1–2:43. 112

[Schulte and Tack, 2001] Schulte, C. and Tack, G. (2001). Implementing efficient propagation control. In *Proceedings of the 3rd workshop on Techniques for Implementing Constraint Programming Systems*. 106

[Schulte and Tack, 2009] Schulte, C. and Tack, G. (2009). Weakly monotonic propagators. In *15th International Conference on Principles and Practice of Constraint Programming (CP 2009)*, pages 723–730. 92

[Scott, 2016] Scott, J. (2016). *Other Things Besides Number: Abstraction, Constraint Propagation, and String Variables*. PhD thesis, University of Uppsala. 98

[Scott, 2017] Scott, J. (2017). Other things besides number: Abstraction, constraint propagation, and string variable types. *Constraints*, 22(1):99–100. 98

[Selman et al., 1994a] Selman, B., Kautz, H. A., and Cohen, B. (1994a). Noise Strategies for Improving Local Search. In Hayes-Roth, B. and Korf, R. E., editors, *12th National Conference on Artificial Intelligence (AAAI'94)*, volume 1, pages 337–343, Seattle, WA, USA. AAAI Press / The MIT Press. 46

[Selman et al., 1994b] Selman, B., Kautz, H. A., and Cohen, B. (1994b). Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, pages 337–343, Seattle. 52

[Selman et al., 1992] Selman, B., Levesque, H. J., and Mitchell, D. (1992). A new method for solving hard satisfiability problems. In Rosenbloom, P. and Szolovits, P., editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California. AAAI Press. 52

[Semerjian and Monasson, 2003] Semerjian, G. and Monasson, R. (2003). Relaxation and metastability in a local search procedure for the random satisfiability problem. *Phys. Rev. E*, 67(6):066103. 46, 48, 51, 52

[Shylo et al., 2011] Shylo, O. V., Middelkoop, T., and Pardalos, P. M. (2011). Restart Strategies in Optimization: Parallel and Serial Cases. *Parallel Computing*, 37(1):60–68. 67

[Simon et al., 2012] Simon, B., Coffrin, C., and van Hentenryck, P. (2012). Randomized adaptive vehicle decomposition for large-scale power restoration. In *Proceedings of the 9th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'12)*, volume 7298 of *Lecture Notes in Computer Science*, pages 379–394. Springer-Verlag. 27

[Simonin et al., 2015] Simonin, G., Artigues, C., Hebrard, E., and Lopez, P. (2015). Scheduling scientific experiments for comet exploration. *Constraints*, 20(1):77–99. 17

[Talbi, 2009] Talbi, E. (2009). *Metaheuristics: From Design to Implementation*. Wiley Series on Parallel and Distributed Computing. Wiley. 37

[Tao et al., 2008] Tao, Y., Christie, M., and Li, X. (2008). Through-the-lens scene design. In *Smart Graphics, 8th International Symposium, SG 2008, Rennes, France, August 27-29, 2008. Proceedings*, pages 142–153. 23

[Thakur and Reps, 2012] Thakur, A. and Reps, T. (2012). A generalization of stØAlmarck's method. In *Proceedings of the 19th International Static Analysis Symposium (SAS'12)*, volume 7460 of *Lecture Notes in Computer Science*. Springer. 98

[Tompkins et al., 2011] Tompkins, D., Balint, A., and Hoos, H. (2011). Captain jack: New variable selection heuristics in local search for sat. In Sakallah, K. and Simon, L., editors, *Theory and Applications of Satisfiability Testing - SAT 2011*, volume 6695 of *Lecture Notes in Computer Science*, pages 302–316. Springer Berlin Heidelberg. 60

[Truchet and Assayag, 2011] Truchet, C. and Assayag, G., editors (2011). *Constraint Programming in Music*. ISTE. 22, 23

[Truchet et al., 2010] Truchet, C., Pelleau, M., and Benhamou, F. (2010). Abstract domains for constraint programming, with the example of octagons. *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 72–79. 98

[Truchet et al., 2013] Truchet, C., Richoux, F., and Codognet, P. (2013). Prediction of Parallel Speed-ups for Las Vegas Algorithms. In Dongarra, J. and Robert, Y., editors, *Proceedings of ICPP-2013, 42nd International Conference on Parallel Processing*. IEEE Press. 64, 74

[United-Nations, 2007] United-Nations (2007). Urban and rural areas 2007, world urbanization prospects: The 2007 revision. 29

[Van Hentenryck, 1989] Van Hentenryck, P. (1989). Parallel Constraint Satisfaction in Logic Programming: Preliminary Results of CHIP within PEPSys. In *ICLP'89, International Conference on Logic Programming*, pages 165–180. MIT Press. 66

[van Hentenryck et al., 1992] van Hentenryck, P., Deville, Y., and Teng, C.-M. (1992). A generic arc-consistency algorithm and its specializations. *Artificial Intelligence*, 57. 106

[Van Hentenryck and Michel, 2005] Van Hentenryck, P. and Michel, L. (2005). *Constraint-Based Local Search*. The MIT Press. 66

[van Hoeve, 2001] van Hoeve, W. (2001). The `AllDifferent` Constraint: A Survey. CoRR cs.PL/0105015. 80, 81

[Van Luong et al., 2010] Van Luong, T., Melab, N., and Talbi, E.-G. (2010). Local Search Algorithms on Graphics Processing units. In *Evolutionary Computation in Combinatorial Optimization*, pages 264–275. LNCS 6022, Springer Verlag. 65

[Vanegas et al., 2010] Vanegas, C. A., Aliaga, D. G., Wonka, P., Müller, P., Waddell, P., and Watson, B. (2010). Modelling the appearance and behaviour of urban spaces. 29(1):25–42. 30

[Verhoeven and Aarts, 1995] Verhoeven, M. and Aarts, E. (1995). Parallel Local Search. *Journal of Heuristics*, 1(1):43–65. 64, 65, 67, 71

[Vitter and Flajolet, 1990] Vitter, J. S. and Flajolet, P. (1990). Average-case analysis of algorithms and data structures. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 431–524. 46

[Waddell, 2002] Waddell, P. (2002). Urbansim: Modeling urban development for land use, transportation, and environmental planning. *Journal of the American Planning Association*, 68(3):297–314. 29

[Watson et al., 2008] Watson, B., Muller, P., Wonka, P., Sexton, C., Veryovka, O., and Fuller, A. (2008). Procedural urban modeling in practice. *Computer Graphics and Applications, IEEE*, 28(3):18–26. 30

[Wolfram, 2003] Wolfram, S. (2003). *The Mathematica Book, 5th Edition*. Wolfram Media. 72

[Xie and Davenport, 2010] Xie, F. and Davenport, A. J. (2010). Massively Parallel Constraint Programming for Supercomputers: Challenges and Initial Results. In *CPAIOR'10, 7th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 6140 of *Lecture Notes in Computer Science*, pages 334–338. Springer. 66

UNIVERSITE BRETAGNE LOIRE

UNIVERSITÉ DE NANTES

Habilitation à Diriger des Recherches

Charlotte TRUCHET

Vers une programmation par contraintes moins contrainte

Outils venant de l'Interprétation Abstraite et de l'Analyse en Moyenne pour définir des solveurs expressifs et faciles à utiliser

Towards A Less Constrained Constraint Programming

Using Abstract Interpretation and Average-case Analysis to Define Expressive, Easy to Tune Solvers

Résumé

La programmation par contraintes (CP) fournit des méthodes efficaces pour modéliser et résoudre des problèmes fortement combinatoires. Les solveurs de contraintes sont rapides, mais difficile à utiliser par des non-informaticiens. Dans ce manuscript, nous présentons d'abord une application de la CP à l'urbanisme, et décrivons les contraintes qui portent sur des utilisateurs non-experts des technologies CP. Nos contributions, organisées en deux axes, visent à relaxer une partie de ces contraintes. Premièrement, nous présentons une série d'analyses en moyenne de certains mécanismes internes des solveurs : le random restart en recherche locale, les speed-ups pour la recherche locale parallèle multiwalk et la consistance incrémentale de la contrainte globale `alldifferent`. Ces analyses théoriques fournissent des résultats utiles pour comprendre et régler ces mécanismes. Deuxièmement, nous introduisons un processus de résolution générique sur des domaines plus expressifs que ceux habituellement utilisés en CP, basés sur les domaines abstraits tels que définis en Interprétation Abstraite. Nous présentons plusieurs nouveaux domaines : relationnels (octogones et polyèdres) et combinés (produits réduits des boîtes-polyèdres et des entiers-réels). Le but est de pouvoir résoudre chaque contrainte dans le domaine abstrait qui lui correspond le mieux. Ces collaborations avec deux autres domaines de recherche, les mathématiques appliquées et la sémantique, permettent d'améliorer à la fois l'expressivité des méthodes de CP et leur facilité d'utilisation.

Mots clés

Programmation par contraintes, Interprétation abstraite, Analyse en moyenne, Urbanisme.

Abstract

Constraint Programming (CP) provides efficient methods to model and solve combinatorial problems. Constraint solvers are fast, yet not easy to use for non-computer scientists. In this thesis, we first present a CP application to urban planning, and describe the constraints holding on non-expert users of the CP technology. Our contributions aim at relaxing (part of) these constraints. They are organized on two axes. Firstly, we present a series of average-case analyses of some inner mechanisms of the CP solving process: random restart for local search, speed-ups of parallel multiwalk local search and incremental consistency of the global `alldifferent` constraint. These theoretical analyses provide useful insights to understand, and tune, these mechanisms. Secondly, we introduce a generic solving process on more expressive domains than the usual CP domains, based on the abstract domains as defined in Abstract Interpretation. We present new such abstract domains: relational ones (octagons and polyhedra) and combined ones (boxes-polyhedra and integer-real reduced products). Ultimately, our goal is to have each single constraint solved in the abstract domain best fit for it. These collaborations with two other fields, Applied Mathematics and Semantic, enhance both CP expressivity and ease-of-use, a necessary condition to widen its application spectrum.

Key Words

Constraint programming, Abstract interpretation, Average-case analysis, Urban planning.