Universitat de Lleida
Departament d'Informàtica i Enginyeria Industrial

TESI DOCTORAL

CSP problems as algorithmic benchmarks: measures, methods and models.

Memòria de treball presentada per en **Carles Mateu Piñol** a la *Universitat de Lleida* per a l'obtenció del títol de *Doctor en Informàtica.* El treball contingut en aquesta memòria ha estat realitzat sota la direcció dels Dr. **Ramón Béjar Torres** i Dr. **Cèsar Fernández Camón**.

Lleida, desembre de 2008

No només t'he de dedicar aquesta tesi, t'he de dedicar tot el que sóc, tot el que puc ser.

Acknowledgements

The list of people that should appear here is too long to fit. Too much people, too much gratitute, too scarse is space allocated to give thanks. I must, however, at least list some of the first ones that come to my mind, I will always be in debt with them.

First of all I'm supposed to thank my advisors, only that, for this work I did not had advisors, I was lucky enough to work with friends, César and Ramón, thanks to them for everything.

The rest of the research group goes now, all of them have been there when I needed something, even where there before I even needed anything, thanks to all: Carlos, Felip, Jordi, Josep, Tere.

To all the friends, those who suffer our stress, those who share our joy.

To my family, they made me be me.

Abstract

On Computer Science research, traditionally, most efforts have been devoted to research hardness for the worst case of problems (proving NP completeness and comparing and reducing problems between them are the two most known). Artificial Intelligence research, recently, has focused also on how some characteristics of concrete instances have dramatic effects on complexity and hardness while worst-case complexity remains the same. This has lead to focus research efforts on understanding which aspects and properties of problems or instances affect hardness, why very similar problems can require very different times to be solved.

Research search based problems has been a substantial part of artificial intelligence research since its beginning. Big part of this research has been focused on developing faster and faster algorithms, better heuristics, new pruning techniques to solve ever harder problems. One aspect of this effort to create better solvers consists on benchmarking solver performance on selected problem sets, and, an, obviously, important part of that benchmarking is creating and defining new sets of hard problems.

This two folded effort, on one hand to have at our disposal new problems, harder than previous ones, to test our solvers, and on the other hand, to obtain a deeper understanding on why such new problems are so hard, thus making easier to understand why some solvers outperform others, knowledge that can contribute towards designing and building better and faster algorithms and solvers.

This work deals with designing better, that is harder and easy to generate, problems for CSP solvers, also usable for SAT solvers. In the first half of the work general concepts on hardness and CSP are introduced, including a complete description of the chosen problems for our study. This chosen problems are, Random Binary CSP Problems (BCSP), Quasi-group Completion Problems (QCP), Generalised Sudoku Problems (GSP), and a newly defined problem, Edge-Matching Puzzles (GEMP). Although BCSP and QCP are already well studied problems, that is not the case with GSP and GEMP. For GSP we will define new creation methods that ensure higher hardness than standard *random* methods. GEMP on the other hand is a newly formalised problem, we will define it, will provide also algorithms to build easily problems of tunable hardness and study its complexity and hardness.

On the second part of the work we will propose and study new methods to increase the hardness of such problems. Providing both, algorithms to build harder problems and an in-depth study of the effect of such methods on hardness, specially on resolution time.

Contents

List of Figures

List of Tables

List of Algorithms

1
Introduction

Although traditional research on problem hardness has been focused on understanding the complexity problems on their worst case (NP-completeness, etc.), ongoing efforts are directed towards understanding why some characteristics of problems, while not varying their inherent worst-case complexity have so dramatic effects on typical case hardness, why varying constraint balancing on very similar problems can suppose increases on hardness of several orders of magnitude. Interesting behaviours of problems are that of the existence of a easy-hard-easy phase transition and of the presence of heavy tails.

In engineering, benchmarking and measuring are, and always have been, critical and crucial processes. Performing benchmarks and getting measures helps engineers refine their designs, compare between competing approaches and solutions, and ascertain that changes suppose, really, benefits and enhancements. In AI CSP solver design and building is, apart from an ongoing research effort, also an engineering process. Because of this benchmarking and measuring is an important aspect of CSP solver design. Benchmarking will help designers and developers to spot issues and potential enhancement to their solvers, showing when, by what and, under which conditions, their algorithms underperform, on which kind of problems solvers do better, on what aspects of problems, and in which amount, affect and influence problem resolution.

This work deals also with that aspect of benchmarking that is crucial for having well crafted and well designed problem sets, that of having at our disposal some set of problem families, with well known properties, tunable hardness, easy to generate, and, potentially being really hard to solve. To this end we have studied some problem families, Sudoku Problems, Edge Matching Puzzles and some kind of Random Problems with some interesting characteristics, from a hardness point of view.

To achieve our goals we have designed some methods to increase hardness of those problems, so researchers will have at their disposal a broad set of possible benchmark hardnesses. We have conducted extensive experimentation on that problems to provide with deep understanding of their hardness characteristics. And, in one case, Edge Matching Puzzles, we have totally defined the problem class, providing the first experimental and analytical study of its hardness and complexity, as well as the first empirical studies on its solvability.

Contents

1.1 Motivation and Objectives

Benchmarking against real-world problems, sometimes called in the research community *industrial-crafted*, is, obviously interesting, but, from a research point of view, not enough. To properly benchmark problems we have to be able to reproduce once and again experiments. We also have to be able to perform several, in the hundreds or thousands order, experiments to asymptotically measure our solver behaviour, having the faster solver in one concrete problem instance doesn't guarantee that we have the faster solver for that class of problems.

Concerning the concrete problems included in our test set, they must be as diverse as possible relating to the characteristic we are interested in measure.

Problem homogeneity poses some handicaps when measuring their hardness characterization, leading in some cases, to misunderstand their particular behavior. Additionally, when performance measurements in algorithms is pursued, as it is usually the case, homogeneity will impede us from obtaining useful measures as the effect of characteristics diversity will not be reflected on the results. Even more, if we use problems not very hard for measuring, then differences between solvers may go unnoticed.

From all the above statements it is clearly needed by the problem solving community to have available as much sets as possible of hard problems, easy to generate (it will not have much sense, otherwise, that problems are as hard to generate as to solve them), of varying hardness, of, potentially, high hardness and with very well defined and studied characteristics.

The aim of this work, then, is to facilitate to researchers willing to do benchmarking some problem classes with a deep study on their hardness characteristics, with fast and convenient generation methods, with several hardness magnitudes, easily decided by the researcher on generation time.

1.2 Publications

Some of the results and findings of this thesis have already been published in journals and conference proceedings. Those publications are listed here, in chronological order:

- Béjar, R., Fernández, C., and Mateu, C. Statistical Modelling of CSP Solving Algorithms Performance In *Proceedings of the CP2005.*
- Ansótegui, C., Béjar, R., Fernández, C., Gomes, C., and Mateu, C. The impact of Balance in a highly structured problem domain. In *Proceedings of the AAAI06.*
- Ansótegui, C., Béjar, R., Fernández, C., and Mateu, C. On Balanced CSPs with High Treewidth. In *Proceedings of AAAI'2007.*
- Ansótegui, C., Béjar, R., Fernández, C., and Mateu, C. Hard SAT and CSP instances with Expander Graphs In *International Symposium on Artificial Intelligence and Mathematics, 2008.*
- Ansótegui, C., Béjar, R., Fernández, C., and Mateu, C. Generating Hard SAT/CSP Instances Using Expander Graphs In *Proceedings of the AAAI08.*
- Ansótegui, C., Béjar, R., Fernández, C., and Mateu, C. From High Girth Graphs to Hard Instances. In *Proceedings of the CP2008.*
- Ansótegui, C., Béjar, R., Fernández, C., and Mateu, C. Edge Matching Puzzles as Hard SAT/CSP Benchmarks. In *Proceedings of the CP2008.*
- Ansótegui, C., Béjar, R., Fernández, C., and Mateu, C. How Hard is a Commercial Puzzle: the Eternity II Challenge. In *volume 184 of Frontiers in Artificial Intelligence and Applications - Artificial Intelligence Research and Development. IOS Press.*
- Béjar, R., Fernández, C., and Mateu, C. Bounding the Phase Transition on Edge Matching Puzzles. Submitted to *ISMVL 2009.*

1.3 Contributions

The main contributions of this work can be summarized as follows:

- We did extensive empirical tests and statistical analysis to identify which problem characteristics have more incidence on problem hardness (see [BFM05]). That study provided us with a handful of parameters that have significant relation with problem hardness (significant from a statistical point of view). From that handful set of parameters there was one, treewidth, with the highest impact on hardness as was expected.
- The first structured problem we have studied is Sudoku (Generalised Sudoku Problems, GSP). In GSP we studied how to balance constraints to increase hardness, and that we did through balancing holes

patterns in GSWH problems. As it arises, obviously, from the encoding, constraints and constraintness depend directly, among some other things, from where are holes located on the problem. We started from a poking model called *singly*, that corresponds to the balancing model already studied for Quasigroup With Holes (QWH) problems. From there we defined two more levels of balancing (doubly and fully), both newly defined. We performed then experimentation that concluded, as we expected, that as balance increases, hardness grows sharply. We provided also explanations on why this effect occurs and why is so sharp (published also in [ABF$^+$06]). For all these hole punching methods we provided algorithms and implementations.

- In the random binary CSP research area we considered how we could increase hardness of CSPs. We considered using the sparsest constraint graphs to increase such hardness. We followed two different approaches for this. First we achieved high treewidth on constraint graphs by using algorithms to create such graphs with high expansion, that we did because previous results linked graph high expansion with high treewidth. We used two different algorithms to get higher expansions, one, using regular or quasi-regular generation methods, the other using high girth graphs. Second we balanced domain values occurrences on constraint tuples achieving highly symmetrical constraints that offer no hints to solving heuristics to choose which values are more promising during early levels of solving tree exploration (see [ABFM07]).

- We designed and implemented methods to generate such problems, for any parameter setting available for traditional B CSP model, this way we could compare our models with model B problems. The result shows that such hardened problems are no marginally harder but orders of magnitude harder.

- We considered also a method to increase expansion of incidence graphs for k-SAT and n-ary CSP formulae. This method works by generating bipartite high-girth graphs. For k-SAT it is known that a high graph expansion implies a high resolution with, consequently implying higher time to solve problems. We studied the impact of such method, comparing it to other balancing methods ([BS96], [BDIS05]), with significative gains.

- We also experimented with using our highly expansive bipartite graphs to increase the hardness of some specially hard satisfiable problem instances for resolution based methods, namely the regular k-XORSAT problem ([Jär06]). Modifying the method to build the incidence matrix of the equation system associated with that problem using our graphs we achieved a dramatic hardness increase.

- We studied also Edge Matching Puzzles (GEMP), a NP-complete problem, and formalised its definition, designing 3 variants (GEMP, one-set GEMP-F and two-set GEMP-F), each of different hardness and characteristics. We have also designed generating algorithms for the 3 variants, and for each one we created both always-satisfiable and regular generators.

- We performed an extensive experimental investigation on GEMP hardness, resulting in an empirical location of the phase transition and characterization of hardness (even for always-satisfiable problems, locating a phase transition from easy to hard to easy). To carry that extensive experimentation, and being GEMP a newly defined problem, we had to devise both a SAT and a CSP encoding for those problem families. After initial experimentation we were also able to defined some novel heuristics, specially designed with GEMP in mind, that perform much better than regular heuristics. At last, we also defined and located the phase transition of Edge Matching Puzzles analytically as that kind of problems is not yet profoundly studied. See ([ABFM08a] and specially [ABFM08b]).

1.4 Overview

This work is structured as follows. First chapter is this one, an introduction, an overview of goals and results of the thesis, a list of resulting publications, and this section that gives an broad overview of the structure of the work. First there is a chapter where some common definitions and concepts on CSP, SAT and graph theory are introduced, these concepts will be used through all the rest of the work. A common notation on CSP, constraint graphs, etc. is also presented, this is the notation used on all the rest of the thesis. There is also a section that gives a broad overview of the state of research on CSP benchmarking, explaining which are the common practices when it comes to benchmark or measure algorithms and solvers, highlighting some shortcomings of such practices and introducing the need for harder easily generable problems. The reader can skip this chapter and go directly into some of the later ones, referring back to this introduction as needed, or can read all the definitions if he is not familiar with some of the concepts. The third chapter presents the chosen problems for our research, introduces their most important characteristics, characterizing them and showing how instances of such problems are generated. Following that, in chapter 4, we will give an detailed analysis on where hardness of CSP, or search problems in general, resides, on what defines problem hardness in the worst case or in the typical case. Later on we will present some methods and algorithms that can be used to generate problems harder than the typical instances for each problem family presented, showing how the fine-tuning of some parameters during problem generation translates directly into very hard problem instances, why this happens and how those parameters affect solvers and algorithms to show this behaviour. Then some global conclusions are drawn and we present some potential enhancements as well as future work and research deriving from our results.

2

Preliminary Concepts and Definitions

A beginning is the time for taking the most delicate care that the balances are correct. This every sister of the Bene Gesserit knows.

Princess Irulan
Dune.
Frank Herbert

In this chapter we will introduce some of the concepts and notations that will be used during the rest of this work. Some of this concepts will be used thoroughly during all the work so it will be interesting to keep an eye on them as future references, although readers already familiar with notations and concepts related to CSP problems can easily skip the first sections and refer to it later if needed. Along with CSP concepts some more general concepts on graph theory are also presented with the same aim, to give reader a context and reference to all the uses later in the work. We will also introduce how and when are CSP problems used as benchmarks. We will also present a brief survey on which problems are usually used in competitions and problem libraries and why is it important to have generators for such problems with tunable hardness, and, if possible, tunable characteristics.

Contents

2.1 Basic Constraint Satisfaction Problem Definitions

First we will define what a Constraint Satisfaction Problem is, modelling it as a graph of constraints. But before entering into the formal definition, one more informal and crude approach can be stated as: A Constraint Satisfaction Problems is that problem of finding which values for some questions (variables) fit without violating some restrictions (constraints)[1].

Definition 2.1 (Constraint Satisfaction Network). *A finite* constraint satisfaction network, *$N = (X, D, C)$ is defined as a triplet consisting, a set of n variables, $X = \{x_1, x_2, \ldots, x_n\}$, a set of domains, $D = \{D(x_1), D(x_2), \ldots, D(x_n)\}$, where $D(x_i)$ is the finite set comprising all possible values that can be assigned to variable x_i, and the set of constraints, $C = \{C_1, C_2, \ldots, C_m\}$. Each constraint $C_i = \langle S_i, R_i \rangle$ is defined as a relation R_i over a subset of variables $S_i = \{x_{i_1}, \ldots, x_{i_k}\}$, called the* constraint scope*. The relation R_i may be represented extensionally as a subset of the Cartesian product $d(x_{i_1}) \times \cdots \times d(x_{i_k})$. Elements $\in R_i$ are called* good *tuples, and elements $\in ((d(x_{i_1}) \times \cdots \times d(x_{i_k})) \setminus R_i)$ are called* nogood *tuples.*

When working with random CSP instances it is a common practice to assume that all variable domains are of the same size, $|D_i| > 2 \; \forall i$. This assumption simplifies notation and working without having, usually, any kind of significance regarding results and conclusions.

Definition 2.2 (Assignment). *An* assignment *v for a CSP instance $\langle X, D, C \rangle$ is a mapping that assigns to every variable $x_i \in X$ an element $v(x_i) \in d(x_i)$. An assignment v satisfies a constraint $\langle \{x_{i_1}, \ldots, x_{i_k}\}, R_i \rangle \in C$ iff $\langle v(x_{i_1}), \ldots, v(x_{i_k}) \rangle \in R_i$.*

[1] We can state it also positively as: A Constraint Satisfaction Problems is that problem of finding which values for some questions (variables) fit while always satisfying some rules (constraints)

Definition 2.3 (Solution of a Constraint Satisfaction Network)**.** *A solution of a Constraint Satisfaction Network, $N = (X, D, C)$, is an assignment of values to all the variables in N such that all the constraints in C are satisfied.*

Definition 2.4 (Constraint Satisfaction Problem)**.** *A Constraint Satisfaction Problem consists in finding a solution to a given Constraint Satisfaction Network or proving that such assignment does not exist.*

During the rest of the text we will refer to a Constraint Satisfaction Problem to the Constraint Satisfaction Network together with the algorithms and methods used to solve it.

Definition 2.5 (Binary Constraint Satisfaction Network)**.** *A binary constraint satisfaction network, $C = (X, D, C)$ is defined as a set of n variables, $X = \{x_1, x_2, \ldots, x_n\}$, a set of domains, $D = \{D(x_1), D(x_2), \ldots, D(x_n)\}$, where $D(x_i)$ is the finite set comprising all possible values that can be assigned to variable x_i, and the set of constraints, $C = \{C_1, C_2, \ldots, C_m\}$, where each $C_k = (x_k, x_j)$ is a subset of the Cartesian product $D(x_k) \times D(x_j)$ that specifies allowed combinations of values for the variables x_k and x_j.*

Definition 2.6 (Constraint Graph)**.** *Given a Constraint Network N with n variables is a graph $G = (V, C)$ with V as the set of n vertexes, one for each variable x_i of N, and C as the set of c edges, one between each pair of variables x_i and x_j such that there exists a constraint between them.*

In case of a CSP with non-binary constraints, an edge in the constraint graph indicates that these two variables appear together in the scope of some constraint. So, for any constraint between k variables it will have a set of $k(k-1)/2$ edges. There is also the more general concept of constraint hypergraph, but we will not use hypergraphs in this thesis.

For binary CSP, and also for CSPs with bounded arity, the most general parameter that allows to classify their complexity is the treewidth of its constraint graph, as it will be discussed in Chapter 4.

Definition 2.7 (Treewidth)**.** *A tree-decomposition of a graph $G = (V, E)$ [RS86] is a pair (T, β), where T is a tree (V^T, E^T) and $\beta : V^T \rightarrow 2^V$ such that:*

1. *For every $v \in V$ the set $\{t \in V^T | v \in \beta(t)\}$ is non-empty and connected in T.*

2. *For every $e \in E$ there is a $t \in V^T$ s.t. $e \subseteq \beta(t)$.*

The width of the tree-decomposition is $max\{|\beta(t)| \; |t \in V^T\} - 1$ and the treewidth of G ($tw(G)$) is the minimum width among all tree-decompositions.

Finally, we define the variable and literal incidence graphs for k-SAT and the literal incidence graph for n-ary CSPs, that are the base for the hardening methods for k-SAT and n-ary CSPs that we present in Chapter 5. It is important to note that our definition of literal incidence graph for n-ary CSPs is not equivalent to the existing one recently used to derive some results of worst-case harndess of n-ary CSPs [SS06].

Definition 2.8 (k-SAT incidence graphs)**.** *Given a k-SAT instance F with set of clauses C, set of variables V and set of literals L, $G(F) = (C \cup V, E)$ is its bipartite variable incidence graph such that $(c, v) \in E$ if and only if variable v appears in clause c. $LG(F) = (C \cup L, E)$ is its bipartite literal incidence graph such that $(c, l) \in E$ if and only if literal l appears in clause c.*

Definition 2.9 (CSP literal incidence graph)**.** *Given a CSP instance $P = \langle X, D, C\rangle$, we define the literal incidence graph as the bipartite graph $LG(P) = (NG \cup L, E)$, where for every variable x_i and domain value $j \in dom(x_i)$ there is a vertex (x_i, j) in L and for every nogood tuple $ng_{i_j} = (v_{j_1} = d_1, v_{j_2} = d_2, \ldots, v_{j_k} = d_k)$ associated with a constraint C_i of arity k there is a vertex ng_{i_j} in NG and k edges, one for every pair $(ng_{i_j}, (v_{j_{j'}}, d_{j'}))$.*

2.2 Basic Graph Theory Definitions

Some definitions and basic concepts from graph theory will be used during this thesis, specially in chapter 5.

Definition 2.10 (Undirected Graph)**.** *An* undirected graph *G is a pair (V, E) where V is the set of vertexes and E is the set of undirected edges $\{u, v\}$. The* degree *$d(u)$ of a vertex u is the number of edges with an endpoint in u. A k*-regular graph *is a graph where the degree of any vertex is k.*

Definition 2.11. *A* bipartite graph *G is a pair $(L \cup R, E)$, where L is the left partition and R is the right partition of the set vertexes, such that any edge is of the form (l, r) with $l \in L$ and $r \in R$. A (k_1, k_2)*-regular bipartite graph *is a bipartite graph $(L \cup R, E)$ such the degree of any l from L is k_1 and the degree of any r from R is k_2. Observe that $|L|k_1 = |R|k_2$. We have a $(k_1, -)$-regular bipartite graph if we only fix the degree of vertexes in L to k_1, but the degrees for R are unfixed.*

Definition 2.12 (Girth of a Graph)**.** *The girth of a graph G ($g(G)$) is the length of the shortest circuit in G. If G is acyclic then, by definition, $g(G) = \infty$.*

There is a limit on how large the girth can be, for a graph with V vertexes and minimum degree d. This limit is $2 \log_{d-1}(|V|)$ [DSV03].

Definition 2.13. *We say that a family F of k-regular graphs has high girth if, for some constant* $0 < C < 2$, $\forall G \in F$, $g(G) \geq (C + o(1)) \log_{k-1} |V|$.

Random k-regular graphs have an expected girth slightly greater than 3 [MWW04], but there exist constructions of graphs with higher girth. The construction with the highest girth, $(4/3) \log_{k-1}(|V|)$, is that of [AS88].

Definition 2.14. *The expansion of a subset $X \subseteq V$ in $G = (V, E)$ is defined to be the ratio $|N(X)|/|X|$, where $N(X) = \{w \in V \backslash X \mid \exists v \in X, \{v, w\} \in E\}$ is the set of outside neighbors of X.*

When all the neighbors of X are inside X, we have expansion 0. We consider a set high expanding when its expansion is greater than 1, that means that the set of different outside neighbors of X is larger than X, so it is well connected with the rest of the graph. So, informally we can define an expander graph as a graph where for any, not too big, subset of vertexes S its set of neighbors outside S is bigger than S.

Definition 2.15 (Expander Graph)**.** *We define an expander graph as a graph G=(V,E) that, for any, not too big, subset of vertexes S, its set of neighbors in $V \setminus S$ is bigger than S.*

The following view of a graph as a communication network is also helpful to understand the expansion property. Consider the graph G, where information retained by some vertex propagates, say in 1 unit of time, to neighboring vertexes. Then the expansion measures the quality of G as a communication network. If the expansion is high, information propagates well [DSV03].

We can also classify graphs depending on their particular level of expansion, as in the two following definitions.

Definition 2.16. *An* (α, c)-expander *is a graph (V, E) such that every subset of size at most $\alpha|V|$ has expansion at least c.*

Usually, smaller sets will have better expansion, the limit being for $\alpha \geq 0.5$, where expansion cannot be greater than 1. For the bipartite graphs considered in this thesis we are mainly interested on the expansion of subsets of the left part. So, we have the next definition.

Definition 2.17. *A* left (α, c)-expander *is a bipartite graph $(L \cup R, E)$ such that every subset of L of size at most $\alpha|L|$ has expansion at least c.*

2.3 State of the art on CSP benchmarking.

When dealing with problem solving, specially if used to compare different solvers/algorithms, benchmarks usually resort to measure only one parameter, time. Simple time benchmarking, provides a fair tool to compare solvers between them, time based benchmarks let us check whether new solvers are better, i.e. faster, than existing ones, and measure the degree of such speedup. But there's more than simply time accounting in benchmarking.

Not all search problems are equal, some are more balanced, some have inner structure than solvers can exploit, some others require huge amounts of memory, while others are small. With properly designed benchmark suites, where those characteristics (balance, structure, etc) are well know along with time spent on solving the benchmark problems will allow solver designers to use results to discover weaknesses in their solvers, showing on which problems their solver fails to perform, and, giving a clue on potential optimization areas. Consequently, for a benchmark to be useful to measure time in solving it must be well documented, thoroughly studied and analysed, and must be representative.

So far benchmarking is continually being used by solver and algorithm designers to compare their work with previously existing work, showing this way the achieved progress. Usually two factors are being used as measures for performance, time and backtracks, although sometimes consistency checks is also taken into account see [van05, vLR06] for recent solver benchmarks and competitions.

The first such parameter, time to solve a given problem or set of problems on equal hardware, somewhat captures efficiency of both algorithm and implementation. It gives a ”real world” measure of solver performance, rewarding both, algorithmic enhancements, better algorithms, propagators, heuristics, etc. and implementation details, faster data structures, and every implementation trick.

On the other hand, measures like number of backtracks try to capture solving methods efficiency on the basis that a solver that makes fewer backtracks is using a better solving strategy and, thus, following a straighter path to the solution. Using such measures obviates the effect that implementation has on performance, equating non-algorithmic details from the measurements such as implementation language and optimization tricks. This kind of benchmarks are very useful when comparing very different solvers with really different implementations and running on different hardware.

The last before mentioned measure, number of consistency checks, comes from the general assumption that consistency checks are very expensive operations and should be minimised. This way methods that do partial consistency or that do not require to do consistency checks along their search tree traversal will have better score on such a measure.

2.3.1 Problem collections

One highly used source of benchmark problems are problem collections or libraries, as CSPLIB[GW99] and SATLIB[HS]. A problem library is a repository, usually a web site, that collects, sorts, and standardises a set of problems related to some research community (SAT in SATLIB case, CSPs in CSPLIB), in order to provide wide circulation and access to such problem collections.

These resources provide developers and designers of solvers with a unified collection of problems to use as benchmarks so programs are tested against a common set of problems, equally defined and specified, thus allowing the comparison of results. Such benchmark libraries are, usually, the source of problems used when designing solver competitions. The greater advantage for a research community, as CSP or SAT communities, on having at their disposal one such resource as a problem library, is that this way researchers have a unified view of sample or reference problems, i.e., when one researcher refers to Social Golfer problem (CSPLIB prob010), the rest of the CSP community has access to the same exact problem definition, from a common repository. This greatly simplifies discussion among the research community.

Although having a common repository is important, the value of such repositories will increase if it fulfills some conditions. Summarily, first problems must be clearly and correctly classified. Second, the library must be kept up to date with state of the art problems to solve. It also must provide several problems in several formats, providing also pointers or references to conversion programs and format specification is of great help for researches, having a code repository for dealing with file formats is, definitively a plus.

Problem taxonomies and classification

All problems in a problem library must be correctly classified and a clear taxonomy used. Problem collections should be as comprehensive as possible, covering as much kinds or classes of problems as possible, of course, without leaving the research field area, an unfocused library can become also non functional. They must also give clear definitions, parameters, characteristics, etc. of all problems contained in the library, providing researchers, if possible, of sample problems in as much standardised formats as possible. Libraries must also be updated regularly with new problems, diverse information on contained problems and solving techniques, specially when linked to solver competitions .

Problem collections can be big, so researchers must have at their disposal, an easy and clear method to locate the problems they are interested in. If we look at existing problem libraries for SAT and CSP problems, SATLIB and CSPLIB, we can see their classification criteria detailed in tables 2.1 and 2.2 respectively. Both libraries, as can be seen, use a very different criteria for providing researchers with problems. SATLIB lists a small amount of problem classes, without much classification, giving several instances of each problem. On the other hand, CSPLIB, gives several problems divided on few classes. When dealing with solver competitions we will see that taxonomies used there are different than those used on problem libraries.

Table 2.1: SATLIB problem classification

Problem Class	**Instances available**
Uniform Random-3-SAT	3700
Random-3-SAT Instances and Backbone-minimal Sub-instances	1000
Random-3-SAT Instances with Controlled Backbone Size	40000
”Flat” Graph Colouring, 3 colourable	1700
”Morphed” Graph Colouring, 5 colourable	901
Planning	10
All Interval Series	4
SAT-encoded Quasigroup (or Latin square) instances	22
SAT-encoded bounded model checking instances	13
DIMACS Benchmark Instances	206
SAT Competition Bejing	16

Table 2.2: CSPLIB problem classification

Problem Class	**Number of problems**
Scheduling	14
Design, configuration and diagnosis	13
Bin packing and partitioning	9
Frequency assignment	3
Combinatorial mathematics	19
Games and puzzles	14
Bio-informatics	1

Problem characterization

One really important aspect of problem repositories is providing researchers not only with a succinct problem description and problem data but also with as much metadata as possible, metadata relevant to the research field, obviously. If, for each problem stored on a problem repository, researchers have at their disposal several characteristic parameters, parameters that can help explaining different behaviour between solvers on a given problem, or different performance of a solver on different problems, benchmarks can provide with an interesting insight on performance bottlenecks, possible enhancement areas, etc.

2.3.2 Solver Competitions

Another asset researchers have at their disposal to benchmark and test their solver designs is competitions. Competitions are events, usually held along some of the main conferences, where a contest confronts proposed solvers between them to check which one is faster solving some predefined problems. This competitions serve a dual purpose, first they stimulate competitive development, there is a challenge, trying to be best that other teams, and second, they suppose a real measure to compare effectiveness of solvers with respect to the rest of the work of the research communities.

Being a much more dynamic and periodical event (competitions are usually held annually or biannually[2]), competitions are a powerful tool for researchers. The results of such competitions are published afterwards and become useful surveys of which is the state of the art on solver design (see for example [ALMP08]).

[2] MaxSAT competitions are held every year, together with SAT Conference. SAT competition are held every other year, the year with no SAT competition a similar event, SAT Race is held. CSP competitions have been held, as of today, in 2005, 2006, and 2008, during CPAI a workshop of CP congress.

3

Chosen CSP problems used as benchmarking problems

The worthwhile problems are the ones you can really solve or help solve, the ones you can really contribute something to. ... No problem is too small or too trivial if we can really do something about it.

Richard Feynman
Letter to Koichi Mano

History teaches the continuity of the development of science. We know that every age has its own problems, which the following age either solves or casts aside as profitless and replaces by new ones.

David Hilbert
Mathematical Problems

As we have seen before, there are several problems commonly used for benchmarking. During this work we have narrowed our study to a few ones that we will study in greater detail. Our election is, first, on fully randomly generated problems, that is, problems without any kind of predefined set of rules on whether variables have or not a constraint among them, or on which values are allowed or not, the only permitted rules are those derived on density or size restraints (mean number of constraints between variables, total number of constraints, etc.) or those that we will define later, relative to balance properties of the problem, in order to selectively harden the problem. Then we have chosen some structured problems, namely Quasigroup Completion Problems (QCP), Sudoku Problems (GSP) and Edge Matching Problems (GEMP). Those chosen problems present some aspects that we consider of the utmost importance for our experimentation; first they are easily generable, we can build as many instances as we need, in a very short time and with few computational resources, second, we have a broad range of possible hardnesses and behaviours, and last, instances of those problems are built programmatically, so there exists the possibility of modifying the building algorithm, on *real world* instances taken from real problems, trying to modify their structure is impossible.

Contents

3.1 Problems choseen for this work

Our first election, random problems, is rather obvious, random problems are the easier to generate, and are the most *generic* ones, being a obvious candidate to compare against more structured[1]? problems. Random problems are also good candidates to benchmarking problems because as they lack any kind of structure solvers cannot use any knowledge provided by problem structure as a lever to shorten resolution time, forcing solver designers to use more generic approaches to solving thus avoiding problem-specific solvers.

Two other problem families chosen for our study are two closely related problem families, Latin Square (LS) problems and Sudoku (GS) problems. Those are easily definable and popular problems, Latin Square consists on filling a number matrix with numbers such that there are no repeated numbers in any row or column of the matrix. Sudoku problems are even more popular than Latin Squares, they are based on LS, but defining some contiguous block regions on the matrix and requiring an additional rule to be satisfied, that all numbers inside a block region should also be different. Those two problem sets are very popular on the research community because, even they are so easily defined, they can be serious challenges to solving techniques, and not so big problems cannot be solved by state of the art solvers.

[1] We understand *structure* from the mathematical point of view, that is, a problem has structure if we can define some kind of rules that describe, more or less accurately, the problem behaviour and characteristics; in our case, number of constraints, or which variables have constraints between them, etc.

The last chosen problem for this work, Generic Edge Mapping Puzzles (GEMP), is, somewhat, related to LS and GS, as it is based on a popular board game, that puzzles where players should place pieces that only match one or various other pieces by their edge. These kind of puzzles received a boost on popularity recently with the release of such a game, named Eternity II, that offered a monetary prize (of 2 Million USD)[2] to the first person to solve it. GEMP problems pose a really hard challenge on solvers.....

3.2 Other interesting problems

There are several other problems usually used for benchmarking. In fact, any problem that can be modelled as a Constraint Network or a SAT problem can be used as a benchmark. Create new challenging problems or modelling new real world problems[3] as SAT or CSP problems is an ever ongoing line of research, as new harder challenges are required to test newer and more powerful solvers.

Although not studied on this work, those problems are worth mentioning as they are often used as benchmarks. An outstanding problem family, XOR-SAT problems, will be briefly described here because, when combined with problems generated using the techniques of section 5.3, that combined problems result in some very hard problems.

3.3 Random models

Random problems are, roughly defined, those created by defining, randomly, constraints between randomly chosen pairs of variables on randomly chosen domain values for such variables. Those kind of random generation can be fine tuned using several parameters, namely, number of constraints to define, tightness of constraints between variables, and obviously number of variables and size of variable domains. As we will see such choices can have a noticeable effect on problem hardness as well as in problem behaviour, and we have, also, a set of choices on the method used for choosing constraints, variables and values. Such models can be equally defined for binary CSPs as well as for n-ary CSPs, as usual with CSP research, as most efficient algorithms and methods are developed for binary CSPs and, usually, all n-ary encodings can be transformed to binary CSPs with good results[], we will devote most of our work to binary models.

[2] Unawarded as of this writing

[3] So-called industrial problems

3.3.1 Standard models

First defined random problem generation models can be classified according to which method is used to generate the constraint graph (§2.6) and to how non valid value pairs are chosen. According to such classification we have these four problem kinds, with corresponding parameters:

model A(p_1,p_2) In this model each of the $n(n-1)/2$ possible edges is independently chosen with probability p_1. Then for each of those edges each of the d^2 possible value pairs, is chosen, independently, with probability p_2, as a non valid value.

model B(p_1,p_2) This time we choose exactly $p_1n(n-1)/2$ edges randomly, and for each one we randomly pick exactly p_2d^2 value pairs as no good pairs.

model C(p_1,p_2) We select independently each one of the $n(n-1)/2$ possible edges with probability p_1 and for each one we randomly pick exactly p_2d^2 pairs of values as no good pairs.

model D(p_1,p_2) We randomly select exactly $p_1n(n-1)/2$ edges, and for each selected edge we pick each p_2d^2 value pairs randomly with probability p_2 as being a no good pair.

3.3.2 Flawless models

Above mentioned standard models have an excellent capability to adjust, independently, constraintness and tightness of the problems, but suffer from a deficiency when the problem size grows; all of them tend to flaw their variables, with higher probability as the size increases. As a flawed variable is a variable that can not satisfy any assignment, the problem becomes unsatisfiable, being such a models useless if the probability of having flawed variables is high enough. In order to avoid this deficiency, rather than create new models, some authors propose to revise standard models determining under which conditions such a models don't degenerate [XL00, Smi01] as well as ensuring satisfiability beyond those limits by adding supports [GMP+01]. In our opinion, below mentioned model E (a variant of standard models) is not strictly speaking a non-degenerate model, as we will see that it requires certain constraintness conditions, as in standard models, to avoid having trivially unsolvable problems.

Fortunately, proving that standard models *degenerate* when size grows is straightforward [AKK+97]. Achlioptas et al. employ Markov's inequality to upper bound the satisfaction probability of a problem. Let's denote X the random variable that determines the number of solutions, then,

$$Pr(|X| \geq 1) \leq E[X]$$

being $Pr(|X| \geq 1)$ the probability of having solution. Compute the mean of X is easy for standard models, resulting

$$E[X] = d^n \cdot (1-p_2)^{p_1\binom{n}{2}} = \left(d \cdot (1-p_2)^{\frac{p_1(n-1)}{2}}\right)^n \tag{3.1}$$

Obviously, avoiding $\lim_{n\to\infty} E[X] = 0$ requires

$$d \geq (1-p_2)^{-\frac{p_1(n-1)}{2}} \tag{3.2}$$

So, condition 3.2 must hold for standard models in order to ensure a probability of having solution greater than zero as the number of variables tend to infinity.

It's worth to mention that for k-SAT, the expected number of solutions is

$$E[X] = \left(2 \cdot (1-2^{-k})^r\right)^n$$

where r is the ratio between the number of clauses and the number of variables. Then, k-SAT problems no degenerates as $n \to \infty$ if the number of clauses grow as $O(n)$ or less.

Beyond this simple approach, [GMP+01] point out that insolubility of standard models relies, partly, on the existence of flawed variable, giving some theoretical insight on the probability of the existence of such variables. Gent at al. make 2 assumptions; variables have the same degree - are connected to the same number of variables each other - and the probability that a variable is flawed does not depend on other variables - that's true for model A but not for model B. Despite these assumptions, computations show a good agreement for the predicted probability.

Let's take model A. The probability that a problem has a flawed variable ($Pr_f w$) can be obtained easy, making the above mentioned assumptions, as

$$Pr_{fw} = 1 - \left(1 - \left(1 - \left(1 - p_2^d\right)^{p_1(n-1)}\right)^d\right)^n$$

Clearly, if $\lim_{n\to\infty} Pr_{fw} = 0$ implies that $\lim_{n\to\infty} \left(1 - (1-p_2^d)^{p_1(n-1)}\right)^d = 0$. Considering that $(1+x)^\alpha \simeq 1 + \alpha x$, for small x, and taken log at both sides of the limit, it can be derived

$$d \geq \frac{\log \frac{1-1/d}{p_1(n-1))}}{\log p_2} \simeq \frac{\log\left(-p_1(n-1)\right)}{\log p_2} \tag{3.3}$$

being valid the right side approach when $d >> 1$. Table 3.1 shows the value of flawness probability, for model A, as a function of n, for $p_1 = 0.5$ and $d = 4$ for values of p_2 such that Eq. 3.2 holds, i.e. the number of solutions is close to 1. As can be observed, the effect of flawed variables is negligible for large n.

Table 3.1: Model A flawness probability when $p_1 = 0.5$ and $d = 4$ for values of p_2 such that Eq. 3.2 holds

n	p_2	Pr_{fw}
4	0.842	0.546
10	0.456	0.012
100	0.054	10^{-12}
1,000	0.005	0

So far, we have shown why standard models degenerate, and some approaches about under which conditions such a degeneration may be avoided. Now let's inspect why some, self claimed, flawless models work, starting by model E. In order to determine bounds for model E, Achlioptas et al. work is based again on the expected number of solutions, but using 2 approaches; one already seen before based on the first moment, and a second that uses the method of local maxima. Achlioptas et al. give an approach to $E[X]$ as follows

$$E[X] = d^n \cdot \left(1 - \frac{1}{d^2}\right)^{rn} = \left(d \cdot \left(1 - \frac{1}{d^2}\right)^r\right)^n \tag{3.4}$$

where rn is the total number contraint tuples out of the total $t = d^k \cdot \binom{n}{k}$ for a k-ary CSP. As we will prove, this approach work reasonably well under certain conditions. Nevertheless, it is no hard to find an exact expression for $E[X]$. Model E chooses m nogoods independently random and with repetition, as a fraction p out of the total number of nogoods t. So, for binary CSPs, $m = p \cdot d^2 \cdot \binom{n}{2}$ and our first finding is the number of distinct generated nogoods (m'). Considering the iterative process of picking the nogood pairs, it is easy to show that after the i-th picked nogood, the number of distinct chosen nogoods (λ_i) can be expressed as the following recursive equation

$$\lambda_{i+1} = 1 + \lambda_i \cdot (1 - 1/t) \tag{3.5}$$

with $\lambda_0 = 0$. Defining the following generation function

$$L(x) = \sum_{i=0}^{\infty} x^i \lambda_i$$

and taking $(1 - 1/t) = a$, we can derive from 3.5

$$\begin{aligned} L(x) &= \frac{1}{(1-x)(1/x-a)} = \frac{x}{1-a}\left(\frac{1}{1-x} - \frac{a}{1-ax}\right) \\ &= \frac{x}{1-a}\left(1 + x + x^2 + \ldots - a - a^2x - a^3x^2 - \ldots\right) \end{aligned}$$

so,

$$\lambda_i = \frac{1-a^i}{1-a} = t \cdot \left(1 - (1 - 1/t)^i\right)$$

Table 3.2: Values of p that give $E[X] = 1$ (p_{crit})

n, d	p_{crit} for Eq. 3.4	p_{crit} for Eq. 3.7
10,3	0.2303	0.2286
10,4	0.2983	0.2967
15,7	0.2751	0.2747
100,3	0.0209	0.0209
100,10	0.0463	0.0462
500,15	0.0108	0.0108

Then, after picking m nogoods, the number of distinct nogoods, m', will be

$$m' = \lambda_m = t \cdot (1 - (1 - 1/t)^m) \tag{3.6}$$

Now, we are able to express exactly the mean number of solutions as

$$E[X] = d^n \cdot \prod_{i=1}^{\binom{n}{2}} \left(1 - \frac{m'}{t - i - 1}\right) \tag{3.7}$$

As observed from Eq. 3.6, $m' \simeq m$ for $t >> 1$, being so whether d or n are large, or both. Under such conditions, Eq. 3.4 and Eq. 3.7 are equivalent as shown in Figure 3.1. In this Figure, we plot the relative error between both expressions of the expected number of solutions (as percentage) against p, where dashed lines depict where $E[X] = 1$ according to Eq. 3.7. It can be noted that the relative error decreases as the domain grows or so it does the number of variables. Then, for large n it turns out that the approach of Achlioptas et al. seems accurate. Employing the first moment method as in 3.2, from Eq. 3.4, we obtain

$$r \leq \frac{\log(1/d)}{\log(1 - 1/d^2)} \tag{3.8}$$

as the first bound of r that avoids having trivial unsatisfiable problems when $n \to \infty$. Table 3.2 shows the values of p (p_{crit}) that leads both expressions of the expected number of solutions to 1, obtained directly from Eq. 3.8 for the Achlioptas et al approach and form numerical computations from Eq. 3.7, being clear that the mentioned approach does very good at predicting the bound of constrainedness for having trivial unsatisfiable problems.

Despite having a good asymptotic behavior with the number of variables, model E reduces one step down the small structural properties of standard models, giving an absolute flat relation graph among its constraints. This absolute lack of structural properties make more desirable maintaining standard models inside a given scope of parametric ranges that ensure the same asymptotic model E properties. This is accomplished in [XL00] for model B,

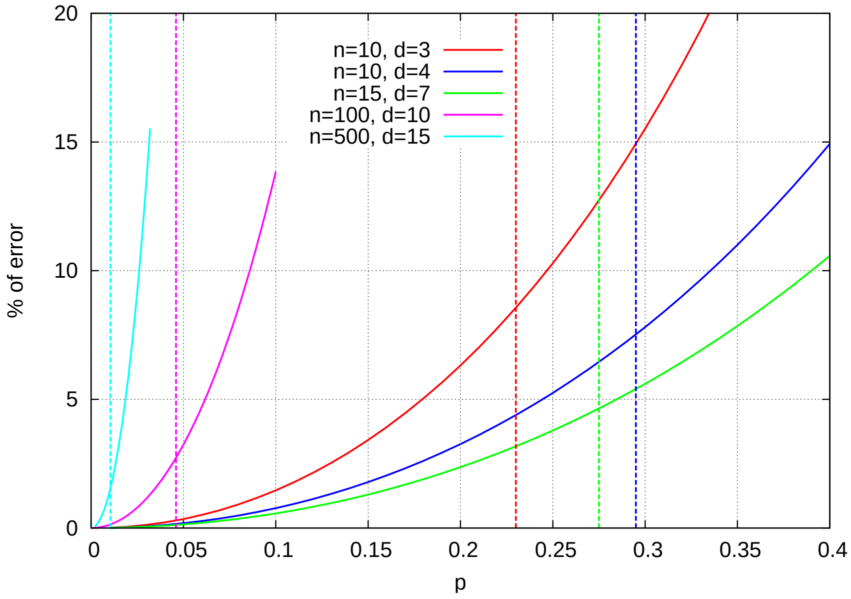


Figure 3.1: Relative error between both Eq. 3.4 and Eq. 3.7 (as percentage) against p, for some values of n and d. Dashed lines depict where $E[X] = 1$ according to Eq. 3.7

leading to revisited model B, or for short, model RB. The new model has no differences with model B, but the limits its parameters can reach, so, it can not be considered indeed a new model. The same expressions for the number of expected solutions and bounds for d than in Eq. 3.1 and 3.2 applies. Actually, the two conditions for the number of constraints and the number of nogoods from Theorem 1 and 2 of [XL00] are equivalent to Eq. 3.2. Xu and Li, therefore proves, using the second moment method, that condition 3.2 determines a sharp transition between satisfiable and unsatisfiable problems when $n \to \infty$.

3.4 Quasigroup Completion Problems (QCP)

In order to study the impact of structure on problem hardness, Gomes and Selman introduced the Quasigroup Completion problem (QCP) as a benchmark problem for evaluating combinatorial search methods: the structure underlying this domain can be found in a range of real-world applications, such as timetabling, routing, and design of statistical experiments [GS97]. QCP (and its variants) has become a widely used benchmark domain and it has led researchers to the discovery of interesting phenomena that characterize combinatorial search and consequently to the design of new search and inference strategies for such domains. Examples include, the so-called heavy-tailed phenomena, randomization and restarts strategies, e.g., [GSC97, GFSB04, Ref04, HO06], the design of efficient global constraints, e.g., [SSW98, RG04], and tradeoffs in different CSP and SAT based representations (see e.g. [DdC03, AdD^{+}04]).

3.5 Generalised Sudoku Problems (GSP)

Generalized Sudoku Problems model and generalize, i.e. extend, the popular Sudoku[4] puzzle. The original Sudoku puzzle consists in a 9×9 number matrix, with some cells filled with numbers from 1 to 9, and the rest of the cells empty, and to solve it, we must fill all the empty cells using numbers from 1 to 9 but following these three restrictions:

- on every column all numbers from 1 to 9 must appear exactly one time,
- on every row all numbers from 1 to 9 must appear exactly one time,
- on each 3×3 block region all numbers from 1 to 9 must appear exactly one time.

		7		6				
	8		9	3	1	6		
	6					4	9	
			5		8			1
	9						2	
1			6		3			
	4	6					1	
		3	2	1	9		6	
				8		5		

Figure 3.2: Typical 9×9 Sudoku puzzle

As it can be easily seen a Sudoku is derived from a 9×9 Latin Square, concretely is a QWH problem with the added restriction of having square sub areas (block regions from now on [5]) that must also contain completely distinct elements. A more formal and general definition of a Sudoku is as follows.

[4] The problem is referred to during this work as Sudoku, although other forms are considered equally valid: Su Doku, Su Do Ku, etc.

[5] We will call them block regions throughout all this thesis but in the literature (on SAT/CSP related literature or in not related one) they are indistinctly called blocks, regions, areas, region blocks, block region.

Definition 3.1 (Generalized Sudoku)**.** *A valid complete Generalized Sudoku (GS) of order s on s symbols, is a LS of order s with the additional restriction that each symbol occurs exactly once in each block region. A block region is a contiguous set of s pre-defined cells; block regions don't overlap, and there are exactly s block regions in a GS of order s. In the case of square block regions, each block region is an $\sqrt{s}\times\sqrt{s}$ matrix (s has to be a square number); in the case of rectangular block regions, each block region is an $m \times n$ matrix (m rows and n columns) with $m \cdot n = s$. Then, a GS with $m \cdot n$ block regions will have n region rows and m region columns (as an illustrating example, Figure 3.2 shows a GS structure with $m = 3$ and $n = 3$, Figure 5.4 shows a GS structure with $m = 3$ and $n = 10$).*

As is easy to see from the previous definition, a GS is a broad class of problems. It provides with a wide range of problems ranging as it subsumes all Quasigroup With Holes (QWH) [AGKS00] (when the block regions correspond to single rows) to Standard Generalized Sudoku Problems (GSP) (when the regions are squares) or Rectangular Sudoku Problems.

We can trivially build a GS of arbitrary order, with arbitrary rectangular or square block regions, using the following method: let S denote the GS, and its coefficients

$$S = (S_{i,j}^{k,l}), 0 \leq i, l \leq n-1, 0 \leq j, k \leq m-1, \tag{3.9}$$

where $S_{i,j}^{k,l}$ corresponds to the coefficient of S located at i-th region row, j-th region column, and inside such a region it is on the k-th row and l-th column. Then, these coefficients are ordered pairs defined as

$$S_{i,j}^{k,l} = (k + j \pmod{m}, i + l \pmod{n}). \tag{3.10}$$

The Generalized Sudoku Problem (GSP) is then defined as follows: given a partially filled Generalized Sudoku instance of order s, can we fill the remaining cells of the $s \times s$ matrix such that we obtain a valid complete Generalized Sudoku instance?

Then, following the current line of research that focus on solvable problems, we define the Generalized Sudoku With Holes Problem (GSWHP). In this case the instance to be solved comes from a GS where some cells have been emptied, so we ensure that a solution exists.

3.5.1 Generating complete Generalized Sudokus

As it is obvious from the previous definition, in order to create a GSWHP instance, first we need to obtain a (valid and complete) GS. Then, to generate such GS, we follow the approach used in [JM96] of building a Markov chain whose set of states includes, as a subset, the set of complete Generalized Sudokus. If we use the Markov chain that considers only *proper* LS as states and using improper LS as pivots, and because any GS is also a LS of the same order, this chain obviously includes a subchain with all the possible GSs. So, for any pair of complete GSs, there exists a sequence of proper moves, of the type mentioned in Theorem 6 of [JM96], that transforms one into the other.

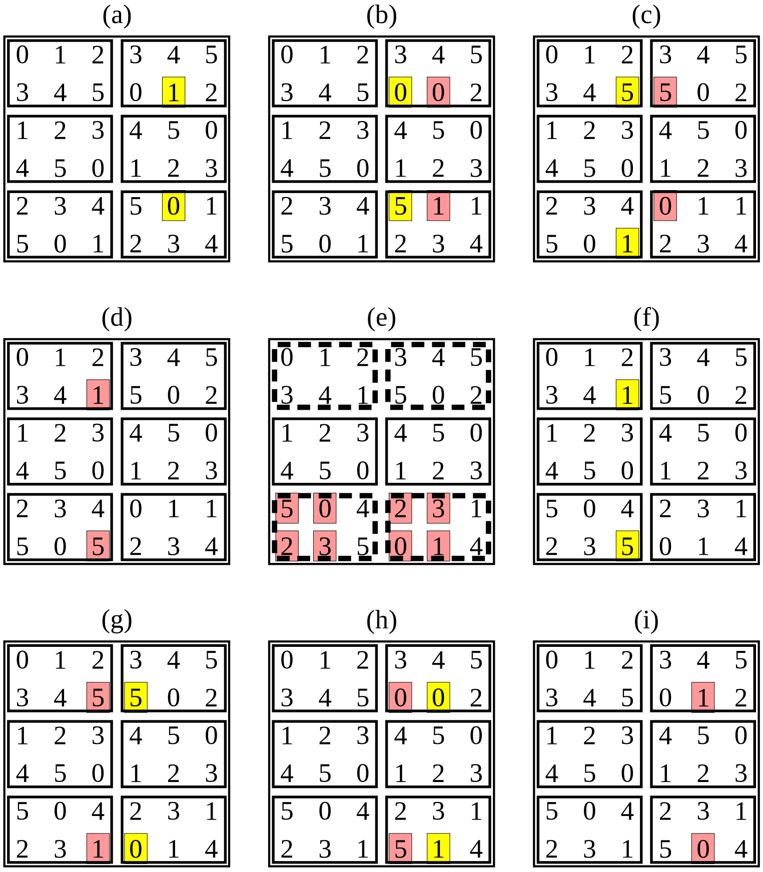


Figure 3.3: GS generation example

However, if we simply use the Markov chain by making proper moves uniformly at random, starting from an initial complete GS, most of the time we will reach LS that are not valid GSs. To cope with this problem, we select the move that minimizes the number of *violated* cells, i.e. cells with a symbol that appears at least twice in the same block region. To minimize those violated cells we proceed using the method defined on [JM96], that moves from one LS to another by choosing initially 4 random parameters $(r1, r2, t, i)$ (two rows, $r1$,$r2$, a symbol, t, and a number of iterations, i), but if through our search for a GS we stop in a LS that is not a GS, instead of starting a new search by choosing those above mentioned parameters at random, we choose the initial rows at random but only among those that contain violated cells, the same is applied to the initial symbol, which is one of the symbols that violated the GS condition in a block.

To escape local minima, if after a certain number of moves[6] we don't get a GS, we restart the move from the previous GS, and we perform the moves described above until we have generated a certain number of valid GSs. Observe that this method does not necessarily generate a GS instance uniformly at random, because we do not always select the next move uniformly at random. However, as we will see in the experimental results, the method provides us with very hard computational instances of the GSWHP, once we punch holes in the appropriate manner.

Figure 3.3 shows an example of an execution of the afore mentioned procedure. From the canonical GS (a), we perform 2 moves ([a-e] and [f-i]) of the second method detailed in [JM96] until we obtain a valid GS (i). Each move is started by randomly selecting 2 rows, a symbol and the number of iterations. For the first move (a-e), the initial random values are $(1, 4, 1, 3)$. At the first iteration (a), symbol 1 points the corresponding symbol to be switched by intersecting its column with the second row (symbol 0). Symbols to be switched are yellow marked, becoming red once the change is produced. Previous symbol 0 is taken in second iteration to designate its corresponding counterpart to be switched (symbol 5 in this case)(b). Same procedure applies up to the last iteration (c). At this point, the last selected symbol (5), is switched with the initial random selected symbol (1) at its corresponding row, giving (d). Once arrived at this point, if the resulting LS is improper (as it is in our example), switches between symbols of the second initial row and the row that contains the last selected symbol (5) are applied until getting a proper LS (e). Dashed blocks show which blocks violate the GS condition, so a second move will be needed. On the second move, the initial random values are $(1, 5, 1, 3)$. Note that in this case, rows 1 and 5 as well as symbol 1 are chosen at random from those that violate the GS condition, i.e. from dashed blocks on (e). The same procedure as before applies from (e) to (i), getting at this last step (i) a proper GS, so no additional shuffles are required unlike in (e).

3.6 Edge Puzzles

Edge matching puzzles are a problem class, that has been shown to be NP-complete [DD07], that can be easily modelled as SAT/CSP problems. Edge matching puzzles have been known for more than a century (E.L. Thurston was granted US Patents 487797 and 487798 in 1892) and there is a number of child toys based on edge matching puzzles. As of 2007, these puzzles

[6] In our experiments, this number has been fixed to 20, because it works reasonable well with all the orders of GSs that we have tested.

have received world wide attention after the publication of an edge matching puzzle with a money prize if resolved (*Eternity II*). This kind of competitions are, on one side a real challenge to SAT/CSP solvers, and on the other side a real showcase to show recent advances in hard problem solving attained by the SAT/CSP community.

3.6.1 Problem description and definitions

Roughly described, an edge matching puzzle is a puzzle where we must place a set of tokens or pieces in a board following a simple rule. Tokens have four sides, in our case for simplicity we assume square tokens, each of a different color or pattern. The rule to follow when placing tokens is that two tokens can be placed side by side iif adjacent sides are of the same color (or pattern). A more formal definition is as follows,

Definition 3.2 (Generic Edge Matching Puzzle (GEMP))**.** *A Generic Edge Matching Puzzle (GEMP) , $P(n \times m, k)$ of size n and k colors, is a tuplet (V, S), where V is the set of variables representing cell positions on the plane, of the form, $V = \{v_{i,j}, 1 \leq i \leq n, 1 \leq j \leq m\}$. Variables in V take values from the domain S, with $S = \{(t, r)/t \in \{T\}, r \in \{1, 2, 3, 4\}\}$, representing r the set of possible rotation of the same token, with T, representing the token set, which is of the form $T = \{(x_1, x_2, x_3, x_4)/x_i \in C\}$ and C is the set of colors, $C = \{c_i, 1 \leq i \leq k\}$.*

One possible variant on GEMPs is that where token rotations are not allowed, that is, all tokens must be placed **exactly** in the same orientation as they are in the puzzle specification.

Definition 3.3 (Generic Edge Matching Puzzle Solution)**.** *A valid solution for a GEMP, $P = (V, T)$ is an assignment of values from T to all the variables in V such that for each pair of neighboring variables, the color value assigned to the adjacent half-edges between those two variables is the same.*

For clarity we define a half-edge to be the part of an edge that belongs to each token or piece in the puzzle, and an edge will be the union of two half-edges of the same color.

Definition 3.4 (Framed GEMP (GEMP-F))**.** *A Framed Generic Edge Matching Puzzle (GEMP-F), $P(n \times m, k)$ is a Edge Matching Puzzle that includes a special color, we represent it in figure 3.5 as 'gray'(0), that, in all valid solutions can only appear in variables located on the frame of the puzzle, i.e., those variables in $\{v_{1,j}, v_{n,j}/1 \leq j \leq m\} \bigcup \{v_{i,1}, v_{i,m}/1 \leq i \leq n\}$ and only on the outside half-edges of those variables.*

One could think on several variants of framed puzzles attending to the sets of colors employed on distinct areas of the puzzle. In this paper we

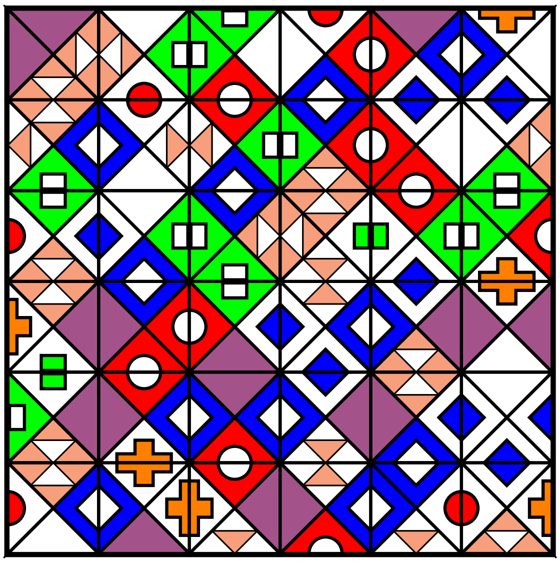


Figure 3.4: 6x6 size GEMP example with 12 inner colors

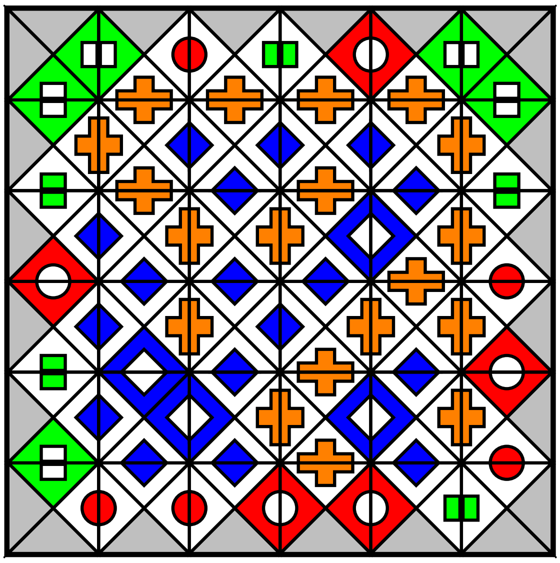


Figure 3.5: 6x6 size two-set GEMP-F example with 4 frame colors and 3 inner colors

deal with two types, that have a profound impact on hardness, **one-set GEMP-F** when colors can be used at any edge of the puzzle, and **two-set GEMP-F** when two disjoint sets of colors are used; one set for edges joining frame pieces and another set for any other edge. As an example take Figure 3.5. One can observe that colors joining frame pieces are different than the rest. As real-world puzzles (as in Eternity II) are usually framed puzzles and due to the interesting effect that the frame has on hardness this work deals with GEMP-F leaving for a future work GEMP problems.

During this work we study square GEMP-F instead of rectangular ones, that is, GEMP-F of the type $P(n \times n, k)$, to simplify experimentation and implementations. Using rectangular puzzles probably will not increase problem hardness as experimental findings in other SAT/CSP approaches to similar problems, such as Sudoku Problems [ABF$^+$06], show that non-square problems are easier than square ones.

3.6.2 Generating Puzzles

The general method for a solvable puzzle generator is detailed in Algorithm 3.1. Roughly explained, the method assigns colors to edges of puzzle pieces (assigning a color to both half-edges). When all edges are colored, tokens are built from the existing color assignment. In the algorithm, $v_{i,j}^{side}$ refers to one of the four half-edges of the variable at position i, j, $N(v_{i,j})$ is the set of up to four neighbors of variable $v_{i,j}$ (at sides up, right, down and left), and $N_s(v_{i,j})$ gives position of neighbor at side s of $v_{i,j}$, if exists, and $0, 0$ otherwise. Finally, $front(s)$ gives the opposite side to s, that is, given two adjacent positions, s and $front(s)$ represent the two adjacent sides that join the two positions.

Special care must be taken on implementing this algorithm because this method does not prevent having repeated tokens nor symmetric tokens (tokens with rotations that leave the token invariant), but for higher enough values of c (as those around the Phase Transition values), repetitions or symmetric tokens are low enough to do not suppose an impact on problem hardness.

Extending this algorithm to generate framed puzzles is easy. First the inner part of the puzzle is generated (tokens without gray color), without taking into account the frame. Then colors are assigned to the half-edges of the frame adjacent to inner tokens, that are already determined by the inner tokens, and then half-edges that join tokens of the frame are filled with random colors, choosing either from the same set of colors used for the inner tokens (getting a one-set GEMP-F) or from a second set of colors with no colors in common with the first set (getting a two-set GEMP-F).

Other extensions can be easily defined for generating problems where the satisfiability is not guaranteed as in the previous example. How to get those puzzles is rather easy. Once we generate one always-SAT puzzle we

Algorithm 3.1: Algorithm for generating GEMP(n, c) puzzles

```
input  : $n, c$
output: an Edge Puzzle of size $n$ with $c$ colors
for $i = 1$ to $n$ do
    for $j = 1$ to $n$ do
        for $side = 1$ to $4$ do
            if $v_{i,j}^{side}$ is empty then
                $v_{i,j}^{side}$= random( $c$ )
                $i', j' = \mathrm{N}_{\mathrm{side}}(i, j)$
                if $i', j' \neq 0, 0$ then
                    $v_{i',j'}^{front(side)}$=$v_{i,j}^{side}$
```

change the color to some half-edges randomly. If we change colors to half-edges always in pairs (that is, maintaining the color count parity) resulting puzzles can be also satisfiable, if we do not maintain that parity, puzzles will always be UNSAT and a solver that checks for color parity will return UNSAT pretty fast.

4

Hardness of CSP problems

What is difficulty? Only a word indicating the degree of effort required to accomplish something! A mere notice of the necessity for exertion; a scarecrow to children and fools and a stimulus to real men.

Samuel Warren

There are two approaches to define the hardness of a CSP problem. One is to study which is the hardness in the worst possible case, the other try to define hardness on the typical case. The first area of research focuses on which is the expected maximum effort we will have to do to solve the problem. The study of worst-case hardness has given us clues on how to use factors present on very hard problems, as high treewidth on constraint graphs, to harden problems generated. The other area, typical case hardness, studies hardness of all problems of a given problem family, not just the worst cases but which is the average hardness. One of the most important concepts on typical case hardness studies is determining the existence or not of a hardness phase transition (from easy to hard to easy), and if that existence is confirmed, to pinpoint both, analytically and experimentally, its location. We have done that experiments, specially on GEMP, as it is a problem defined formally for first time in this work, locating precisely the PT point was of utmost importance.

Contents

4.1 Worst case hardness

4.1.1 Complexity of binary CSPs

In this section we present the theoretical worst-case complexity results upon we have based our methods for hardening CSP problems, mainly for binary CSPs. We do not try to present here a complete picture of worst-case complexity results for CSPs, but only to present those results directly linked with the ideas behind our methods. However, we have tried to focus on results that provide fairly general classifications of the complexity of CSPs, such that many other complexity results are just special cases of the results we present here, or are subsumed by the results we present here.

We divide the different restrictions studied on CSPs in two classes: structural restrictions on their constraint graph (or constraint hypergraph for non-binary CSPs) and constraint language restrictions.

Constraint graph restrictions

The most general results about the complexity of CSPs based on structural restrictions, are based on structural properties of either its constraint graph (in the case of binary or bounded arity CSPs) or its contraint hypergraph (in the case of unbounded arity CSPs). As the work of this thesis focus mainly on binary CSPs, we present mainly results about binary CSPs.

The main structural restriction that gives tractable (binary) CSPs is their treewidth (see Definition 2.7). Observe that the treewidth of a tree is 1, because we can put any edge of the tree in exactly one bag, such that any vertex appears in at most two (intersecting) bags. The first link between structural graph restrictions and complexity of CSPs is the result presented in [Fre82], where they show that when the constraint graph is a tree, we can solve it in a backtrack free manner, using only arc-consistency. If a constraint graph contains at least one cycle, we cannot use directly the results of Freuder to solve the CSP. However, if the CSP has a tree decomposition of width w, we can still solve them with time $O(||I||^w)$, where $||I||$ is the size of the instance, thanks to a generalization of the algorithm of Freuder [Fre90]. That means, for example, that if the constraint graph is a simple cycle we can solve it in time $O(||I||^2)$, given that a cycle has treewidth 2. But as the treewidth increases, this time bound increases. Results presented in [Gro03] show that the treewidth of a graph is the most relevant structural parameter concerning the complexity of binary CSPs with arbitrary constraint languages. These results basically show that, assuming that a conjecture from parametrized computational complexity to be true ($FPT \neq W[1]$), then CSPs with bounded arity and arbitrary constraint languages can only be solved in polynomial time if and only if their treewidth is bounded by a constant.

These results indicate that for CSPs of a given size and number of constraints, the ones with highest possible treewidth should be the hardest ones. In general, the more edges a graph has, the higher its treewidth, because the treewidth can only increase or stay the same as we insert more edges to a graph, reaching the maximum treewidth of $|V| - 1$ when the graph is complete. But then the interesting question is how to have a high treewidth with a low number of edges. This observation has lead us to consider how high can the treewidth be when we have sparse constraint graphs. In this thesis, by sparse we mean with the minimum needed number of edges such that the constraint graph is connected. We will see that this question is directly connected with very nice areas of graph theory, in particular with expander graphs. In chapter 5 we will present some consequences of well-known results in expander graphs that give us the main tools we have used towards locating, and generating, high treewidth graphs with a low number of edges. Moreover, the methods for expander graph generation will be also used to increase the hardness of k-SAT and n-ary CSPs instances through the use of their incidence graph.

The following lemma gives us an indication of a necessary condition for the treewidth of a graph to be low.

Lemma 4.1 ([Bod98]). *Every graph of treewidth at most k contains a vertex of degree at most k.*

So, having low treewidth implies having at least one vertex of low degree. That means that, for a set of graphs with the same number of edges, the ones that do not allow the treewidth to be lower than their average degree are the ones where the degrees are all equal. This observation indicates that a desirable property of high treewidth graphs is the balance of the degrees of its vertexes.

Constraint language restrictions

Many of the current known restrictions on the constraint language that determine tractability are based on properties of so called polymorphisms of the constraint language [BJK05]. Basically, a polymorphism is a function defined over the set of tuples of a relation, such that each tuple of the relation is mapped to another tuple of the relation. The set of polymorphisms of a constraint language is the set of functions that are polymorphisms of all the relations of the constraint language. For the purposes of our work, the following results that link the concepts of m-looseness and m-tightness of a constraint and the maximum arity of the constraints with the $k-$consistency needed to solve a problem, are very relevant towards finding ways to increase the hardness of the constraints.

Definition 4.1 (m-tight)**.** *A constraint relation R of arity k is called m-tight if, for any variable x_i constrained by R and any instantiation $\overline{a}$ of the remaining $k-1$ variables constrained by R, either there are at most m extensions of $\overline{a}$ to x_i that satisfy R, or there are exactly $|D_i|$ such extensions.*

Definition 4.2 (m-loose)**.** *A constraint relation R of arity k is called m-loose if, for any variable x_i constrained by R and any instantiation $\overline{a}$ of the remaining $k-1$ variables constrained by R, there are at least m extensions of $\overline{a}$ to x_i that satisfy R.*

Next, we consider two important results that link the m-tightness and m-looseness with the complexity of solve a CSP, related the notions of local and global consistency.

Theorem 4.2 ([vD97])**.** *If a constraint network with relations that are m-tight and of arity at most r is strongly $((m+1)(r-1)+1)$-consistent, then it is globally consistent.*

So, the lower the value of m for the m-tightness of the constraints of a CSP, the lower the level of consistency that is enough to ensure also global consistency. That indicates that one should avoid constraints with a low m-tightness in order to escape from problems trivially solvable with a low consistency level.

Theorem 4.3 ([vD97, ZY03])**.** *A constraint network with domains that are of size at most d and relations that are m-loose and of arity at least r, $r \geq 2$, is strongly $k-$consistent, where k is the maximum value such that the following inequality holds*

$$\binom{k-1}{r-1} \leq \lceil \frac{d}{d-m} \rceil - 1$$

So, the higher the $m-$looseness of the constraints, the higher the inherent level of consistency already present on the problem, so higher levels of consistency will need to be enforced in order to discover inconsistencies in the problem.

As we will see in Chapter 5, we can use these results about m-tightness and $m-$looseness to partially explain the increased hardness of balanced constraint languages, with respect to random constraint languages.

4.1.2 Hardness of k-SAT

We do not provide here a complete picture of the complexity of SAT, but we explain only the relevant results that are the main motivation behind the methods we have developed for generating hard k-SAT and n-ary CSP instances in Chapter 5.

Concerning the resolution complexity of a 3-SAT instance F, Ben-Sasson and Wigderson [BSW01] proved that if every resolution refutation of F requires width w, then every resolution refutation of F requires size $2^{\Omega(w^2/n)}$. The width of a resolution refutation is the length of the longest clause in the refutation. Thus, lower bounds on width imply lower bounds on size. Finally, there is a connection between graph expansion and 3-SAT resolution complexity based on this width-size relationship. Consider a 3-SAT instance F with set of clauses C and set of variables V and its bipartite variable incidence graph $G(F) = (C \cup V, E)$. Results presented in [Ats04] imply that any resolution refutation will have width lower bounded by $\lfloor ((c-1)\alpha|C|)/((2+c)d) \rfloor$, where d is the maximum right-degree of $G(F)$, if $G(F)$ is a left (α, c)-expander. So, any resolution refutation of F will have exponential size if $d = o(|C|)$ and $c > 1, \alpha > 0$, given the width-size relationship. So, the higher the expansion of the graph and the smaller the maximum right-degree d, the higher the refutation size lower bound. Moreover, the results also imply that more powerful proof algorithms based on strong k-consistency will also require exponential time for solving the 3-SAT instance under the same circumstances.

Concerning n-ary CSPs, there are results about their complexity related to the treewidth of their incidence graph [SS06]. The incidence graph for CSPs considered in these results is not the same generalization of incidence graph that we will consider for hardening n-ary CSPs in Chapter 5. In the work [SS06] the incidence graph simply links constraints with the variables in their scope. However, looking at the results about k-SAT hardness related to the expansion of their incidence graph, it seems more natural to consider linking nogoods with their variables, or even nogoods with the many-valued literals they contain. This is what we will do in Chapter 5.

4.2 Typical case hardness

4.2.1 Phase Transition

The phase transition concept has its origins on thermodynamics, denoting when a thermodynamic systems changes from one phase to another, out of the four possible phases of matter. This concept has been extended to several areas of physics and science areas in general, such as dynamic systems, science of materials and computational complexity as a few examples.

One the most amazing topics where phase transitions are detected are percolation systems. It is amazing because percolation theory links phase transition phenomena with fractal behavior of real systems, being this point not achieved yet on computer complexity theory. Percolation is a physical process that describes for a system, a transition from one state to another, such as the filtering of fluid (e.g. petroleum) between rock layers. In [PJS04], a more *natural* paradigm is described for such a systems, the forest fires. Considering a forest as a lattice of trees of size N, and modelling fire propagation as a probability p between adjacent trees, three fundamental observations can be done:

1. Easy-hard-easy patterns. The time elapsed until a fire extinguishes, depends on p, showing a maximum duration at a given value p_c
2. Phase transition of the maximum cluster. Measuring the burned area $M(N)$ in relation to the lattice size (N) an abrupt transition can be observed at p_c. This would be an indication of the probability that a random picked tree was burned.
3. Fractal shapes. At p_c, can be observed that the fractal dimension of the maximum cluster is less than 2, acquiring the clusters this way certain fractal shapes.

Figures 4.1 and 4.2 show the easy-hard-easy and phase transition behavior respectively for some lattice sizes depending on probability p. The simulations are done over 200 samples per point. Figure 4.3 show a burned forest of size 1000 for $p = 0.5$, where the fractal shapes of its boundaries can be guessed.

The relevance of these observations are extended, constantly, to computer complexity analysis as we will see in the following sections, observing that some of such behaviors systematically arise in several SAT/CSP problem classes.

Phase transition on K-SAT problems

Beyond physical systems, phase transition effects linked to AI problems (graph coloring, k-satisfiability, Hamilton circuits, ...) were already reported on [Pur83, CKT91]. Subsequently, some authors [BHHW96, MZK$^+$96, Hay97] report phase transition characterization for K-SAT problems and explaining how such transition phenomena are intrinsic to SAT problems and not related to any particular solving algorithm.

In [MZK$^+$99], it is shown that there exists a threshold for complexity transition on K-SAT problems for any $K \geq 3$, although an accurate determination of such a threshold remains still unclear. Moreover, phase transition effect in K-SAT problems are linked to both; percentage of SAT/UNSAT instances and the backbone fraction (i.e. the number of variables assigned to

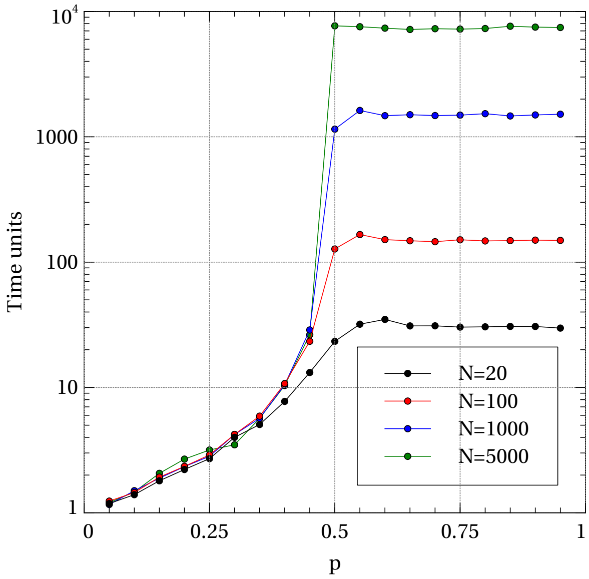


Figure 4.1: Easy-hard-easy characterization on a forest fire simulation lattice of size N as a function of p

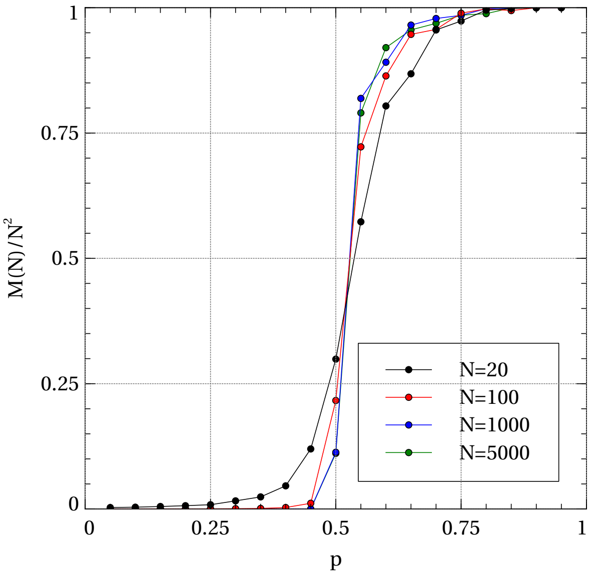


Figure 4.2: Phase transition behavior on a forest fire simulation lattice of size N as a function of p

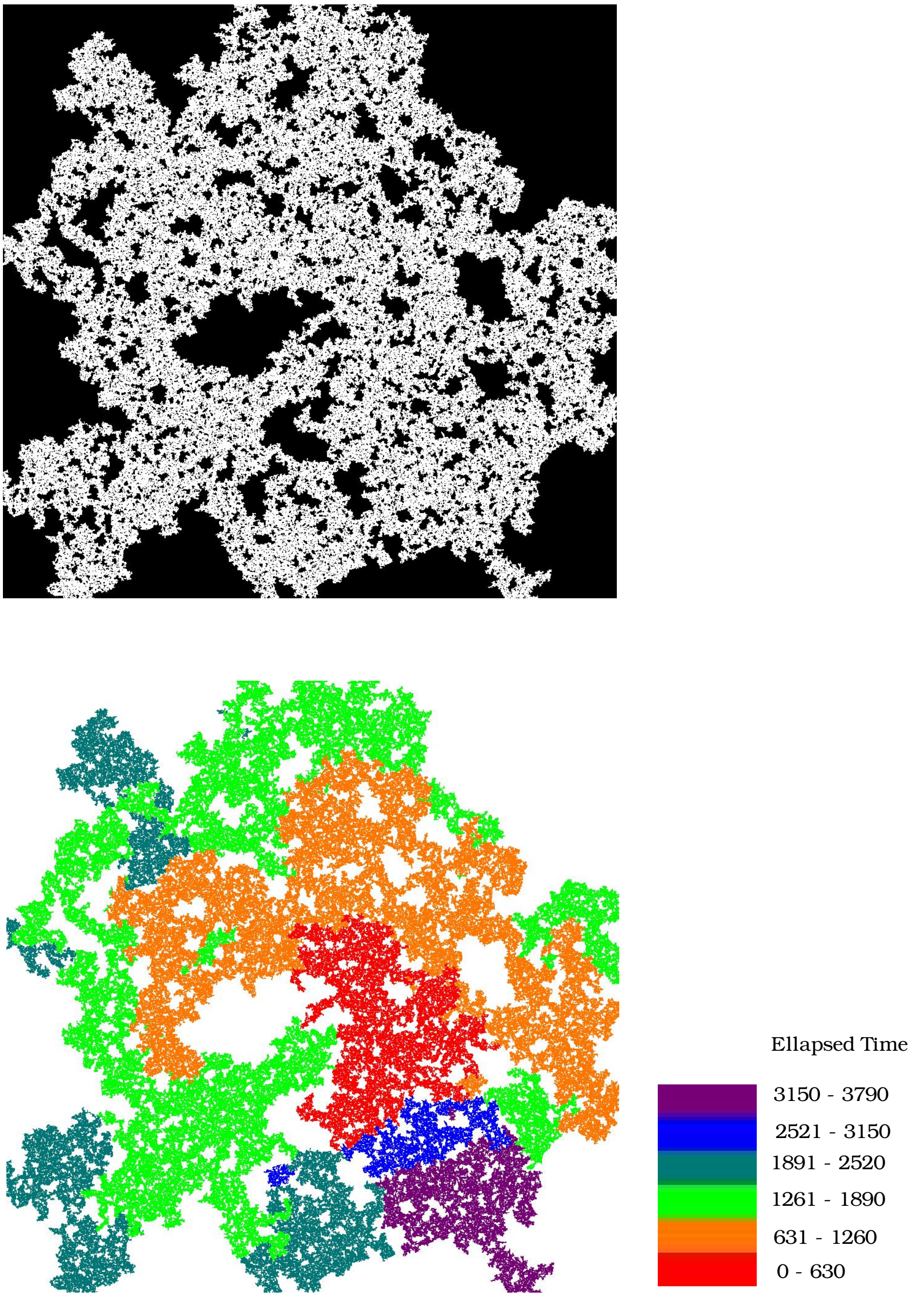


Figure 4.3: Fractal shapes of the maximum cluster and the corresponding propagation time on a forest fire simulation lattice of size $N = 1000$ for $p = 0.5$

the same value for all the solutions), being this last concept of capital importance when determining phase transition effects on only satisfiable problems as we will see later.

Since [CF86], a lot of work has been devoted to the location of the phase transition threshold. With an easy analysis, an upper bound to the threshold shows that for a ratio of clauses to variables (r) such that, $r \geq 2^k \log 2$, the problem is unsatisfiable with high probability. At this point the *Satisfiability Threshold Conjecture* arises, stating that a lower bound exists (r^*) being asymptotically equal to r. Meanwhile successive analysis of r showed that it was no far from its original computation, an accurate analysis of r^* required more sophisticated tools, giving those approaches not small enough boundaries able to prove the conjecture. This lower bound have been improved by [AP04, NAP05, AM06] introducing the analysis based on the second moment of the expected number of solutions while applying the Cauchy Inequality, and proving an asymptotic form of the Satisfiability Conjecture.

Phase transition on Random CSP problems

[Pro96, SD96] both are empirical studies of the PT on BCSP (model B), estimating the PT location for the combination of model B parameters where the expected number of solutions is 1. This predictor seems to work reasonably well as experimental results show. Model B number of solutions can easily be shown (see 3.3.1 for a detailed explanation of the parameters)

$$E[X] = d^n(1-p_2)^{\frac{n(n-1)p_1}{2}}$$

obtaining for $E[X] = 1$

$$p_{crit} = 1 - d^{\frac{-2}{(n-1)p_1}} \tag{4.1}$$

Figure 4.4 is a three dimensional plot of the hardness characterization of model B problems depending on p_1 and p_2. We can obtain several conclusions from the plot: hardest instances align effectively around the predicted value of p_2 upon Eq. 4.1; transition from under-constrained instances is sharper than those at the other side of the peak; and hardness peak decreases with p_1, showing that constrainedness is correlated with hardness.

Second moment analysis is also used on [SD96] to give upper and lower bounds of p_{crit} as mentioned on 4.2.1. There, an expression for the second moment of the number of solutions ($E[X^2]$) is provided, giving numerical computation of the second moment for small values of n.

But the asymptotic threshold reported on K-SAT problems no longer exists for RBCSP. As discovered on [AKK+97], standard RBCSP models are flawed (flaw probabilities are computed on [GMP+01]), and almost all instances, generated when the number of variables tend to infinity, have no solution, so proposing a new model, called model E (already detailed in

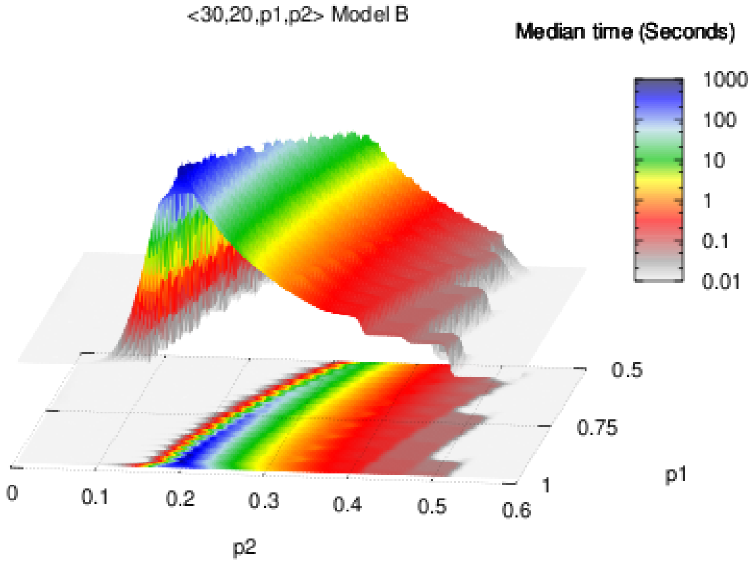


Figure 4.4: Three dimensional representation of the hardness characterization of model B problems as a function pf p_1 and p_2 for 30 variables, domain 20 problems.

3.3.1). In model E, the two step procedure leading to models A-D is omitted and constraints are chosen among all the possible set, $\binom{n}{k}d^k$, at random, uniformly with repetition, being n the number of variables, k the arity and d the domain size. [AKK+97] also gives bounds for the underconstrainted and the overconstrained region, allowing to theoretically determined the PT location. [GMP+01] gives more experimental evidence about the flawlessness of the standard RBCSP models.

A second try in order to define non-degenerated (as the number of variables tend to infinity) random BCSP problems is the work of [XL00]. There, model RB is defined as a revision of the standard model B, showing that a phase transition exists. Its construction is the same, but extended to k-arity problems and defining 2 control parameters (r and p) linked to the constrainedness and tightness of the problem respectively. Those parameters, when conveniently established, ensure the existence of a phase transition. Parameter r defines the total number of k-ary constraints as $r\,n\,\log(n)$, meanwhile p is the fraction of tuples per constraint, so each constraint has $p\,d^k$ tuples, being d the domain size. Its interrelation is given as follows:

- Being α a constant that relates number of variables and domain, i.e.

$d = n^{\alpha}$, when $\alpha > \frac{1}{k}$, $0 < p < 1$ and $k \geq \frac{1}{1-p}$, then there exists a r_{crit} than defines a phase transition such that

$$r_{crit} = \frac{-\alpha}{\ln(1-p)} \tag{4.2}$$

- Otherwise, if $\alpha > \frac{1}{k}$, $r > 0$ and $k\,e^{\frac{-\alpha}{r}} \geq 1$ hold, then the phase transition is defined by the parameter p as

$$p_{crit} = 1 - e^{\frac{-\alpha}{r}} \tag{4.3}$$

Both critical points are obtained by an asymptotical analysis of the number of solutions second moment.

The work of [XL00] is a good indicator about the findings of [Smi01] on constructing non-degenerate RBCSPs. In this work, [Smi01] proves, at least for standard model D, that the objections introduced by [AKK^{+}97] about the standard models for large values of n can be conveyed by adjusting conveniently the model parameters. And so is done for model D, proposing to increase slowly parameters m and d as n grows. In this sense, a computable revision of model D is reported on [XBHL05], called model RD, extended to k-ary problems and finding that same relations Eq. 4.2 and Eq. 4.3 apply, but in this case for $p \leq \frac{k}{k-1}$.

As we will see in Chapter 5, several techniques have been developed for hardening random K-SAT, as well as random CSP problems. Independently of such techniques, phase transition phenomena and easy-hard-easy patterns have been observed on the new derived problem classes. Fig.4.5 is an example. Note that according to Eq. 4.1, for Fig. 4.5 example it results, $p_{crit} = 0.4742$, meaning 38.4 nogoods per constraint.

Phase transition on QCP and GSP problems

As seen in Section 3.4, the quest for new benchmarks that bridge the gap between purely random generated problems and highly structured and real ones, lead to definition of a new generation of problems built from mathematical structures (such as Latin Squares) that once randomly perturbed were able to give interesting instance sets. An so it was the definition of the Quasigroup Completion Problem [GS97]. Beyond practical applications, QCP present a handful of interesting properties such as their NP hard completeness as well as a clear phase transition effect when the ratio of preassigned cells varies. This transition is determined experimentally to be located around a given fraction of the number of cells; 42% of preassigned cells. It's worth to note that as far as we know, no theoretical approaches for the PT location exist on structured problems as QCP, QWH or GSP.

The requirement of only satisfiable instances make QCP not appropriate on certain benchmarks. So, QWH problems are introduced [AGKS00,

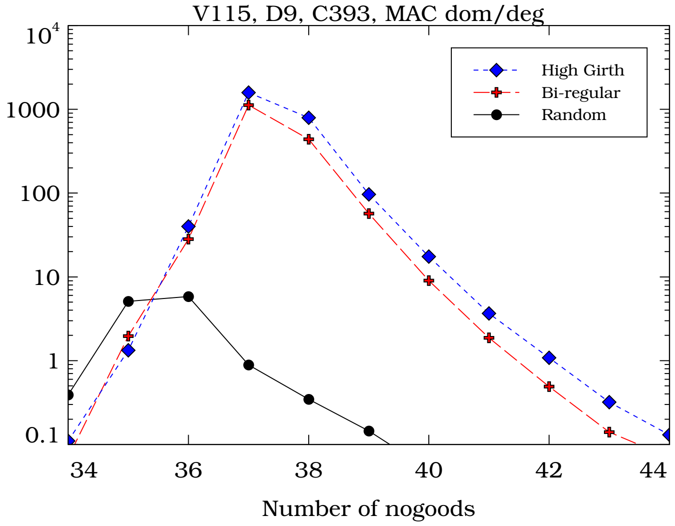


Figure 4.5: An example of easy-hard-easy pattern for RBCSP model B problems and two derived hardening schemes, with 115 variables, domain 9 and 393 constraints as a function of the tightness. See Chapter 5 for details of the new models.

GS02], proving to be NP-hard and showing also a PT characterization. This PT is shown to occur at some point around 1.6 $N^{1.55}$ preassigned cells, being N^2 the number of total QWH cells. But it's relevant to mention that PT no longer can be associated to the percentage of satisfiable instances because we are dealing with only satisfiable problems. For QWH problems, a new parameter is introduced by [AGKS00], linking the peak search in complexity with phase transition phenomena. Such a parameter is the fraction of the backbone, being the backbone the set of variables that keeps invariant for all the assignments of a given instance. The existence of the backbone phase transition was already linked to SAT problems by [MZK$^+$99].

Keeping on structured problems, GSPs also present a PT characterization when the backbone fraction of variables is considered. On [ABF$^+$06], is presented the computational behavior for different GSP sizes, showing that a GSP is much heavier than its corresponding same order QWH as depicted in Fig. 4.6. Furthermore, the more squared are the compounding Sudoku blocks, the harder the problem is for the same size. Actually, for Fig. 4.6 can be observed that hardness increases more than 3 orders of magnitude from a order 30 QWH to its corresponding 6x5 GSP. The link of this easy-hard-easy pattern with the backbone fraction is plot in Fig. 4.7.

In order to have a complete hardness characterization, we performed an exhaustive experimentation to locate the phase transition point, at least for

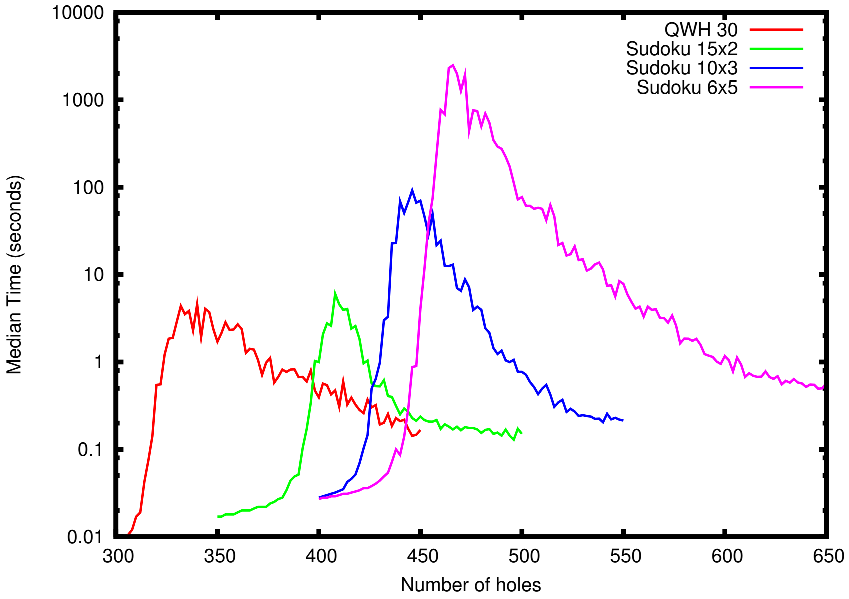


Figure 4.6: Easy-Hard-Easy computational complexity patterns for QWH order 30 problems and different GSP with same size

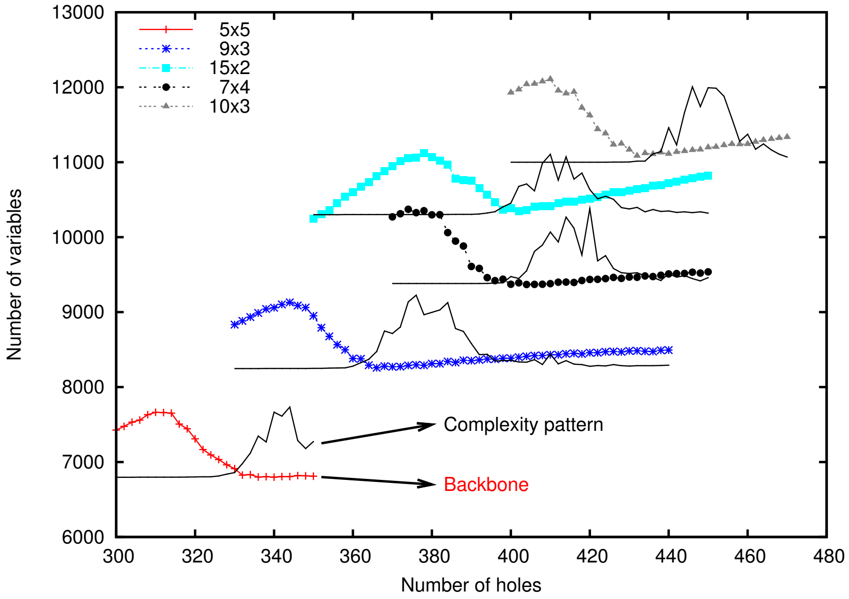


Figure 4.7: Backbone and computational complexity representation of different size GSP

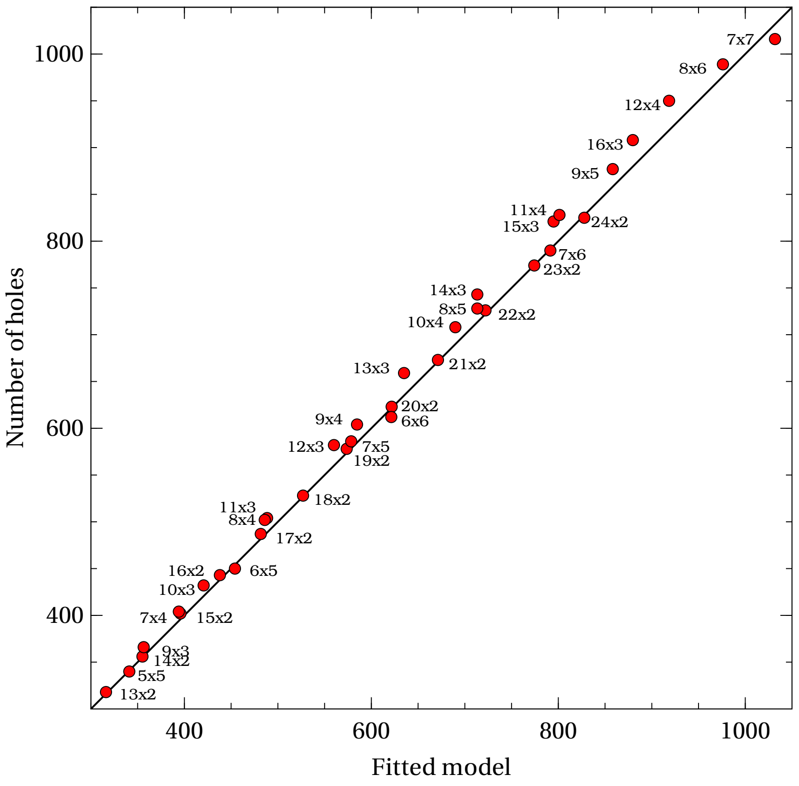


Figure 4.8: Regression model for some computed hard peak GSP sizes

the GSP sizes we were able to deal with. Table 4.1 shows the number of holes for GSP that lead those problems to the hardest instances. Taking these data and preforming a doubly exponential regression, we were able to find an exponential model for the number of holes as a function of the GSP block size. The resulting model for all the possible block sizes ranging from order 26 to 49 is the following:

$$holes = e^{0.537} \cdot n^{1.57} \cdot l^{1.72}$$

with a coefficient of regression $R^2 = 0.989$. Fig. 4.8 plots the model fit.

Phase transition on Edge Puzzle problems

Edge Matching Puzzles, seen on Section 3.6, also present a PT characterization linked to their backbone. We have reported experimental evidence of the PT and the hardness characterisation of GEMP benchmark [ABFM08a, ABFM08c], as well as, to our best knowledge, the first theoretical analysis of such a kind of problems.

Block size (nxl)		Number of holes
n	l	
13	2	318
14	2	356
15	2	402
16	2	443
17	2	487
18	2	528
19	2	578
20	2	623
21	2	673
22	2	726
23	2	774
24	2	825
9	3	366
10	3	432
11	3	504
12	3	582
13	3	659
14	3	743
15	3	821
16	3	908
7	4	404
8	4	502
9	4	604
10	4	708
11	4	828
12	4	950
5	5	340
6	5	450
7	5	586
8	5	728
9	5	877
6	6	612
7	6	790
8	6	989
7	7	1016

Table 4.1: Number of holes for the GSP that determines the phase transition location for different block sizes

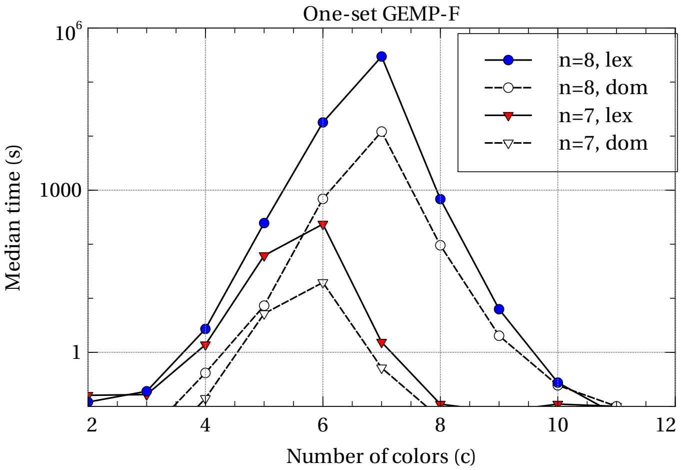


Figure 4.9: Hardness characteristic for a one-set GEMP-F as a function of the number of colors

One-set and two-set GEMP-F present a hardness characterization depending on their constituent number of colors. For one-set GEMP-F problems, once fixed the problem size, problem hardness only depends on one parameter, i.e. the number of colors, c. Fig. 4.9 plots the median time to solve 1-set GEMP-F problems of size 7x7 and 8x8 for different variable selection heuristics, showing clearly a peak of hardness as a function of c. A similar characterisation is found for two-set GEMP-F problems, but in this case, problem hardness depends on both parameters; number of colors in the mid and at the frame, c_m and c_f respectively. Fig. 4.10 is an example for 7x7 size puzzles.

We conjecture this could be due the clustering effect produced by the frame over the solutions space, similar to the one observed in other problems [MZ02, NAP05, LS05], i.e., the resulting solutions space of the inner puzzle could be splitted into disjoint clusters of solutions surrounded by many assignments that are very close to some solution. As detailed in previous analyzed only satisfiable problems, one can link this hardness characterization to the backbone. Figure 4.11 shows this phase transition plotting the fraction of the backbone as a function of the number of inner colors (c_m) for two-set GEMP-F with 3 frame colors ($c_f = 3$).

From an analytical point of view, we can derive some expressions that predict the phase transition location. For the sake of tractability, we consider tokens generated randomly, unregarding adjacency constraints that give only SAT puzzles. Of course, this is only an approach, but experimental results and numerical evaluations agree for both models. As usual in SAT/UNSAT models, the point where the expected number of solutions ($E[X]$) is small,

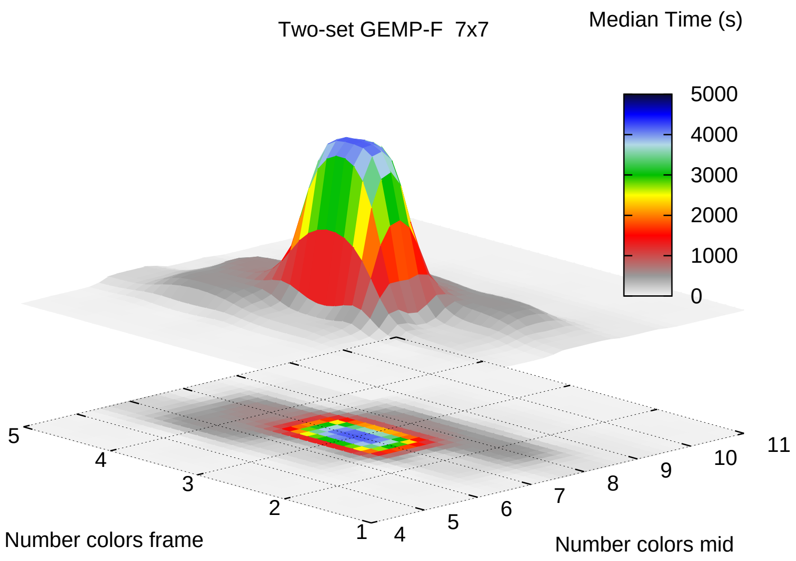


Figure 4.10: Hardness characteristic for a two-set GEMP-F as a function of the number of colors

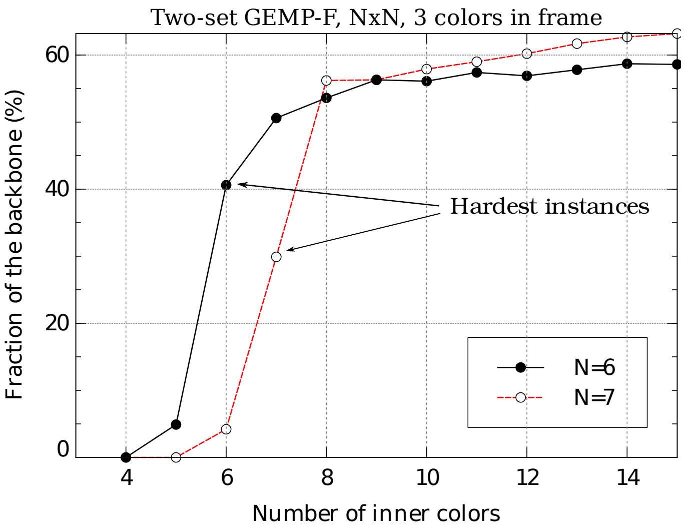


Figure 4.11: Phase transition of the percentage of backbone variables for a two-set GEMP-F

but not negligible, marks the phase transition [SD96, Pro96] for random CSP problems, being proved by [XL00] that such a transition occurs for $E[X] = 1$ on Model RB. Then, our first step is to find analytical expressions for the expected number of solutions on both models, one-set and two-set.

For a two-set GEMP-F, according to Definition 3.2, one can think on set T as $T = T_c \cup T_f \cup T_m$, being T_c, T_f and T_m the set of tokens corresponding to the corners, rest of the frame and mid of the board respectively.

Lets denote as $\mathcal{S} = \mathcal{S}_c \times \mathcal{S}_f \times \mathcal{S}_m$ the set of possible locations on the board for T_c, T_f and T_m jointly, and $\mathcal{C}$ the subset of $\mathcal{S}$ that satisfies 2-set GEMP-F rules. Clearly, considering a $n \times n$ board, and that only elements of the set T_m can be rotated:

$$|T_c| = 4, \quad |T_f| = 4(n-2), \quad |T_m| = (n-2)^2$$
$$|\mathcal{S}_c| = 4!, \quad |\mathcal{S}_f| = (4(n-2))!, \quad |\mathcal{S}_m| = 4^{(n-2)^2} \cdot ((n-2)^2)!$$

We define X as the random variable that denotes the number of satisfying locations according to the rules of 2-set GEMP-F puzzles, (i.e. the elements of $\mathcal{C}$). So, its expectation can be expressed as

$$\begin{aligned} E[X] &= E\left[\sum_{\sigma\in\mathcal{S}} \mathbf{1}_{\mathcal{C}}(\sigma)\right] = \sum_{\sigma\in\mathcal{S}} E\left[\mathbf{1}_{\mathcal{C}}(\sigma)\right] \\ &= \sum_{\sigma_c\in\mathcal{S}_c}\sum_{\sigma_f\in\mathcal{S}_f}\sum_{\sigma_m\in\mathcal{S}_m} E\left[\mathbf{1}_{\mathcal{C}}(\sigma_c\times\sigma_f\times\sigma_m)\right] \\ &= 4!\cdot(4(n-2))!\cdot 4^{(n-2)^2}\cdot(n-2)^2!\cdot E\left[\mathbf{1}_{\mathcal{C}}(\sigma_c\times\sigma_f\times\sigma_m)\right], \end{aligned}$$

being $\mathbf{1}_A(x)$ the indicator function, i.e., takes value 1 if $x \in A$ and 0 if $x \notin A$. We claim that $E\left[\mathbf{1}_{\mathcal{C}}(\sigma_c\times\sigma_f\times\sigma_m)\right]$ is the probability that a given arrangement of tokens satisfies a 2-set GEMP-F puzzle. If tokens are build randomly, such a probability is

$$E\left[\mathbf{1}_{\mathcal{C}}(\sigma_c\times\sigma_f\times\sigma_m)\right] = \left(\frac{1}{c_f}\right)^{4(n-1)} \cdot \left(\frac{1}{c_m}\right)^{2(n-1)(n-2)},$$

being c_f and c_m the number of colors in frame and mid, respectively, and giving

$$E[X] = 4!\cdot(4(n-2))!\cdot 4^{(n-2)^2}\cdot(n-2)^2!\cdot\left(\frac{1}{c_f}\right)^{4(n-1)}\cdot\left(\frac{1}{c_m}\right)^{2(n-1)(n-2)} \tag{4.4}$$

Analogously, one can derive an exact expression for one-set GEMP-F, resulting

$$E[X] = 4!\cdot(4(n-2))!\cdot 4^{(n-2)^2}\cdot(n-2)^2!\cdot\left(\frac{1}{c}\right)^{2n(n-1)} \tag{4.5}$$

Table 4.2: Round of $\log_{10}(E[X])$ according to Eq. 4.4 for two-set GEMP-F. Shadowed cells shows where the hardest problems have been experimentally found

	n = 6				**n = 7**					**n = 8**		
$\mathbf{c_f \setminus c_m}$	4	5	6	7	4	5	6	7	8	6	7	8
3	4	0	-3	-6	12	7	2	-2	-6	10	4	-1
4	2	-2	-6	-8	9	4	-1	-5	-9	6	1	-4
5					7	1	-3	-7	-11	3	-2	-7
6					5	-1	-5	-9	-13	1	-4	-9

Of course, we have not the same level of granularity on GEMP problems than in Random CSP models, and we are not able to tune our parameters to lead $E[X]$ to a desired point, but we can observe in Table 4.2 how the point where $E[X]$ changes from many to few solutions predicts where the harder instances are. Furthermore, it is worth to note that for $n = 16$ and $c_f = 5$ the predicted phase transition occurs at $c_m = 17$ that is exactly the number of inner colors of the two-set GEMP-F puzzle used in Eternity II contest.

As shown on Table 4.2, hard instances may be found for one or two contiguous values of c_m, meaning that their respective median times to solve are equivalent. That is usual for small orders, tending to disappear for larger n and therefore concentrating their hard problems for a given value of c_m. Actually, using Markov inequality that gives an upper bound to the probability of having a satisfiable instance, $P(Sat) \leq E[X]$, it can be shown that $\lim_{n\to\infty} P(Sat) = 0$ beyond a critical value of $c_m > c_{m_{cr}}$. It is straightforward to prove from Equations 4.5 and 4.4 that $c_{m_{cr}} = \frac{2n}{\sqrt{e}}$. Same result stands for one-set GEMP-F model.

5 Hardening CSP problems

> It is one of man's curious idiosyncracies to create difficulties for the pleasure of resolving them.
>
> Joseph de Maistre

Hardening search problems is, from an abstract point of view, pretty simple. If we use the time to solve as a measure on how hard problems are we can easily deduce one simple method to harden them, make the solver spend a lot of time exploring the search tree. A general knowledge on how do solvers work give us a first insight on how to make those problems hard, make hard for the solver, in every step of its exploration of the search tree hard to choose which is the best move towards the solution, and make those branching even harder by increasing, as much as possible, the branching factor.

If we look at CSP problems we can easily see how to achieve, at least partially, those goals. One method is to *balance*, i.e. equalize, the number of constraints, of variables involved in each constraint, of domain sizes for each variable, and so on. This will make hard for the solver to chose which constraint try to satisfy, which domain value use to propagate a variable, etc. thus increasing the uncertainity level on each branching decision.

Contents

5.1 Balancing GSP

5.1.1 Balanced Hole Patterns

Once a valid GS is generated, to create a GSWH we must punch holes to be filled. The most simple method to remove values from the GS is to choose which cells will be removed randomly. This creates problems that will be, usually, easier to solve than if we choose these holes following a pattern, specially if such pattern is balanced, that is, that the number of holes in every row, column or block region is the same (or very similar). We will present here three methods to punch those holes, each one progressively providing a more refined pattern, and we will see later, in the experimental results, that this every time more refined balance, has a heavy influence on problem hardness.

Singly Balanced

First, we consider the balanced pattern used in [KRA$^+$01] for QWH instances, that we call here *singly balanced*. In a singly balanced pattern we have, when possible, the same number of holes in every row and column of the Sudoku. Given the total number of holes h, we can distribute $q = h/s$ holes in each row and column of the Sudoku using an algorithm for regular bipartite graph generation, based on a Markov Chain algorithm [KTV97]. Observe that a hole pattern with q holes in every row and every column is equivalent to a $q-$regular bipartite graph $(R \cup C, E)$, with R the set of rows of the Sudoku and C its set of columns and $(r, c) \in E$ indicates that there is a hole in position (r, c) of the Sudoku. We move along the Markov chain, where every state is a pattern that satisfies that the number of holes in each row and column is the same, using a "switch" [KTV97]. A switch is defined by two entries $(i, j), (i', j')$ of the GS instance, such that there is a hole in (i, j) and (i', j') but not in (i, j') and (i', j). If we change both holes to positions (i, j') and (i', j) the row and column hole number does not change. When $q = h/s$ is not an integer, we can still generate, with the same algorithm, an almost balanced pattern with a bipartite graph where the degree of the vertices either $\lfloor q \rfloor$ or $\lfloor q \rfloor + 1$. In this case, we create the initial hole pattern putting $\lfloor q \rfloor$ holes in every row and column of the GS. Then we distribute the remaining $(h \ mod \ s)$ holes by randomly selecting $(h \ mod \ s)$ additional cells with no rows or columns in common.

Doubly Balanced

Because the distribution of holes between different blocks can make a difference in the difficulty of the problem, we propose a new method that ensures that both; the number of holes in every row and column, and the number of holes in every block will be the same. Our new *doubly balanced* method is based on the Markov chain of §5.1.1, but now every state is a hole pattern that also satisfies that the number of holes is the same in all blocks. So, we use this Markov chain, but we restrict the moves to those moves that also maintain the number of holes present in each block, i.e., moves such that the blocks of (i, j) and (i', j') are either in the same block row or the same block column, or both. Such moves always exist, even for a hole pattern with only one hole in each block. With this, we have the code detailed in Algorithm 5.1 for generating a hole pattern H with $q = h/s$ holes in each row, column and block, using a GS $S(i, j)$ (with symbols $\{1, \ldots, s\}$) to create the initial pattern considering each symbol as a hole, and then performing t moves trough the Markov chain in order to sample from the set of possible doubly balanced hole patterns.

Algorithm 5.1: Algorithm to create Doubly Balanced hole patterns in a given GS

```
input : $s, h, t$
output: a doubly balanced hole pattern $H$ of order $s$ with $h$ holes
$H = \{\ (i,j) \mid S(i,j) \in [1, \lfloor h/s \rfloor]\ \}$
for $1 \ldots t$ do
    $T = \{\ switch((i,j),(i',j'))\ of\ H \mid \lfloor i/l \rfloor = \lfloor i'/l \rfloor \ \vee\ \lfloor j/n \rfloor = \lfloor j'/n \rfloor\ \}$
    pick a uniformly random $switch((i,j),(i',j'))$ from $T$
    $H = (H - \{(i,j),(i',j')\}) \ \cup\ \{(i,j'),(i',j)\}$
```

Observe that the Sudoku S considered to create the initial hole pattern can be any arbitrary Sudoku, and has no any relation with the Sudoku to which we want to apply the hole pattern. Building such an arbitrary Sudoku can be done using Equations 3.9 and 3.10.

This code, obviously, only generates a perfect doubly balanced pattern when h/s is an integer. If this is not the case, we generate a hole pattern that is almost a doubly balanced one. That is, a hole pattern H that contains $h \ mod \ s$ rows, columns and blocks with $\lfloor h/s \rfloor + 1$ holes each one and the remaining $s-(h \ mod \ s)$ rows, columns and blocks with $\lfloor h/s \rfloor$ holes each one. To get this hole pattern, we generate the initial pattern as in the Algorithm 5.1, but also selecting a random subset M (with $|M| = (h \ mod \ s)$) of the positions (i, j) of the Sudoku with $S(i, j) = \lfloor h/s \rfloor + 1$ to select the positions of the additional $(h \ mod \ s)$ holes. So, the initial pattern will be:

$$\begin{aligned} H &= \{ \ (i,j) \mid S(i,j) \in [1, \lfloor h/s \rfloor] \ \} \cup \\ &\quad \{ \ (i,j) \mid S(i,j) = \lfloor h/s \rfloor + 1 \wedge (i,j) \in M \} \end{aligned}$$

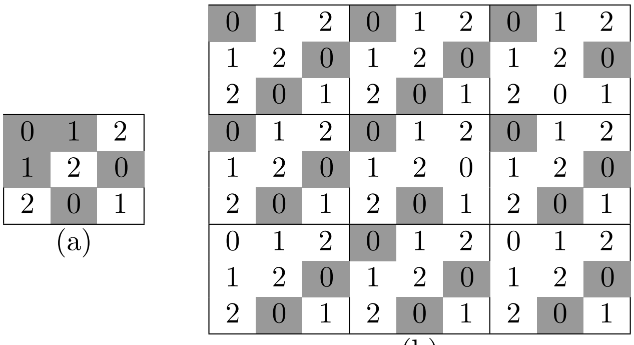
Fully Balanced

Our last balanced method, that we call *fully balanced*, is a generalization of the previous one. Here, we force also that the number of holes in each row and column inside each block to be the same. Of course, this is only possible if blocks are squared, so in this subsubsection we assume that $m = n$ and therefore $n = \sqrt{s}$. This method produces a fully balanced hole pattern if the total number of holes h satisfies that $q_1 = h/s$ is an integer (the number of holes in each block, row and column of the Sudoku) and if $q_2 = q_1/n$ is also an integer (q_2 is the number of holes in each row and column inside any block). If these conditions are met, because what we indeed need in every block is a hole pattern that is singly balanced inside the block, the following simple code generates a fully balanced Sudoku:

Algorithm 5.2: Algorithm to create Fully Balanced hole patterns in a given GS

```
input  : $s, h$
output: a fully balanced hole pattern $H$ of order $s$ with $h$ holes
for $i \in 1 \dots \sqrt{s}$ do
    for $j \in 1 \dots \sqrt{s}$ do
        $H'$ := Singly balanced hole pattern of order $\sqrt{s}$ with $\lfloor h/s \rfloor$ holes
        set the hole pattern of $H$ in block $(i, j)$ equal to $H'$
```

In case that $q_1 = h/s$ is not an integer, that means that we have an additional set of $(h \ mod \ s)$ holes that we need to distribute as uniformly as possible between the s blocks. To do it, consider a Latin Square $R(i, j)$ of order $\sqrt{s}$, with symbols $\{0, \dots, \sqrt{s}-1\}$, such that entry $R(i, j)$ is associated with the block in block row i and block column j of our desired hole pattern. We use $R(i, j)$ to decide which blocks will contain one additional hole. Let



(a) (b)

Figure 5.1: Fully balanced GS hole poking example. (a) LS of order $\sqrt{s}$, every cell is associated with a region block on the resulting GS instance. (b) Resulting hole pattern where grayed cells will be holes in the GS instance.

$q = \lfloor((h \ mod \ s) \ / \ \sqrt{s})\rfloor$ and $r = ((h \ mod \ s) \ mod \ \sqrt{s})$. Then, the set of blocks (i, j) that will contain one additional hole will be:

$$\{(i, j) \mid R(i, j) < q\} \cup \{(i, j) \mid R(i, j) = q \ \wedge \ i < r\} \qquad (5.1)$$

So, observe that we try to distribute the remaining $(h \ mod \ s)$ holes giving almost the same number to every block row and block column (when $r > 0$ some blocks rows and blocks columns will be assigned with one more hole than others).

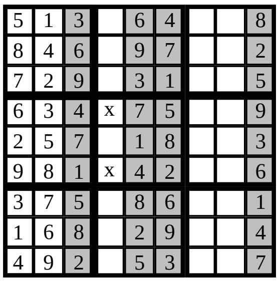
Finally, given a block (i, j) and the number of holes $h_{i,j}$ we have decided to distribute in the block, we basically use the Markov chain algorithm of §5.1.1 in order to have the same number of holes in every subrow and subcolumn of the block.

This hole punching method can be easily seen in Figure 5.1. In this example we poke 23 holes in a 9×9 sudoku puzzle, i.e. a 81 cell sudoku. As the number of holes is not a multiple of the number of block regions, we must punch 2 holes in every block region, and the remainding 5 holes will be punched using Equation 5.1. For this example, we have $h = 23$, $s = 9$, $q = 1$ and $r = 2$, thus Equation 5.1 results in

$$\{(i, j) \mid R(i, j) < 1\} \cup \{(i, j) \mid R(i, j) = 1 \ \wedge \ i < 2\}$$

this gives the following (i, j) values, refering to the following cells of LS in Figure 5.1(a) that will have one additional hole: $\{(0, 0), (1, 2), (2, 1)\} \cup \{(0, 1), (1, 0)\}$, shown in the figure as grayed cells.

Once the number of holes to poke in every block region is set forth, we proceed as if the block region was a LS of $\sqrt{s}$ order and punching holes. The results can be seen on Figure 5.1(b), where holes have been grayed, and, as



5	1	3		6	4			8
8	4	6		9	7			2
7	2	9		3	1			5
6	3	4	x	7	5			9
2	5	7		1	8			3
9	8	1	x	4	2			6
3	7	5		8	6			1
1	6	8		2	9			4
4	9	2		5	3			7

Figure 5.2: Grayed cells represent the initial assignment. The cells marked with × cannot be completed after the two first columns are completed in the way shown

it can be easily seen, block regions with one additional hole (that is, block regions with 3 holes), correspond to grayed cells in 5.1(a).

Rectangular Hole Poking Method

We also considered a different method for punching holes: This method is based on the rectangular model presented also in [KRA^{+}01]. The rectangular model selects a set of columns (or rows) and punches holes in all the cells of these columns: in the case of QWH, this method produces tractable instances [KRA^{+}01]; they can be solved using an algorithm based on bipartite graph matching. It is proved in this work that a similar hole pattern in the case of GSP instances corresponds to an untractable class. Figure 5.2 shows an example of a solvable 3x3 GSP instance, with a rectangular hole pattern. The initial assignment is indicated by the grayed cells. After two rounds of the bipartite graph matching algorithm, a possible outcome corresponds to the first two columns in the way shown; this configuration cannot be completed into a valid Sudoku (even though there exists a valid completion of the initial partially filled instance). So, the rectangular hole pattern could still provide hard instances for Sudoku. For that reason, we propose a rectangular model for Generalized Sudoku, that distributes a set of c hole columns between the different region columns of the Generalized Sudoku, in a uniform way. The uniform distribution of the hole columns tries to minimize the clustering of holes.

5.1.2 Complexity patterns of Balanced GSP

We will study the complexity of solving the balanced GSP generated with the methods above mentioned. First we will study the worst case and then we will present experimental results of hardness.

Worst-case Complexity of GSP

It has been shown in [YS02], that GSP is NP-complete for the particular case of square regions (n columns and n rows). Given that our empirical complexity results show that, on average, GSP with rectangular regions is easier than with square regions (the complexity decreasing the larger the ratio n/m) the next natural question to answer was if NP-completeness still applied to the rectangular case. We show here that this is the case.

Theorem 5.1. *GSP with block regions with m rows, n columns and $n \neq m$ is NP-complete.*

Proof. The proof for this case is a generalisation of the proof of [YS02], and shows a reduction from QCP (Quasigroup Completion Problem) to GSP (Generalized Sudoku Problem).

The following construction uses a GSP with $n > m$ but can be transformed to a isomorphic GSP with $m > n$, by simply rotating the GSP 90 degrees.

Given an instance of the QCP of order m, the reduction follows by constructing an instance of GSP with region blocks with m rows and n columns such that the GSP instance has solution iff the QCP instance has solution. The coefficients of the QCP instance are embedded into the first columns of the regions of the first region row.

Let L be the QCP instance and S the GSP instance, and let denote their coefficients

$$\begin{aligned} L &= (L_{i,j}), \quad 0 \leq i,j \leq m-1; \\ S &= (S_{i,j}^{k,l}), \quad 0 \leq i,l \leq n-1, 0 \leq j,k \leq m-1, \end{aligned}$$

where $S_{i,j}^{k,l}$ corresponds to the coefficient of S located at k-th row and l-th column inside i-th region row, j-th region column.

Then, these coefficients are ordered pairs defined as

$$S_{i,j}^{k,l} = \begin{cases} (L_{k,j}, 0), \text{ if } i = l = 0 \\ (k + j \,(\text{mod } m), i + l \,(\text{mod } n)), \text{ otherwise.} \end{cases}$$

Under this construction, the original (or empty) coefficients of the QCP instance are placed at positions with $i = l = 0$.

It is straightforward to observe the GSP instances with rectangular block regions have solution if and only if the QCP instance has solution. ∎

Finally, there is still one last case that makes GSP even more interesting than the problem it generalizes (QCP). In the work [KRA$^+$01] it was shown that QCP, when every column is either empty or full of symbols, can be solved in polynomial time. By contrast, GSP with every column either empty or full of symbols (referred as GSP with column hole pattern) is NP-complete. An equivalent result is also true for row hole patterns.

Theorem 5.2. *GSP with block regions with n rows and n^2+1 columns and with column hole pattern, is NP-complete.*

In this appendix we show NP-completeness of GSP with column hole pattern (proof of Theorem 5.2). The same proof can be adapted to show a similar result for row hole patterns.

The main idea of the proof is to define a reduction from QCP to GSP with column hole patterns. Given a QCP instance of order n, the reduction follows by constructing a GSP instance of order $n(n^2+1)$, which has n region columns and n^2+1 region rows, and column hole pattern. The overall idea of this reduction is the same as the proof for Theorem 5.1. That is, columns in the QCP instance are mapped to the first column of each rectangular region in the first region row. The remaining cells are filled using a particular class of Latin Squares (named Canonical Zero-Diagonal Latin Square, CZD-LS), of order n^2+1. Finally, the first column of each region column is emptied, after conveniently swapping the cells mapped from the original QCP instance to preserve their effect. It can be shown that a solution of such a GSP instance exists if, and only if, a solution of the original QCP instance also exists.

In the following paragraphs we define and construct Canonical Zero-Diagonal Latin Squares. Afterwards we will present the reduction from QCP to GSP with column hole pattern.

Canonical Zero-Diagonal Latin Square, CZD-LS

A Canonical Zero-Diagonal Latin Square (CZD-LS) of order n is a Latin Square $L = (L_{i,j}) \in M_{n\times n}(\mathbb{Z}_n)$, $0 \leq i, j \leq n-1$ such that the coefficients of its main diagonal are 0, i.e. $L_{i,i} = 0$, and that the coefficients in the first row and first column are the elements of the sequence $(0, 1, \ldots, n-1)$, that is $L_{i,0} = i$ and $L_{0,j} = j$, for $0 \leq i, j \leq n-1$.

The following matrix corresponds to a CZD-LS of order $n = 10$,

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 0 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 2 & 3 & 0 & 5 & 6 & 7 & & & & \end{pmatrix}$$

Proposition 5.3. *For any positive integer n there exists a CZD-LS of order n.*

Proof. The existence of such a matrix will be proven by providing an explicit construction. For such a construction, we will distinguish between two cases:

- *Case n even.*
  In this case, the coefficients $L_{i,j}$ of a Canonical Zero-Diagonal Latin Square L can be constructed as follows,

$$L_{i,j} = \begin{cases} 0, & \text{if } i = j, \\ i+j \pmod n, & \text{if } j \leq n-i-1 \text{ and } i \neq j, \\ i+j+1 \pmod n, & \text{if } n-i-1 < j < n-1 \text{ and } i < n-1, \\ 2i \pmod n, & \text{if } 0 < i < \frac{n}{2} \text{ and } j = n-1, \\ 2j \pmod n, & \text{if } i = n-1 \text{ and } 0 < j < \frac{n}{2}, \\ 2(i-\frac{n}{2})+1 \pmod n, & \text{if } \frac{n}{2} \leq i < n-1 \text{ and } j = n-1, \\ 2(j-\frac{n}{2})+1 \pmod n, & \text{otherwise.} \end{cases}$$

Notice that this matrix is symmetric. Hence, to show that it is a Latin Square, it is enough to show that the values in each row are different. We should distinguish between the following three situations:

- Case $0 \leq i < n/2$.
  In row i, the consecutive values are

  $$i, i+1, \ldots, 2i-1, 0, 2i+1, \ldots, n-1, 1, 2, \ldots i-1, 2i,$$

  hence, they cover all possible values from 0 to $n-1$.

- Case $n/2 \leq i < n-1$.
  In this case, the values for row i are:

  $$i, i+1, \ldots, n-1, 1, 2, \ldots, 2i, 0, 2i+2, \ldots i-1, 2i+1.$$

  Again, they cover all possible values.

- Case $i = n-1$.
  In the last row, the values are

  $$n-1, 2, 4, \ldots, n-2, 1, 3, 5, \ldots, n-3, 0,$$

  so they also are pairwise different.

- *Case n odd.*
  In the case that n is odd, consider a Latin Square L' of even order $n-1$. Then, a CZD-LS L with order n can be obtained from L' proceeding in the following way:

  - To obtain L, add to L' an extra column and an extra row.
  - Then $L_{0,n-1} = n-1$, $L_{n-1,0} = n-1$ and $L_{n-1,n-1} = 0$.

For example, in the case $n = 11$,

$$L = \begin{pmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
1 & 0 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 & \\
2 & 3 & 0 & 5 & 6 & 7 & 8 & 9 & 1 & 4 & \\
3 & 4 & 5 & 0 & 7 & 8 & 9 & 1 & 2 & 6 & \\
4 & 5 & 6 & 7 & 0 & 9 & 1 & 2 & 3 & 8 & \\
5 & 6 & 7 & 8 & 9 & 0 & 2 & 3 & 4 & 1 & \\
6 & 7 & 8 & 9 & 1 & 2 & 0 & 4 & 5 & 3 & \\
7 & 8 & 9 & 1 & 2 & 3 & 4 & 0 & 6 & 5 & \\
8 & 9 & 1 & 2 & 3 & 4 & 5 & 6 & 0 & 7 & \\
9 & 2 & 4 & 6 & 8 & 1 & 3 & 5 & 7 & 0 & \\
10 & & & & & & & & & & 0
\end{pmatrix}$$

- Consider the submatrix of L consisting of columns from 2 until $n-2$, and rows also from 2 until $n-2$.
  In the previous example,

$$L = \left(\begin{array}{c|ccccccccc|c}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \cline{2-10}
1 & 0 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 & \\
2 & 3 & 0 & 5 & 6 & 7 & 8 & 9 & 1 & 4 & \\
3 & 4 & 5 & 0 & 7 & 8 & 9 & 1 & 2 & 6 & \\
4 & 5 & 6 & 7 & 0 & 9 & 1 & 2 & 3 & 8 & \\
5 & 6 & 7 & 8 & 9 & 0 & 2 & 3 & 4 & 1 & \\
6 & 7 & 8 & 9 & 1 & 2 & 0 & 4 & 5 & 3 & \\
7 & 8 & 9 & 1 & 2 & 3 & 4 & 0 & 6 & 5 & \\
8 & 9 & 1 & 2 & 3 & 4 & 5 & 6 & 0 & 7 & \\
9 & 2 & 4 & 6 & 8 & 1 & 3 & 5 & 7 & 0 & \\ \cline{2-10}
10 & & & & & & & & & & 0
\end{array}\right)$$

- In such submatrix, consider a *latin transversal*[1]. Such a transversal always exists in this submatrix. One possible choice would be taking the following coefficients: $L_{1,n-2}, L_{2,n-4}, L_{i,i-2}$ for $i \in \{3, \dots, n-3\}$, and $L_{n-2,n-3}$.
  Following with the example, such coefficients are the ones typed in bold,

[1] A *transversal array* is a set of n cells in an $n \times n$ square such that no two come from the same row and no two come from the same column. Then, a *latin transversal* is a transversal such that no two cells contain the same element.

$$L = \left(\begin{array}{c|ccccccccc|c}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \cline{2-10}
1 & 0 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \mathbf{2} & \\
2 & 3 & 0 & 5 & 6 & 7 & 8 & \mathbf{9} & 1 & 4 & \\
3 & \mathbf{4} & 5 & 0 & 7 & 8 & 9 & 1 & 2 & 6 & \\
4 & 5 & \mathbf{6} & 7 & 0 & 9 & 1 & 2 & 3 & 8 & \\
5 & 6 & 7 & \mathbf{8} & 9 & 0 & 2 & 3 & 4 & 1 & \\
6 & 7 & 8 & 9 & \mathbf{1} & 2 & 0 & 4 & 5 & 3 & \\
7 & 8 & 9 & 1 & 2 & \mathbf{3} & 4 & 0 & 6 & 5 & \\
8 & 9 & 1 & 2 & 3 & 4 & \mathbf{5} & 6 & 0 & 7 & \\
9 & 2 & 4 & 6 & 8 & 1 & 3 & 5 & \mathbf{7} & 0 & \\ \cline{2-10}
\multicolumn{1}{c}{10} & & & & & & & & & & 0
\end{array}\right)$$

– Then, the value of each coefficient in the transversal is placed at the last column and last row, and replaced by value $n-1$.
In our example,

$$L = \left(\begin{array}{c|ccccccccc|c}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ \cline{2-10}
1 & 0 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & \mathbf{10} & 2 \\
2 & 3 & 0 & 5 & 6 & 7 & 8 & \mathbf{10} & 1 & 4 & 9 \\
3 & \mathbf{10} & 5 & 0 & 7 & 8 & 9 & 1 & 2 & 6 & 4 \\
4 & 5 & \mathbf{10} & 7 & 0 & 9 & 1 & 2 & 3 & 8 & 6 \\
5 & 6 & 7 & \mathbf{10} & 9 & 0 & 2 & 3 & 4 & 1 & 8 \\
6 & 7 & 8 & 9 & \mathbf{10} & 2 & 0 & 4 & 5 & 3 & 1 \\
7 & 8 & 9 & 1 & 2 & \mathbf{10} & 4 & 0 & 6 & 5 & 3 \\
8 & 9 & 1 & 2 & 3 & 4 & \mathbf{10} & 6 & 0 & 7 & 5 \\
9 & 2 & 4 & 6 & 8 & 1 & 3 & 5 & \mathbf{10} & 0 & 7 \\ \cline{2-10}
\multicolumn{1}{c}{10} & 4 & 6 & 8 & 1 & 3 & 5 & 9 & 7 & \multicolumn{1}{c}{2} & 0
\end{array}\right)$$

Following this construction, it can be easily proven that coefficients in the same row or in the same column are always different. ∎

A reduction from PLS to GSP with rectangular hole pattern

Let L be the original PLS of size n and C the CZD-LS of size n^2+1. Their coefficients are denoted as:

$$\begin{array}{ll} L = (L_{i,j}), & 0 \le i,j \le n-1; \\ C = (C_{i,j}), & 0 \le i,j \le n^2. \end{array}$$

Empty cells in L are assigned the value $\perp$. Notice that the first row and column are numbered as zero.

Then the GS of size $n(n^2+1)$ is denoted as

$$S = (S_{i,j}^{k,l}), \quad 0 \le i,l \le n^2, \quad 0 \le j,k \le n-1,$$

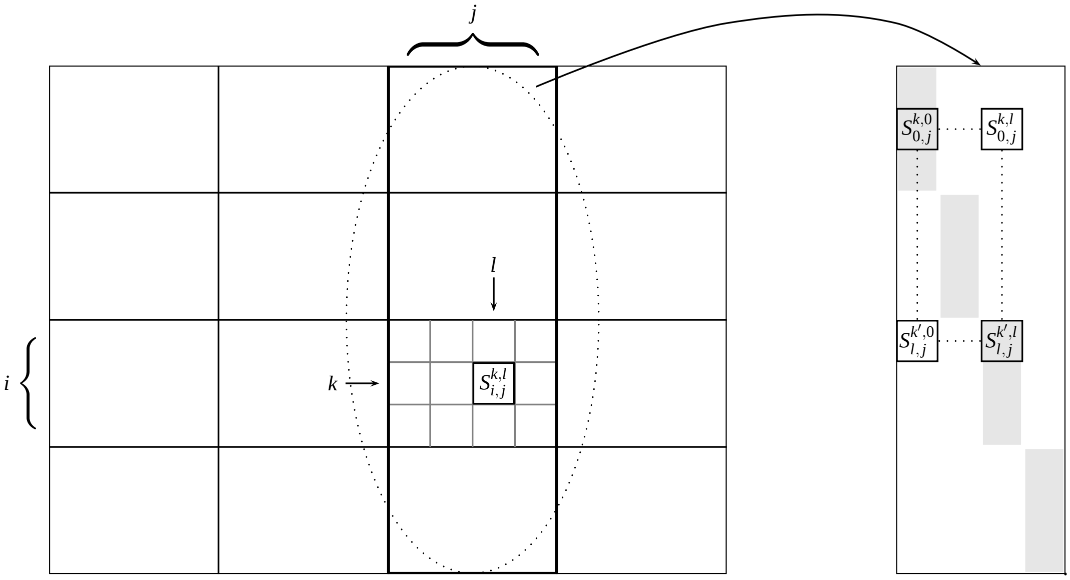


Figure 5.3: Detail of the structure of the region columns. By construction we have $S^{k,0}_{0,j} = S^{k',l}_{l,j}$ and $S^{k,l}_{0,j} = S^{k',0}_{l,j}$

where $S^{k,l}_{i,j}$ corresponds to the coefficient located at i-th region row, j-th region column, and inside such a region it is placed on the k-th row and l-th column. These coefficients are ordered pairs, defined as follows

$$S^{k,l}_{i,j} = \begin{cases} (L_{k,j}, C_{0,0}) = (L_{k,j}, 0), & \text{if } i = l = 0, \\ (j + k \pmod{n},\, C_{i,i}) = (j + k \pmod{n}, 0), & \text{if } i = l \neq 0, \\ (i + j + k \pmod{n},\, C_{i,l}), & \text{otherwise.} \end{cases}$$

In fact, the first condition is the responsible for embedding the information from the PLS to the GS. When in the PLS the cell is empty, it remains also empty in the GS. Figure 5.4 shows an example of this construction for a PLS of size $n = 3$.

It can be easily seen that this construction is a sudoku with holes. This is due to the fact that the second component corresponds to the cells of the CZD-LS, which is combined with a first component whose values are $0, 1, \ldots, n-1$, which ensures that the contents are different in each row, each column and each region.

At the end, the aim of the construction is obtaining a GS with rectangular pattern, that is, where columns with parameter $l = 0$ are empty. For such a purpose, since at the moment these columns contain the information embedded from the original PLS, the construction goes on swapping this cells, but preserving the properties of being a GS. We proceed as follows:

- Consider every non–empty cell $S^{k,0}_{0,j}$ (that is, the ones placed on the first columns of each region in the first region row). Notice that these cells are the ones that have inherited the information in the PLS, $S^{k,0}_{0,j} = (L_{k,j}, 0)$.

- For each $S^{k,0}_{0,j}$ that is different to $\perp$, we will take one of its *mates*, that is, another cell in the same row, $S^{k,l}_{0,j}$, $l \neq 0$, such that there exists k' for which it holds that $S^{k,0}_{0,j} = S^{k',l}_{l,j}$ and $S^{k,l}_{0,j} = S^{k',0}_{l,j}$ (that is, the opposite corners of the rectangle defined by these positions coincide). In fact, Proposition 5.4 shows that there always exist exactly n mates.

- Then, for each $S^{k,0}_{0,j}$, we take one of its mates, and swap their contents, and we also swap the contents of the cells in the other two corners of the rectangle (that is $S^{k',0}_{l,j}$ and $S^{k',l}_{l,j}$). In the case that there exists two cells $S^{k,0}_{0,j}$ and $S^{k',0}_{0,j'}$ with the same value, one should take care in choosing different mates (next results show that it is always possible).

Proposition 5.4. *For every values $0 \leq j, k \leq n-1$, every non-empty cell $S^{k,0}_{0,j}$ has exactly n mates.*

Proof. Let us fix the values j and k. From the construction of S it follows that $S^{k,0}_{0,j} = (L_{k,j}, 0)$. Hence, the cells that can be located at its opposite corner have to be assigned value 0 in its second component, which are, by construction, $S^{k',l}_{l,j}$, for $0 \leq k' \leq n-1$ and $1 \leq l \leq n^2$

Notice that the contents of the n cells $S^{k',l}_{l,j}$ is $(j + k' (\text{mod } n), 0)$, for $0 \leq k' \leq n-1$. Hence, $S^{k,0}_{0,j}$ will be equal to $S^{k',l}_{l,j}$ for $k' \equiv L_{k,j} - j \pmod n$

We are looking forward the existence of values for l such that verify that the two other corners coincide, that is, $S^{k',0}_{l,j} = S^{k,l}_{0,j}$. The values of these cells are

$$\begin{array}{rcl} S^{k',0}_{l,j} & = & (l + j + k' (\text{mod } n), C_{l,0}) = (l + j + k' (\text{mod } n), l) \\ S^{k,l}_{0,j} & = & (j + k (\text{mod } n), C_{0,l}) = (j + k (\text{mod } n), l) \end{array}$$

Hence, equality will hold if, and only if,

$$l + j + k' \equiv j + k (\text{mod } n) \Longleftrightarrow l \equiv k - k' (\text{mod } n) \equiv j + k - L_{k,j} (\text{mod } n).$$

It straightforwardly follows that there exist $n^2/n = n$ values for l that satisfy this condition, which determine the position of the mates.

∎

Proposition 5.5. *Given the set of values $S^{k_1,0}_{0,j_1}, S^{k_2,0}_{0,j_2}, \ldots, S^{k_n,0}_{0,j_n}$ embedded from the PLS, their mates can be taken pairwise different.*

Proof. Firstly, notice that the sets of possible mates for two different values $S^{k,0}_{0,j} \neq S^{k',0}_{0,j'}$ are disjoint. The first component of the mates will be $j + k \pmod n$ and $j' + k' \pmod n$, respectively. If these values are different, obviously the sets of possible mates also will be. In the case that $j+k \equiv j'+k' \pmod n$, the sets of mates will be in columns $l \equiv j + k - L_{j,k} \pmod n$ and $l \not\equiv j' + k' - L_{j',k'} \pmod n$, respectively, so they are also disjoint.

Hence, the problem could appear for cells with the same value, namely $S^{k,0}_{0,j} = S^{k',0}_{0,j'}$. Again, when $j+k \not\equiv j'+k' \pmod n$, their set of possible mates will be disjoint. In the case that $j + k \equiv j' + k' \pmod n$, their set of mates will be located at columns $l \equiv j + k + L_{k,j} \equiv j' + k' + L_{k',j'} \pmod n$. In this last case, the repeated value can appear at most n times, so n different mates can be taken.

∎

In the example in Figure 5.4, it can be seen that, for instance, the mates of $S^{1,0}_{0,2} = (2, 0)$ will be located at positions

$$l \equiv k - k' \equiv 1 - 0 \equiv 1(\text{mod } 3) \Longrightarrow l = 1, 4, 7.$$

We take one of its three mates, for instance the one with $l = 7$. Indeed, we have $S^{1,7}_{0,2} = (0, 7)$, $S^{0,0}_{7,2} = (0, 7)$ and $S^{0,7}_{7,2} = (2, 0)$. Hence, contents of these pairs of cells in each row can be swapped, while maintaining the property of being a sudoku with holes.

Then, in the last step of the construction, the first column of each column region is emptied, namely those positions with $l = 0$.

Finally, the following proposition shows that the Partial Latin Square Problem can be reduced to the General Sudoku with Column Holes Problem.

Theorem 5.6. *Let L be a PLS, and let T be the GS with holes obtained under the previously detailed construction. The Latin Square L has a solution if, and only if, T has a solution.*

Proof. On the one hand, notice that if a solution of L exists, a solution of T can be also obtained: originally non–empty cells located at $l = 0$ can be assigned the values obtained in the construction, before emptying the columns, and those originally empty cells (located at $i = l = 0$) can be assigned the values of the solution of L, along with second component 0.

On the other way round, if a solution of T exists, each empty hole $L_{k,j}$ can be assigned the first component of $T^{k,0}_{0,j}$. Firstly, notice that, in cells with $i = l = 0$ in T which were originally non–empty, in the solution must appear the chosen mate (since every mate was taken at most once). After the swapping, an original value $L_{k,j}$ of the PLS is embedded in some cell $T^{k,l}_{0,j}$. So, every value in its same row and region must be different. Hence, the solution values assigned to originally empty cells conform a solution of the PLS. ∎

Finally, from this result, Theorem 5.2 straightforwardly follows.

		0
		2
1		

→

	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9		1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9	0,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9
	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9		2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9
1,0	2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9		0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9		1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
1,1	0,0	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,2	2,1	1,0	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,2	0,1	2,0	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,2
2,1	1,0	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,2	0,1	2,0	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,2	1,1	0,0	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,2
0,1	2,0	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,2	1,1	0,0	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,2	2,1	1,0	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,2
2,2	2,3	0,0	2,5	2,6	2,7	2,8	2,9	2,1	2,4	0,2	0,3	1,0	0,5	0,6	0,7	0,8	0,9	0,1	0,4	1,2	1,3	2,0	1,5	1,6	1,7	1,8	1,9	1,1	1,4
0,2	0,3	1,0	0,5	0,6	0,7	0,8	0,9	0,1	0,4	1,2	1,3	2,0	1,5	1,6	1,7	1,8	1,9	1,1	1,4	2,2	2,3	0,0	2,5	2,6	2,7	2,8	2,9	2,1	2,4
1,2	1,3	2,0	1,5	1,6	1,7	1,8	1,9	1,1	1,4	2,2	2,3	0,0	2,5	2,6	2,7	2,8	2,9	2,1	2,4	0,2	0,3	1,0	0,5	0,6	0,7	0,8	0,9	0,1	0,4
0,3	0,4	0,5	0,0	0,7	0,8	0,9	0,1	0,2	0,6	1,3	1,4	1,5	1,0	1,7	1,8	1,9	1,1	1,2	1,6	2,3	2,4	2,5	2,0	2,7	2,8	2,9	2,1	2,2	2,6
1,3	1,4	1,5	1,0	1,7	1,8	1,9	1,1	1,2	1,6	2,3	2,4	2,5	2,0	2,7	2,8	2,9	2,1	2,2	2,6	0,3	0,4	0,5	0,0	0,7	0,8	0,9	0,1	0,2	0,6
2,3	2,4	2,5	2,0	2,7	2,8	2,9	2,1	2,2	2,6	0,3	0,4	0,5	0,0	0,7	0,8	0,9	0,1	0,2	0,6	1,3	1,4	1,5	1,0	1,7	1,8	1,9	1,1	1,2	1,6
1,4	1,5	1,6	1,7	0,0	1,9	1,1	1,2	1,3	1,8	2,4	2,5	2,6	2,7	1,0	2,9	2,1	2,2	2,3	2,8	0,4	0,5	0,6	0,7	2,0	0,9	0,1	0,2	0,3	0,8
2,4	2,5	2,6	2,7	1,0	2,9	2,1	2,2	2,3	2,8	0,4	0,5	0,6	0,7	2,0	0,9	0,1	0,2	0,3	0,8	1,4	1,5	1,6	1,7	0,0	1,9	1,1	1,2	1,3	1,8
0,4	0,5	0,6	0,7	2,0	0,9	0,1	0,2	0,3	0,8	1,4	1,5	1,6	1,7	0,0	1,9	1,1	1,2	1,3	1,8	2,4	2,5	2,6	2,7	1,0	2,9	2,1	2,2	2,3	2,8
2,5	2,6	2,7	2,8	2,9	0,0	2,2	2,3	2,4	2,1	0,5	0,6	0,7	0,8	0,9	1,0	0,2	0,3	0,4	0,1	1,5	1,6	1,7	1,8	1,9	2,0	1,2	1,3	1,4	1,1
0,5	0,6	0,7	0,8	0,9	1,0	0,2	0,3	0,4	0,1	1,5	1,6	1,7	1,8	1,9	2,0	1,2	1,3	1,4	1,1	2,5	2,6	2,7	2,8	2,9	0,0	2,2	2,3	2,4	2,1
1,5	1,6	1,7	1,8	1,9	2,0	1,2	1,3	1,4	1,1	2,5	2,6	2,7	2,8	2,9	0,0	2,2	2,3	2,4	2,1	0,5	0,6	0,7	0,8	0,9	1,0	0,2	0,3	0,4	0,1
0,6	0,7	0,8	0,9	0,1	0,2	0,0	0,4	0,5	0,3	1,6	1,7	1,8	1,9	1,1	1,2	1,0	1,4	1,5	1,3	2,6	2,7	2,8	2,9	2,1	2,2	2,0	2,4	2,5	2,3
1,6	1,7	1,8	1,9	1,1	1,2	1,0	1,4	1,5	1,3	2,6	2,7	2,8	2,9	2,1	2,2	2,0	2,4	2,5	2,3	0,6	0,7	0,8	0,9	0,1	0,2	0,0	0,4	0,5	0,3
2,6	2,7	2,8	2,9	2,1	2,2	2,0	2,4	2,5	2,3	0,6	0,7	0,8	0,9	0,1	0,2	0,0	0,4	0,5	0,3	1,6	1,7	1,8	1,9	1,1	1,2	1,0	1,4	1,5	1,3
1,7	1,8	1,9	1,1	1,2	1,3	1,4	0,0	1,6	1,5	2,7	2,8	2,9	2,1	2,2	2,3	2,4	1,0	2,6	2,5	0,7	0,8	0,9	0,1	0,2	0,3	0,4	2,0	0,6	0,5
2,7	2,8	2,9	2,1	2,2	2,3	2,4	1,0	2,6	2,5	0,7	0,8	0,9	0,1	0,2	0,3	0,4	2,0	0,6	0,5	1,7	1,8	1,9	1,1	1,2	1,3	1,4	0,0	1,6	1,5
0,7	0,8	0,9	0,1	0,2	0,3	0,4	2,0	0,6	0,5	1,7	1,8	1,9	1,1	1,2	1,3	1,4	0,0	1,6	1,5	2,7	2,8	2,9	2,1	2,2	2,3	2,4	1,0	2,6	2,5
2,8	2,9	2,1	2,2	2,3	2,4	2,5	2,6	0,0	2,7	0,8	0,9	0,1	0,2	0,3	0,4	0,5	0,6	1,0	0,7	1,8	1,9	1,1	1,2	1,3	1,4	1,5	1,6	2,0	1,7
0,8	0,9	0,1	0,2	0,3	0,4	0,5	0,6	1,0	0,7	1,8	1,9	1,1	1,2	1,3	1,4	1,5	1,6	2,0	1,7	2,8	2,9	2,1	2,2	2,3	2,4	2,5	2,6	0,0	2,7
1,8	1,9	1,1	1,2	1,3	1,4	1,5	1,6	2,0	1,7	2,8	2,9	2,1	2,2	2,3	2,4	2,5	2,6	0,0	2,7	0,8	0,9	0,1	0,2	0,3	0,4	0,5	0,6	1,0	0,7
0,9	0,2	0,4	0,6	0,8	0,1	0,3	0,5	0,7	0,0	1,9	1,2	1,4	1,6	1,8	1,1	1,3	1,5	1,7	1,0	2,9	2,2	2,4	2,6	2,8	2,1	2,3	2,5	2,7	2,0
1,9	1,2	1,4	1,6	1,8	1,1	1,3	1,5	1,7	1,0	2,9	2,2	2,4	2,6	2,8	2,1	2,3	2,5	2,7	2,0	0,9	0,2	0,4	0,6	0,8	0,1	0,3	0,5	0,7	0,0
2,9	2,2	2,4	2,6	2,8	2,1	2,3	2,5	2,7	2,0	0,9	0,2	0,4	0,6	0,8	0,1	0,3	0,5	0,7	0,0	1,9	1,2	1,4	1,6	1,8	1,1	1,3	1,5	1,7	1,0

Figure 5.4: First step in the reduction. The PLS is mapped to the set of positions B of the Sudoku build as explained in the text. Every symbol $(a, 0)$ in the positions B will be swapped with the symbol that appears in bold face in the same row and region of $(a, 0)$.

		0
		2
1		

→

	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9		1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9		2,1	2,2	2,3	2,4	2,5	2,6	2,7	0,0	2,9
	1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9		2,1	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9		0,1	0,2	0,3	0,4	0,5	0,6	2,0	0,8	0,9
	1,0	2,2	2,3	2,4	2,5	2,6	2,7	2,8	2,9		0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9		1,1	1,2	1,3	1,4	1,5	1,6	1,7	1,8	1,9
	0,0	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,2		1,0	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,2		2,0	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,2
	2,1	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,2		2,0	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,2		0,0	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,2
	2,0	0,3	0,4	0,5	0,6	0,7	0,8	0,9	0,2		0,0	1,3	1,4	1,5	1,6	1,7	1,8	1,9	1,2		1,0	2,3	2,4	2,5	2,6	2,7	2,8	2,9	2,2
	2,3	0,0	2,5	2,6	2,7	2,8	2,9	2,1	2,4		0,3	1,0	0,5	0,6	0,7	0,8	0,9	0,1	0,4		1,3	2,0	1,5	1,6	1,7	1,8	1,9	1,1	1,4
	0,3	1,0	0,5	0,6	0,7	0,8	0,9	0,1	0,4		1,3	2,0	1,5	1,6	1,7	1,8	1,9	1,1	1,4		2,3	0,0	2,5	2,6	2,7	2,8	2,9	2,1	2,4
	1,3	2,0	1,5	1,6	1,7	1,8	1,9	1,1	1,4		2,3	0,0	2,5	2,6	2,7	2,8	2,9	2,1	2,4		0,3	1,0	0,5	0,6	0,7	0,8	0,9	0,1	0,4
	0,4	0,5	0,0	0,7	0,8	0,9	0,1	0,2	0,6		1,4	1,5	1,0	1,7	1,8	1,9	1,1	1,2	1,6		2,4	2,5	2,0	2,7	2,8	2,9	2,1	2,2	2,6
	1,4	1,5	1,0	1,7	1,8	1,9	1,1	1,2	1,6		2,4	2,5	2,0	2,7	2,8	2,9	2,1	2,2	2,6		0,4	0,5	0,0	0,7	0,8	0,9	0,1	0,2	0,6
	2,4	2,5	2,0	2,7	2,8	2,9	2,1	2,2	2,6		0,4	0,5	0,0	0,7	0,8	0,9	0,1	0,2	0,6		1,4	1,5	1,0	1,7	1,8	1,9	1,1	1,2	1,6
	1,5	1,6	1,7	0,0	1,9	1,1	1,2	1,3	1,8		2,5	2,6	2,7	1,0	2,9	2,1	2,2	2,3	2,8		0,5	0,6	0,7	2,0	0,9	0,1	0,2	0,3	0,8
	2,5	2,6	2,7	1,0	2,9	2,1	2,2	2,3	2,8		0,5	0,6	0,7	2,0	0,9	0,1	0,2	0,3	0,8		1,5	1,6	1,7	0,0	1,9	1,1	1,2	1,3	1,8
	0,5	0,6	0,7	2,0	0,9	0,1	0,2	0,3	0,8		1,5	1,6	1,7	0,0	1,9	1,1	1,2	1,3	1,8		2,5	2,6	2,7	1,0	2,9	2,1	2,2	2,3	2,8
	2,6	2,7	2,8	2,9	0,0	2,2	2,3	2,4	2,1		0,6	0,7	0,8	0,9	1,0	0,2	0,3	0,4	0,1		1,6	1,7	1,8	1,9	2,0	1,2	1,3	1,4	1,1
	0,6	0,7	0,8	0,9	1,0	0,2	0,3	0,4	0,1		1,6	1,7	1,8	1,9	2,0	1,2	1,3	1,4	1,1		2,6	2,7	2,8	2,9	0,0	2,2	2,3	2,4	2,1
	1,6	1,7	1,8	1,9	2,0	1,2	1,3	1,4	1,1		2,6	2,7	2,8	2,9	0,0	2,2	2,3	2,4	2,1		0,6	0,7	0,8	0,9	1,0	0,2	0,3	0,4	0,1
	0,7	0,8	0,9	0,1	0,2	0,0	0,4	0,5	0,3		1,7	1,8	1,9	1,1	1,2	1,0	1,4	1,5	1,3		2,7	2,8	2,9	2,1	2,2	2,0	2,4	2,5	2,3
	1,7	1,8	1,9	1,1	1,2	1,0	1,4	1,5	1,3		2,7	2,8	2,9	2,1	2,2	2,0	2,4	2,5	2,3		0,7	0,8	0,9	0,1	0,2	0,0	0,4	0,5	0,3
	2,7	2,8	2,9	2,1	2,2	2,0	2,4	2,5	2,3		0,7	0,8	0,9	0,1	0,2	0,0	0,4	0,5	0,3		1,7	1,8	1,9	1,1	1,2	1,0	1,4	1,5	1,3
	1,8	1,9	1,1	1,2	1,3	1,4	0,0	1,6	1,5		2,8	2,9	2,1	2,2	2,3	2,4	1,0	2,6	2,5		0,8	0,9	0,1	0,2	0,3	0,4	0,7	0,6	0,5
	2,8	2,9	2,1	2,2	2,3	2,4	1,0	2,6	2,5		0,8	0,9	0,1	0,2	0,3	0,4	2,0	0,6	0,5		1,8	1,9	1,1	1,2	1,3	1,4	0,0	1,6	1,5
	0,8	0,9	0,1	0,2	0,3	0,4	2,0	0,6	0,5		1,8	1,9	1,1	1,2	1,3	1,4	0,0	1,6	1,5		2,8	2,9	2,1	2,2	2,3	2,4	1,0	2,6	2,5
	2,9	2,1	2,2	2,3	2,4	2,5	2,6	0,0	2,7		0,9	0,1	0,2	0,3	0,4	0,5	0,6	1,0	0,7		1,9	1,1	1,2	1,3	1,4	1,5	1,6	2,0	1,7
	0,9	0,1	0,2	0,3	0,4	0,5	0,6	1,0	0,7		1,9	1,1	1,2	1,3	1,4	1,5	1,6	2,0	1,7		2,9	2,1	2,2	2,3	2,4	2,5	2,6	2,8	2,7
	1,9	1,1	1,2	1,3	1,4	1,5	1,6	2,0	1,7		2,9	2,1	2,2	2,3	2,4	2,5	2,6	0,0	2,7		0,9	0,1	0,2	0,3	0,4	0,5	0,6	1,0	0,7
	0,2	0,4	0,6	0,8	0,1	0,3	0,5	0,7	0,0		1,2	1,4	1,6	1,8	1,1	1,3	1,5	1,7	1,0		2,2	2,4	2,6	2,8	2,1	2,3	2,5	2,7	2,0
	1,2	1,4	1,6	1,8	1,1	1,3	1,5	1,7	1,0		2,2	2,4	2,6	2,8	2,1	2,3	2,5	2,7	2,0		0,2	0,4	0,6	0,8	0,1	0,3	0,5	0,7	0,0
	2,2	2,4	2,6	2,8	2,1	2,3	2,5	2,7	2,0		0,2	0,4	0,6	0,8	0,1	0,3	0,5	0,7	0,0		1,2	1,4	1,6	1,8	1,1	1,3	1,5	1,7	1,0

Figure 5.5: Second step in the reduction.

Typical case complexity

In this section, we experimentally analyze the complexity behavior of GSWH problems depending on the employed method for punching holes. We first compare the complexity patterns among singly balanced GSWH problems for distinct factor forms on their constituent blocks, looking as well to the performance of different solving algorithms at the hardest zone of the complexity pattern. Finally, we show how this complexity is raised when more balanced puncturing methods for generating holes (doubly and fully) are used.

Singly Balanced

We consider first the complexity of solving GSWH instances generated with the Singly Balanced method. Our first set of results shows the complexity of Solving GSWH instances with different block factor forms, comparing it with the complexity of solving QWH instances. Figure 5.6 shows the results for GSWH with blocks 15×2, 10×3 and 6×5 (size 30) and QWH also of size 30 (the encoding used for QWH is 3D). We employ 100 instances per point and MiniSAT solver with 5,000 seconds cutoff. Complexity patterns are quite similar, being worth to note that the closer to a square is the block region shape, the greater is its peak complexity. So, for the same size, the easiest instances are from QWH and the hardest ones those from GSWH with almost square blocks[2]. Observe that the difference between QWH and GSWH with square blocks is about three orders of magnitude for this size. Our conjecture is the following; for a GSWH instance with blocks $n \times l$ and l fixed, when a cell is assigned, the larger n, the more cells in the same block become constrained in those blocks that intersect with the row of the fixed cell. So, when fixing a cell, for a large n some blocks will become more constrained than others, and this may be an advantage for simplifying the problem where look-ahead heuristics can take advantage of. By contrast, for square blocks, fixing a cell constrains the same number of cells (if the current hole pattern is balanced) in all the blocks that intersect with the row or the column of the fixed cell. So, again it seems that balance is making a difference in the complexity of this problem.

We observe the same qualitative behavior when using different SAT algorithms. The main difference is the magnitude of the peak of the complexity curve. Figure 5.7 shows a plot with the performance of different algorithms in the critically constrained area for different GSWH problems. The plot shows, for different sizes and different algorithms, the percentage of solved instances from a test-set of 200 instances, when using a cutoff of 10^4 seconds. For small sizes, all algorithms solve almost all the instances. But as the size increases, the solver MiniSAT clearly outperforms the other solvers.

[2] Since 30 is not a perfect square, we cannot have perfectly square blocks.

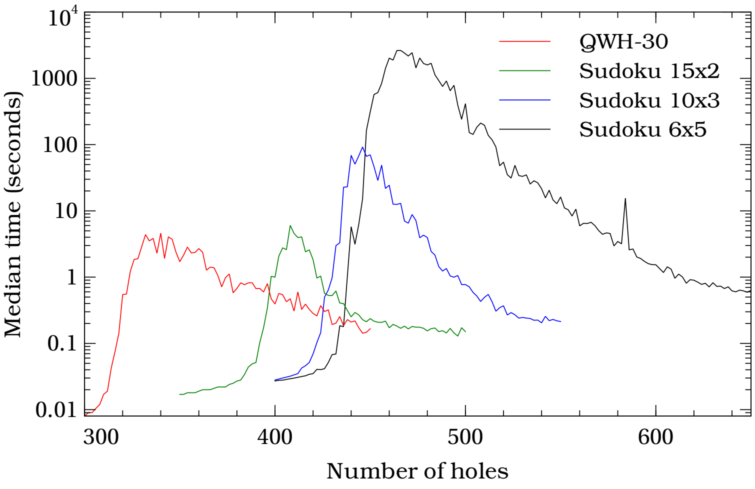


Figure 5.6: Empirical complexity patterns for singly balanced GSWH instances with different block regions form factor and same size

Doubly and Fully Balanced Next, we consider the doubly and fully balanced method for punching holes. When using doubly balanced, the typical hardness of the GSWH instances seems to be very similar to the singly balanced method for small sizes, however, as we increase the size of the instances, we see a dramatically difference in computational hardness. This is probably due to the fact that the singly balanced method tends

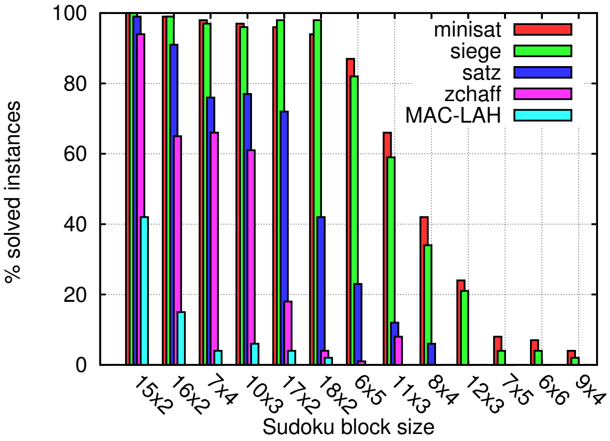


Figure 5.7: Empirical complexity patterns for singly balanced GSWH instances with different block shapes. 10^4 seconds time out

Table 5.1: Comparison of percentage of solved GSWH instances generated with three methods (singly, doubly, and fully balanced) for putting holes, for instances at the peak of hardness. 500 instances per row with a 5,000 seconds time out

			Satz			**Minisat**	
block	holes	singly	doubly	fully	singly	doubly	fully
5×5	344	98.8	98.8	77.0	100.0	100.0	95.6
7×4	414	71.6	67.4	n/a	98.8	97.0	n/a
17×2	504	48.6	37.8	n/a	85.2	76.0	n/a
18×2	572	33.8	24.9	n/a	91.6	86.0	n/a
6×5	480	18.6	11.6	n/a	81.4	71.4	n/a
8×4	518	2.6	1.8	n/a	37.4	31.6	n/a
6×6	686	0.0	0.0	0.0	3.2	1.6	0.0

Table 5.2: Median time (in seconds) for GSWH problems from Table 5.1 where the percentage of solved instances is greater than 50%

			Satz			**Minisat**	
block	holes	singly	doubly	fully	singly	doubly	fully
5×5	344	13	12	733	3	3	129
7×4	414	1,236	1,480	n/a	55	65	n/a
17×2	504	–	–	n/a	220	497	n/a
18×2	572	–	–	n/a	103	213	n/a
6×5	480	–	–	n/a	663	1,481	n/a

to distribute the holes uniformly between blocks in such a way that the difference with respect to the doubly balanced method is not significant for small instances. This can be quantified by looking at the percentage of solved instances from a test-set with 500 instances, for both methods, when working with a cutoff of 5,000 seconds. Table 5.1 shows these values. Solved ratios are almost the same for 5×5 Sudokus, but as the order is increases, also does the relative difference between doubly and singly balance methods in terms of ratio of solved instances as well as in terms of time to solve them as reflected on Table 5.2. Our doubly balanced method, then, gives harder instances than those produced by balanced QWH, guaranteeing satisfiability as well, and therefore constituting a good benchmark for the evaluation of local and systematic search methods.

Besides, when applicable (squared blocks), fully balanced method generates even harder instances. Due to the hardness of the problems generated by this method, we are able to compute results only for small GSWH of size 5×5, but even in this sort of problems, fully balanced method is able to pro-

Table 5.3: Comparison of solved GSWH instances with Minion [GJM06a] and MAC [BR96] solvers (dual encoding used)

		MAC		**Minion**	
block	holes	% solved	median	% solved	median
4×4	148	100	0.19	100	7.98
5×4	222	100	1.7	11	> 10.000
5×5	346	42	> 10.000	0	> 10.000

duce instances several orders of magnitude harder (in time) than singly and doubly balanced. In order to remark such differences, Figure 5.8 compares the hardness of the three balanced methods for GSWH sizes of 5×5 and 6×6, along a broad range of holes, above and below the peak of hardness.

5.2 Balancing Random BCSP

In this section we propose some methods to increase the hardness of typical instances of random binary CSPs, by modifying the methods for creating the constraints and the constraint graph, by equalizing some of the values that characterize a CSP problem. We will consider as our reference model the random binary CSP model B $\langle V, d, C, t \rangle$, already introduced (see § 3.3.1 in page 23).

Although previous work has identified some reasons for the hardness of the typical instances on the phase transition [HHW96, SD96, SK96, Pro96] or for the occurrence of *ehps* (exceptionally hard problems) on the under-constrained region [HW94, GW94, SG95, SG97, GFSB04], we will deal with hardening only typical instances on the phase transition, so we do not study *ehps* in our CSP models.

As for the the constraint language, we will consider the restriction of balancing the constraints, that is, balancing the number of occurrences of every domain value in the tuples in each side of the constraint.

We will, additionally, also consider the restriction of symmetry, that can be imposed simultaneously to the balance condition. An example of balanced symmetric constraint is the $k-$alldiff binary constraint associated with a graph $k-$coloring problem, that is NP-complete only when $k > 2$. So, not all balanced or symmetric balanced constraint languages will be hard, but empirical results show that they seem to be harder than general random constraint languages.

We will consider two models to harden constraint graphs. The first one, in which vertices have at most two different degrees, tries to minimize the probability that any vertex is more relevant than any other when finding a solution. The goal is to makes harder for heuristics to choose graph vertices

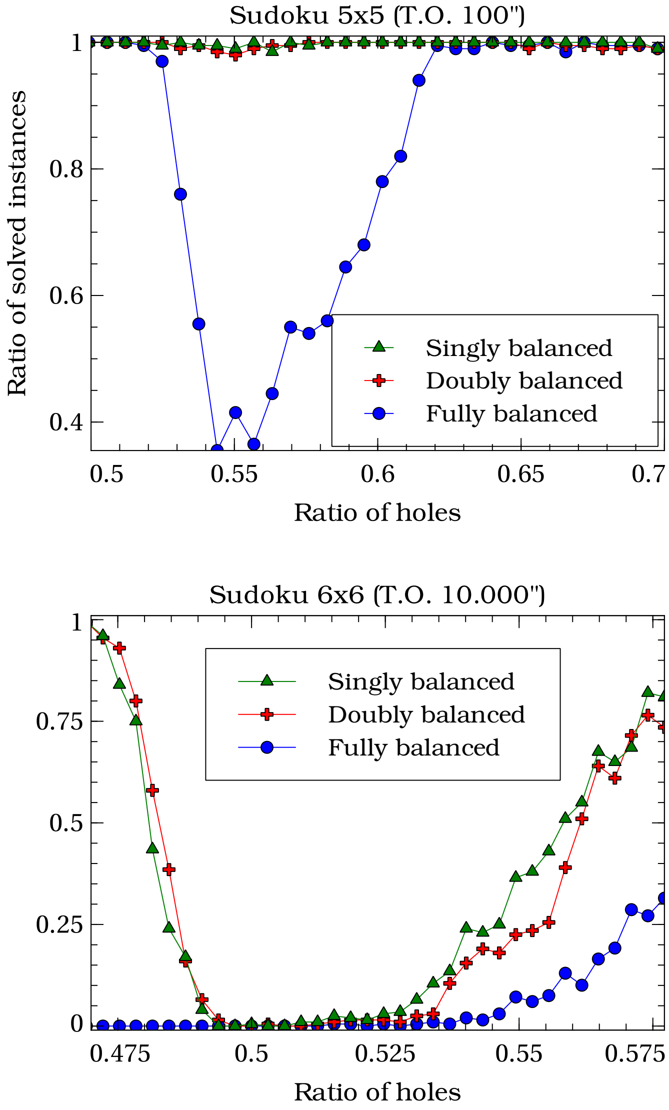


Figure 5.8: Comparison of the hardness of instances generated using single, doubly balanced and fully balanced methods of punching holes. Plot shows the rate of solved instances (using Minisat over 200 generated instances) for a specified time out in seconds as a function of the number of holes

as there are no much differences between them. The second model, where vertices have at most three different degrees, generates graphs with a high girth value, a structural property linked with the graph treewidth. The treewidth has been found to be the most general structural parameter that defines the boundary between polynomial-time and NP-hard restrictions for binary CSPs with arbitrary constraint languages [Gro03].

We will focus on CSPs with sparse, but connected, constraint graphs. With sparse graphs it is more difficult to obtain high treewidth graphs.

5.2.1 Balancing the constraint language

Balanced constraints

The first way to increase the hardness is to balance the number of occurrences of every domain value on each side of the constraint. A constraint $R_{i,j}$ with the same number of occurrences of every domain value on each side of the constraint can be seen as a regular bipartite graph $(V_i \cup V_j, E)$, where part V_i of the graph is associated with side i of the constraint, having an edge $\{a, b\}$ in E, with $a \in V_i$ and $b \in V_j$, iff $(a, b) \in R_{i,j}$. That is, the set of allowed tuples (a, b) of the constraint is determined by the set of edges of the regular bipartite graph.

We can see a partial explanation of why balanced constraints should provide harder problems looking at the concepts of m-looseness and m-tightness. The more balanced a constraint is, the higher will be its m-looseness and m-tightness. Observe that the m-looseness is always smaller or equal than its m-tightness, and that for a binary constraint they can be equal precisely when the constraint is perfectly balanced.

In order to sample from the set of regular bipartite graphs we use a Markov chain algorithm that walks between regular bipartite graphs [KTV97]. Observe that a perfectly balanced constraint with t tuples must satisfy that t/d is an integer, otherwise we can have an almost balanced constraint. That is, a constraint where $d-(t\ mod\ d)$ domain values appear $\lfloor t/d \rfloor$ times in each side of the constraint and $(t\ mod\ d)$ domain values appear $\lfloor t/d \rfloor + 1$ times in each side of the constraint. This more general constraint is associated with a bi-regular bipartite graph (vertices can have either degree $\lfloor t/d \rfloor$ or $\lfloor t/d \rfloor + 1$), that can be generated with the Markov chain algorithm as well.

Symmetric balanced constraints

Here we consider an additional restriction over the constraints. In addition to being balanced, we also impose the constraints to be symmetric ($(a, b) \in R_{i,j} \Leftrightarrow (b, a) \in R_{i,j}$) and not to include any tuple of the form (a, a), for any value a of the domain. In this case, a constraint can be associated with a regular (undirected) graph (V, E), where the vertices are the domain values, and $\{a,b\} \in E \Leftrightarrow (a,b) \in R_{i,j} \wedge (b,a) \in R_{i,j}$. It is interesting to consider this additional restriction because it was shown in [HN90] that a constraint language having only one symmetric constraint (but not necessarily balanced) is NP-hard only when the associated graph is non-bipartite and does not contain loops[3]. Because asymptotically a random graph will not be bi-partite, it is interesting to check empirically whether the additional restriction of symmetric language creates typical instances of equivalent hardness to the previous model, or the frequency of hard instances decreases.

Analogously to the previous model, we need to consider bi-regular graphs, because depending on the number of tuples t not all values can appear the same number of times [4]. To generate the bi-regular graph associated with a symmetric balanced constraint, we use a generalization of the algorithm presented in [SW99]. In principle, that algorithm is presented only for regular graphs, but it can be easily generalized for the case of graphs with vertices with two different degrees k and $k+1$. Since the two degrees are contiguous the performance of the generalization of the algorithm seems to be almost identical to the regular case, although in this case we cannot guarantee a perfect uniform distribution of graphs.

An interesting question related to balanced, symmetric or not symmetric, constraints is which additional conditions ensure the CSP is NP-complete. For example, the symmetric balanced $k-$alldiff constraint associated with the binary CSP encoding of a graph $k-$coloring problem, is an NP-hard case, but only if $k > 2$.

5.2.2 Balancing the constraint graph

Pure random graphs

As far as we know, in typical random distributions of CSPs, the constraint graph has been built following either the $G(V,p)$ or the $G(V,M)$ random graph model. We consider here the model $G(V,M)$, because we are interested in comparing the hardness of this model with others in which also the number of edges is an exact input parameter, and not an expected value as in the $G(V,p)$ model[5].

[3] Observe that using a bi-partite graph for building a symmetric constraint is not equivalent to what we do in the previous model, because there the bipartite graph is used in a different way.

[4] Also, if t is odd there will be a tuple $(a,b) \in R_{i,j}$ s.t. $(b,a) \notin R_{i,j}$ ($R_{i,j}$ will be almost symmetric).

[5] For $G(V,p)$ the expected number of edges is $pV(V-1)/2$ but in $G(V,M)$ the number of edges is always M.

Random regular graphs

As the first model for a harder constraint graph we propose regular graphs, i.e. graphs where all the vertices have the same degree. The most obvious reason why these graphs can be harder is that since all the degrees are equal, degree-based search heuristics will be less effective. A less trivial reason can be the following. A graph $G = (V, E)$ is perfectly balanced iff for any subgraph (V', E') of G:

$$\frac{|E|}{|V|} \geq \frac{|E'|}{|V'|}$$

That is, the average degree (or edge density), of any subgraph is never greater than in the whole graph G. We consider a graph more balanced than other if the number of its subgraphs that satisfy that relation is higher. For graphs where some vertices have a higher degree than others, as in $G(V, p)$ or $G(V, M)$, one may find subgraphs with bigger edge density than the whole graph. However, *any* connected regular graph is perfectly balanced. Balanced graphs will tend to be harder for heuristics, because it will be less easy to isolate potentially overconstrained subproblems during the search, as the edge density of any subproblem will be never higher. Of course, the fact of finding potentially overconstrained subproblems will also depend on the particular constraints between the variables of the subproblem. This is why we have also considered balancing the constraints. Because we want to consider graphs with any desired number of edges, we use the same more general model of bi-regular graphs (and regular graphs when $E * 2/V$ is an integer) that we use for building a symmetric balanced constraint. Not every bi-regular graph will be perfectly balanced, but in general they will be much more balanced than sparse pure random graphs.

As we have mentioned in chapter 4, the treewidth of a graph is the most relevant structural parameter concerning the complexity of binary CSPs with arbitrary constraint languages [Gro03]. Concerning the treewidth of random regular graphs, we can use the following lower bound, that is also valid for general graphs, based on the spectrum of the (combinatorial) Laplacian of the graph [CS03]:

$$tw(G) \geq \left\lfloor \frac{3V}{4} \left(\frac{\mu}{\Delta + 2\mu} \right) \right\rfloor - 1$$

Where Δ is the maximum degree of G, the Laplacian of G is the matrix $D - A$, where D is the diagonal matrix in which $D_{i,i}$ is the degree of v_i, A is the adjacency matrix of G and μ is the second smallest eigenvalue of the Laplacian.

For the case of a $k-$regular graph, where $\mu = k - \lambda_2(G)$, this lower bound can be rewritten as:

$$tw(G) \geq \left\lfloor \frac{3V}{4} \left(\frac{k - \lambda_2(G)}{3k - 2\lambda_2(G)} \right) \right\rfloor - 1$$

Where $\lambda_2(G)$ is the second largest eigenvalue of the adjacency matrix of G (k is the largest eigenvalue for a $k-$regular graph). So, the higher the value of $\mu = k - \lambda_2(G)$, also called the spectral gap, the closer the lower bound gets to $V/4$. This shows a connection between expander graphs and graphs with a high value for this treewidth lower bound, because this spectral gap also gives a lower bound on the expansion of a graph.

For random $k-$regular graphs, with k fixed, it is known that asymptotically (as $V \to \infty$) for $\epsilon > 0$:

$$Probability(\lambda_2(G) \leq 2\sqrt{k-1} + \epsilon) \to 1$$

where $2\sqrt{k-1}$ is, asymptotically, the lowest possible value for $\lambda_2(G)$ [Fri04]. Regular graphs with $\lambda_2(G) \leq 2\sqrt{k-1}$ are called Ramanujan graphs. For Ramanujan graphs, we have:

$$tw(G) \geq \left\lfloor \frac{3V}{4} \left(\frac{k - 2\sqrt{k-1}}{3k - 4\sqrt{k-1}} \right) \right\rfloor - 1$$

So, for a fixed and sufficiently high k the lower bound will get very close to $V/4$. If we consider regular (or bi-regular) graphs with the degrees growing with V, we cannot infer directly that asymptotically our graphs will have a spectral gap of order $k - 2\sqrt{k-1}$. Actually, as the degree grows with V, the expanding properties of the graphs can be better for higher degrees. For example, some existing constructions of Ramanujan graphs based on certain Cayley graphs are obtained with a degree $\Omega(\sqrt{V})$, whereas similar Cayley graphs with degree $O(\log V)$ are also expander graphs but not as good as Ramanujan graphs [AR94]. As far as we know the best result for non-constant degree random $k-$regular graphs is, if $k > \sqrt{V \log V}$ then asymptotically $\lambda_2(G) = o(k)$ [KSVW01]. So those regular graphs have a high spectral gap, and a high treewidth spectral lower bound.

One important corollary of the treewidth spectral lower bound of [CS03] is that, given that it is derived trough a lower bound on graph expansion, we have also the following lower bound based directly on any expansion lower bound we have for the graph, and not necessarily trough spectral lower bounds:

Corollary 5.7. *If the size of the subset of outside neighbors, for any subset X with $0.25|V| \leq |X| \leq 0.5|V|$, is at least $N(X)$, then we have that*

$$tw(G) \geq \lfloor N(X) \rfloor - 1$$

This is important, due to the existence of some graph constructions that achieve high expansion but not using spectral bounds to show it. See [SHW96] for a survey about expander graphs constructions. However, for general graphs, checking whether a graph is an expander is co-NP complete [BKV$^+$81], and spectral methods are still the best we have to bound the expansion of graphs. For example, Kahale [Kah95] shows that the best we can prove for Ramanujan k-regular graphs, using the second eigenvalue technique, is to have an expansion as high as $k/2$. No better expansion lower bounds are known for Ramanujan k-regular graphs.

For the random graph model $G(V,p)$, recent results [FV04, CO05] indicate that only when $pV >> \log^2 V$, the expected spectral gap tends to be high (although the results do not clarify if as high as with regular graphs). For $pV = O(\log V)$ (when we have the sparsest but still connected random graphs) the results only indicate that the graph will contain a large *subgraph* with large spectral gap. Moreover, most of the existing constructions of good expander graphs (or even Ramanujan graphs) consist of regular graphs of constant degree (See chapter 6 of [Chu97] for some examples). By contrast, the results of [CO05] indicate that random graphs $G(V,p)$ with expected constant degree have a spectral gap of $o(1)$, so they are neither expander graphs nor have a high treewidth spectral lower bound. Therefore, theoretical results show a clear difference between random graphs and random regular graphs when considering graphs with expected constant degree, but such a difference is less obvious as we increase the expected degree.

High girth graphs

As we have mentioned, lower bounds on expansion imply lower bounds on treewidth, so any method that assures high expansion will also assure high treewidth. Previous results indicate that an indirect way of achieving high expansion is trough achieving high girth [Kah93, Kah95]. In [CS05] it is shown that if the girth of a graph G is at least g and the average degree at least d, then we have:

$$tw(G) = \Omega\left(\frac{1}{g+1}(d-1)^{\lfloor (g-1)/2 \rfloor}\right)$$

For random $k-$regular graphs it is known that the average girth is only slightly greater than 3 [MWW04], so that means that forcing a high girth is not necessarily equivalent to generating a regular graph. Actually, the high girth graphs that we present here are not necessarily regular, but almost regular.

In [Cha03], the authors present an algorithm that, for given V and k (with $k < V/3$), it builds a graph with girth g that satisfies:

$$g \geq \log_k(V) + O(1)$$

Where the vertices of the graph have one of these possible degrees: $k-1$, k or $k+1$. So, the graph is almost regular but may be less balanced than biregular graphs. With such a bound on the girth, we have that these graphs satisfy:

$$tw(G) \geq \Omega\left(\frac{(k-1)^{\lfloor(\log_k(V)/2\rfloor}}{\log_k(V)+1}\right) = \Omega\left(\frac{\sqrt{V}}{\log_k(V)+1}\right)$$

So, we can build graphs, for any value of V and $k < V/3$, and even with k growing with V, such that every graph will satisfy that lower bound. Actually, the graphs obtained with this algorithm could have an even higher treewidth[6], because there are graphs with low girth that, however, have a high treewidth, i.e. the complete graph K_V has girth 3 and treewidth $V-1$.

The algorithm for the generation of such graphs basically proceeds in a greedy fashion, starting with an initially empty graph and adding edges one by one, connecting vertices which are at large distances in the current graph. Here, we use their algorithm with a slight modification. The original algorithm works with k integer, giving a number of edges $\lfloor Vk/2 \rfloor$. So, we cannot use it directly to get graphs with any desired number of edges. However, one can generalize the algorithm to proceed by adding as many edges as desired, instead of stopping when $|E| = \lfloor Vk/2 \rfloor$. Observe that adding more edges will not reduce the treewidth although this can produce a reduction of the girth of the graph.

5.2.3 Experimental investigation

All the plots contained in this subsection are obtained using Forward-Checking with Conflict Back-Jumping (FCCBJ) and Maintaining Arc-Consistency (MAC) search algorithms with variable selection heuristic *dom/deg* [Pro93, BR96]. Note that other algorithms and heuristics have also been used, for different sizes and constrainedness, showing similar qualitative behavior. More specifically, the same set of problems have been solved with previous mentioned algorithms with *dom+deg* and *dom/deg* heuristics, as well as with Forward Checking [HE80] and a MAC solver using a Satz-like heuristic as described in [AdD+04].

Balanced constraints

Figure 5.9 shows the hardness of solving CSPs with either random constraints, balanced constraints or symmetric balanced constraints for sparse constraint graphs ($|edges| = (V/2)\log_2 V$) using MAC and FCCBJ. Differences between random and balanced constraints almost achieve an order of magnitude, meanwhile differences between balanced and symmetric balanced are irrelevant.

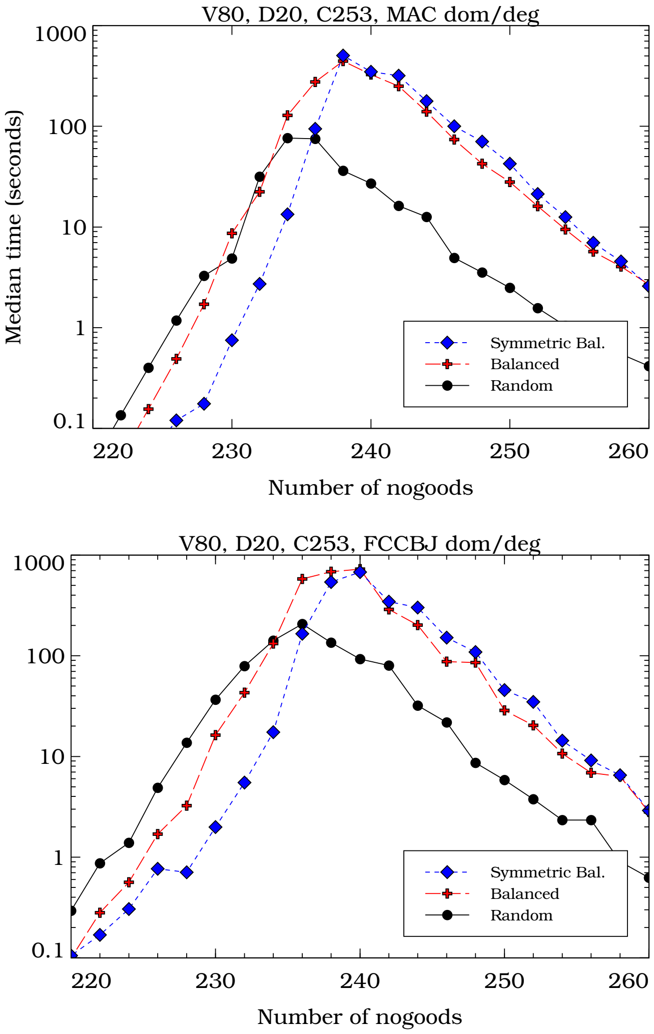


Figure 5.9: Random constraints versus balanced and symmetric balanced constraints. On top, V stands for number of variables, D for domain size and C for number of constraints.

Balanced graphs

Figure 5.10 shows the hardness of solving CSPs with either pure random, bi-regular or high girth constraint graphs for random constraints. Clearly, our graph models give typical instances of higher complexity than pure random graphs. Between pure random and bi-regular graphs the difference is of several orders of magnitude, and less than 1 order between bi-regular and high girth graphs.

[6] Our empirical results seem to confirm a higher treewidth.

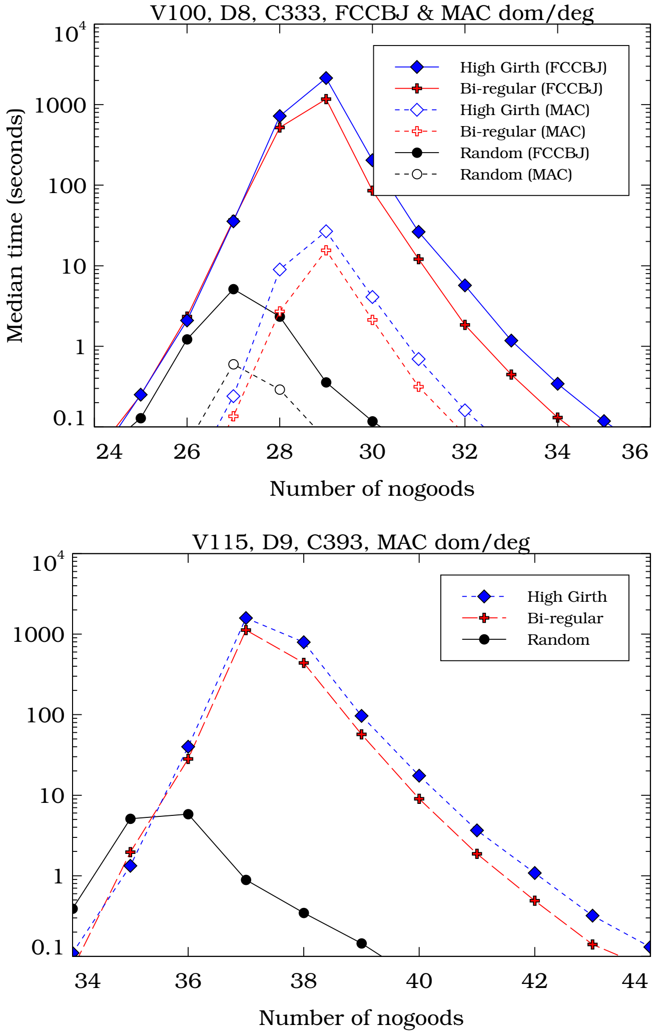


Figure 5.10: Random graphs versus bi-regular and high girth graphs.

Right plot on Figure 5.10 shows the hardness for a larger problem solved with the best performing algorithm (MAC). As noted, differences between pure random and our graph models increase with the problem size.

Bounds on the treewidth

Calculating the treewidth of a graph is NP-hard, and even to approximate it to within a constant absolute additive error [BGHK95], so we cannot get exact results within reasonable time limits. Table 5.4 shows two different lower bounds and one upper bound on the treewidth of typical instances from the three graph models considered in this paper. The values represent

	$G(V,M)$			bi-regular			high girth		
V	Heu LB	Spec LB	UB	Heu LB	Spec LB	UB	Heu LB	Spec LB	UB
100	15	3	36	15	13	44	15	14	46
200	22	6	80	22	28	96	23	30	98
300	28	14	127	28	43	153	29	45	154
400	33	12	175	33	61	206	34	63	209
500	38	22	224	38	79	263	39	81	265
600	42	28	274	42	92	323	43	94	324

Table 5.4: Comparison of lower and upper bounds on the treewidth for the different graphs, with $|edges| = (V/2)\log_2 V$

the median of 51 graphs obtained from each model. The first lower bound (Heu LB) is the heuristic lower bound *minimum maximum degree (mmd+)* with *least-c* neighbor selection strategy, as in [KvK02]. The second lower bound (Spec LB) is the spectral lower bound explained before. The upper bound (UB) shows the best heuristic upper bound obtained with either *Lexicographic Breadth First Search, variant Minimal (LEX_M)* used in [KBv01] or with the min-fill heuristic [Bod05][7].

We observe that as the size increases the spectral lower bound becomes much higher for our more balanced graph models than for pure random graphs. For pure random graphs the heuristic lower bound is the best, but never better than the spectral lower bound for our graph models. So, our graph models seem to have a high spectral gap, and thus a high treewidth lower bound. For the upper bound, we clearly observe that higher values are obtained for our models and that the difference increases with the size. To conclude, even if the gap between the lower and upper bounds is too big to infer exact results about the treewidth, they seem to indicate higher values of the treewidth for our graph models.

Combined effect

Figure 5.11 shows the hardness of solving pure random CSPs, CSPs with bi-regular constraint graphs and balanced constraints and CSPs with high girth constraint graphs and balanced constraints. This time, the differences between the models are even higher than before: more than 3 orders of magnitude for the best algorithm (MAC) and 4 orders for FCCBJ. The right plot shows that the differences for the best algorithm increase with a

[7] We thank Arie Koster for providing us with the implementations of the treewidth algorithms and the referees for suggesting to use the min-fill heuristic.

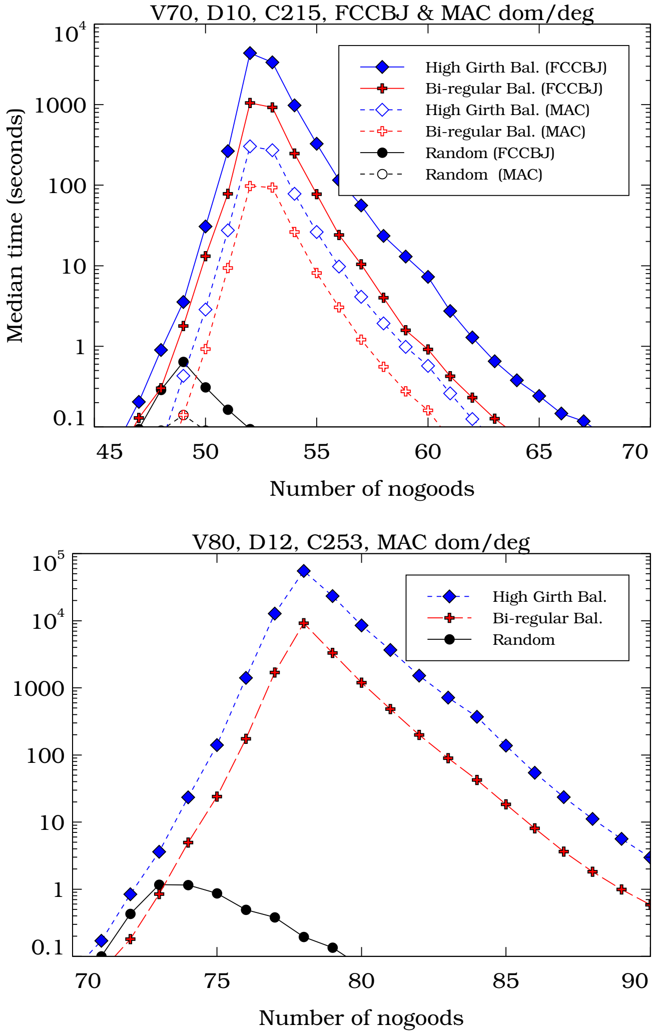


Figure 5.11: Random CSPs versus balanced CSPs

larger problem.

So, we observe that the increase in hardness is much more significant when we balance both; the constraint graph and the constraints, and that although the high girth graphs give the hardest instances, more dramatic differences arise between random and bi-regular graphs than between bi-regular and high girth graphs. Our high girth graphs seem to have the highest treewidth, although the bounds are very similar with bi-regular graphs. Also, probably they are not as balanced as bi-regular graphs, although it is not clear that having three contiguous degrees, instead of just two, makes a significant difference on the balance of the graphs. This may partially explain the lower difference between bi-regular and high girth graphs than between bi-regular and random graphs. Probably, high girth graphs as balanced as bi-regular graphs could give an even higher hardness.

5.3 Balancing k-SAT models

In this section we will study a new method to generate hard problems, in this case hard k-SAT and n-ary CSP instances. The proposed method, basically, consists in building a bipartite incidence graph representing the k-SAT or n-ary CSP instance. The left side of the graph represents clauses (or nogoods) and the right side represents the literals of our Boolean formula or the many-valued literals of the CSP. Finally, edges indicate which literals are present in each clause or nogood. Our algorithm will try to keep the girth of the incidence graph as high as possible by filling incrementally the edges of the graph while keeping the distance between pairs of non-connected vertices as high as possible. Generating a high girth graph we assure that the expansion of the graph will be also high.

This section relies heavily on basic graph theory definitions introduced in section 2.2 in page 12, specially those definitions concerning graph girth and expansion. To achieve our goal, i.e. generate hard k-SAT instances, we will resort to results from the field of propositional proof complexity. We rely on the fact that graph expansion has been established as a key to hard k-SAT formulas for resolution (e.g. [Ats04]), but also for other proof systems like the polynomial calculus [AR01]. We will make intensive use of expander graphs (definition 2.15, on §2.2). See section 4.1.2 for the relevant results concerning complexity of k-SAT related to the expansion of their incidence graph, and that are also the main motivation for our method for hardening n-ary CSPs.

5.3.1 Related Work

In this section we survey some previous theoretical results about the expansion of bipartite random graphs, and the related work in the SAT and CSP communities.

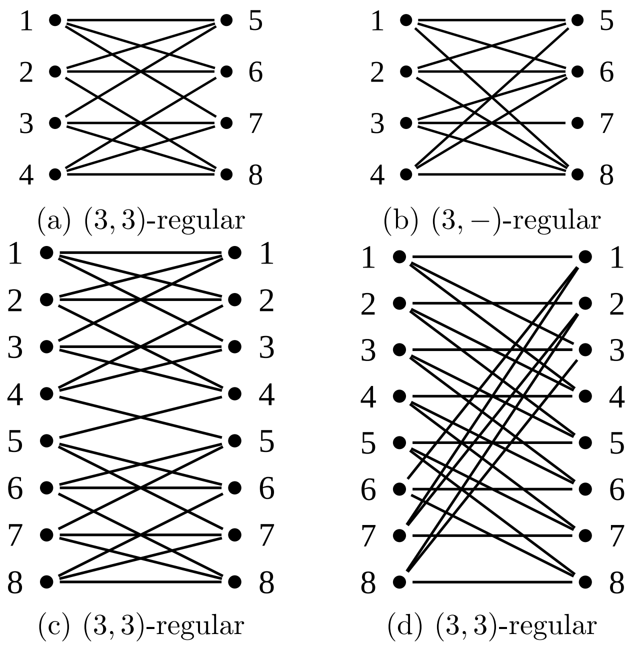
Expansion of bipartite random graphs

We have seen before that general (not necessarily bipartite) graphs that are either regular or high girth have high expansion. In this subsection we are interested in the particular case of bipartite graphs. The particular case of (k_1, k_2)-regular bipartite graphs have received special attention in the communications community (e.g. [Chu78, SS96]), and such bipartite graphs are good expanders almost always. So, it seems that regular bipartite graphs are promising towards obtaining good expanders, although we will see that *almost* regular graphs can also be excellent good expanders, even better than regular graphs. For the case of bipartite graphs we have, for example, that a random $(k, -)$-regular bipartite graph $(L \cup R, E)$ with $|L| = |R|$ will be a good expander *with probability* $> 1/2$. So, when only the vertexes of one part have the same degree, the expansion properties seem to degrade. Observe that this last graph can represent the incidence graph of a random k-SAT instance.

Related models of hard SAT formulas

As we have discussed, regular graphs tend to be better expanders than general graphs. So, it is not surprising that previous random models for k-SAT based on balancing the literal and variable occurrences generate harder instances, as their incidence graph will tend to have higher expansion. The model described in [BS96] for 3-SAT (lit-bal-1), generates instances with m clauses by putting $\lfloor \frac{3 \cdot m}{2n} \rfloor$ occurrences of each literal plus an additional random set of unique literals so that there are exactly $3 \cdot m$ literals in it. To construct each clause, 3 literals on distinct variables are removed from the bag. If there are less than 3 distinct variables mentioned in literals remaining in the bag, additional distinct variables are randomly selected from the set of all variables and negated with probability $\frac{1}{2}$. This model easily generalizes for $k > 3$. The model described in [BDIS05] for k-SAT (lit-bal-2) is very similar, being the main difference that every literal in the resulting formula appears exactly $\lfloor \frac{k \cdot m}{2n} \rfloor$ or $\lfloor \frac{k \cdot m}{2n} \rfloor + 1$ times. By contrast with lit-bal-1 the occurrences of literals can be less balanced.

Recently, models of hard satisfiable k-SAT instances, based on variants of the XORSAT model, have been introduced [RTWZ01, JMS05, Jär06, HJKN06]. The basic ingredient in all these models is that a system of linear equations (mod 2) with at least one solution is converted to an equivalent set of clauses, such that the solutions of the SAT formula correspond to the solutions of the linear system. All these models provide very challenging instances, being the hardest one regular k-XORSAT [Jär06, HJKN06], where the running time of DPLL algorithms seems to scale exponentially in the number of variables. It seems that the key for the hardness of regular k-XORSAT is the high expansion they get in the system of linear equations, thanks to the use of a regular bipartite graph for building it, such that the resulting system has n variables and n equations with k variables per equation, and every variable appears in k equations. We will see that by using our high-girth graph generation algorithm to generate the system of linear equations we increase the hardness of regular k-XORSAT instances even more.



(a) $(3,3)$-regular

(b) $(3,-)$-regular

(c) $(3,3)$-regular

(d) $(3,3)$-regular

Figure 5.12: Example of bipartite graphs with different expansion

5.3.2 Hard SAT and n-ary CSP instances

In this section we introduce our generation method for hard k-SAT and n-ary CSP instances, based on generating bipartite graphs with high girth, and so with high expansion.

Expansion, balance and girth

To get an idea about what is the typical structure of a good bipartite expander graph, consider the expansion of subsets of the left partition of the bipartite graphs (a) and (b) of Figure 5.12. As the vertexes in the left partition of both graphs have degree 3, the expansion when $|S| = 1$ is 3. Consider now sets with $|S| = 2$. In the graph (a), the set $N(S)$ for any left subset S with $|S| = 2$ is always the entire right partition, so the expansion is $4/2$. But for graph (b) the set $N(\{1,4\})$ does not contain the vertex 7, and so the expansion is only $3/2$ due to the poor connectivity of vertex 7. For $|S| = 3$ the situation is similar. For graph (a) any left subset with $|S| = 3$ is connected to the whole right partition (its expansion is $4/3$), but for graph (b) $N(\{2,1,4\}) = \{5,6,8\}$, so the expansion is 1. Thus, we observe that, due to the unbalanced degrees of the right partition of graph (b), the vertex expansion of the left subsets is not as good as in graph (a), where all the degrees are equal.

However, the balance of the degrees does not provide a complete characterization of good expander graphs. Consider the graphs (c) and (d) of Figure 5.12, that are both equally balanced. Graph (c) has several cycles of length 4, and thus girth 4, and its expansion for some left subsets of size 4 is 5/4. By contrast, in graph (d) the minimum expansion for left subsets of size 4 is 7/4. The main structural difference with the graph (c) is actually its girth, that in this case is 6. Actually, Kahale [Kah93] shows that a high girth ($O(\log_{k-1}(|V|))$) implies high expansion, at least for subsets of size at most $|V|^{\delta}$, with $\delta < 1$. So, one way to obtain graphs with good expansion is to get high girth graphs. In [Cha03] it is presented an algorithm for building graphs with degrees $k-1$, k and $k+1$ and high girth. The algorithm we present in the next subsubsection follows the same approach to build bipartite graphs with high girth. This graph will be used to build hard k-SAT and n-ary CSPs instances .

High girth bipartite graphs

The algorithm presented by Chandran in [Cha03] works for general (non-bipartite) graphs. It builds the graph by introducing edges one by one, connecting vertexes which are at large distances in the current graph, in such a way that the degrees are maintained almost balanced and the girth obtained is $O(\log_{k-1}(|V|))$. The algorithm initiates the construction by building a matching between the vertices, and then starts to insert edges between edges with maximum distance between them.

For building the literal incidence graph of a k-SAT formula with C clauses and L literals (and similarly for a k-ary CSP formula), we need to build a $(k,-)$-regular bipartite graph $(V_1 \cup V_2, E)$, where V_1 is C and V_2 is L. Algorithm 5.3 does this, but trying to keep the girth as high as possible, using the same technique of linking vertexes which are at large distances in the current graph. It starts the process by creating a random matching from V_1 to V_2, such that every vertex from V_1 will have degree 1 and every vertex from V_2 will have degree either $\lfloor |V_1|/|V_2| \rfloor$ or $\lfloor |V_1|/|V_2| \rfloor + 1$. Because this matching does not create any cycles, it starts with girth equal to ∞. Then, at every step it selects an edge from the subset of edges (u, v) with $u \in V_1$ and $v \in V_2$, such that $degree(u) < k$ and $degree(v)$ is minimum among all the current degrees in V_2. From this subset of edges, it selects one (u', v') with the maximum distance between u' and v', because this way the new created cycle is of maximum length. This process ends when the graph has $|V_1|k$ edges.

One of the keys of the algorithm of Chandran is that at any stage of the process maintains the degrees of the vertices almost balanced. Similarly, the algorithm in [GS06] creates a bipartite graph with balanced degrees and guaranteed high girth, but it only works for $|V_1| = |V_2|$. By contrast, for the purpose of using a high girth bipartite graph for generating SAT instances with any possible ratio $r = |C|/|V|$, we need a more general algorithm. That is, that works with not necessarily equal partition sizes. Observe that this implies that the degrees of the vertices of the right partition (literals) will be higher than on the left, the bigger the ratio r, the bigger the difference between the degrees on the left and right partitions.

Algorithm 5.3: Algorithm for generation of high girth $(k,-)$-regular bipartite graphs $(V_1 \cup V_2, E)$

```
input  : $V_1, V_2, k$
output: a bipartite $(k,-)$-regular graph $(V_1 \cup V_2, E)$
Initialize $E$ with a random matching from $V_1$ to $V_2$
(every vertex from $V_1$ will have degree 1)
for $i = |V_1| + 1$ to $k|V_1|$ do
    $L_T := \{u \in V_1 \mid degree(u) < k\}$
    $R_T := \{u \in V_2 \mid degree(u) \leq degree(v), \forall v \in V_2\}$
    $maxdist := 1$
    while $(maxdist = 1)$ do
        $T :=$
        $\{(u,v) \mid (u,v) \in L_T \times R_T \; dist(u,v) \geq dist(x,y) \forall (x,y) \in L_T \times R_T\}$
        $d_{min} := degree(u)$, where $u \in R_T$
        $maxdist := dist(u,v)$, where $(u,v) \in T$
        if $(maxdist = 1)$ then
            $R_T := \{u \in V_2 \mid degree(u) = d_{min} + 1\}$
    $E := E \cup (u,v)$, where $(u,v) \in T$
```

However, we can show that the degrees of the right vertexes (V_2) of our bipartite graph will be almost balanced.

Lemma 5.8. *For any fixed k and r and $|V_1| = r|V_2|$ this algorithm creates a $(k,-)$-regular bipartite graph $(V_1 \cup V_2, E)$, where the degrees of the vertices in V_2 will be asymptotically (as $|V_2| \to \infty$), from the set $\{d-1, d, d+1\}$, where $d = \lfloor rk \rfloor$.*

Proof. After inserting the initial matching from V_1 to V_2, the degree of any vertex from V_2 will be $\lfloor r \rfloor$ or $\lfloor r \rfloor + 1$. If the algorithm always succeds in selecting a minimum degree vertex from V_2, then at the end of the process any vertex will have degree $\lfloor rk \rfloor = d$ or $d+1$. We define a minimum degree vertex from V_2 that is linked with all the current available vertices from V_1 (vertices with less than k edges) as a blocked vertex.

Consider a blocked vertex v from V_2. The number of available vertices from V_1, but already linked to some vertex, when $k|V_1| - E$ edges are already inserted, will be, at least, $\lceil E/(k-1) \rceil$. This situation corresponds to the extremal case when all the available vertices are linked with only one vertex, and the rest of vertices from V_1 have degree k.

So, the first time when a blocked minimum degree vertex v from V_2 can appear is when this minimum number of available vertices can appear all linked with v. Thats is, when $\lceil E/(k-1)\rceil$ coincides with the current minimum degree from V_2:

$$\left\lceil \frac{E}{(k-1)} \right\rceil = \left\lfloor \frac{k|V_1| - E}{|V_2|} \right\rfloor$$

Then, this will never occur before E satisfies:

$$E = \frac{k(k-1)|V_1|}{|V_2| + (k-1)} < \frac{k(k-1)|V_1|}{|V_2|} = rk(k-1)$$

So, this is at the end of the process because $rk(k-1) = O(1)$. At this time, we have two posibilities. By one hand, if $R = kr|V_2|\ mod\ |V_2| > 0$ because r is fixed we have $R = O(|V_2|) >> O(1)$. Let i be the number of remaining minimum degree vertices from V_2 (number of minimum degree vertices from V_2 that should receive one more edge). Then, the degree of $R-i$ vertices will be $d+1$, and the degree of $|V_2| - (R-i)$ vertices will be d. Observe that a third degree $d+2$ will only be introduced if all the minimum degree vertices from V_2 are blocked. But the maximum number of blocked vertices is always $k-1$, so only if $|V_2| - R < k-1$ will be possible to block, at most, the last $k-1$ vertices from V_2. But $R = O(|V_2|)$, so $|V_2| - R = O(|V_2|) >> k-1$ and asymptotically no vertex of degree $d+2$ will appear.

By the other hand, if $R = 0$ we will have $|V_2| - i$ vertices with degree d and i vertices with degree $d-1$. In this case, only if $i < k$ we could have all the minimum degree vertices blocked, and this would lead to the introduction of at most $k-1$ vertices of degree $d+1$. ∎

So, the degrees of the vertices in V_2 will be almost balanced. Actually, the instances we have obtained with this method in our experiments almost always have only two distinct degrees in V_2, and only exceptionally three distinct degrees.

Regarding the girth, although we cannot ensure the same conditions that guarantee a logarithmic girth like in [Cha03] and [GS06], our empirical results show that this is the case. Table 5.5 shows the girth for graphs obtained with our algorithm, compared with the minimum girth we would obtain for a general $d-$regular graph, with d equal to the floor of the average degree of our bipartite graphs ($d = \lfloor (2|V_1|k)/(|V_1| + |V_2|) \rfloor$), if we used the algorithm of Chandran.

The table does not show the results for $|V| = 1000$ for 4-SAT and 5-SAT because the size of the corresponding literal incidence graph is too big in such cases for our generation algorithm to work in reasonable time, even if we are using the best performing polynomial-time algorithm we have found in [DI06], for the dynamic all-pairs shortest distance problem (DAPSD). For our bipartite graph $(V_1 \cup V_2, E)$ with $|V_1| = |C|$, $|V_2| = 2|V|$ and $n = |V_1| + |V_2|$, the worst-case running time of our algorithm is $O((k-1)^2 n^3)$, thanks to using the variant of the algorithm of [RR96] described in [DFMSN00].

Table 5.5: Girth of bipartite graphs created by our algorithm, corresponding to 3-SAT, 4-SAT and 5-SAT literal incidence graphs for instances at the peak of hardness

	3-SAT		
$\|V\|$	$\|C\|$	g	$\log_d(2\|V\| + \|C\|)$
62	221	8	5.6
125	447	8	6.2
250	895	10	6.9
500	1790	10	7.6
1000	3560	10	8.3

	4-SAT		
$\|V\|$	$\|C\|$	g	$\log_d(2\|V\| + \|C\|)$
62	539	6	3.8
125	1087	6	4.3
250	2175	6	4.6
500	4350	6	5.1

	5-SAT		
$\|V\|$	$\|C\|$	g	$\log_d(2\|V\| + \|C\|)$
62	1091	4	3.4
125	2467	4	3.8
250	4935	6	4.1
500	9870	6	4.5

Theoretically, there is a best worst-case algorithm for DAPSD [DI04], but empirically for our particular graphs the chosen algorithm was the best performing.

By contrast, we also computed the girth for the literal incidence graphs obtained with the balanced SAT model Lit-bal-1, and the girth obtained in all the instances was always 4, the minimum possible girth for a bipartite graph. Remember that actually there are theoretical results that imply that for random $k-$regular graphs the average girth is slightly greater than 3 [MWW04].

We can easily use this algorithm to generate the literal incidence graphs of k-SAT and n-ary CSPs, with the goal of obtaining harder instances than the ones we obtain with regular (balanced) graph models. Moreover, we can also use it to generate the systems of linear equations for the XORSAT instances, instead of using a random regular bipartite graph like it is done in [Jär06], with the left partition representing the equations, and the right partition the variables of the equations.

5.3.3 Experimental investigation

We have divided our experimental investigation into three parts. The first one presents a comparison of our method against the most recent k-SAT generators and the classical random k-SAT generator. The second one shows a comparison between model E and our method high-girth for n-ary CSPs.

Hard k-SAT instances

For generating the k-SAT instances we have used four methods: the classical random k-SAT (Random), the generalization of the method described in [BS96] (Lit-bal-1) for k-SAT, the method described in [BDIS05] (Lit-bal-2), and our method (High-Girth). We have solved the instances with four SAT solvers: satz [LA97], minisat [ES03], kcnfs [DD01], walksat [SKC94] and adaptg2wsat [LWZ07].

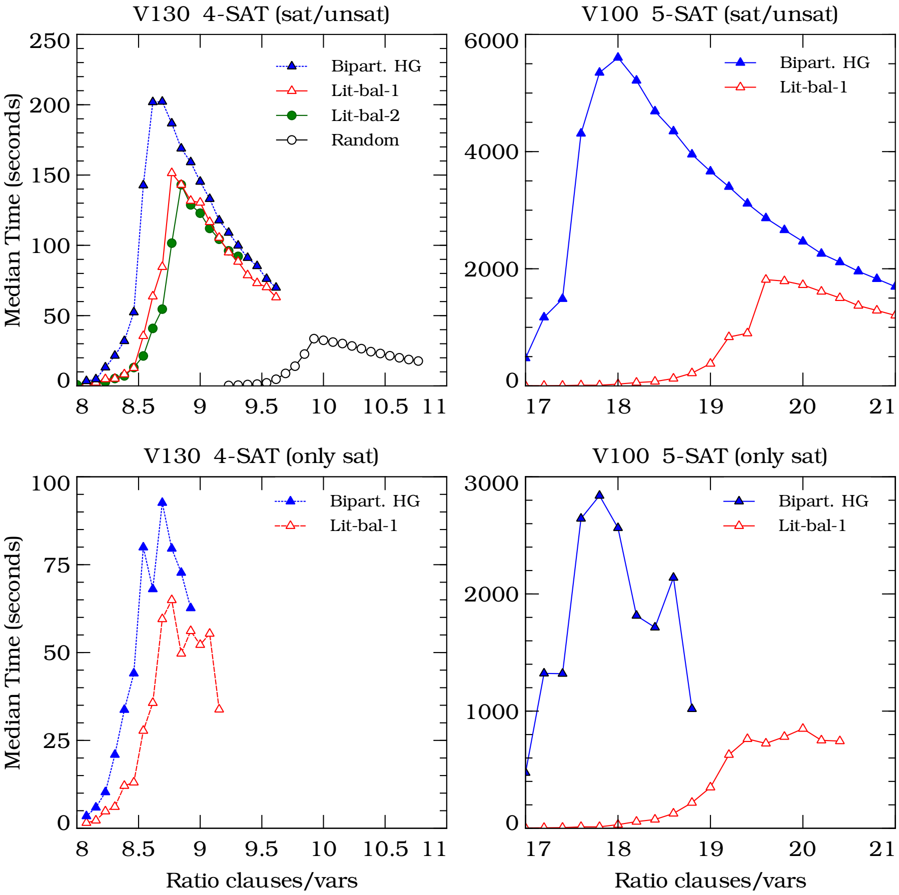


Figure 5.13: Comparison of SAT generators.

Table 5.6: Ratio of median time to solve all/only_sat instances on peak hardness between High Girth Bipartite, Literal and Random generation methods

3-SAT

Num. vars.	300	330
HG/Lit	1.29/1.02	1.44/1.19
Lit/Ran	80.2/126	132/162

4-SAT

Num. vars.	130	150
HG/Lit	1.34/1.42	1.39/1.68
Lit/Ran	4.58/7.41	5.59/10.47

5-SAT

Num. vars.	70	100
HG/Lit	1.64/1.48	3.09/3.34
Lit/Ran	1.73/2.65	2.09/2.48

Figure 5.13 shows the results for the complete SAT solver kcnfs on 4-SAT instances. As we can see High-Girth is the best generator, while Lit-bal-1 and Lit-bal-2 are almost identical. We only report the results for the SAT solver kcnfs since it was the fastest and it reported the least difference between the two best generators. Table 5.6 shows the ratios for the median time between High-Girth and Lit-bal-1 (HG/Lit) and between Lit-bal-1 and Random (Lit/Ran) for different arities when all and only satisfiable instances are considered. We observe that the higher arity (k), the higher the ratio HG/Lit results, particularly for larger number of variables. This can be due to the differences in the expansion of the bipartite graphs of the different models, because as we increase k, it is possible to obtain more drastic differences in the expansion of the bipartite graphs of the different models. That is, the higher k, the higher the maximum expansion of a subset of clauses S. At the same time, the ratio Lit/Ran seems to decrease, but this could be due to the fact that as we increase k, more variables may be needed in order to observe a difference for such ratio.

We also wanted to check if we could observe the same behavior when using a local search SAT solver. Table 5.7 reports results for 4-SAT and 5-SAT. On 4-SAT we run the solver walksat, on the most 50 difficult satisfiable instances at the underconstrained and the phase transition zones. We report the median time and the best noise parameter setting (median-time/best-noise at table) for the heuristics *best* and *novelty*. On 5-SAT we run one of the best performing local search solvers at the SAT07 solver competition, Adaptg2wsat. We report the median time on the satisfiable

Table 5.7: Local search. Median time (seconds). Cutoff 1000s (10000s) for 4SAT (5SAT)

	4-SAT walksat		
50 most	High-Girth	Lit-bal-1	Random
70 vars / best	1000/—	0.12/29	0.08/27
70 vars / novelty	84/21	0.05/45	0.03/41
90 vars / best	1000/—	0.17/29	0.07/27
90 vars / novelty	76/15	0.06/47	0.04/47

	5-SAT adaptg2wsat		
peak of hardness	High-Girth	Lit-bal-1	Random
130 vars	16	2.5	0.4
150 vars	53	8.7	0.6

instances at the peak of hardness. In order to filter out the unsatisfiable instances from the sets of 100 instances we applied a cutoff of 10000 seconds, and we assumed those instances lasting more to be unsatisfiable. For all the generation methods we obtained around 50 satisfiable instances. As we can see, High-Girth dominates Lit-bal-1, and Lit-bal-1 dominates Random instances. As a possible explanation of why expansion of the incidence graph also affects the performance of local search, we could find one looking at the results in [WS02]. In that work, the existence of what they call chains of short range connections between clauses is identified as a cause for bad performance of local search, due to the long range not-explicit dependencies they create. In turns out that big cycles in the incidence graph could create these problematic chains on the formula.

Finally, we have also compared the hardness of the satisfiable instances obtained with High-Girth, at the peak of hardness, with the ones obtained with regular k-XORSAT and with our modification of k-XORSAT where we generate the system of linear equations with our high girth algorithm (HG-XORSAT). Table 5.8 shows the results when solving test-sets of 100 satisfiable instances with a cutoff of $2 * 10^4$ seconds per instance. The instances from XORSAT are harder than the satisfiable ones from High-Girth, but when we use our high girth algorithm for the k-XORSAT instances is when we obtain the hardest instances, with orders of magnitude of difference, and even in some cases we have not been able to reach the median with our cutoff time ($median > 2 \cdot 10^4$).

Hard n-ary CSP instances

For generating the n-ary CSP instances we have used two methods: the model E described in [AKK$^+$97] and our method High-Girth. We have solved

Table 5.8: Median time (in seconds) for 3-SAT, 4-SAT and 5-SAT for High-Girth with kcnfs, and regular XORSAT and HG-XORSAT with minisat. Results only for best solver among satz, minisat and kcnfs.

	3-SAT					
Num. vars	200	250	270	300	330	350
High-Girth	0	7	14	91	368	1125
XORSAT	14	386	2322	19778	$>2 \cdot 10^4$	$>2 \cdot 10^4$
HG-XORSAT	642	$>2 \cdot 10^4$	$>2 \cdot 10^4$	$>2 \cdot 10^4$	$>2 \cdot 10^4$	$>2 \cdot 10^4$

	4-SAT			**5-SAT**		
Num. vars	100	130	150	80	90	100
High-Girth	3	59	1180	64	405	2839
XORSAT	4	201	2543	51	290	2528
HG-XORSAT	66	8018	$>2 \cdot 10^4$	586	3186	$>2 \cdot 10^4$

the n-ary CSP instances with the CSP solver minion [GJM06b] using the dynamic heuristic *sdf* (smaller domain first). We also report results on the direct SAT encoding [8] of the n-ary CSP instances for the SAT solvers minisat and kcnfs (some competitive solvers submitted to the CSP competition are built on top of minisat). We have generated two set of instances, one of 25 variables, domain 3, and arity 4, and the other set of 40 variables, domain 3 and arity 3. At Figure 5.14 plot the results for arity 4, seeing again that our generation method high-girth produces the hardest instances. In this figure, the results are shown in log-scale, in contrast with Figure 5.13 for $k-$SAT, because here the differences are even more significant than in Figure 5.13. However, observe that we do not have previous existing balanced models for n-ary CSPs, like Lit-bal-1 and Lit-bal-2 for $k-$SAT, that are the ones that are closer to our high girth model for $k-$SAT.

[8] For more details see [Wal00].

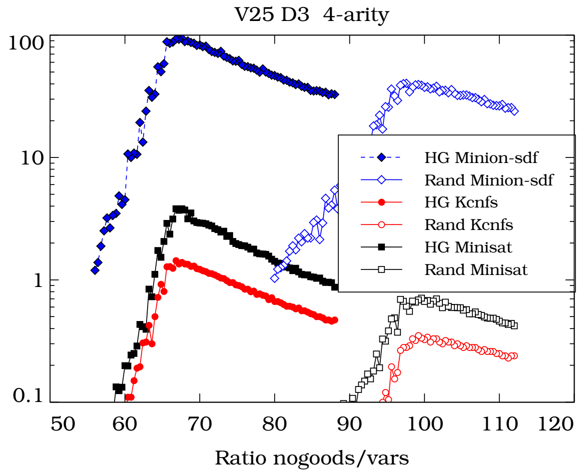


Figure 5.14: Comparison of CSP generators.

6 Conclusions and future work

The Road goes ever on and on
Down from the door where it began.
Now far ahead the Road has gone,
And I must follow, if I can,
Pursuing it with eager feet,
Until it joins some larger way
Where many paths and errands meet.
And whither then? I cannot say.

Bilbo Baggins
The Lord of The Rings. Book I.
J. R. R. Tolkien

Don't adventures ever have an end? I suppose not. Someone else always has to carry on the story.

Bilbo Baggins
The Lord of The Rings. Book II.
J. R. R. Tolkien

This work follows two main goals, and we think that, although not completely, a surmounting task for just a PhD thesis, we have achieved great progress towards fulfilling our goals. On one side there is the ever going research for better and finer benchmark sets, more fine grained, better known and characterised, and more readily available. On the other side, in our pursuit of the hardest problems, our effort o to harden, more and more, the problems, controlling the parameters that made them grow in complexity and hardness, we attained a deeper knowledge of what describes and characterised these hard problems.

Being able to harden a problem implies having a profound knowledge on some defining characteristic of problem hardness. That is, we can harden a problem because we know what aspect of the problem definition we have to fine tune in order to increase complexity, and consequently solving time. This insight can now be put to use, first to better comprehend problems and its nature, second to understand solver behaviour, to be able to spot weaknesses on solvers, and offer hints on potential enhancements.

One important, yet often neglected, aspect of benchmarking is that benchmarks should be easy and simple to run. The easier is to create a benchmark set, to choose which aspects will change and how while we are increasing hardness, the easier is that researchers will use that benchmarks. If benchmarks are well defined, well studied, well formalised and normalised, more conclusions and knowledge can be obtained from using them to measure. This is why we felt important to create as much simpler and fast as possible algorithms to generate our hard problems, being generator simplicity one of the keystones on our generator design work.

As it is obvious, there is no end in sight on the pursuit of every time harder problems. As soon as implemented solvers and solving techniques are beefed up, as new heuristics, learning techniques, optimizations, and so are incorporated into more sophisticated, and better performing solvers, problems today considered as unattainable will become easy problems. At that time, newer and harder problems should be already available, and hard problems collections should, as diverse as possible, to pinpoint where those yet-to-be solvers are trailing behind, where they can be enhanced.

A fact that has surprised us, and a worrying one, is the state of maintenance and update of the public libraries of problems that we have perceived during the realization of this work. The big libraries and collections of problems of the both areas of research (SATLIB and CSPLIB) are not updated with the advisable periodicity, and the available data is not enough extensive. One possible line of work, along with the rest of the research community is ensuring that instances and descriptions, as well as, descriptive and characterization data, are added to those libraries, making available to the rest of the research community all the assets derived from this work: formalizations, characterizations, instance generators, reference benchmarks, etc.

Another achieved goal is the definition, for the first time, of a new family of problems, Generic Edge Matching Puzzles. GEMPs are easily defined puzzles, like Sudoku problems, their formulation does not require much space, they are *human scale* puzzles. Despite this, even small puzzles, of the kind that are sold in stores within a box as a board game, can be of a surprising hardness[1].

[1] We should not be confused by the *simplicity* of solving typical "picture" kids puzzles, picture printed on top makes all pieces unique and constraints so tough that only one piece fits in every place. Try to paint a kid puzzle totally white or turn it upside down

One distinguishing trait of this kind of puzzles, along with Generalised Sudoku Problems, Quasigroup Completion Problems, and Latin Square problems, is that they can be build with a wide range of sizes and hardness, from the small and simple problems solvable by humans to medium sized very hard problems challenging to state of the art solvers, to the big, almost impossible to solve problems with today knowledge as is the case with Eternity II challenge. It is rather simple to increase or decrease size. It is trivial to augment hardness (or decrease it) just changing the number of different edges present. One advantage of Edge Matching Puzzles over Sudoku[2] is that, without changing size of the problem, one can have a wide range of hardnesses, just changing one parameter, colors i.e. domain size.

As initially stated, this work is not the end, just a milestone on the road, ahead of us lie some interesting and important tasks, and further ahead, some challenges await us.

First of all we have some jobs to do in the near future, to help results from this work reach as much part of the community as possible, namely, we must pack, clean and make available all the code and some instances to the research community, including using some of our instances in the solver competitions held.

Then, some of the extensions to this work that can be done are summarized here as follows:

- The methods for high expanding graph generation we have used in this thesis, allow to generate graphs of any size and number of edges. In the literature of expander graphs, there exists some methods that achieve a very high expansion, but they produce graphs with only some specific sizes and number of edges, so one cannot use them as the basis for hard instance generators. It would be interesting to study the modification of some of these methods to generate graphs with a wider range of parameters in order to use them to generate hard instances. One promising family of graphs are the ones introduced in [CRVW02].

- We could tighten the boundaries of our analytical expression for the Phase Transition of GEMP puzzles. While it is not straightforward, having an even more accurate bounding of that expression could be very interesting.

- A similar analysis of that of GEMP could be done towards bounding the expression for the Phase Transition on GSP, as of now, none such analysis exists. The results about lower bounds for k-SAT through the use of the second moment method have discovered important properties of the space of solutions of a formula, with nice implications with

and solve it.

[2] We include Latin Squares and Quasigroups as they are similar in this aspect to Sudokus

respect to the complexity of finding solutions. We expect that a similar analysis for more structured problems will provide similar benefits for these other problems.

- Obviously better heuristics, better consistency methods, better or even novel approaches, could be, and should, be designed to solve GEMP problems.

- Our method for hard n-ary CSPs is based on a generalization of the model for hard k-SAT instances, based on the results that link k-SAT complexity with the expansion of its incidende graph. However, the incidende graph we have used for n-ary CSPs does not allow a straightforward explanation of why the expansion of this graph increases the hardness of the generated n-ary CSP instances. This is an interesting future question to answer, given that the previous existing definition of incidence graph in the literature for n-ary CSPs is not equivalent to ours.

- Studies similar to those done here for SAT and CSP solvers could be done for High Girth based balanced instances and MaxSAT and MaxCSP solvers. We suspect that the reasons that make hard the high expanding SAT instances, the fact that they have many partial satisfying assingments that cover a big subset of the variables, will probably increase the difficulty of distinguising optimal solutions from non-optimal ones.

- Deeper understanding of GEMP problems behaviour when using local search methods, MaxSAT, and/or MaxCSP, and their hardness characterization could deepen our understanding of GEMP and similar problems.

- Although treewidth, and related parameters as expansion and high girth, have proven that can easily influence hardness, it could be interesting to check whether other parameters (clustering, for example), can be used to harden problems, or, conversely, as a speed up by solvers.

- It is important that all knowledge gained with works like this, using *laboratory* built problems, now finds its way into real world problems. One possible way to make this happen is studying which characteristics are present in real world problems and search for similarities between real world problems and studied problems, thus allowing solvers make use of learned techniques.

- Given that expansion can be hard to measure, and parameters such as girth only provide lower bounds, it could be interesting to search for other parameters that can be used to approximate treewidth, or others, not related to treewidth but that can be used to predict or increase hardness.

Bibliography

[ABF$^+$06] C. Ansótegui, R. Béjar, C. Fernández, C. Gomes, and C. Mateu. The impact of Balance in a highly structured problem domain. In *Proc. of AAAI06*, pages 438–443. AAAI Press, 2006.

[ABFM07] Carlos Ansótegui, Ramón Béjar, Cèsar Fernández, and Carles Mateu. On Balanced CSPs with High Treewidth. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada*, pages 161–166, 2007.

[ABFM08a] Carlos Ansótegui, Ramón Béjar, César Fernàndez, and Carles Mateu. Edge Matching Puzzles as Hard SAT/CSP Benchmarks. In *CP '08: Proceedings of the 14th International Conference on Principles and Practice of Constraint Programming*, pages 560–565, Berlin, Heidelberg, 2008. Springer-Verlag.

[ABFM08b] Carlos Ansótegui, Ramón Béjar, César Fernàndez, and Carles Mateu. Edge matching puzzles as hard SAT/CSP benchmarks (extended version). Technical Report TR-1-08, Dept. of Computer Science, Universitat de Lleida, 2008. http://ccia.udl.cat/images/stories/Papers/techrep1_08.pdf.

[ABFM08c] Carlos Ansótegui, Ramón Béjar, Cèsar Fernández, and Carles Mateu. How Hard is a Commercial Puzzle: the Eternity II Challenge. volume 184 of *Frontiers in Artificial Intelligence and Applications - Artificial Intelligence Research and Development*, pages 99–108. IOS Press, 2008.

[AdD$^+$04] Carlos Ansótegui, Alvaro del Val, Iván Dotú, Cèsar Fernández, and Felip Manyà. Modelling Choices in Quasigroup Completion: SAT vs CSP. In *Proc. of AAAI-04*, 2004.

[AGKS00] Dimitris Achlioptas, Carla Gomes, Henry Kautz, and Bart Selman. Generating Satisfiable Problem Instances. In *Proc. of AAAI-00*, pages 193–200, 2000.

[AKK$^+$97] D. Achlioptas, L. M. Kirousis, E. Kranakis, D. Krizanc, M. Molloy, and Y. C. Stamatiou. Random Constraint Satisfaction: a More Accurate Picture. In *Proceedings CP'97*, pages 107–120, Linz, Austria, 1997.

[ALMP08] J. Argelich, C.M. Li, F. Manya, and J. Planes. The first and second max-sat evaluations. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:251–278, 2008.

[AM06] Dimitris Achlioptas and Cristopher Moore. Random k-sat: Two moments suffice to cross a sharp threshold. *SIAM J. Comput.*, 36(3):740–762, 2006.

[AP04] Dimitris Achlioptas and Yuval Peres. The Threshold for Random k-SAT is $2^k \log 2 - O(k)$. *Journal of the American Mathematical Society*, 17(4):947–973, 2004.

[AR94] Noga Alon and Yuval Roichman. Random Cayley Graphs and Expanders. *Random Structures and Algorithms*, 5:271–284, 1994.

[AR01] M. Alekhnovich and A. Razborov. Lower bounds for polynomial calculus: non-binomial case. In *Proceedings of 42nd Annual Symposium on Fondutations of Computer Science*, pages 190–199, 2001.

[AS88] R. Phillips A. Lubotzky and P. Sarnak. Ramanujan Graphs. *Combinatorica*, 8:261–277, 1988.

[Ats04] Albert Atserias. On Sufficient Conditions for Unsatisfiability of Random Formulas. *Journal of the ACM*, 51(2):281–311, 2004.

[BDIS05] Yacine Boufkhad, Olivier Dubois, Yannet Interian, and Bart Selman. Regular Random -SAT: Properties of Balanced Formulas. *J. Autom. Reasoning*, 35(1-3):181–200, 2005.

[BFM05] R. Béjar, C. Fernández, and C. Mateu. Statistical Modelling of CSP Solving Algorithms Performance. In *Proceedings CP'05*, 2005.

[BGHK95] Hans L. Bodlaender, John R. Gilbert, HjÃ¡lmtÃœr Hafsteinsson, and Ton Kloks. Approximating treewidth, pathwidth, frontsize, and shortest elimination tree. *Journal of Algorithms*, 18-2:238–255, 1995.

[BHHW96] Daniel G. Bobrow, Tad Hogg, Bernardo A. Huberman, and Colin P. Williams, editors. *Special volume on frontiers in problem solving: phase transitions and complexity*, volume 81. Elsevier Science Publishers Ltd., Essex, UK, 1996.

[BJK05] Andrei A. Bulatov, Peter Jeavons, and Andrei A. Krokhin. Classifying the complexity of constraints using finite algebras. *SIAM J. Comput.*, 34(3):720–742, 2005.

[BKV$^+$81] M. Blum, R. M. Karp, O. Vornberger, C. H. Papadimitriou, and M. Yannakakis. The complexity of testing whether a graph is a superconcentrator. *Inf. Process. Letters*, 13(4/5):164–167, 1981.

[Bod98] Hans L. Bodlaender. A partial -arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209(1-2):1–45, 1998.

[Bod05] Hans L. Bodlaender. Discovering Treewidth. In *Proc. of SOFSEM 2005 (LNCS 3381)*, pages 1–16, 2005.

[BR96] Christian Bessière and Jean-Charles Régin. MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems. In *CP*, pages 61–75, 1996.

[BS96] R. J. Bayardo and Robert Schrag. Using CSP Look-Back Techniques to Solve Exceptionally Hard SAT Instances. In *CP'96*, pages 46–60, 1996.

[BSW01] E. Ben-Sasson and A. Wigderson. Short proofs are narrow-resolution made simple. *Journal of the ACM*, 48(2):149–169, 2001.

[CF86] Min-Te Chao and John Franco. Probabilistic analysis of a generalization of the unit clause literal selection heuristic for the k-satisfiability problem. *SIAM J. Comput.*, 15:1108–1118, 1986.

[Cha03] L. Sunil Chandran. A High Girth Graph Construction. *SIAM journal on Discrete Mathematics*, 16(3):366–370, 2003.

[Chu78] F. R. K. Chung. On Concentrators, superconcentrators, generalizers and nonblocking networks. *Bell Systems Tech. Journal*, 58:1765–1777, 1978.

[Chu97] Fan Chung. *Spectral Graph Theory.* Number 92 in Regional Conference Series in Mathematics. AMS, 1997.

[CKT91] Peter Cheeseman, Bob Kanefsky, and William M. Taylor. Where the Really Hard Problems Are. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence, IJCAI-91, Sidney, Australia*, pages 331–337, 1991.

[CO05] Amin Coja-Oghlan. On The Laplacian Eigenvalues of G_n, p. prepint, 2005.

[CRVW02] Michael R. Capalbo, Omer Reingold, Salil P. Vadhan, and Avi Wigderson. Randomness conductors and constant-degree lossless expanders. In *IEEE Conference on Computational Complexity*, page 15, 2002.

[CS03] L. Sunil Chandran and C. R. Subramanian. A Spectral Lower Bound for the Treewidth of a Graph and its Consequences. *Information Processing Letters*, 87(4):195–200, 2003.

[CS05] L. Sunil Chandran and C. R. Subramanian. Girth and treewidth. *Journal of Combinatorial Theory, Series B*, 93:23–32, 2005.

[DD01] Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3-SAT formulae. In *Proc. of IJCAI'01*, pages 248–253, 2001.

[DD07] Erik D. Demaine and Martin L. Demaine. Jigsaw Puzzles, Edge Matching, and Polyomino Packing: Connections and Complexity. *Graphs and Combinatorics*, 23(s1):195, 2007.

[DdC03] Iván Dotú, Alvaro del Val, and Manuel Cebrián. Redundant Modeling for the QuasiGroup Completion Problem. In *CP03*, 2003.

[DFMSN00] Camil Demetrescu, D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Maintaining shortest paths in digraphs with arbitrary arc weights: An experimental study. In *Proceedings of the 4-th Workshop on Algorithm Engineering (WAE'00)*, September 2000.

[DI04] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the Association for Computing Machinery (JACM)*, 51(6):968–992, 2004.

[DI06] Camil Demetrescu and Giuseppe F. Italiano. Experimental analysis of dynamic all pairs shortest path algorithms. *ACM Transactions on Algorithms*, 2(4):578–601, 2006. Special issue devoted to selected papers from the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'04).

[DSV03] Giuliana Davidoff, Peter Sarnark, and Alain Valette. *Elementary Number Theory, Group Theory, and Ramanujan Graphs.* Number 55 in London Mathematical Society Student Texts. Cambridge University Press, 2003.

[ES03] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In *Proc. of SAT'03*, pages 502–518, 2003.

[Fre82] Eugene C. Freuder. A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32, 1982.

[Fre90] Eugene C. Freuder. Complexity of k-tree structured constraint satisfaction problems. In *AAAI'90*, pages 4–9, 1990.

[Fri04] Joel Friedman. A proof of Alon's second eigenvalue conjecture and related problems. *Memoirs of the A.M.S.*, 2004.

[FV04] Linyuan Lu Fan Chung and Van Vu. The Spectra of Random Graphs with Given Expected Degrees. *Internet Mathematics*, 1(3):257–275, 2004.

[GFSB04] Carla Gomes, Cèsar Fernández, Bart Selman, and Christian Bessière. Statistical Regimes Across Constrainedness Regions. In *Proceedings CP'04*, 2004.

[GJM06a] Ian P. Gent, Christopher Jefferson, and Ian Miguel. Minion: A fast scalable constraint solver. In *ECAI 2006, 17th European Conference on Artificial Intelligence*, pages 98–102, 2006.

[GJM06b] Ian P. Gent, Christopher Jefferson, and Ian Miguel. Watched Literals for Constraint Propagation in Minion. In *Proc. of CP'06*, pages 182–197, 2006.

[GMP$^+$01] I. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. Random constraint satisfaction: flaws and structure. *Constraints*, 6(4):345–372, 2001.

[Gro03] Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. In *Proc. of FOCS'03*, pages 552–561, 2003.

[GS97] Carla Gomes and Bart Selman. Problem Structure in the Presence of Perturbations. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 221–227, New Providence, RI, 1997. AAAI Press.

[GS02] C. Gomes and D. Shmoys. Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem. In *Proceedings Computational Symposium on Graph Coloring and Extensions*, 2002.

[GS06] Joachim Gudmundsson and Michiel Smid. On spanners of geometric graphs. In *Algorithm Theory – SWAT 2006*, volume 4059 of *LNCS*, pages 388–399, 2006.

[GSC97] Carla P. Gomes, Bart Selman, and Nuno Crato. Heavy-tailed distributions in combinatorial search. In *Proceedings of the Third International Conference of Constraint Programming (CP-97)*, Linz, Austria., 1997. Springer-Verlag.

[GW94] I. Gent and T. Walsh. Easy Problems are Sometimes Hard. *AI Journal*, 70:335–345, 1994.

[GW99] I.P. Gent and T. Walsh. Csplib: a benchmark library for constraints. Technical report, Technical report APES-09-1999, 1999. Available from http://csplib.cs.strath.ac.uk/. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99).

[Hay97] Brian Hayes. Can't Get No Satisfaction. *American Scientist*, 85:108–112, 1997.

[HE80] R. M. Haralick and G. L. Elliott. Increasing tree search efficiency for constraint satisfaction problems. *AI Journal*, 14:263–313, 1980.

[HHW96] T. Hogg, B. Huberman, and C. Williams. Phase Transitions and Search Problems. *Artificial Intelligence*, 81 (1-2):1–15, 1996.

[HJKN06] Harri Haanpää, Matti Järvisalo, Petteri Kaski, and Ilkka Niemelä. Hard satisfiable clause sets for benchmarking equivalence reasoning techniques. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):27–46, 2006.

[HN90] P. Hell and J. NešetRil. On the complexity of H-coloring. *Journal of Combinatorial Theory, Series B*, 48:92–110, 1990.

[HO06] Tudor Hulubei and Barry O'Sullivan. The Impact of Search Heuristics on Heavy-Tailed Behaviour. *Constraints*, 11(2), 2006.

[HS] Holger H. Hoos and Thomas Stützle. SATLIB: An Online Resource for Research on SAT. pages 283–292.

[HW94] T. Hogg and C. P. Williams. The Hardest Constraint Problems: a Double Phase Transition. *AI Journal*, 69:359–377, 1994.

[Jär06] Matti Järvisalo. Further investigations into regular xorsat. In *Proceedings of the AAAI 2006,*. AAAI Press / The MIT Press, 2006.

[JM96] Mark T. Jacobson and Peter Matthews. Generating uniformly distributed random Latin squares. *Journal of Combinatorial Design*, 4:405–437, 1996.

[JMS05] H. Jia, C. Moore, and Bart Selman. From spin glasses to hard satisfiable formulas. In *Proceedings of SAT'05*, volume 3452 of *Lecture Notes in Computer Science.*, pages 199–210. Springer, 2005.

[Kah93] N. Kahale. *Expander Graphs.* PhD thesis, MIT, 1993.

[Kah95] N. Kahale. Eigenvalues and expansion of regular graphs. *Journal of the ACM*, 42(5):1091–1106, 1995.

[KBv01] Arie M. C. A. Koster, Hans L. Bodlaender, and Stan P. M. van Hoesel. Treewidth: Computational Experiments. *ZIB-Report*, 01-38, 2001.

[KRA$^+$01] Henry Kautz, Yongshao Ruan, Dimitris Achlioptas, Carla Gomes, Bart, Stickel, and Mark. Balance and Filtering in Structured Satisfiable Problems. In *Proc. of IJCAI-01*, pages 193–200, 2001.

[KSVW01] M. Krivelevich, B. Sudakov, V. H. Vu, and N. Wormald. Random Regular graphs of High Degree. *Random Structures and Algorithms*, 18:346–363, 2001.

[KTV97] Ravi Kannan, Prasad Tetali, and Santosh Vempala. Simple Markov-chain algorithms for generating bipartite graphs and tournaments. In *Proc. of the eighth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 193–200, 1997.

[KvK02] A. M. C. A. Koster, S. P. M. van Hoesel, and A. W. J. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40:170–180, 2002.

[LA97] Chu Min Li and Anbulagan. Look-ahead versus look-back for satisfiability problems. In *Proc. of CP'97*, pages 341–355, 1997.

[LS05] Ashish Sabharwal Lukas Kroc and Bart Selman. Survey Propagation Revisited . In *23rd Uncertainty in Artificial Intelligence 2007*, pages 217–226, 2005.

[LWZ07] Chu Min Li, Wanxia Wei, and Harry Zhang. Combining adaptive noise and look-ahead in local search for sat. In *SAT*, pages 121–133, 2007.

[MWW04] B. D. McKay, N. C. Wormald, and B. Wysocka. Short cycles in random regular graphs. *Elect. J. Combinatorics*, 11:–66, 2004.

[MZ02] G. Parisi M. Mézard and R. Zecchina. Analytic and Algorithmic Solution of Random Satisfiability Problems. *SCIENCE*, 297:812–815, 2002.

[MZK$^+$96] R. Monasson, R. Zecchina, S. Kirkpatrick, B. Selman, and L. Troyansky. Phase transitions and search cost in the 2 + p-sat problem, 1996.

[MZK$^+$99] Rémi Monasson, Riccardo Zecchina, Scott Kirkpatrick, Bart Selman, and Lidror Troyansky. Determining computational complexity from characteristic phase transitions. *Nature*, 400:133–137, 1999.

[NAP05] Assaf Naor, Dimitris Achlioptas, and Yuval Peres. Rigorous location of phase transitions in hard optimization problems. *Nature*, 435(7043):759, 2005.

[PJS04] Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe. *Chaos and Fractals. New Frontiers of Science.* Springer, second edition, 2004.

[Pro93] P. Prosser. Domain filtering can degrade intelligent backtracking search. In *Proceedings IJCAI'93*, pages 262–267, Chambéry, France, 1993.

[Pro96] P. Prosser. An empirical study of phase transitions in binary constraint satisfaction problems. *AI Journal*, 81:81–109, 1996.

[Pur83] P. W. Purdom. Search rearrengement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.

[Ref04] Philippe Refalo. Impact-Based Search Strategies for Constraint Programming. In *Proceedings CP'04*, 2004.

[RG04] Jean Charles Régin and Carla Gomes. The Cardinality Matrix Constraint. In *Proceedings CP'04*, 2004.

[RR96] G. Ramalingam and Thomas Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21(2):267, 1996.

[RS86] N. Roberston and P. D. Seymour. Graph minors II: Algorithmic aspects of treewidth. *Journal of Algorithms*, 7:309–322, 1986.

[RTWZ01] F. Ricci-Tersenghi, M. Weight, and R. Zecchina. Simplest random k-satisfiability problem. *Physical Review*, E 63:026702, 2001.

[SD96] B. Smith and M. Dyer. Locating the Phase Transition in Binary Constraint Satisfaction Problems. *AI Journal*, 81:155–181, 1996.

[SG95] B. Smith and S. A. Grant. Sparse Constraint Graphs and Exceptionally Hard Problems. In *Proc. IJCAI'95*, pages 646–651, Montréal, Canada, 1995.

[SG97] B. Smith and S. A. Grant. Modelling Exceptionally Hard Constraint Satisfaction Problems. In *Proceedings CP'97*, pages 182–195, Linz, Austria, 1997.

[SHW96] Nathan Linial Sholomo Hoory and Avi Wigderson. Expnder graphs and their applications. *Bulletin of the AMS*, 43:439–561, 1996.

[SK96] B. Selman and S. Kirkpatrick. Critical behaviour in the computational cost of satisfiability testing. *AI Journal*, 81:273–295, 1996.

[SKC94] Bart Selman, Henry A. Kautz, and Bram Cohen. Noise Strategies for Improving Local Search. In *Proc. of AAAI'94*, pages 337–343, 1994.

[Smi01] Barbara M. Smith. Constructing an asymptotic phase transition in random binary constraint satisfaction problems. *Theoretical Computer Science*, 265(1–2):265–283, 2001.

[SS96] M. Sipser and D. A. Spielman. Expander codes. *IEEE Trans. on Information Theory*, 43(6):1710–1722, 1996.

[SS06] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. In *Principles and Practice of Constraint Programming - (CP'2006)*, pages 499–513, 2006.

[SSW98] Paul Shaw, Kostas Stergiou, and Toby Walsh. Arc Consistency and Quasigroup Completion. In *Proceedings of the ECAI-98 workshop on non-binary constraints*, 1998.

[SW99] A. Steger and N. C. Wormald. Generating random regular graphs quickly. *Probability and Computing*, 8:337–396, 1999.

[van05] M.R.C. van Dongen. Introduction to the solver competition. In *Proceedings of the Second International Workshop on Constraint Propagation and Implementation, Volume II, Solver Competition*, pages 1–5, 2005.

[vD97] Peter van Beek and Rina Dechter. Constraint tightness and looseness versus local and global consistency. *J. ACM*, 44(4):549–566, 1997.

[vLR06] M.R.C. van Dongen, Christophe Lecoutre, and Olivier Roussel. Results of the second csp solver competition. In *Proceedings of the Second International CSP Solver Competition*, 2006.

[Wal00] T. Walsh. SAT vs CSP. In *Proceedings CP'00*, pages 441–456, Singapore, 2000.

[WS02] Wei Wei and Bart Selman. Accelerating random walks. In *CP'2002*, pages 216–232, 2002.

[XBHL05] Ke Xu, Frédéric Boussemart, Fred Hemery, and Christophe Lecoutre. A Simple Model to Generate Hard Satisfiable Instances. In *IJCAI'05*, pages 337–342, 2005.

[XL00] K. Xu and W. Li. Exact Phase Transition in Random Constraint Satisfaction Problems. *JAIR*, 12:93–103, 2000.

[YS02] Takayuki Yato and Takahiro Seta. Complexity and Completness of Finding Another Solution and Its Application to Puzzles. In *Proc. of National Meeting of the Information Processing Society of Japan (IPSJ)*, 2002.

[ZY03] Yuanlin Zhang and Roland H. C. Yap. Erratum: P. van beek and r. dechter's theorem on constraint looseness and local consistency. *J. ACM*, 50(3):277–279, 2003.