Estimating the number of solutions on cardinality constraints

Giovanni Christian Lo Bianco Accou

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THESE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE
BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE
COMUE UNIVERSITE BRETAGNE LOIRE

ECOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : *Informatique*

Par

Giovanni LO BIANCO

ESTIMATING THE NUMBER OF SOLUTIONS ON CARDINALITY CONSTRAINTS

Thèse présentée et soutenue à IMT Atlantique – campus de Nantes, le 31/10/2019 – Amphi Besse
Unité de recherche : Laboratoire LS2N
Thèse N° : 2019 IMTA 0155

Rapporteurs avant soutenance :

Jean-Charles Régin	Professeur à Université Nice Sophia Antipolis
Willem-Jan van Hoeve	Professeur à Carnegie Mellon University – Tepper School of Business

Composition du Jury :

Président :	Vlady Ravelomanana	Professeur à Université Paris VII
Examinateurs :	Jean-Charles Régin	Professeur à Université Nice Sophia Antipolis
	Willem-Jan van Hoeve	Professeur à Carnegie Mellon University – Tepper School of business
	Nicolas Beldiceanu	Professeur à l'IMT Atlantique
Dir. de thèse :	Xavier Lorca	Directeur centre Génie Industriel à IMT Mines Albi-Carmaux
Co-dir. de thèse :	Charlotte Truchet	Maître de conférences à Université de Nantes

Invités

Gilles Pesant	Professeur à l'Ecole Polytechnique de Montreal
Frédéric Saubion	Professeur à l'Université d'Angers

Acknowledgements

First of all, I want to show my gratitude to Xavier Lorca and Charlotte Truchet who have proposed this wide thesis subject. It was quite hard not to get lost in the beginning as the subject took multiple directions. I want to thank my supervisors for guiding me all along my research and for discussing delightful theoretical problems together. I have learned a lot.

I am very honored for having such an outstanding jury. I want to thank Jean-Charles Régin and Willem-Jan van Hoeve for accepting to review my manuscript. I really appreciated their comments that made me even more confident about the theoretical importance of my research. I thank Gilles Pesant for not getting upset when I found this little mistake in one of his article and for collaborating together on the correction. This was not an easy piece of work. I also thank Vlady Ravelomanana for his expertise in random graphs and our brainstorming sessions on probabilistic models for constraints.

I want to thank every member and former PhD students of TASC as well as other people that I met in the lab: Nicolas Beldiceanu, Charles Prud'homme, Philippe David, Gilles Simonin, Eric Monfroy, Pierre Talbot, Arthur Godet, Mathieu Vavrille, Ekaterina Arafailova, Gilles Madi-Wamba, Anicet Bart, Quentin Delmée and Vivian Sicard for all these conversations, coffee breaks, lunch breaks and, overall, for making my working environment a better place. I especially thank Charles Prud'homme for his very relevant ideas on my Choco problems. I also thank Thaddeus Leonard, the intern I was in charge of during my second year, for his serious work on the statistical study of `alldifferent`. Of course, I thank my officemates Arthur Godet and Pierre Talbot for the great conversations we had as much as for all these pints we drank together. They were not from the lab, but I met them many times in Nantes or in conferences, so I thank Ghiles Ziat and Dimitri Justeau-Allaire for all the nice exchanges we had and for the crazy times we had during conferences.

I have very good friends. Some of them were encouraging me and some others are gifted for keeping me doing anything else than working. For these reasons I should thank: Alex, Hélèna, Krini, Lauren, Romain, Antoine, Diane, Styve, Kiana, Yoann, Augustin, Alan, Tipo, Chris, Pierrick, Schiebel, Sawik, Thomas, Manu, Runa, Mélyss, Delaf and many others. If any of my friends cannot find his/her name in this list, please be sure that I fairly like him/her less than those quoted above.

I finish by thanking my family, my mother Angélique and her partner Alain, my father François and his partner Pierrette, my sisters Natacha and Anouchka, my grandmother Calaine and my late grandfather Pape. I know I can be distant sometimes but you always encouraged me and I know you are proud of me.

Contents

Introduction

Energy production and delivery, public transportation networks, air traffic, financial trades, DNA sequencing, molecular biology, cybersecurity, language processing, machine learning are a small part of problems related to tremendous combinatorial systems. These structures can be very large and sophisticated and capturing all of their complexity is a challenging issue. Each of these combinatorial structures is covered by one or several engineering fields and/or research topics, and all converge to one point: they need mathematical tools to be explored. A various number of combinatorial problems comes from these scientific areas. The public transportation field raises the questions of the optimal bus lines networks, the optimal drivers rostering, the optimal passage frequencies on each line and each period of the days, *etc.* Genomics questions properties of different combinations of genes and their consequences for health, agriculture, *etc.* In data analysis, we are looking for correlation between properties or variables by analyzing huge amounts of observations. These problems are all about finding one solution or subset of solutions that match some conditions inside a very large combinatorial structure, called search space.

In this thesis, we are interested in the case of problems as expressed in Constraint Programming (CP), a framework to declaratively express and solve hard combinatorial problems. A combinatorial problem can be stated as follows. A set of variables represent the unknowns of the problem: who drives a given bus at a given date, at which times a bus passes on a given station. These variables can take their values in a finite domain: the set of all buses for instance, a time between the beginning and the end of the workdays. The problem is then expressed by adding constraints, that are logical and mathematical relations on the variables. For instance, the drivers cannot be assigned to several buses at the same time, or a driver cannot take too many shifts during a certain time period, or the buses must respect some frequencies on a given line and a given period of the day. The problem is finally to find assignments of values of the domains to the variables, such that the constraints are satisfied, for instance to build a schedule which is feasible for the buses and respects the drivers' constraints.

CP solvers explore the search space recursively by alternating two steps. First, a reasoning is done on the constraints, to reduce the possible instantiations. If a bus starts its tour at 8am and that it takes 30min to serve all the stations on the line, then this bus cannot be scheduled on another tour before at least 8.30am. Each time a variable is instantiated or its domain is reduced, the constraint solver calls recursively the filtering algorithms associated with the constraints that apply on this variable. For each constraint, there is one or several dedicated filtering algorithms that remove values of the domain that cannot appear in a solution. This stage is called the propagation. Once, we cannot reduce domains anymore, a fix point is reached and we must select (or branch on) a variable to instantiate, to explore the search space further. This stage is called the search. If a domain is empty after applying a filtering, then we have found a contradiction and the previous choices of instantiation must be revised: we must backtrack. On the opposite, if we have instantiated every variable, then we have found a solution.

One of the main assets of constraint programming is the wide range of constraints that can be used to model combinatorial problems and the various filtering algorithms that come from many fields like operations research, artificial intelligence and graph theory. The filtering algorithms can be parameterized to be more or less clever. There is a balance to find between the cleverness of these algorithms and their computation time. Also, the strategy that selects the variables to instantiate once the propagation stage is over must be parameterized. It is called the search strategy as it selects a region of the search space to explore first. The configuration of a CP solver is most of the time problem dependent and requires a deep knowledge of CP background. Therefore, constraint programming is far from being used as a black-box optimization tool. Ideally, constraint solvers should be automatically configured to fit the problem's structure.

In this thesis, we introduce mathematical models to compute the number of solutions of a constraint without solving it, and better understand the structure of the search space. It makes the link between constraint programming and model counting, which aims at counting the number of solutions of a given problem (here the constraints) without solving it. Model counting has numerous applications such as probabilistic reasoning and machine learning (C. P. Gomes, Hoffmann, Sabharwal, and Selman 2007; Meel, Vardi, Chakraborty, Fremont, Seshia, Fried, Ivrii, and Malik 2015). Counting the number of solutions on constraints gives us important information on their structure and helps configuring CP solvers (Pesant, Quimper, and Zanarini 2012). We focus on counting the number of solutions on cardinality constraints. Cardinality constraints are mostly used to model assignment problems and can all be represented as matching problems. Many filtering algorithms for cardinality constraints come from graph theory and matching theory. Graph theory is a powerful tool of modelization, for which there also exists many counting applications, that we will use to count solutions on cardinality constraints.

There are two main contributions in this thesis. The first contribution concerns counting methods for the global cardinality constraint (Régin 1996). The global cardinality constraint carries on the number of occurrences of each value that the variables may be assigned to and it can be represented as a flow graph problem. This is the most general cardinality constraint: most of the cardinality constraints can be expressed as a special case of the global cardinality constraint. It requires very sophisticated filtering algorithms and Pesant et al. (2012) showed that developing counting methods for the global cardinality constraint is as much difficult as developing its filtering algorithms. Actually, the problems that consist in counting solutions in such combinatorial structures are classified in the $\#P-$complete complexity class. Counting the number of solutions on the global cardinality constraint requires exponential time methods. Therefore, Pesant et al. (2012) did not develop exact counting methods but estimations of the true number of solutions. They established an upper bound as an estimation of the number of solutions for this constraint and integrated their result within counting-based search.

For our first contribution, we show that the estimator developed by Pesant et al. (2012) is not an upper bound, as they did not properly consider symmetries. We fix the error and develop a true upper bound on the number of solutions for the global cardinality constraint. Then, we show experimentally that the corrected upper bound and the former result behave similarly within the `maxSD` counting-based

strategy. Thereafter, we tackle the problem of counting solutions on the global cardinality constraint under the scope of group theory. More precisely, we use Burnside's lemma (Burnside 2012), which aims at counting elements in a set considering some symmetries. We deduce from this theoretical study another upper bound and we show that it is actually dominated by the upper bound that we first developed. Moreover, we show that this new upper bound is not suited for the use of counting-based heuristics. Yet, this approach is an original way to deal with counting problems on constraints and has to be investigated.

The estimators of the number of solutions for the global cardinality constraint are based on upper bounds. In our second part, we elaborate probabilistic models for every cardinality constraint and develop estimators based on the expectancy of the number of solutions. We first apply probabilistic models on the *alldifferent* constraint (Régin 1994) and show how to compute estimates of the number of solutions. The structure of the *alldifferent* constraint is well suited for the introduction of probabilistic reasoning on cardinality constraints. We also show how to use some results on random graphs of Erdős and Renyi (1963) to predict the satisfiability of an *alldifferent* instance. Secondly, we extend this probabilistic reasoning to every cardinality constraint. As developing one estimator for one constraint requires complex counting models, we factorize the work by using *range* and *roots* decomposition (Bessière, Hebrard, Hnich, Kiziltan, and Walsh 2005c). The constraints *range* and *roots* have been introduced to decompose cardinality constraints and improve the propagation stage for these constraints. We propose to apply Erdős-Renyi random graph model on these two constraints and to compute the expectancy of the number of solutions as an estimate. Thus, using the *range* and *roots* decomposition, we compute in a systematic way an estimate of the number of solutions on many cardinality constraints: *alldifferent*, *nvalue*, *atmost_nvalue*, *atleast_nvalue*, *occurrence*, *atmost*, *atleast*, *among*, *uses*, *disjoint*. We integrate these estimators within counting-based heuristics and show that, on certain class of problems, they perform as well as other generic heuristics such as `dom/wdeg`, impact-based search and activity-based search.

Outline In the first part of this thesis, we introduce the key ingredients for the understanding of these researches. In Chapter 1, we introduce the necessary tools in graph and matching theory on which cardinality constraints are based and all the mathematical methods that we will develop later. In Chapter 2, we get deeper into constraint programming and introduce the main mechanisms behind the solving process, focusing on cardinality constraints propagation and counting-based strategies.

In the second part, we present our work on the global cardinality constraint. Chapter 3 fixes the upper bound proposed by Pesant et al. (2012) and propose an experimental analysis to compare the corrected bound to the former one. This chapter is in great parts taken from our article (Lo Bianco, Lorca, Truchet, and Pesant 2019). In Chapter 4, we develop our group theory approach to count the number of solutions on the global cardinality constraint.

The third part of this thesis concerns our probabilistic approach for cardinality constraints. In Chapter 5, we introduce random graph models applied to the *alldifferent* constraint. We extend this probabilistic reasoning to many other cardinality constraints in Chapter 6, by using their *range* and *roots* decomposition. This last chapter is in great parts taken from our second article (Lo Bianco, Lorca, and Truchet 2019).

Part I

State of the art

Chapter 1

Matching Theory Material

In the field of combinatorial optimization, many problems can be modeled with graphs. For instance, the well-known Travelling Salesman Problem (Applegate, Bixby, Chvatal, and Cook 2007) aims at finding the shortest route for a salesman that goes through a set of cities and comes back to his/her initial town. This problem can be seen as a graph problem, every city being represented by a vertex, in which we are looking for the shortest Hamiltonian cycle. Another example of combinatorial problem that can be modeled with graphs are assignment problems. These problems aims at finding an assignment of tasks to some agents. For example, consider a factory, where there are several workers and several operations to be achieved during the day. Some operations require specific qualifications that only some of the workers have. The chief of the plant want to assign each operation to a worker. This problem can be modeled with a bipartite graph with on one side, a vertex per worker, and on the other side, a vertex per operation and there is an edge between a worker and a operation, if the worker has the required qualifications to perform this operation. Finding a solution to this problem is equivalent to finding a matching covering every operation in the bipartite graph.

There is a wide range of assignment problems and, in the field of constraint programming, a whole family of constraints, called cardinality constraints, are dedicated to these problems (see Subsection 2.2.2). Cardinality constraints can be modeled with bipartite graphs and are thus closely tied to matching problems. Filtering and counting algorithms on these constraints are based on matching theory. This chapter aims at giving the basic knowledge in this latter field that will be used all along this thesis.

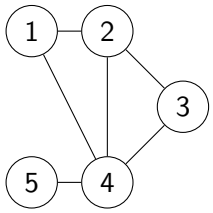
Section 1.1 and Section 1.2 introduce general definitions and theorems in Graph and Matching Theory. Then, we will get interested in counting perfect matchings for balanced bipartite graphs in Section 1.3 and for unbalanced bipartite graphs in Section 1.4.

1.1 Basic Terminology

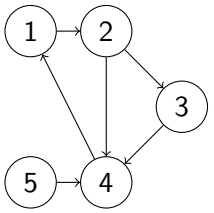
This section introduces the basic definitions and notations in graph theory and matching theory. Most of them can be found in *Matching Theory* (2009).

1.1.1 Graph Theory

We note $G = (V, E)$ the **graph** G with the set of **vertices** $V = \{v_1, \dots, v_n\}$ and the set of **edges** $E \subseteq V^2$. We call **size** of G, its number of vertices. If the pairs of vertices in E are ordered then G is a **directed** graph, if not, G is **undirected**. Let $(v_i, v_j) \in E$, then



(A) Undirected graph G_A



(B) Directed graph G_B

FIGURE 1.1: Examples of undirected and directed graphs

- undirected case: we say that v_i and v_j are **neighbors**
- directed case: we say that v_i is the **predecessor** of v_j and that v_j is the **successors** of v_i

In the undirected case, we note $D(v_i) = \{v_j \in V | (v_i, v_j) \in E\}$ the set of neighbors of v_i, also called the **neighborhood** of v_i, and $d(v_i)$ the size of the neighborhood, also called **degree** of v_i. In the directed case, we similarly note $D^-(v_i)$ the set of successors of v_i and $D^+(v_i)$ the set of predecessors of v_i. And we note $d^-(v_i) = |D^-(v_i)|$, the **outdegree** of v_i and $d^+(v_i) = |D^+(v_i)|$, the **indegree** of v_i.

Example 1.1. *Let $G_A = (V_A, E_A)$ be an undirected graph with $V_A = \{1, 2, 3, 4, 5\}$ and $E_A = \{(1, 2), (1, 4), (2, 3), (2, 4), (3, 4), (4, 5)\}$. The graph G_A is represented in Figure 1.1a. The neighbors of node 2 are $D(2) = \{1, 3, 4\}$ and its degree is 3.*

Let $G_B = (V_B, E_B)$ be a directed graph with $V_B = \{1, 2, 3, 4, 5\}$ and $E_B = \{(1, 2), (2, 3), (2, 4), (3, 4), (4, 1), (5, 4)\}$. The graph G_B is represented in Figure 1.1b. The successors of node 2 are $D^-(2) = \{3, 4\}$ and its outdegree is $d^-(2) = 2$. The predecessors of node 4 are $D^+(4) = \{2, 3\}$ and its indegree is $d^+(4) = 2$.

A finite sequence of edges of E is a **walk**. If every edge is distinct, then the walk is a **trail** and if the trail does not go through a same vertex more than once, then it is a **path**. If the first and the last vertex of a path is the same, then the path is a **cycle**. A graph that has no cycle is called **acyclic**. If an undirected graph is such that for every pair of vertices, there is a path between these two vertices, then the graph is **connected**. If this property is true for a directed graph, it is called **strongly connected**.

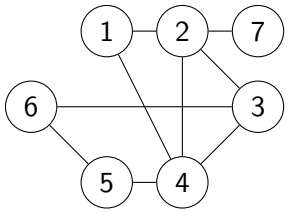
Let $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$, then G' is a **subgraph** of G. We call **strongly connected component** a maximal strongly connected subgraph G' of a directed graph G. A strongly connected component cannot be extended by adding a node of G, such that G' remains strongly connected.

An acyclic graph is also a **forest** and if it is connected, then the forest is a **tree**. If a forest $F = (V_F, E_F)$ is a subgraph of $G = (V, E)$ and $V_F = V$, then F is a **spanning forest** and if, in addition, F is connected, then it is a **spanning tree**.

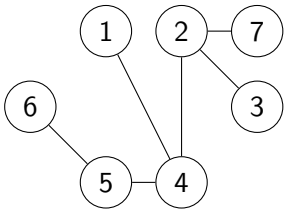
We conclude this subsection by illustrating the notions above on an undirected graph in Example 1.2 and then on a directed graph in Example 1.3.

Example 1.2. *Let $G = (V, E)$ be an undirected graph with $V = \{1, 2, 3, 4, 5, 6, 7\}$ and $E = \{(1, 2), (1, 4), (2, 3), (2, 4), (2, 7), (3, 4), (3, 6), (4, 5), (5, 6)\}$. The graph G is represented in Figure 1.2a.*

- *The sequence $((1, 4), (4, 2), (2, 3), (3, 4), (4, 2), (2, 7))$ is a walk*
- *The sequence $((1, 4), (4, 2), (2, 3), (3, 4), (4, 5))$ is a trail*

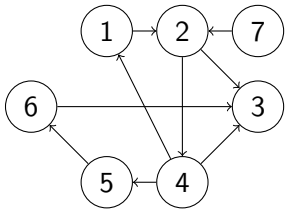


(A) An undirected graph G

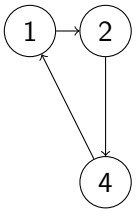


(B) Spanning tree of G

FIGURE 1.2: Paths, cycles in an undirected graph and spanning tree



(A) An directed graph G



(B) Strongly connected component of G

FIGURE 1.3: Paths, cycles in a directed graph and strongly connected component

- *The sequence* $((1,4),(4,3),(3,6),(6,5))$ *is a path*
- *The sequence* $((4,2),(2,3),(3,6),(6,5),(5,4))$ *is a cycle*

The subgraph of G represented in Figure 1.2b is a spanning tree.

Example 1.3. *Let $G = (V, E)$ be a directed graph with $V = \{1,2,3,4,5,6,7\}$ and $E = \{(1,2),(2,3),(2,4),(4,1),(4,3),(4,5),(5,6),(6,3),(7,2)\}$. The graph G is represented in Figure 1.3a.*

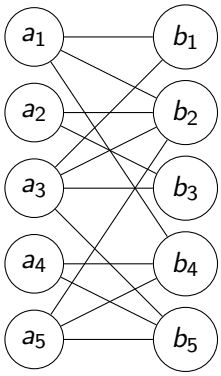
- *The sequence* $((1,2),(2,4),(4,1),(1,2),(2,3))$ *is a walk*
- *The sequence* $((4,1),(1,2),(2,4),(4,5))$ *is a trail*
- *The sequence* $((2,4),(4,5),(5,6),(6,3))$ *is a path*
- *The sequence* $((2,4),(4,1),(1,2))$ *is a cycle*

There is not any path between vertex 6 and 7, then G is not strongly connected. The subgraph of G represented in Figure 1.3b is a strongly connected component of G. There exist a path from any two vertices of this subgraph and we cannot extend it with any vertex of G.

1.1.2 Bipartite Graphs and Matchings

This thesis focuses on problems that have a bipartite graph structure in which we are searching for matchings. We give here the definitions on bipartite graphs and matchings that will be used afterwards.

Definition 1.1 (Bipartite graph)**.** *Let $G = (V, E)$, G is **bipartite** iff there exist two disjoint and non-empty subsets $A \subset V$ and $B \subset V$ such that $A \cap B = \emptyset$ and $A \cup B = V$ and every edge of E are in $A \times B$. Such a bipartite graph is also noted $G = (A \cup B, E)$ and A and B are also called **partitions** of G.*



(A) A bipartite graph $G = (A \cup B, E)$

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

(B) Biadjacency matrix of G

FIGURE 1.4: Bipartite graph and biadjacency matrix

A bipartite graph is entirely defined by its biadjacency matrix. We note $\mathbb{M}_{n,m}$ the set of matrices of size $n \times m$.

Definition 1.2 (Biadjacency matrix)**.** *Let $G = (A \cup B, E)$ be a bipartite graph with $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_m\}$. We define $\mathcal{B}(G) = (c_{ij}) \in \mathbb{M}_{n,m}$, the **biadjacency matrix** of bipartite graph G, with coefficients c_{ij} defined as follows:*

$$c_{ij} =: \begin{cases} 1, \ \textit{if } (a_i, b_j) \in E \\ 0, \ \textit{otherwise} \end{cases}$$

When enumerating matchings, the case of balanced and unbalanced bipartite graph will be treated separately.

Definition 1.3 (Balanced bipartite graph)**.** *Let $G = (A \cup B, E)$ be a bipartite graph. G is called **balanced** iff $|A| = |B|$ (and unbalanced otherwise).*

Example 1.4. *Let $G = (A \cup B, E)$ be a balanced bipartite graph such as on Figure 1.8a. We represent its biadjacency matrix in Figure 1.8b. Each row corresponds to a node from partition A and each column corresponds to a node from partition B.*

We now give some definitions on matchings.

Definition 1.4 (Matching)**.** *Let $G = (V, E)$ and $M \subseteq E$. M is called a **matching** (or an **independent set**) iff no edge have a common vertex.*

Definition 1.5 (Maximal matching)**.** *Let $G = (V, E)$ and $M \subseteq E$, a matching in G. M is called **maximal** iff there is not any larger subset $M' \subseteq E$ containing M such that M' is also a matching.*

Definition 1.6 (Maximum matching)**.** *Let $G = (V, E)$ and $M \subseteq E$, a matching in G. M is called **maximum** iff there is not any subset $M' \subseteq E$ such that M' is a matching. The size of any maximum matching in G is called the **matching number** of G.*

A maximum matching is necessarily maximal. A maximal matching cannot be "extended" to a larger matching, but it is not necessarily a maximum matching.

Definition 1.7 (Perfect matching)**.** *Let $G = (V, E)$ and $M \subseteq E$, a matching in G. M is called **perfect** iff M covers every vertex of V.*

A perfect matching is necessarily a maximum matching. We note $\Phi(G)$ the number of perfect matchings in G. We conclude this section with Example 1.5 that illustrates the definitions above:

(A) Maximal matching

(B) Maximum and perfect matching

FIGURE 1.5: Example of maximal and maximum matchings

Example 1.5. *Let's take the bipartite graph $G = (A \cup B, E)$ of example 1.4. In Figure 1.5a, we highlight in red a maximal matching of G. This matching is indeed not extendable as the node a_2 can not match any node of B anymore. In Figure 1.5b, we highlight a maximum matching, which is also a perfect matching, as it covers every node of G.*

1.2 Fundamental Results in Matching Theory

In this section, we are interested in necessary and sufficient conditions for the existence of perfect matchings or the existence of maximum matchings covering one partition. We also give some insights on how to construct such matchings.

1.2.1 Existence of Maximum and Perfect Matchings

Let $G = (A \cup B, E)$ be a bipartite graph and $X \subseteq A$ be a set of vertices of G. We note $\Gamma(X)$, the set of every vertex of V that are adjacent to one of the vertices of X. The Hall's theorem gives an necessary and sufficient condition for the existence of a matching covering one of the partition in a bipartite graph.

Theorem 1 (Hall's theorem)**.** *Let $G = (A \cup B, E)$ be a bipartite graph. There exists a matching of G covering A iff $\forall X \subseteq A, |\Gamma(X)| \geq |X|$.*

We do not detail the proof of Theorem 1. The first implication is trivial. If we have a maximum matching that covers A, then a given subset $X \subseteq A$, $\Gamma(X)$ contains at least the corresponding nodes of B given by this maximum matching. Then $|\Gamma(X)| \geq |X|$. The other implication is much more complicated. Halmos and Vaughan (1950) proposed to proceed by induction on $|A|$. The whole proof can be found in Chapter 1 of *Matching Theory*.

The Marriage theorem gives a necessary and sufficient condition for the existence of a perfect matching in a bipartite graph.

Theorem 2 (The Marriage theorem)**.** *Let $G = (A \cup B, E)$ be a bipartite graph, then there exists a perfect matching in G iff $\forall X \subseteq A, |\Gamma(X)| \geq |X|$ and G is balanced.*

Proof. Admitting Theorem 1, we know that there exists a matching of G covering A iff $\forall X \subseteq A, |\Gamma(X)| \geq |X|$.

Here, G is balanced then a matching covering A is necessarily a perfect matching. □

(A) Existence of perfect matching

(B) Absence of perfect matching

FIGURE 1.6: Applications of the Marriage theorem

Example 1.6. *The bipartite graph in Figure 1.6a is the same as in Figure 1.5b. We know that there is a perfect matching and the conditions of Theorem 2 are validated.*

Looking at the bipartite graph in Figure 1.6b, we see that $\Gamma(\{a_1, a_2, a_3\}) = \{b_2, b_3\}$*, then there are not any perfect matching in this graph.*

In practice, verifying the conditions of the Marriage theorem or Hall's theorem is very time-consuming since it requires to consider every subsets of a partition. Besides, the proof of existence does not return any maximum matching. In Subsection 1.2.2, we will see how to find a maximum matching in a given bipartite graph.

1.2.2 Building a maximum matching

In this subsection, we give some insight on how to build step by step a maximum matching in a bipartite graph. We introduce the notion of alternating path and augmenting path:

Definition 1.8 (Alternating path)**.** *Let* $G = (V, E)$ *and* $M \subseteq E$ *a matching in* G*. Let* P *a path in* G*.* P *is called a* M***-alternating path*** *if the edges of* P *are alternately in and not in* M*.*

Definition 1.9 (Augmenting path)**.** *Let* $G = (V, E)$ *and* $M \subseteq E$ *be a matching in* G*. Let* P *be a* M*-alternating path.* P *is a* M***-augmenting path*** *iff* P *starts and ends with vertices not covered by* M*.*

Lemma 1 gives a necessary and sufficient condition for matching to be a maximum matching.

Lemma 1 (Berge's lemma, Claude Berge 1957)**.** *Let* M *be a matching in a graph* G*. Then* M *is a maximum matching iff there exists no* $M-$*augmenting path in* G*.*

If, for a given matching M, a $M-$augmenting path P can be found, then we can extend M to a larger matching, by removing the common edges $M \cap P$ from M and adding the edges $P \backslash M$ in M (see Example 1.7).

The Hungarian method (*Matching Theory*, Chapter 1), which aims to construct a maximum matching in a bipartite graph, is based on Berge's lemma. It first takes an arbitrary matching in the graph and looks for an augmenting-path related to this matching. If such an augmenting-path is found, we can then extend the initial matching. We repeat this process until no augmenting-path is found. More details

(A) Alternating path (B) Augmenting path

FIGURE 1.7: Example of alternating and augmenting paths

are given in Chapter 1 of *Matching Theory* to compute efficiently such augmenting-paths.

Example 1.7. *On Figure 1.7a, we represented in red the perfect matching M and in dotted lines the $M-$alternating path $((b_1, a_1), (a_1, b_2), (b_2, a_5), (a_5, b_5))$. M is a perfect matching, then we cannot find an $M-$augmenting path.*

On Figure 1.7b, we represented in red a matching M and in dotted lines the $M-$augmenting path $((b_1, a_1), (a_2, b_2), (b_2, a_2))$. The matching M is then not maximum. We can actually extend M by removing (a_1, b_2) and adding the edges (a_1, b_1) and (a_2, b_2)

In this section, we have given the results in Matching Theory that are required for the rest of this chapter and of this thesis. In next sections, we focus on counting maximum and perfect matchings in bipartite graphs.

1.3 Counting Perfect Matchings in Bipartite Graphs

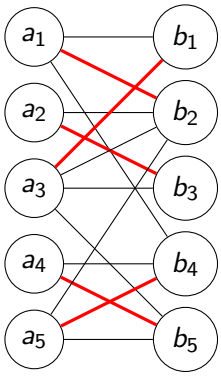
In order to give more insights on how to count perfect matchings in a bipartite graph, we first study Example 1.8. In this section we will see how permutations and matching theory are connected. We note $\mathfrak{S}_n$ the set of every permutation over $\{1, \dots, n\}$, also called the **symmetric group**.

Example 1.8. *We take the bipartite graph $G = (A \cup B, E)$ from example 1.4 and also the perfect matching (in red) in Figure 1.5b : $M = \{(a_1, b_1), (a_2, b_3), (a_3, b_5), (a_4, b_4), (a_5, b_2)\}$. We represent the coefficients of the corresponding edge of the perfect matching in red in the biadjacency matrix:*

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

We notice that every red coefficients are placed in different rows and columns. It actually corresponds to the permutation $\sigma = (1, 3, 5, 4, 2)$ over the columns. A perfect matching corresponds to a permutation, such that every corresponding coefficient is "1".

The permutation $\sigma = (2, 3, 1, 5, 4)$ corresponds to the perfect matching $\{(a_1, b_2), (a_2, b_3), (a_3, b_1), (a_4, b_5), (a_5, b_4)\}$ (see Figure 1.8)



(A)

$$\begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \end{pmatrix}$$

(B)

FIGURE 1.8: Perfect matching corresponding to $\sigma = (2, 3, 1, 5, 4)$

The permutation $\sigma = (1, 3, 4, 2, 5)$ does not correspond to any perfect matching in G because (a_3, b_4) and (a_4, b_2) are not in E.

Let $G = (A \cup B, E)$ be a balanced bipartite graph with $A = \{a_1, \dots, a_n\}$ and $B = \{b_1, \dots, b_n\}$. Finding a perfect matching in G is equivalent to finding n coefficients equal to 1 in its biadjacency matrix $\mathcal{B}(G) = (c_{ij}) \in \mathbb{M}_{n,n}$ such that all coefficients 1 are located on different rows and columns.

One permutation $\sigma \in \mathfrak{S}_n$ such that every corresponding coefficients in the biadjacency matrix are all 1 corresponds to one unique perfect matching in the bipartite graph. Let us call these permutations **consistent**. Then, counting every perfect matching in a bipartite graph is equivalent to counting every consistent permutations.

Let $\sigma \in \mathfrak{S}_n$ and $M = \{(a_1, b_{\sigma(1)}), \dots, (a_n, b_{\sigma(n)})\}$, then:

$$M \text{ is a perfect matching } \Leftrightarrow \forall i \in \{1, \dots, n\}, c_{i\sigma(i)} = 1 \Leftrightarrow \prod_{i=1}^{n} c_{i\sigma(i)} = 1 \tag{1.1}$$

We can now introduce the permanent of a matrix.

1.3.1 Permanent

The permanent of a matrix is defined as follows:

Definition 1.10 (Permanent)**.** *Let $B = (c_{ij}) \in \mathbb{M}_{n,n}$, we define the **permanent** of B as follows:*

$$Per(B) = \sum_{\sigma \in \mathfrak{S}_n} \prod_{i=1}^{n} c_{i\sigma(i)} \tag{1.2}$$

Analyzing the formula of the permanent and Equation 1.1, we note that computing the permanent is equivalent to counting the number of consistent permutations, and then, is equivalent to counting the number of perfect matchings. Proposition 1.1 makes the link between the number of perfect matchings in a bipartite graph and the permanent of its biadjacency matrix, and hence is one of the most important propositions of this chapter.

Proposition 1.1. *Let $G = (A \cup B, E)$ be a balanced bipartite graph and $\mathcal{B}(G)$ be its biadjacency matrix, then the number of perfect matchings in G is equal to the permanent of $\mathcal{B}(G)$:*

$$\Phi(G) = Per(\mathcal{B}(G)) \tag{1.3}$$

Example 1.9. *We take the bipartite graph and the associated biadjacency matrix from Example 1.8.*

$$Per\left(\mathcal{B}\left(G\right)\right) = 8$$

There are 8 perfect matchings in G.

The permanent of a matrix almost has the same definition as the determinant of a matrix. We just do not consider the signatures of the permutations. Some properties of the determinant also apply on the permanent: the permanent is a multilinear operator and an evaluation can be made by expanding by a row or by a column. However, an evaluation of the determinant can be made in polynomial time by computing the adjugate and the inverse matrices. As for the permanent of a matrix, there is not such formula and we must consider every permutation of the symmetric group, which is an exponential-size set. Computing the permanent is a $\#P-$complete problem (Valiant 1979). Thus, we are interested in approximating the permanent with polynomial-time methods.

1.3.2 Approximating the Permanent

Approximating the permanent is an active research topic in a lot of areas: mathematical analysis, graph theory, probability theory, approximation algorithms, physics... A first approach to approximating the permanent is upper-bounding. We give here two upper bounds that are used by Pesant et al. (2012). This reference is one the main paper on which this work is based and we will refer to it many times along this thesis, and more particularly in Subsection 2.3.2. We do not give any proof for the following propositions, as they use very advanced results in mathematical analysis but they can be found in Friedland (2008) and Liang and Bai (2004).

Proposition 1.2 (Brégman-Minc upper bound, Brégman 1973)**.** *Let $B = (c_{ij}) \in \mathbb{M}_{n,n}$ and $\forall i \in \{1, \dots, n\}$ let $r_i = \sum_{j=1}^{n} c_{ij}$, the sum over row i, we have:*

$$Per\left(B\right) \leq UB^{BM}(B) = \prod_{i=1}^{n}(r_i!)^{\frac{1}{r_i}} \tag{1.4}$$

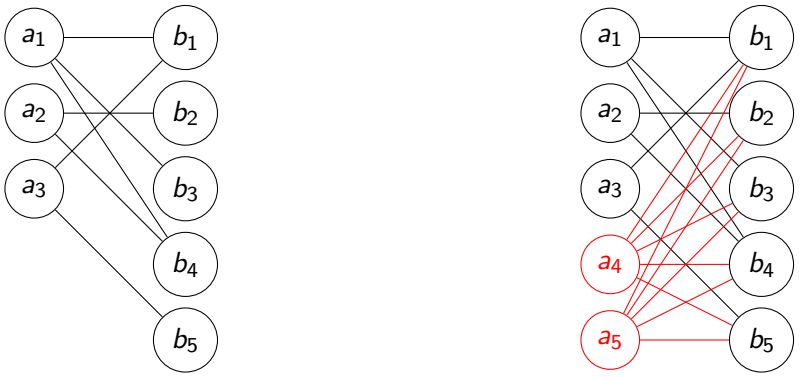
Proposition 1.3 (Liang-Bai upper bound, Liang et al. 2004)**.** *Let $B = (c_{ij}) \in \mathbb{M}_{n,n}$ and $\forall i \in \{1, \dots, n\}$ let $r_i = \sum_{j=1}^{n} c_{ij}$ be the sum over row i and $q_i = min(\lceil \frac{r_i+1}{2} \rceil, \lceil \frac{i}{2} \rceil)$, we have then:*

$$Per\left(B\right)^2 \leq UB^{LB}(B) = \prod_{i=1}^{n} q_i(r_i - q_i + 1) \tag{1.5}$$

Both upper bounds are based on the sum over the rows in the matrix. We have an upper bound of the number of perfect matchings in bipartite graphs based on the degree of the vertices. Also, Pesant et al. (2012) states that none of these two bounds dominate the other.

Another approach to approximate the permanent is to compute an estimate based on probabilistic models. Erdős et al. (1963) introduce random graphs models and propose an estimate of the permanent for random matrices. We do not detail these models here because we describe these random graphs models in Part III with a direct application of these results on cardinality constraints.

Two other approaches should also be mentioned. In the random graphs area, it has been shown that it is possible to estimate the permanent as the expectancy of the determinant of a random orientation (Godsil and Gutman 1981). Also, there exists,



(A) An unbalanced bipartite graph G

(B) Balanced bipartite graph G^b

FIGURE 1.9: Bipartite graphs G and G^b from Example 1.10

approximation algorithms that propose an estimate of the permanent in probabilistic polynomial time and that guarantee a bounded error (Jerrum, Sinclair, and Vigoda 2004; Eldar and Mehraban 2018).

1.4 The Case of Unbalanced Bipartite Graphs

Subsection 1.3 presented how to evaluate and approximate the number of perfect matchings in a balanced bipartite graph. In unbalanced bipartite graphs, there is no perfect matchings (Theorem 2). Yet, we are interested in enumerating matchings covering one of the partitions. Pesant et al. (2012) give some insights to deal with the unbalanced bipartite graphs' case. In this section, we detail this reasoning. Example 1.10 illustrates how we proceed to enumerate such matchings in an unbalanced bipartite graph.

Example 1.10. *Let $G = (A \cup B, E)$ be the unbalanced bipartite graph such as on Figure 1.9a. We want to count the number of matchings that cover the partition $A = \{a_1, a_2, a_3\}$. A way to do this is to **balance** G by adding 2 fake vertices in partition A and adding an edge between each fake vertex and vertices of the partition B. This operation results in the graph G^b. We know how to count perfect matchings on the balanced bipartite graph G^b, with Proposition 1.1, by applying the permanent on its biadjacency matrix represented in Figure 1.10b.*

Of course, adding fake vertices introduces a bias that increases the size of combinatoric structure. More precisely, for every matching that covers $\{a_1, a_2, a_3\}$ in G, there are two perfect matchings in G^b, as vertices a_4 and a_5 can match any vertex of B. To fix this problem, we simply divide the number of perfect matchings of G^b by 2 to obtain the number of matchings covering A in G. More generally, we must divide by the number of every possible permutations among fake vertices.

Definition 1.11. *Let $G = (A \cup B, E)$ be an unbalanced bipartite graph with $|A| = n_A$ and $|B| = n_B$. We note G^b, the graph in which we added $|n_B - n_A|$ vertices in the smallest partition, in order to balance G. There is an edge between each added vertex and the vertices of the other partition.*

For a balanced bipartite graph G, $\Phi(G)$ refers to its number of perfect matchings. We will keep the same notation, extended in the following way: if G is unbalanced, then $\Phi(G)$ refers to the number of matchings covering the smallest partition.

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

(A) Biadjacency matrix of G

$$\begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

(B) Biadjacency matrix of G^b

FIGURE 1.10: Biadjacency matrices of G and G^b from Example 1.10

Proposition 1.4. *Let $G = (A \cup B, E)$ be an unbalanced bipartite graph with $|A| = n_A < n_B = |B|$, then:*

$$\Phi(G) = \frac{\Phi(G^b)}{(n_B - n_A)!} = \frac{Per\left(\mathcal{B}\left(G^b\right)\right)}{(n_B - n_A)!} \tag{1.6}$$

Proof. Some insights of the proof are given by Pesant et al. 2012. We detail more formally the reasoning here.

Let $A' = \{a_{n_A+1}, \dots, a_{n_B}\}$ be the set of fake vertices that balances graph G and $E' = \{(a_{n_A+1}, b_1), \dots (a_{n_B}, b_1), \dots, (a_{n_A+1}, b_{n_B}), \dots, (a_{n_B}, b_{n_B})\}$ be the set of fake edges that connect every fake vertex of A' to every vertex of B. We assume there exists a matching $M = \{(a_1, b_{j_1}), \dots, (a_{n_A}, b_{j_{n_A}})\} \subseteq E$ that covers partition A. Then we can extend this matching to a perfect matching:
$M^* = M \cup \{(a_{n_A+1}, b_{j_{n_A+1}}), \dots, (a_{n_B}, b_{j_{n_B}})\}$. Actually we can extend M to a lot of perfect matchings. Let σ be any permutation over the vertices $\{b_{j_{n_A+1}}, \dots, b_{j_{n_B}}\}$, then $M_\sigma = M \cup \{(a_{n_A+1}, \sigma(b_{j_{n_A+1}})), \dots, (a_{n_B}, \sigma(b_{j_{n_B}}))\}$ is also a perfect matching, as the subgraph $(A' \cup B, E')$ is complete. Then, for each matching M covering A in G, there are $(n_B - n_A)!$ perfect matchings in G^b. Then we have:

$$\Phi(G) = \frac{\Phi(G^b)}{(n_B - n_A)!}$$

□

In this chapter, we have seen the basic definitions and theorems in matching theory, how to build maximum matchings and how to count perfect matchings in both balanced and unbalanced bipartite graphs. These results are used to design filtering algorithm (see Section 2.2) and counting algorithms (see Subsection 2.4 and the contributions of this thesis in Part II and Part III) for cardinality constraints.

Chapter 2

Constraint Programming

Constraint programming is a powerful tool for describing and solving combinatorial problems. It has been attracting a growing interest for the last decades, both in the academic and industrial areas. Constraint programming (CP) gathers several techniques from different research fields: operations research, artificial intelligence, graph theory, etc. It is a declarative paradigm, in which the user inputs a mathematical model of his/her problem and then gets a solution, if there exists one, of this problem. It aims at being used as a black-box optimization tool.

A CP model is made of variables and relations among these variables, called constraints. In Section 2.1, we present the basics of constraint programming modeling and give a toy problem example.

Constraints represent substructures inside the problem and are the core of constraint programming. We will see in Section 2.2 how constraint programming takes advantage of these constraints to reduce the search space step by step.

The solving process in CP is most of the time based on a backtracking algorithm. It instantiates a variable to a certain value, propagates the information inside the constraints network and repeats the process until every variable is instantiated. If a contradiction is found, the solving process backtracks to another variable instantiation. We detail in Section 2.3 this backtracking algorithm. We will also see that there exist different strategies to explore the search space. We focus on a specific kind of strategy called counting-based search, that is based on counting solutions on constraints. In Section 2.4, we show the filtering and counting algorithm for the constraint *alldifferent*.

2.1 Modelling

Constraint programming aims at being a black-box optimization tool. Yet, the problem must be properly stated to the solver and constraint programming users need at least some notions in modeling. A same problem can be modeled in several ways and choosing a model can have a great impact on the solving process. In this section, we give the basic notions in modeling and we illustrate them on a toy problem called the Word Sum problem.

2.1.1 Basic Notions

In a Constraint Satisfaction Problem (**CSP**), we aim at instantiating every **variable** of a set $X = \{x_1, \dots, x_n\}$ to a value of their respective **domain** $D = \{D_1, \dots, D_n\}$. In this thesis, we will only consider integer variables that take their values in a finite domain: $\forall x_i \in X, D_i \subset \mathbb{N}$ and $|D_i| \in \mathbb{N}$. The operator $|.|$ denotes the size (or cardinal) of a set. Some variables of X can be linked together by a relation, called **constraint**. We note $C = \{c_1, \dots, c_p\}$, the set of constraints of the CSP. A constraint

satisfaction problem P is entirely defined by the triplet $P = (X, D, C)$. For each $c_k \in C$, we note $scope(c_k) \subseteq X$, the set of variables that are involved in the constraint c_k. We note $\mathcal{S}_{c_k}$, the set of tuples allowed by c_k. Noting $|scope(c_k)| = q$, we define formally $\mathcal{S}_{c_k}$:

$$\mathcal{S}_{c_k} = \{(v_{i_1}, ..., v_{i_q}) \in D_{i_1} \times ... \times D_{i_q} | c_k \text{ is satisfied } \}$$

A constraint can be defined in **intention**, by a logical relation (like arithmetic ones, for example) or in **extention**, by the set of its allowed tuples.

A CSP is said **satisfiable** if there exists an instantiation of the variables of X such that every constraint of C is satisfied, and it is called unsatisfiable otherwise. We can formally define the set of solutions of a CSP P:

$$\mathcal{S}_P = \{(v_1, ..., v_n) \in D_1 \times ... \times D_n | \forall c_k \in C, c_k \text{ is satisfied } \}$$

Example 2.1 presents a very simple CSP illustrating the notions above.

Example 2.1. *Let $P = (X, D, C)$ be a CSP with $X = \{x_1, x_2, x_3\}$. The domains are $D_1 = \{1, 3, 4\}$, $D_2 = \{1, 2, 3\}$, $D_3 = \{3, 4\}$. And the set of constraints $C = \{c_1, c_2\}$ with c_1 is the constraint $x_1 \neq x_2$ and c_2 is the constraint $x_2 < x_3$.*

Then, $scope(c_1) = \{x_1, x_2\}$ and $scope(c_1) = \{x_2, x_3\}$. We also have:

- $\mathcal{S}_{c_1} = \{(1,2), (1,3), (3,1), (3,2), (4,1), (4,2), (4,3)\}$
- $\mathcal{S}_{c_2} = \{(1,3), (1,4), (2,3), (2,4), (3,4)\}$

And the set of solutions of P is

$$\mathcal{S}_P = \{(1,2,3), (1,2,4), (1,3,4), (3,1,3), (3,1,4), (3,2,3), (3,2,4), (4,1,3), (4,1,4), (4,2,3), (4,2,4), (4,3,4)\}$$

Most of time, in a constraint satisfaction problem, we aim at finding one solution among $\mathcal{S}_P$. For specific purposes, it can also be interesting to enumerate every solution. Sometimes, we are interested in finding the variables assignment that minimize or maximize a function f over some variables of X. We call such a problem a Constraint Optimization Problem (**COP**) and it is defined as a quadruplet $P = (X, D, P, f)$. However, this thesis only focuses on CSPs.

2.1.2 Example: A Word Sum Problem

This section presents how to model a word sum problem, which is a classical toy problem in constraint programming and a bit more sophisticated than the CSP presented in Example 2.1. Of course, the model that we present here is not unique. To solve a word sum problem, we need to find, for each letter, the corresponding digit so that the final addition is correct. Let us have the following example:

```
  M E N T A L
+ H E A L T H
-------------
M A T T E R S
```

We need to find the corresponding digits for letters `M`, `E`, `N`, `T`, `A`, `L`, `H`, `R`, `S`. Then we define the corresponding variables $X_{Let} = \{x_M, x_E, x_N, x_T, x_A, x_L, x_H, x_R, x_S\}$, which take their values in the interval $[0, 9]$, except x_M and x_H, which can not be 0. Thus, we have the following domains:

- $D_M = D_H = \{1, ... 9\}$.
- $D_E = D_N = D_T = D_A = D_L = D_R = D_S = \{0, ..., 9\}$.

There are still some variables to define, as we need to consider carries. There are five carries in this addition, so we introduce five other variables $X_{Car} = \{x_1, x_2, x_3, x_4, x_5\}$. As it is a two terms addition, the carries can only be 0 or 1, hence the domains are: $D_1 = D_2 = D_3 = D_4 = D_5 = \{0, 1\}$.

The variables of X_{Let} are the **decision** variables, by opposition to X_{Car}, which are the **auxiliary** variables, that only exist to model the problem.

Now that we have defined every variables and domains, we need to link them with some constraints. The first constraint to consider is that every variables of X_{Let} must be different. In constraint programming, there is a constraint to specifically model this:

$$alldifferent(x_M, x_E, x_N, x_T, x_A, x_L, x_H, x_R, x_S) \tag{2.1}$$

Detailed explanations on the behaviour of the constraint $alldifferent$ will be given in Section 2.2 and Section 2.4.

The others constraints to consider are simply arithmetic relations that rule the addition:

$$x_L + x_H = x_S + 10x_1 \tag{2.2}$$
$$x_A + x_T + x_1 = x_R + 10x_2 \tag{2.3}$$
$$x_T + x_L + x_2 = x_E + 10x_3 \tag{2.4}$$
$$x_N + x_A + x_3 = x_T + 10x_4 \tag{2.5}$$
$$2x_E + x_4 = x_T + 10x_5 \tag{2.6}$$
$$x_M + x_H + x_5 = x_A + 10x_M \tag{2.7}$$

We have defined all the variables, their domains and every constraint. Putting this model into a CP solver will give a solution. For example:

$$\texttt{M} = 1;\ \texttt{E} = 2;\ \texttt{N} = 3;\ \texttt{T} = 4;\ \texttt{A} = 0;\ \texttt{L} = 8;\ \texttt{H} = 9;\ \texttt{R} = 5;\ \texttt{S} = 7$$

$$\begin{array}{r} 1\ 2\ 3\ 4\ 0\ 8 \\ +\ 9\ 2\ 0\ 8\ 4\ 9 \\ \hline 1\ 0\ 4\ 4\ 2\ 5\ 7 \end{array}$$

In next sections, we give further details on how constraints manage to reduce the domains of the variables step by step and how a CP solver search a solution of a given problem.

2.2 Constraint Propagation

In constraint programming, each constraint is associated with a filtering technique (or algorithm), which aims at deleting, from the domains, values that are not allowed by the constraint. We call these values **inconsistent**.

If we take the simple arithmetic constraint $x_1 < x_2$ with $D_1 = \{2, 3, 4, 5, 6\}$ and $D_2 = \{1, 2, 5\}$ then we can filter domains D_1 and D_2 so they become $D_1 = \{2, 3, 4\}$

and $D_2 = \{5\}$. The deleted values were inconsistent: removing these values did not eliminate any solution from the initial state of the CSP.

Now, let us take a CSP $P = (X, D, C)$ with several constraints $C = \{c_1, ... c_p\}$. Once a constraint $c_k \in C$ has performed its filtering algorithm, the domains of variables from $scope(c_k)$ may have been modified. Hence, some values from domains of variables in $X \backslash scope(c_k)$ may have become inconsistent. It is generally not sufficient to remove inconsistent values for one constraint only once. As long as at least one domain change during the filtering, there might be other inconsistent values to detect.

Example 2.2. *Let $X = \{x_1, x_2, x_3\}$ with $D_1 = D_2 = D_3 = \{1, 2, 3\}$.*
Let $c_1 = \texttt{alldifferent}(x_1, x_2, x_3)$, $c_2 = x_1 < x_2$ and $c_3 = x_2 < x_3$ be three constraints over X. If we filter domains considering the order c_1, c_2, c_3:

- *For c_1, every value of each domain is consistent. No value is removed.*
- *For c_2, we remove the value 3 from D_1 and the value 1 from D_2.*
- *For c_3, we remove the value 3 from D_2 and values 1 and 2 from D_3.*

Finally we have $D_1 = \{1, 2\}$, $D_2 = \{2\}$ and $D_3 = \{3\}$. We see that we can deduce again inconsistent values from constraints c_1 and c_2. Indeed D_1 can be reduced to $\{1\}$.

A CSP can be seen as a network of constraints, in which a constraint is adjacent to another one if both constraints share at least one variable. Performing the filtering algorithm on one constraint may wake up the adjacent constraints. We keep filtering the domains until a **fix point** is reached. This process is called the **propagation**.

We write in Algorithm 1 a possible pseudo-code for a propagation stage to give some insights about how it works. The *propagate*() method returns false if one of the domains becomes empty during the filtering and true otherwise. Let a CSP $P = (X, D, C)$, we first initialize a queue of constraints by inserting every constraint of C. This queue is meant to store all the constraints which still need to be propagated. We pop one by one every constraint of the queue and we apply the associated filtering algorithm. Then, for each domain that has changed after the filtering, we push constraints, which contains the corresponding variable in their scopes, into the

queue. We keep filtering until the queue is empty: until the fix point is reached.

Algorithm 1: *propagate*()

Data: A CSP $P = (X, D, C)$
Result: *true* if and only if no domain became empty when filtering

```
constraintsQueue ← C ;
while constraintsQueue ≠ ∅ do
    c ← constraintsQueue.pop() ;
    D' ← filter(c, D) ;
    if ∃D'_i ∈ D', D'_i = {} then
        return false
    end
    for x_i ∈ scope(c) do
        if D_i ≠ D'_i then
            for c_k ∈ C\{c ∪ constraintsQueue} do
                if x_i ∈ scope(c_k) then
                    constraintsQueue.push(c_k) ;
                end
            end
        end
    end
    D ← D'
end
return true
```

Remark 2.1. *The propagation itself is not enough to solve the whole CSP:*

- *It is most of the time too hard to delete every inconsistent values from domains for some constraints. In that case, only a subset of them are removed.*
- *Even if every inconsistent value is removed, we cannot deduce which combinations of the remaining values are solutions.*

It is necessary to properly define the fix point of the propagation stage we aim to reach. This fix point characterizes how deep the propagation is performed. This is called the consistency.

2.2.1 Consistency

A **consistency** is a property that a constraint must satisfied at the fix point. There are different level of consistency. We will first focus on binary CSP, which are only composed of binary constraints. One of the most known and used consistency is called **arc consistency** (AC):

Definition 2.1 (arc consistency)**.** *A binary constraint $c(x_1, x_2)$ is arc consistent iff:*

- *For every value $v_1 \in D_1$, there is a value $v_2 \in D_2$, such that (v_1, v_2) is an allowed tuple in c.*
- *For every value $v_2 \in D_2$, there is a value $v_1 \in D_1$, such that (v_1, v_2) is an allowed tuple in c.*

*We call such an allowed tuple, a **support** of c.*

There are many ways to achieve a same level of consistency on a given constraint. The consistency is independent from the filtering algorithm. A binary CSP is called arc consistent if every constraint of the problem is arc consistent. arc consistency can be extended to k-nary constraints. We call it **generalized arc consistency** (GAC):

Definition 2.2 (Generalized arc consistency)**.** *A constraint c is GAC iff for every variable $x_i \in scope(c)$ and every value $v \in D_i$, the instantiation $x_i = v$ is allowed: there exists an allowed tuples in $\mathcal{S}_c$ such that $x_i = v$.*

In the general case, achieving GAC on a CSP is a NP-hard problem. There exists other levels of consistency, that are less demanding. One of the classical low-level consistency is the **bound consistency**. It consists in only removing values from the domains such that the bounds of the domains remain consistent. Then, there are possibly inconsistent values inside the domains that are not removed. There are also many higher levels of consistency such as path consistency or $k-$consistencies. We do not detail here these consistencies, as they are not relevant for the rest of this thesis, but most of them can be found in chapter 3 of *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*.

Practically, when solving a CSP, every constraint is associated with a filtering algorithm. Some of these algorithms achieve AC (or GAC). For some constraints, it is too hard to achieve arc consistency, then we use a filtering algorithm that achieves a lower level of consistency such as bound consistency. In solvers, we sometimes use filtering algorithms that do not achieve any particular level of consistency, but that are a good compromise between time complexity and filtering quality.

In Subsection 2.2.2, we will see that it might be preferable to model some subproblems with a same constraint instead of decomposing them into more primitive and atomic constraints.

2.2.2 Global Constraints

Global constraints exploit substructures of the main problem. Running the propagation loop on a set of constraints independently can be less efficient than dealing with the whole set of constraints instantly. Example 2.3 illustrates this by applying (generalized) arc consistency on two equivalent CSPs.

Example 2.3. *Let us take $X = \{x_1, x_2, x_3\}$ with $D_1 = \{1, 2, 3\}$ and $D_2 = D_3 = \{1, 2\}$. Let $c_1 = x_1 \neq x_2$, $c_2 = x_2 \neq x_3$ and $c_3 = x_1 \neq x_3$ be three constraints over X. The CSP $P_{neq} = (X, D, \{c_1, c_2, c_3\})$ is arc consistent. Indeed, examining each constraint individually, we cannot remove any value from the domains. However, it is obvious that domain D_1 can be reduced to $\{3\}$, since either $x_2 = 1$ and $x_3 = 2$ or vice-versa.*

Now, let us take the CSP $P_{alldifferent} = (X, D, \{alldifferent(x_1, x_2, x_3)\})$. Actually $alldifferent(x_1, x_2, x_3)$ encapsulates the whole problem P_{neq}. We have seen that some values were inconsistent when considering the whole problem P_{neq}. The constraint $alldifferent(x_1, x_2, x_3)$, and then $P_{alldiff}$, is not GAC.

In Example 2.3, we see that it can be more efficient to encapsulate some subproblems into a single constraint (here $alldifferent$ instead of several $\neq$) to capture more inconsistent values. This is one of the main purpose of global constraints such as $alldifferent$. Also, using global constraints instead of more primitive constraints allows more natural modeling. For an assignment problem, it is more natural to say that we want each agent to be assigned to different tasks than considering the difference constraints between each pair of agents.

There are many global constraints for a wide range of categories of problems:

- Graph-based constraint : `circuit` (Laurière 1978), `tree` (Nicolas Beldiceanu, Flener, and Lorca 2005),...
- Sequencing constraint : `sliding_sum` (Nicolas Beldiceanu and Carlsson 2001), `regular` (Pesant 2004),...
- Scheduling constraint : `cumulative` (Aggoun and Beldiceanu 1992),...
- Cardinality constraints : see Subsection 2.2.3.
- etc.

In Subsection 2.2.3, we focus on the cardinality constraints, which are the global constraints that will interest us in this thesis.

2.2.3 Cardinality Constraints

Cardinality constraints concern the number of occurrences of certain values or the number of different values in a solution. We list in this subsection the definitions of some of the most used cardinality constraints in the literature and that we will study in the next parts of this thesis.

For the whole section, we use the following notations: let $X = \{x_1, \dots, x_n\}$ be the variables in the scope of the constraint and $D = \{D_1, \dots, D_n\}$, the domains of these variables. We also note $Y = \{y_1, \dots, y_m\} = \bigcup_{i=1}^{n} D_i$ the union of the domains. We start with the $\mathtt{alldifferent}$ constraint that we already used previously in this chapter.

Definition 2.3 ($\mathtt{alldifferent}$, Régin 1994)**.** *A constraint $\mathtt{alldifferent}(X)$ is satisfied iff each variable $x_i \in X$ is instantiated to a value of its domain D_i and each value $y_j \in Y$ is chosen at most once. We define formally the set of allowed tuples:*

$$\mathcal{S}_{\mathtt{alldifferent}(X)} = \{(v_1, \dots, v_n) \in D \mid \forall i, j \in \{1, ..., n\}, i \neq j \Leftrightarrow v_i \neq v_j\} \tag{2.8}$$

We will detail in Section 2.4 the GAC filtering and the counting method on $\mathtt{alldifferent}$.

Another common global cardinality constraint is called $\mathtt{gcc}$, and restrains very specifically each of the values taken by the variables in Y:

Definition 2.4 (`global_cardinality` ($\mathtt{gcc}$), Régin 1996)**.** *Let $l, u \in \mathbb{N}^m$ be two m-dimension vectors. The constraint $\mathtt{gcc}(X, l, u)$ holds iff each value $y_j \in Y$ is taken at least l_j times and at most u_j times. Formally:*

$$\mathcal{S}_{\mathtt{gcc}(X,l,u)} = \{(v_1, \dots, v_n) \in D \mid \forall y_j \in Y, l_j \leq |\{v_i | v_i = y_j\}| \leq u_j\} \tag{2.9}$$

Many cardinality constraints can be seen as a specific case of $\mathtt{gcc}$, hence it is considered as the most general cardinality constraint. Chapter 3 and Chapter 4 focus on two approaches to count solutions on $\mathtt{gcc}$.

The contributions of this thesis are all about counting solutions techniques for $\mathtt{alldifferent}$, $\mathtt{gcc}$ and the following cardinality constraints: $\mathtt{nvalue}$, $\mathtt{atmost\_nvalue}$, $\mathtt{atleast\_nvalue}$, $\mathtt{occurrence}$, $\mathtt{atmost}$, $\mathtt{atleast}$, $\mathtt{among\ uses}$ and $\mathtt{disjoint}$.

Definition 2.5 ($\mathtt{nvalue}$, Pachet and Roy 1999)**.** *The constraint $\mathtt{nvalue}(X, N)$ holds iff exactly N values from Y are assigned to the variables. Formally:*

$$\mathcal{S}_{\mathtt{nvalue}(X,N)} = \{(v_1, \dots, v_n) \in D \mid N = |\{y_j \in Y | \exists x_i \in X, v_i = y_j\}|\} \tag{2.10}$$

Definition 2.6 (`atmost_nvalue`, Bessière, Hebrard, Hnich, Kiziltan, and Walsh 2005b)**.** *The constraint* $\texttt{atmost\_nvalue}(X, N)$ *holds if at most* N *values from* Y *are assigned to the variables. Formally:*

$$\mathcal{S}_{\texttt{atmost\_nvalue}(X,N)} = \{(v_1, \dots, v_n) \in D |\ N \geq |\{y_j \in Y | \exists x_i \in X, v_i = y_j\}|\} \tag{2.11}$$

Definition 2.7 (`atleast_nvalue`, Bessière et al. 2005b)**.** *The constraint* $\texttt{atleast\_nvalue}(X, N)$ *holds if at least* N *values from* Y *are assigned to the variables. Formally:*

$$\mathcal{S}_{\texttt{atleast\_nvalue}(X,N)} = \{(v_1, \dots, v_n) \in D |\ N \leq |\{y_j \in Y | \exists x_i \in X, v_i = y_j\}|\} \tag{2.12}$$

Definition 2.8 (`occurrence`, Carlsson and Fruehwirth 2014)**.** *Let* $y \in Y$*, the constraint* $\texttt{occurrence}(X, y, N)$ *holds iff exactly* N *variables are assigned to value* y*.*

$$\mathcal{S}_{\texttt{occurrence}(X,y,N)} = \{(v_1, \dots, v_n) \in D |\ N = |\{x_i \in X | v_i = y\}|\} \tag{2.13}$$

Definition 2.9 (`atmost`, Dincbas, Hentenryck, Simonis, Aggoun, Graf, and Berthier 1988)**.** *Let* $y \in Y$*, the constraint* $\texttt{atmost}(X, y, N)$ *holds iff at most* N *variables are instantiated to the value* y*. Formally:*

$$\mathcal{S}_{\texttt{atmost}(X,y,N)} = \{(v_1, \dots, v_n) \in D |\ N \geq |\{x_i \in X | v_i = y\}|\} \tag{2.14}$$

Definition 2.10 (`atleast`, Dincbas et al. 1988)**.** *Let* $y \in Y$*, the constraint* $\texttt{atleast}(X, y, N)$ *holds iff at least* N *variables are instantiated to the value* y*. Formally:*

$$\mathcal{S}_{\texttt{atmost}(X,y,N)} = \{(v_1, \dots, v_n) \in D |\ N \leq |\{x_i \in X | v_i = y\}|\} \tag{2.15}$$

Definition 2.11 (`among`, Bessière, Hebrard, Hnich, Kiziltan, and Walsh 2005a)**.** *Let* $Y' \subseteq Y$*. The constraint* $\texttt{among}(X, Y', N)$ *holds iff exactly* N *variables are assigned to value from* Y'*. Formally:*

$$\mathcal{S}_{\texttt{among}(X,Y',N)} = \{(v_1, \dots, v_n) \in D |\ N = |\{x_i \in X | v_i \in Y'\}|\} \tag{2.16}$$

Definition 2.12 (`uses`, Bessière et al. 2005c)**.** *Let* $X_1 \subseteq X$ *and* $X_2 \subseteq X$ *be two subsets of variables. The constraint* $\texttt{uses}(X, X_1, X_2)$ *holds iff the set of values taken by variables of* X_2 *is a subset of the set of values taken by variables of* X_1*. Formally:*

$$\mathcal{S}_{\texttt{uses}(X,X_1,X_2)} = \{(v_1, \dots, v_n) \in D |\ \{y_j | \exists x_i \in X_2, v_i = y_j\} \subseteq \{y_j | \exists x_i \in X_1, v_i = y_j\}\} \tag{2.17}$$

Definition 2.13 (`disjoint`, Nicolas Beldiceanu 2001)**.** *Let* $X_1 \subset X$ *and* $X_2 \subset X$*, such that* $X_1 \cap X_2 = \emptyset$*,* $\texttt{disjoint}(X, X_1, X_2)$ *holds iff none of the variables from* X_1 *is assigned to a value that one of the variable from* X_2 *takes. Formally:*

$$\mathcal{S}_{\texttt{disjoint}(X,X_1,X_2)} = \{(v_1, \dots, v_n) \in D |\ \{v_i | x_i \in X_1\} \cap \{v_i | x_i \in X_2\} = \emptyset\} \tag{2.18}$$

We have seen in Section 2.1 how to model a problem in CP and in this section, how constraints communicate to remove step by step inconsistent values from the domains. In Section 2.3, we present the required final stage to find a solution: the search procesS.

2.3 Search

After performing the propagation, we reach a fixed a point, in which no value can be removed from the domains without loosing solutions. At this stage, some variables are instantiated and some are not fixed yet. We must choose a subset of non-instantiated variables and instantiate them to a value of their domain, reducing these latter to a singleton. We call this step the **branching**. Here, we will only focus a single type of branching, which consists in branching only one variale at a time. Reducing the domain of a variable calls the propagation method one more time. If, during the propagation, a domain becomes empty, then we meet a **contradiction** and the last choice of instantiation must be revised. We call that a **backtrack**. If no contradiction is met, we reach a new fixed point and we repeat the process until every domain is reduced to a singleton, that is when a solution is found. This algorithm is called propagate_and_search and in Algorithm 2 we write a possible pseudo-code for it.

Algorithm 2: *propagate_and_search*()

```
   Data: A CSP P = (X, D, C)
   Result: true if and only if P has a solution
 1 if propagate() then
 2     if X is fully instantiated then
 3         printSolution();
 4         return true;
 5     end
 6     x_i ← chooseVariable();
 7     y ← chooseValue();
 8     D_i^save ← D_i;
 9     D_i ← {y};
10     if propagate_and_search() then
11         return true ;
12     else
13         D_i ← D_i^save \ {y};
14         return propagate_and_search();
15     end
16 else
17     return false;
18 end
```

First, we filter the domains (line 1) with the propagate() method presented in Algorithm 1. If a domain becomes empty, then the current CSP has no solution, we return false (line 17). If, after the propagation, every variable has been instantiated (line 2), we return true (line 4). If no solution has been found, we must choose a pair variable/value (line 6 and 7) to branch on (line 9). We try recursively to find a solution with $x_i = y$ (line 10). If no solution is found, we backtrack and we branch on the negation, by removing y_j from the initial domain D_i (line 13) and recursively call the method *propagate_and_search*() (line 14).

Solving a CSP can be seen as applying a depth-first search on a binary search tree. A node of the search tree corresponds to a state of the CSP. Its left son corresponds to the state of the CSP after branching on a decision (line 9) and propagating. The left subtree is visited when calling *propagate_and_search*() on line 10. The right son corresponds to the state of the CSP after branching on the negative decision (line 13). The right son is visited when calling *propagate_and_search*() on line 14. The leafs of the search tree correspond to contradictions (line 17) or solutions (line 4).

The two methods *chooseVariable*() and *chooseValue*() will be further discussed in Subsection 2.3.1.

2.3.1 Search strategies

In this subsection, we give some insights on how to choose a pair variable/value to branch on. The methods to choose such a pair are called **search strategies** (or search heuristics). In Algorithm 2, we first select a variable and then a value. It is also possible to select directly a pair variable/value. Search strategies can have a great influence on the efficiency of the solving process. The best strategies aim at visiting the least possible nodes in the search tree.

There is not, a priori, any strategy that could solve efficiently every CSP, in terms of number of visited nodes. Some strategies are designed for specific problems and require specific knowledge. Some other strategies are problem-independent. They are called **generic**.

A quite naive generic strategy is to consider the input order for the variables (we first select x_1, then x_2,... and finally x_n) and to instantiate them to their lower bound. Such a strategy is called **static** because the order of variables and values do not change during the search. By opposition, there also are **dynamic** strategies, that adapt the ordering according to some indicators on the search. We present in the following three generic dynamic heuristics that are often used in CP, and that have been proved to be efficient on a wide range of problems.

2.3.1.1 Domains over Weighted Degrees (`dom/wdeg`)

This search strategy has been first presented by Boussemart, Hemery, Lecoutre, and Sais (2004a). For each constraint $c \in C$, we associate a weight $weight_c$. In the beginning, every weight is set to 1. During the search every time a constraint raises a contradiction, its weight is increased. At one state of the search, we can compute the weight $weight_i$ of an unassigned variable x_i as the sum of the weights of every constraint that applies on this variable. The `dom/wdeg` heuristic aims at choosing the variable x_i with the smallest ratio $\frac{|D_i|}{weight_i}$.

This strategy selects first the variables with the smallest domains, which tends to minimize the number of nodes in the search tree. The idea is that the biggest domains are likely to be treated in the bottom of the search tree, when they will be reduced. Also it selects first the variables which are in a lot of constraints that generated a lot of contradictions. This tends to take more benefits of the propagation stage. The strategy `dom/wdeg` does not describe the value selection.

2.3.1.2 Impact-Based Search (`ibs`)

This search strategy has been first presented by Refalo (2004). Each time we select a pair variable/value (x, y), we measure the impact of the branching. The impact $I_k(x = y)$ of a branching at state (or node) k is measured by comparing the size of the CSP before and after the propagation: $I_k(x = y) = 1 - \frac{T_{P_k}}{T_{P_{k-1}}}$ where S_{T_k} is the size of the CSP P at state k. The size of a CSP can be computed as the product of the domains sizes. If assigning x to y raises a contradiction, then the impact of this branching is 1, and if this branching does not remove a lot of values, the impact will be close to 0.

We can then compute an estimate of the impact of a branching as the average of its impacts all along the search. The `ibs` strategy proposes to choose first the branching that has the higher impact estimate. This strategy chooses firsts the assignment that shrinks the most the space search. This strategy follows the **first-fail** principle, which aims at pruning the search tree as soon as possible in order to visit less nodes.

2.3.1.3 Activity-Based Search (`abs`)

This search strategy has been first presented by Michel and Hentenryck (2012). Each variable $x_i \in X$ is associated to an activity $A(x_i)$. The activity measures how often the domain of the variable changes during the search. Each time the variable's domain is affected, its activity increases and each time its domain does not change after propagation, its activity decreases with a predefined decay. The `abs` strategy chooses first the variables with the highest ratio $\frac{A(x_i)}{|D_i|}$.

2.3.2 Counting-Based Search

Counting-based search, introduced by Pesant et al. (2012), are generic heuristics that are based on the number of solutions remaining in the visited subtrees. The most known counting based heuristic is **maximum solutions density** (`maxSD`). It aims at choosing assignments that preserve most of the solutions by estimating the proportion of solutions in the subtrees. These heuristics require dedicated counting algorithms for each constraint.

Let $c \in C$ be a constraint and X the variables on which the constraint c applies, we note $\#c(X)$ the number of tuples allowed by c for X. We will need to compare the number of allowed tuples $\#c(X)$ and the number of remaining allowed tuples after instantiating a variable x_i to a value y and propagating, that we note $\#c_{x_i=y}(X)$. We now introduce the notion of solution density:

Definition 2.14 (Solutions density)**.** *The solution density of an assignment $x_i = y$ in a constraint c is defined as the ratio between the number of remaining tuples after the instantiation and the propagation and the number of tuples before the instantiation:*

$$\sigma(x_i, y, c) = \frac{\#c_{x_i=y}(X)}{\#c(X)} \tag{2.19}$$

At each node of the search tree, the heuristic `maxSD` computes the solution density for each constraint and each possible assignment in this constraint. Then it selects

the maximum solution density (among every constraint) and branches on the corresponding pair variable/value. We detail this heuristic in Algorithm 3.

Algorithm 3: `maxSD` search

Data: A CSP $P = (X, D, C)$
Result: The next assignment

```
1  maxDensity ← 0;
2  for each c ∈ C do
3      for each x_i ∈ scope(c) do
4          for each y ∈ D_i do
5              if σ(x_i, y, c) > maxDensity then
6                  (x*, y*) ← (x_i, y);
7                  maxDensity ← σ(x_i, y, c);
8              end
9          end
10     end
11 end
12 return (x*, y*);
```

There are other counting-based heuristics described in Pesant et al. (2012), that does not select the maximum density. For example, the `aAvgSD` heuristic selects the pair variable/value with the highest solution density average.

One of the main limitations of these heuristics is that they require a quite heavy process. Computing the solution density for each possible assignment in each constraint can be very costly. Gagnon and Pesant (2018) propose not to use the heuristic at each node of the search tree but to re-compute the ordering of the variables only when the product of the domains size have decreased enough (according to a predefined threshold).

Computing the solution densities requires counting techniques on each constraint. Pesant et al. (2012) state that it is very hard to compute exactly the number of solutions on some constraints. They propose counting algorithms that approximates the number of solutions for the constraint *alldifferent*, *gcc*, `regular` and `knapsack` constraints.

These counting-based heuristics are the first motivation for counting solutions on constraints. In Section 2.4, we detail the method to approximate the number of solutions on *alldifferent*.

2.4 Counting and Filtering on *alldifferent*

This section makes the link between matching theory and the *alldifferent* constraint. We first develop the method to count solutions then the method to filter inconsistent values.

2.4.1 Counting Solutions

Counting solutions on *alldifferent* has been studied by Pesant et al. (2012) as part of counting-based heuristics. We take the same notations as in Subsection 2.2.3: let $X = \{x_1, \dots, x_n\}$ be the variables, $D = \{D_1, \dots, D_n\}$, the domains of these variables and $Y = \{y_1, \dots, y_m\} = \bigcup_{i=1}^{n} D_i$ the union of the domains.

We use a graphic representation of the constraint. On one side, there are the variables of X and on the other side, every value of Y. And we draw an edge between

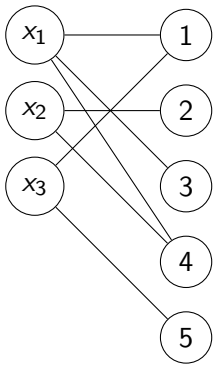


FIGURE 2.1: Value graph $G_{X,Y}$ of Example 2.4

a variable and a value if and only if the domain of the variable contains this value. We call this graph the **value graph** of the set of variables X.

Definition 2.15 (Value graph, Régin 1994)**.** *The bipartite graph $G_{X,Y} = (X \cup Y, E)$ with $E = \{(x_i, y_j) | y_j \in D_i\}$ is the value graph of X.*

The following proposition makes the equivalence between finding a matching in the value graph and finding a solution of `alldifferent` (van Hoeve 2001).

Proposition 2.1. *Let $M = (x_1, y_{j_1}), \dots, (x_n, y_{j_n}) \subseteq E$. Then,*

$$M \text{ is a matching of } G_{X,Y} \text{ covering } X \Leftrightarrow \tau = (y_{j_1}, \dots, y_{j_n}) \in \mathcal{S}_{\texttt{alldifferent}(X)} \tag{2.20}$$

Proof. By construction of the value graph. □

We note $\#\texttt{alldifferent}(X)$, the number of allowed tuples by $\texttt{alldifferent}(x)$. According to proposition 2.1, we can directly deduce the following proposition:

Proposition 2.2. *$\texttt{alldifferent}(X)$ has as many solutions as there are matchings covering X in $G_{X,Y}$:*

$$\#\texttt{alldifferent}(X) = \Phi(G_{X,Y}) \tag{2.21}$$

Example 2.4. *Let $X = \{x_1, x_2, x_3\}$ with $D_1 = \{1, 3, 4\}$, $D_2 = \{2, 4\}$ and $D_3 = \{1, 5\}$. We obtain the value graph $G_{X,Y}$ depicted on Figure 2.1*

This bipartite graph is actually the same as on Figure 1.9a. There are 8 matchings covering X, then there are 8 solutions for $\texttt{alldifferent}(X)$

counting perfect matchings in a bipartite graph is a $\#P-$complete problem. Pesant et al. (2012) propose to compute an estimate of the number of solutions of `alldifferent` as the following upper bound:

Proposition 2.3. *Let $G^b_{X,Y}$ be the value graph after balancing it and $\mathcal{B}\left(G^b_{X,Y}\right)$ its biadjacency matrix.*

$$\#\texttt{alldifferent}(X) \leq \frac{min\left(UB^{BM}\left(\mathcal{B}\left(G^b_{X,Y}\right)\right), UB^{LB}\left(\mathcal{B}\left(G^b_{X,Y}\right)\right)\right)}{(m-n)!} \tag{2.22}$$

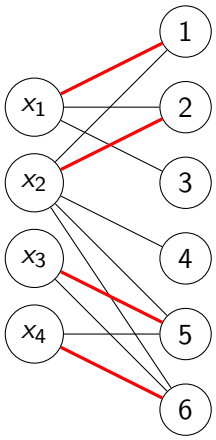


FIGURE 2.2: Example of application of Theorem 3

This upper bound is based on the Brégman-Minc and Liang-Bai upper bounds (see Section 1.3). We first balance the value graph by adding fake vertices (here, fake variables) and we compute the two permanent upper bounds for this balanced graph. We divide by $(m-n)!$ to remove the symmetric solutions induced by the fake vertices.

2.4.2 GAC Filtering for `alldifferent`

The GAC algorithm, given by Régin (1994), aims at removing every inconsistent value for a given instance of `alldifferent`. We do not detail the algorithm here but only its main ingredients, and the theory it is taken from.

We have seen in Subsection 2.4.1 that a solution of `alldifferent` corresponds to a matching covering the variables in the value graph. An edge (x_i, y_j) belongs to such a matching if and only if y_j is consistent. Then, removing every inconsistent value is equivalent to removing every edge that do not belongs to a matching covering X. The problem is that there is an exponential number of such matchings and it would be very naive to compute all of them. The filtering technique proposed by Régin (1994) is based on the following theorem:

Theorem 3 (Berge's theorem, C. Berge 1973)**.** *Let $G = (A \cup B, E)$ be a bipartite graph and $M \subseteq E$ be a maximal matching in G. Then $e \in E$ belongs to a maximal matching iff:*

- *$e \in M$,*
- *or e belongs to an even $M-$alternating path starting at a free vertex,*
- *or e belongs to an even $M-$alternating cycle.*

Theorem 3 states that we can deduce every consistent values (and thus every inconsistent values) from an arbitrary solution. We show in Example 2.5 an application of Theorem 3 with some graphical intuitions on an `alldifferent` instance.

Example 2.5. *We take the instance `alldifferent`(X) with $X = \{x_1, x_2, x_3, x_4\}$ and $D_1 = \{1,2,3\}$, $D_2 = \{1,2,4,5,6\}$, $D_3 = \{5,6\}$ and $D_4 = \{5,6\}$. We represent the corresponding value graph in Figure 2.2.*

We consider the solution in red: $(x_1 = 1, x_2 = 2, x_3 = 5, x_4 = 6)$. From Theorem 3, we can deduce the other solutions:

- *The edges* $(x_1, 2)$ *and* $(x_2, 1)$ *belong to the alternating cycle* $((x_1, 1), (1, x_2), (x_2, 2), (2, x_1))$*, then the partial instantiation* $(x_1 = 2, x_2 = 1)$ *also has a support. For example* $(x_1 = 2, x_2 = 1, x_3 = 5, x_4 = 6)$.

- *Equivalently, the partial instantiation* $(x_3 = 5, x_4 = 6)$ *can be switched to* $(x_3 = 6, x_4 = 5)$.

- *The edge* $(x_2, 4)$ *belongs to the augmenting path* $((4, x_2), (x_2, 2))$ *then, we can switch the partial instantiation* $(x_2 = 2)$ *to* $x_2 = 4$.

- *Idem for the edge* $(x_1, 3)$.

- *However, the edges* $(x_2, 5)$ *and* $(x_2, 6)$ *do not belong to any alternating cycles. Then values 5 and 6 are inconsistent in* D_2 *and can be removed.*

This theorem requires to have an initial solution. Such a solution can be easily found, for example, by applying the Hungarian method (Lovasz and Plummer 2009). If the resulting maximum matching does not cover entirely X, then the constraint does not have a solution. Since the initial maximum matching must cover every variable, there is no free variable but only free values. If a value is free, then every edge connected to this value corresponds to a consistent value (the second item of the theorem is trivially verified). The difficulty here is to find alternating cycles and more precisely, edges that do not belong to such cycles. Régin (1994) proposes a method to find these edges

- We orientate the graph this way: an edge (x_i, y_j) becomes the oriented edge (x_i, y_j) if it belongs to the initial maximum matching and become the oriented edge (y_j, x_i) if it does not.

- We compute the strongly connected components of the oriented graph.

- The edges that link two different strongly connected components corresponds to inconsistent values.

We have given here the main insights for the GAC filtering. The detailed algorithm can be found in Régin (1994).

2.5 Random Structure and Constraints

In this section, we give some insights on how random structures are related to constraint programming. The research on random structures provides a deeper understanding of hard combinatorial problem's behaviour. This allows to design specific and/or accurate algorithms to solve these kind of problems. Most of the time, to perform a probabilistic study for such problems, we first build a random process to generate instances of this problem and we analyze some properties on the generated instances. If the random process is uniform—every instance of the class of problems has the same probability to be chosen—, then we can perform an average-case analysis. Given an instance of problem with some parameters (number of variables, number of constraints, edge density,...), we can evaluate the probability that this instance has a given property.

Let's take the case of the k-SAT problem: given a Boolean formula over n Boolean variables and m clauses in a conjunctive normal form, where each clause is limited to at most k literals, we aim at finding an instantiation for each Boolean variable that

satisfy the whole Boolean formula. This problem is known to be NP-complete and is often used to prove the NP-completeness of other combinatorial problems. One way to draw uniformly a random instance of the k-SAT problem with n Boolean variables and m clauses is to choose m clauses without replacement among the set of $\binom{n}{k}$ possible clauses (Mitchell, Selman, and Levesque 1992).

This probabilistic approach of the k-SAT problem highlights a relation between the hardness of the instance and the ratio $\frac{m}{n}$ between the number of clauses m and the number of Boolean variables n. Intuitively, the higher the ratio is, the often a Boolean variable occurs in different clauses and the more likely the instance is satisfiable. When the ratio is between these two extremes, it is hard to predict if the instance is satisfiable or not: the question of the satisfiability is harder. This phenomenon characterizes a phase transition between the two states "satisfiable" and "unsatisfiable". For this class of problems, another properties of this transition phase is that, it gets sharper when the size of the instance grows—when n tends to $+\infty$. In other words, the bigger the instance is, the easier is the satisfiability prediction.

It is much easier to decide whether the instance is satisfiable or not if the instance is big enough. For the 3-SAT problem, the phase transition occurs when the ratio $\frac{m}{n}$ is in $[3.42, 5.51]$ (Friedgut, An, and Bourgain 1999): if $\frac{m}{n} < 3.42$, the problem is likely satisfiable, if $\frac{m}{n} > 4.51$, the problem is unlikely satisfiable and if $\frac{m}{n} \in [3.42, 5.51]$, it is hard to decide.

These phase transitions and asymptotic behaviours are recurrent when studying the hardness of combinatorial optimization problems in average. In Section 5.3, we study the behaviour of the satisfiability for `alldifferent` instances and identify a phase transition. Such a probabilistic approach can be helpful to predict the satisfiability of a given instance. The solving strategy can also be chosen depending on whether the instance is likely satisfiable or not. For example, it might be interesting to choose a fail-first strategy to prove the unsatisfiability of an instance, if it is likely unsatisfiable. Also, it helps a lot for the generation of benchmarks when the hardness of the instances must be parametrized.

Another study concerns the constraint `alldifferent`: Du Boisberranger, Gardy, Lorca, and Truchet (2013) noticed that performing the filtering algorithm on this constraint does not necessarily remove values from the domains and can be costly. The authors made an average-case analysis on this constraint and study when it is worthwhile to use the filtering algorithm in the case of the bound consistency (BC). An `alldifferent` instance is bound-consistent when the lower bound and upper bound of each domain is consistent (appears in one solution). The authors describe the following process to generate randomly `alldifferent` instances: let n be the number of variables and m be the number of values, for each variable, they pick uniformly a random interval among the $\frac{m\cdot(m+1)}{2}$ possible intervals. This study only focuses on the bound consistency and the case where the domains are intervals. They highlight a relation between the ratio $\frac{n}{m}$ and the probability of remaining BC after instantiating one variable - the probability that the BC filtering does not reduce any domain. If the number of variables n is much lower than the number of values m, then it is very likely that the instance remains BC after instantiating a variable. When the ratio $\frac{n}{m}$ gets closer to 1, the filtering algorithm will likely have an impact on the domains. More precisely, the authors identify that the phase transition occurs when $m - n \simeq \sqrt{2m}$. This probabilistic study of the effectiveness of the filtering algorithm can be integrated in the solving process to gain time in average: the filtering algorithm is performed only if the probability that it actually removes values is big enough.

In this thesis, we will make an average-case study on the *alldifferent* constraint, but we are not interested in the efficiency of the filtering algorithm. In Chapter 5, we use random graphs to make an average-case study of the number of solutions of *alldifferent* instances and we extend this study to other cardinality constraints in Chapter 6. This probabilistic study of the number of solutions will then be integrated into counting-based heuristics.

Part II

Counting Solutions on the Global Cardinality Constraint

Chapter 3

Revisiting Counting Solutions for the Global Cardinality Constraint

This chapter focuses on the global cardinality constraint (Régin 1996), or *gcc*, presented in Subsection 2.2.3. Such a constraint restricts the number of times a value is assigned to a variable to be in a given integer interval. It has been known to be very useful in many real-life problems derived from the generalized assignment problems (Ford and Fulkerson 1962), such as scheduling, timetabling and resource allocation (Nuijten 1994). We show in Example 3.1 an instance of `gcc` represented with a flow model. Pesant et al. (2012) state an upper bound on the number of solutions for an instance of *gcc*, as well as exact evaluations for other constraints like `regular` or `knapsack`.

Example 3.1. *Suppose we have a* `gcc` *defined on* $X = \{x_1, \dots, x_6\}$ *with domains* $D_1 = D_4 = \{1, 2, 3\}$, $D_2 = \{2\}$, $D_3 = D_5 = \{1, 2\}$ *and* $D_6 = \{1, 3\}$*; lower and upper bounds for the values are respectively* $l_1 = 1, l_2 = 3, l_3 = 0$ *and* $u_1 = 2, u_2 = 3, u_3 = 2$*. We can model this gcc as the flow problem depicted by Figure 3.1.*

The labels $l_j - u_j$ *between each value node* y_j *and the sink* `t` *represent the lower bound and upper bound for the flow between* y_j *and* `t`*. In order to make Figure 3.1 easier to read, we did not represent the labels for edges between variables and values and between the source* `s` *and the variables. The flow between a variable and a value must be* 0 *or* 1 *and the flow between the source* `s` *and each variable is* 1*.*

Then $(1, 2, 1, 2, 2, 3)$ *and* $(3, 2, 1, 2, 2, 3)$ *are solutions but* $(1, 2, 1, 1, 2, 3)$ *is not because the value* 2 *is only taken twice and it must be taken at least three times.*

This chapter present a work that have been published in Journal of Artificial Intelligence Research (Lo Bianco, Lorca, Truchet, and Pesant 2019). We first present

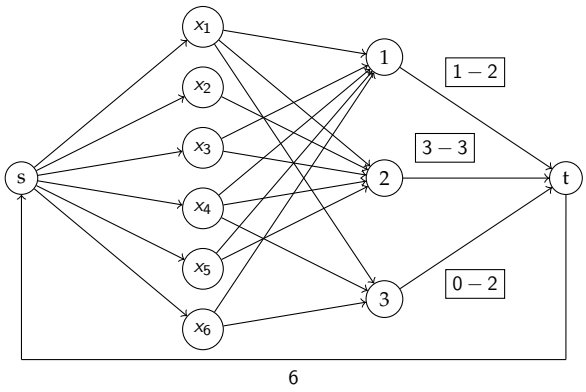
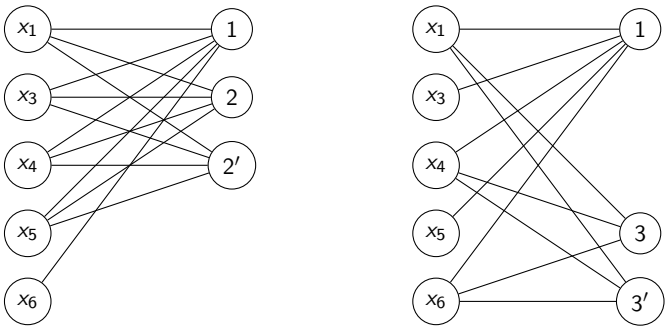


FIGURE 3.1: Flow model of the instance of gcc presented in Example 3.1



(a) (b)

FIGURE 3.2: Lower Bound Graph (a) and Residual Graph Upper Bound (b) of the `gcc` instance described in Example 3.1

in Section 3.1 the method develop by Pesant et al. (2012) to upper bound the number of solutions on a `gcc` instance. In Section 3.2, we show that the upper bound for `gcc` is actually not correct and in Section 3.3, we present a correction formulated as a non-linear minimization problem. We give a detailed algorithm and a complexity study for the computation of the new bound. The corrected bound is then adapted for counting-based search and its behavior is compared to the former result (Pesant et al. 2012). In Section 3.4, we conclude with an experimental analysis of the efficiency of both estimators within the search heuristic `maxSD`.

3.1 Lower Bound and Upper Bound Graphs

We present here the method proposed by Pesant et al. (2012) to compute an upper bound of the number of solutions of a `gcc` instance. The authors first count partial instantiations that satisfy the lower bound restriction. Then, for each of these partial instantiations, they count how many possibilities there are to complete it to a full instantiation satisfying the upper bound restriction.

We take the same notations as in Subsection 2.2.3: let $X = \{x_1, \dots, x_n\}$ be the variables, $D = \{D_1, \dots, D_n\}$, the domains of these variables and $Y = \{y_1, \dots, y_m\} = \bigcup_{i=1}^{n} D_i$ the union of the domains. Pesant et al. (2012) only consider instances in which every fixed variable (that can be instantiated to only one value) has been removed and the lower and upper bounds have been adjusted accordingly. Let $X' = \{x_i \in X |\ x_i \text{ is not fixed}\}$ be the set of unfixed variables and lower bounds are l' where $l'_j = l_j - |\{x_i | x_i \text{ is fixed and } y_j \in D_i\}|$ and upper bounds u' are defined similarly. We assume that, $\forall y_j \in Y$, $l'_j \geq 0$ and $u'_j \geq 0$.

The first stage of the method counts the partial instantiations satisfying the lower bound restriction. For this purpose, the notion of **Lower Bound Graph** is introduced:

Definition 3.1 (Lower Bound Graph, Pesant et al. 2012)**.** *Let $G_l(X' \cup Y_l, E_l)$ be a bipartite graph where X' is the set of unfixed variables and Y_l, the extended value set, that is for each $y_j \in Y$ the graph has l'_j vertices $y_j^1, y_j^2, \dots y_j^{l'_j}$ (l'_j possibly equal to zero). There is an edge $(x_i, y_j^k) \in E_l$ if and only if $y_j \in D_i$.*

Figure 3.2a represents the Lower Bound Graph for the instance described in Example 3.1 and its computation is detailed in Example 3.2.

By construction, a matching covering every vertex of Y_l corresponds to a partial instantiation of the variables satisfying the lower bound restriction. The matching

$\{(x_1, 2), (x_4, 2'),$
$(x_5, 1)\}$ corresponds to the partial assignment $(2, 2, \_, 2, 1, \_)$ (x_2 is already instantiated) and this partial instantiation satisfies the lower bound restriction. Note that the matching $\{(x_1, 2'), (x_4, 2), (x_5, 1)\}$ leads to the same partial instantiation. Switching two duplicated values does not change the resulting partial instantiation. Actually, every combination of permutations among duplicates of a same value must be considered. There are $\frac{\Phi(G_l)}{\prod_{y_j \in Y} l'_j!}$ partial instantiations satisfying the lower bound restriction.

The authors are now interested in counting every possibility to complete a partial instantiation to a full instantiation. Similarly, the **Residual Upper Bound Graph** is introduced:

Definition 3.2 (Residual Upper Bound Graph, Pesant et al. 2012)**.** *Let $G_u(X' \cup D_u, E_u)$ be an undirected bipartite graph where X' is the set of unfixed variables and Y_u the extended value set, that is for each $y_j \in Y$, the graph has $u'_j - l'_j$ vertices $y_j^1, y_j^2, \dots y_j^{u'_j - l'_j}$ (possibly $u'_j = l'_j$). There is an edge $(x_i, y_j^k) \in E_u$ if and only if $y_j \in D_i$.*

Figure 3.2b represents the Residual Upper Bound Graph for the instance described in Example 3.1. At this point, $K = \sum_{y_j \in Y} l'_j$ variables are already instantiated (without counting fixed variables). They are removed from the Residual Upper Bound Graph and, by construction, a matching covering the remaining variables corresponds to a completion of the partial assignment. On the partial instantiation $(2, 2, \_, 2, 1, \_)$, x_3 and x_6 remain to instantiate. The matching $\{(x_3, 1), (x_6, 3')\}$ leads to the full instantiation $(2, 2, 1, 2, 1, 3)$, which satisfies both the lower bound and upper bound restrictions.

However, in general, there is an exponential number of partial assignments satisfying the lower bound restriction. It is not reasonable to compute the number of maximum matchings on G_u for each of them. Pesant et al. (2012) propose an over-approximation of the number of possibilities to complete each partial assignment: the K variables already instantiated are the variables that contribute the less to the combinatorial complexity of the problem. They remove the K variables such that the number of maximum matchings in G_u covering the remaining variables is maximized. The resulting graph is noted $\overline{G_u}$ (see Figure 3.3). Like in the Lower Bound Graph, there are symmetries among duplicated values in Y_u. There are at most $\frac{\Phi(\overline{G_u})}{\prod_{y_j \in Y}(u'_j - l'_j)!}$ ways to complete a partial instantiation satisfying the lower bound restriction to a full instantiation also satisfying the upper bound restriction. We can conclude on the following upper bound:

$$\#gcc(X, l, u) \leq \frac{\Phi(G_l) \cdot \Phi(\overline{G_u})}{\prod_{y_j \in Y} l'_j! \cdot (u'_j - l'_j)!} \tag{3.1}$$

We apply this upper bounding method on the instance represented in Figure 3.1 in Example 3.2:

Example 3.2. *We consider the same gcc instance as in Example 3.1: $X = \{x_1, \dots, x_6\}$ with domains $D_1 = D_4 = \{1, 2, 3\}$, $D_2 = \{2\}$, $D_3 = D_5 = \{1, 2\}$ and $D_6 = \{1, 3\}$; lower and upper bounds for the values are respectively $l_1 = 1, l_2 = 3, l_3 = 0$ and $u_1 = 2, u_2 = 3, u_3 = 2$.*

Considering that $x_2 = 2$, the lower and upper bounds for the value 2 are respectively $l'_2 = 2$ and $u'_2 = 2$. The Lower Bound Graph is shown in Figure 2a: variable x_2 is fixed and thus does not appear in the graph, value vertex 2 is represented by two vertices because

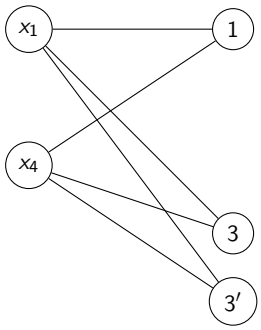
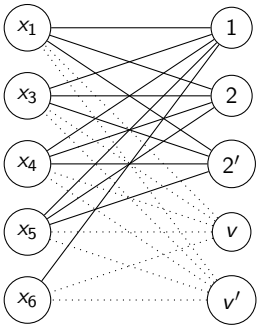


FIGURE 3.3: Residual Upper Bound Graph after removing x_3, x_5 and $x6 : \overline{G_u}$



(A) Lower Bound Graph after adding fake vertices, G_I^b

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

(B) Biadjacency matrix of G_I^b

FIGURE 3.4: Examples of undirected and directed graphs

$I'_2 = 2$ *; finally value vertex 3 does not appear because* $I'_3 = 0$*. We construct similarly the Residual Upper Bound Graph in Figure 3.2b.*

We already have instantiated 3 variables at this stage (4, if we count x_2*, which is fixed). To construct* $\overline{G_u}$*, we remove* x_3*,* x_5 *and* x_6 *from* G_u*, which contribute the less in the combinatorial complexity of the problem (they have the smallest domains). We actually could have chosen* x_1 *or* x_4 *instead of* x_6*.* $\overline{G_u}$ *is such as in Figure 3.3.*

First, we compute $\Phi(G_I)$*. We add two fake vertices on the value part,* v *and* v'*, in order to balance* G_I *(Figure 3.4a) and we compute the permanent of its biadjacency matrix (Figure 3.4b). The permanent of* $\mathcal{B}\left(G_I^b\right)$ *is 72. Then we divide by* $2!$ *to deal with the combinatorial complexity induced by the fake vertices, there are then* $\Phi(G_I) = 36$ *matching covering the value partition in* G_I*.*

Similarly, we can compute $\Phi(\overline{G_u}) = 6$*. Then, we can conclude that:*

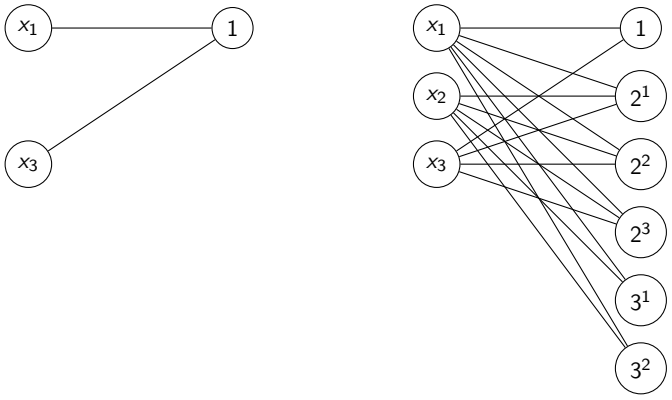
$$\#gcc(X, l, u) \leq \frac{36 \cdot 6}{2! \cdot 2!} = 54$$

The true number of solutions of this problem is 19. Scaling and also fake vertices used with the permanent bounds are factors that degrade the quality of the upper bound.

In practice, on bigger instances, we do not compute directly the number of perfect matchings, but we use the Bregman-Minc or Liang-Bai upper bound (Subsection 1.3). Also we do not compute exactly $\overline{G_u}$, because it would require to consider $\binom{|X'|}{K}$ graphs and to compute the number of perfect matchings for each of them. Instead, we remove the K variables that contribute the less to the computation of the

Instantiation	x_1	x_2	x_3
ι_1	1	2	1
ι_2	1	2	2
ι_3	1	3	1
ι_4	1	3	2
ι_5	2	2	1
ι_6	2	3	1
ι_7	3	2	1
ι_8	3	3	1

FIGURE 3.5: Array of every instantiation



(a) (b)

FIGURE 3.6: Lower Bound Graph G_l (a) and Residual Upper Bound Graph G_u (b)

Bregman-Minc or Liang-Bai upper bound (*i.e.* the ones having the smaller factors in the product).

3.2 Properly Considering Symmetries

The method described in Section 3.1 does not work in general. We develop here a counter-example that proves it wrong and we analyze the error.

3.2.1 Counter-Example

Let $X = \{x_1, x_2, x_3\}$, $D_1 = \{1, 2, 3\}$, $D_2 = \{2, 3\}$, $D_3 = \{1, 2\}$ and $l = \{1, 0, 0\}$ and $u = \{2, 3, 2\}$. A list of every instantiation is given by Figure 3.5. The Lower Bound Graph and Residual Upper Bound Graph are presented by Figure 3.6. Only value node "1" is in the Lower Bound Graph because $l_1 = 1$, $l_2 = 0$ and $l_3 = 0$. Also variable x_2 cannot take value 1, this is why it is not in the Lower Bound Graph. There is only one value in the Lower Bound Graph, which means that only one variable is instantiated at this stage, then we need to delete one variable from G_u. We choose to delete x_3, as this is the variable which has less impact on the combinatorial complexity. It ensures that we are computing an upper bound. After deleting one variable from G_u, we have $\overline{G_u}$ presented by Figure 3.7.

If we apply directly the upper bound of Equation 3.1 , we have:

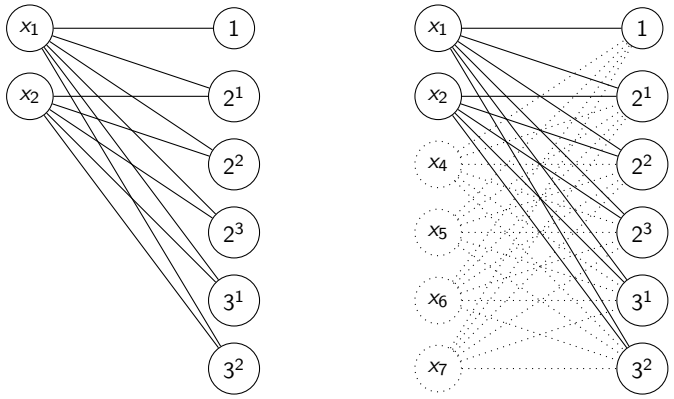


FIGURE 3.7: $\overline{G_u}$ (a) and $\overline{G_u}^b$ (b)

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

FIGURE 3.8: Biadjacency matrix of $\overline{G_u}^b$

Perfect Matching	x_1	x_2	x_4	x_5	x_6	x_7
μ_1	1	3^1	2^1	2^2	2^3	3^2
μ_2	1	3^1	2^1	2^3	2^2	3^2
μ_3	1	3^1	2^2	2^1	2^3	3^2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
μ_{24}	1	3^1	3^2	2^3	2^2	2^1
μ_{25}	1	3^2	2^1	2^2	2^3	3^1
μ_{26}	1	3^2	2^1	2^3	2^2	3^1
μ_{27}	1	3^2	2^2	2^1	2^3	3^1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
μ_{48}	1	3^2	3^1	2^3	2^2	2^1

FIGURE 3.9: Array of every perfect matchings on $\overline{G_u}^b$ corresponding to $x_1 = 1$ and $x_2 = 3$

$$\#gcc(X, I, u) \leq \frac{\Phi(G_I) \cdot \Phi(\overline{G_u})}{\prod_{y_j \in Y} l'_j! \cdot (u'_j - l'_j)!} = \frac{2 \cdot 25}{2! \cdot 3!} = \frac{25}{6} = 4,1666$$

It is obvious that $\Phi(G_I) = 2$, as for $\Phi(\overline{G_u})$, we compute the permanent of the biadjacency matrix of $\overline{G_u}^b$ (Figure 3.8) , $Per\left(\mathcal{B}\left(\overline{G_u^b}\right)\right) = 600$, and, as we added 4 fake variables, we divide by 4!. Thus, we get $\Phi(\overline{G_u}) = \frac{600}{4!} = 25$.

But, we know that $\#gcc(X, I, u) = 8$, as we enumerated every instantiation in Figure 3.5. There is an error when computing the number of matchings covering x_1 and x_2 in $\overline{G_u}^b$. Figure 3.9 lists every perfect matching of $\overline{G_u}^b$ corresponding to $\{x_1 = 1, x_2 = 3\}$.

We can notice, for example, that μ_1 and μ_2 are symmetric by transposition of x_5 and x_6 and also by transposition of 2^2 and 2^3. In the method proposed, we first deal with the fake variables symmetry and then with the duplicated values symmetry. The symmetry between μ_1 and μ_2 is thus counted twice instead of once. Fake variables symmetry and duplicated values symmetry may actually offset each other and should not be treated separately.

However, the problem does not appear for the Lower Bound Graph, as the number of variables is greater than or equal to the number of values (including the duplicates). In that case, the fake vertices symmetry comes from the same partition than the duplicated values symmetry. It is then impossible to have two perfect matchings being symmetric in both ways. In Subsection 3.2.2, we modify the model to fit the case where the duplicated values symmetry and fake variables symmetry are conflictual.

3.2.2 A Model that Fits our Specific Problem

In the previous subsection, we saw that the method does not work for the Residual Upper Bound Graph (when there are fewer variables than duplicated values). In this subsection, we formally present and refine the problem to fit this case. This model and notations will be retained for the rest of this chapter.

Let $\omega \in \mathbb{N}^m$ be the vector of occurrences, such that $\sum_{j=1}^{m} \omega_j \geq n$. In order to satisfy the global cardinality constraints, the variables of X must be instantiated in such a way that each value $y_j \in Y$ is taken at most ω_j times (ω_j are the former $u'_j - l'_j$). In order to work with balanced bipartite graph, we add fake variables that can take

any value from Y. Let $\overline{X} = \{x_{n+1}, \dots x_{n+n'}\}$ be the set of fake variables, such that $n + n' = \sum_{j=1}^{m} \omega_j$ and $\forall x_i \in \overline{X}, D_i = Y$. We introduce now the ω-multiplied value graph, which is the value graph in which each value node y_j has been duplicated ω_j times and wich has been balanced by adding fake variables:

Definition 3.3 (ω-multiplied value graph)**.** *Let $G(X, \omega) = ((X \cup \overline{X}) \cup Y_\omega, E_\omega)$ be the $\omega-$multiplied value graph with $E_\omega = \{(x_i, y_j^k) | y_j \in D_i\}$ and*
$Y_\omega = \{y_1^1, \dots, y_1^{\omega_1}, \dots, y_m^1, \dots y_m^{\omega_m}\}$.

Figure 3.10a represents the ω-multiplied value graph for the instance described in Example 3.3. We will prove that, by construction, a perfect matching in the ω-multiplied value graph, corresponds to an instantiation over X satisfying the restriction on the occurrences. We first define formally the set of perfect matchings in $G(X, \omega)$ and the set of instantiations over X satisfying the restriction on the occurrences:

Definition 3.4 (Set of perfect matchings)**.** *We note $\mathcal{PM}(X, \omega)$, the set of perfect matchings in $G(X, \omega)$:*

$$\mathcal{PM}(X, \omega) = \{\{e_1, \dots, e_{n+n'}\} \subseteq E_\omega | \forall a, b \text{ if } a \neq b,\ e_a = (x_{i_a}, y_{j_a}^{k_a}) \text{ and } e_b = (x_{i_b}, y_{j_b}^{k_b}), \text{ then } x_{i_a} \neq x_{i_b} \text{ and } y_{j_a}^{k_a} \neq y_{j_b}^{k_b}\}$$

Definition 3.5 (Set of instantiations over X)**.** *We note $\mathcal{I}(X, \omega)$, the set of instantiations over X that satisfy the restriction on the occurrences ω_j:*

$$\mathcal{I}(X, \omega) = \{(v_1, \dots, v_n) | v_i \in D_i \text{ and } |\{v_i | v_i = y_j\}| \leq \omega_j\}$$

To simplify the notations, $\mathcal{PM}(X, \omega)$ and $\mathcal{I}(X, \omega)$ will be referred to as $\mathcal{PM}$ and $\mathcal{I}$, when there is no ambiguity on the considered instances. By construction of the ω-multiplied value graph, a perfect matching of $\mathcal{PM}$ corresponds to an instantiation of $\mathcal{I}$:

Proposition 3.1. *Let $\mu = \{(x_1, y_{j_1}^{k_1}), \dots, (x_{n+n'}, y_{j_{n+n'}}^{k_{n+n'}})\} \in \mathcal{PM}$, then $\iota = (y_{j_1}^{k_1}, \dots, y_{j_n}^{k_n})$ is an instantiation from $\mathcal{I}$.*

Proof. Let $\mu = (e_1, \dots, e_{n+n'}) \in \mathcal{PM}$. For each $x_i \in X \cup \overline{X}$, we write the edge e_i as $e_i = (x_i, y_{j_i}^{k_i})$. Let $\iota = (v_1, \dots, v_n)$, such that $\forall x_i \in X, v_i = y_{j_i}^{k_i}$.

We have $\forall x_i \in X, e_i = (x_i, y_{j_i}^{k_i}) \in \mathcal{PM} \subseteq E_\omega$, thus $v_i = y_{j_i}^{k_i} \in D_i$.

Also, for a $y_j \in Y$, we have $|\{e_i \in \mu | y_{j_i}^{k_i} = y_j\}| = \omega_j$, since every value nodes is covered once in μ and there are ω_j duplicated nodes for each value.

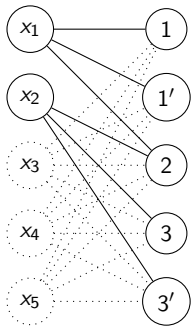
Then $\forall \mu' \subseteq \mu, |\{e_i \in \mu' | y_{j_i}^{k_i} = y_j\}| \leq \omega_j$ and, in particular, $|\{e_i \in \{e_1, \dots e_n\} | y_{j_i}^{k_i} = y_j\}| \leq \omega_j$. As $\forall x_i \in X, v_i = y_{j_i}^{k_i}$, we can conclude $\forall y_j \in Y, |\{v_i | v_i = y_j\}| \leq \omega_j$. Then $\iota \in \mathcal{I}$ □

Notation. *Let $\mu \in \mathcal{PM}$, we note $\pi(\mu) = \iota$ the corresponding instantiation as introduced above.*

Example 3.3 illustrates these definitions and properties:

Example 3.3. *Let $X = \{x_1, x_2\}, D_1 = \{1, 2\}, D_2 = \{2, 3\}$, $\omega = \{2, 1, 2\}$ and $\overline{X} = \{x_3, x_4, x_5\}$. Then we obtain the $\omega-$multiplied value graph in Figure 3.10a. We also list every instantiation of $\mathcal{I}$ in Figure 3.10b.*

$\{(x_1, 1'), (x_2, 2), (x_3, 3'), (x_4, 1), (x_5, 3)\}$ is a perfect matching that leads to the instantiation $\iota_1 = (1, 2)$. ι_1 is also reached by $\{(x_1, 1), (x_2, 2), (x_3, 3), (x_4, 1'), (x_5, 3')\}$ and $\{(x_1, 1'), (x_2, 2), (x_3, 3), (x_4, 3'), (x_5, 1)\}$.



(A) $\omega-$multiplied Value Graph

instantiations	x_1	x_2
ι_1	1	2
ι_2	1	3
ι_3	2	3

(B) List of every elements of $\mathcal{I}$

FIGURE 3.10: ω-multiplied value graph and list of every instantiation for Example 3.3

Several perfect matchings can lead to a same instantiation. The size of $\mathcal{I}$ and the size of $\mathcal{PM}$ are correlated. In the next section, we present how to compute an upper bound of $|\mathcal{I}|$.

3.3 Computation of an Upper Bound as a Minimisation Problem

This section introduces a correction for the bound proposed in Pesant et al. 2012 and recalled in Section 3.1. Then we propose an algorithmic analysis of this correction.

3.3.1 Correction of the Upper Bound

Basically, the approach is to count how many perfect matchings in $G(X, \omega)$ lead to a specific instantiation, over the true variables, $\iota \in \mathcal{I}$. However there is an exponential number of such instantiations, so we will find a lower bound of the number of such perfect matchings, by solving a non-linear minimization problem, in order to upper bound the number of instantiations in $\mathcal{I}$.

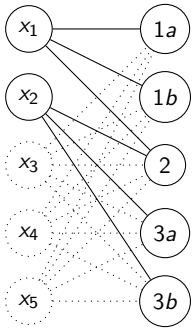
Proposition 3.2. *Let $\iota = (v_1, \dots, v_n)$ be an instantiation of the variables of X and $\forall y_j \in Y$, $c_j(\iota) = |\{i | y_{j_i} = y_j\}|$ be the number of occurrences of each value y_j in ι. There are $n'! \cdot \prod_{j=1}^{m} A_{c_j(\iota)}^{\omega_j}$ perfect matchings on $G(X, \omega)$ that lead to the instantiation ι, where $A_{c_j(\iota)}^{\omega_j}$ is the number of possible arrangements of $c_j(\iota)$ objects among ω_j.*

Proof. For each value $y_j \in Y$, there are $\binom{\omega_j}{c_j(\iota)}$ ways to pick up nodes in G to have as many occurrences of y_j as in ι and there are $c_j(\iota)!$ ways to order these nodes, then there are $\binom{\omega_j}{c_j(\iota)} \cdot c_j(\iota)! = A_{c_j(\iota)}^{\omega_j}$ ways to assign variables of X that are equal to y_j in the instantiation ι.

These choices are independent, thus this result can be extended to every value y_j: there are $\prod_{j \in \{1,\dots,m\}} A_{c_j(\iota)}^{\omega_j}$ ways to assign the variables in X to get the instantiation ι.

We need now to consider fake variables assignment. For each assignment of X leading to the instantiation ι, there are $n'!$ ways to assign every variable of $\overline{X}$, then, there are $n'! \cdot \prod_{j \in \{1,\dots,m\}} A_{c_j(\iota)}^{\omega_j}$ perfect matchings in $G(X, \omega)$, that lead to the instantiation ι. □

The following example illustrates Proposition 3.2.



(A) $\omega-$multiplied Value Graph

x_1	x_2	x_3	x_4	x_5
$1a$	2	$1b$	$3a$	$3b$
$1a$	2	$1b$	$3b$	$3a$
$1a$	2	$3a$	$1b$	$3b$
$1a$	2	$3a$	$3b$	$1b$
$1a$	2	$3b$	$1b$	$3a$
$1a$	2	$3b$	$3a$	$1b$
$1b$	2	$1a$	$3a$	$3b$
$1b$	2	$1a$	$3b$	$3a$
$1b$	2	$3a$	$1a$	$3b$
$1b$	2	$3a$	$3b$	$1a$
$1b$	2	$3b$	$1a$	$3a$
$1b$	2	$3b$	$3a$	$1a$

(B) List of perfect matchings corresponding to the solution $\{x_1 = 1, x_2 = 2\}$

FIGURE 3.11: ω-multiplied graph and list of perfect matchings for Example 3.4

Example 3.4. *Let $X = \{x_1, x_2\}, D_1 = \{1, 2\}, D_2 = \{2, 3\}$, and $\omega = \{2, 1, 2\}$, then we must add 3 fake variables, $\overline{X} = \{x_3, x_4, x_5\}$. We obtain the $\omega-$multiplied value graph in Figure 3.11a.*

Let us now consider this instantiation of the variables x_1 and x_2 of this problem: $\{x_1 = 1, x_2 = 2\}$, and let us call it ι, we want to count how many perfect matching there are in $G(X, \omega)$ that lead to this instantiation. The array in Figure 3.11b lists those perfect matchings.

Note that we can remove every symmetric perfect matching by permutation of the variables of $\overline{X}$. There are $3! = 6$ such permutations. After deleting these perfect matchings, there remain 2 possibilities: either $\{x_1 = 1a, x_2 = 2\}$ or $\{x_1 = 1b, x_2 = 2\}$. In general, we need to count every possibility there is to instantiate variables in X. In ι, we choose one value "1" among two, one value "2" among one and zero value "3" among two. Also the order in which we pick these values is important, then there are $3! \cdot A_1^2 \cdot A_1^1 \cdot A_0^2 = 12$ perfect matchings leading to the instantiation ι.

We can directly deduce the following corollary, when extending Proposition 3.2 to every instantiation of $\mathcal{I}$:

Corollary 3.1.

$$\Phi(G(X, \omega)) = n'! \cdot \sum_{\iota \in \mathcal{I}} \prod_{j=1}^{m} A_{c_j(\iota)}^{\omega_j} \tag{3.2}$$

The following proposition gives an upper bound for $|\mathcal{I}|$:

Proposition 3.3.

$$|\mathcal{I}| \leq \frac{\Phi(G(X, \omega))}{n'! \cdot \min\limits_{\iota \in \mathcal{I}}(\prod_{j=1}^{m} A_{c_j(\iota)}^{\omega_j})} \tag{3.3}$$

Proof. The proof follows directly from Corollary 3.1. □

Remark 3.1. *What suggested the bound developed by Pesant et al. (2012) is that*

$$|\mathcal{I}| \leq \frac{\Phi(G(X,\omega))}{n'! \cdot \prod_{j=1}^{m} \omega_j!}$$

We can check that we have, $\forall \iota \in \mathcal{I}$,

$$\prod_{j=1}^{m} A_{c_j(\iota)}^{\omega_j} \leq \prod_{j=1}^{m} \omega_j!$$

and then,

$$\frac{\Phi(G(X,\omega))}{n'! \cdot \min\limits_{\iota \in \mathcal{I}}(\prod_{j=1}^{m} A_{c_j(\iota)}^{\omega_j})} \geq \frac{\Phi(G(X,\omega))}{n'! \cdot \prod_{j=1}^{m} \omega_j!}$$

Our new objective is to compute $\min\limits_{\iota \in \mathcal{I}}(\prod_{j=1}^{m} A_{c_j(\iota)}^{\omega_j})$. In the following, $c_j(\iota)$ will be simply denoted as c_j. This problem can be modeled as minimization problem:

$$IP =: \begin{cases} min \ \prod_{j=1}^{m} \frac{\omega_j!}{(\omega_j - c_j)!} \\ s.t. \sum_{j=1}^{m} c_j = n \\ \forall j \in \{1, ..., m\}, c_j \in \{0, ..., \omega_j\} \end{cases} \tag{3.4}$$

Note that for a high ω_j, choosing a high c_j, makes the objective function rise much more than choosing a high c_j, for a small ω_j. Also choosing a small c_j for a high ω_j makes the objective function rise much more than choosing the same c_j for a smaller ω_j.

Example 3.5. *Let* $\omega_1 = 5$ *and* $\omega_2 = 2$ *and* $n = 4$, *then* $c_1 \in [0, 5]$ *and* $c_2 \in [0, 2]$, *then taking* $c_1 = 4$ *and* $c_2 = 1$, *for example,* $\frac{5!}{(5-4)!} = 120 > 2 = \frac{2!}{(2-2)!}$.
And taking $c_1 = 1$ *and* $c_2 = 1$, *we have* $\frac{5!}{(5-1)!} = 5 > 2 = \frac{2!}{(2-1)!}$
As we must satisfy $c_1 + c_2 = n = 4$, *it seems better to, first, assign the biggest value possible for* c_2 *and then assign the rest to* c_1. *We would have* $c_1 = 2$ *and* $c_2 = 2$ *and* $A_2^2 * A_2^5 = 40$, *which is the minimum.*

This reasoning is generalized in the next proposition.

Proposition 3.4. *If* $\omega_1 \leq ... \leq \omega_m$, *then* $c^* = (\omega_1, ..., \omega_{k-1}, c_k^*, 0, ..., 0)$ *is a minimum of IP, with* $c_k^* = n - \sum_{j=1}^{k-1} \omega_j$.

Proof. Let $\omega = (\omega_1, ..., \omega_m)$ and let consider that $\omega_1 \leq \cdots \leq \omega_m$ and

$$f =: \begin{cases} \mathbb{N}^m \to \mathbb{N} \\ c \mapsto \prod_{j=1}^{m} \frac{\omega_j!}{(\omega_j - c_j)!} \end{cases} \tag{3.5}$$

We want to minimize $f(c)$ such that $\sum_{j=1}^{m} c_j = n$. Then, ω being in ascending order, we want to prove that f reaches a global minimum for:

$$c^* = (\omega_1, ..., \omega_{k-1}, c_k^*, 0, ..., 0) \tag{3.6}$$

with $c_k^* \in \{0, ..., \omega_k\}$.

Let $c = (c_1, ..., c_m)$ be a vector such that $\sum_{j=1}^{m} c_j = n$. We want to prove that $f(c^*) \leq f(c)$. Let us rewrite c as a modification of c^*. We can only decrease (not strictly) c_j^* for $j \in \{1, ..., k-1\}$ and increase (not strictly) c_j^* for $j \in \{k+1, ..., m\}$. As

for c_k^*, there are two possibilities. We first deal with the case where c_k^* is increased. We rewrite c:

$$c = (\omega_1 - d_1, ..., \omega_{k-1} - d_{k-1}, c_k^* + u_k, u_{k+1}, ..., u_m) \tag{3.7}$$

with $\forall j \in \{1, ..., m\}, d_j \geq 0$ and $u_j \geq 0$ and $\sum_{j=1}^{m} u_j = \sum_{j=1}^{m} d_j$. We consider that $\forall j \in \{k, ..., m\}, d_j = 0$ and $\forall j \in \{1, ..., k-1\}, u_j = 0$. The last condition states that everything that has been removed, has also been added. Thus, we still have $\sum_{j=1}^{m} c_j = n$.

Now, we will see a method to transform vector c^* to c in several steps such that, at every step, the function f increases. Let c^j be the vector at step j.

For each $j \in \{1, k-1\}$, we remove d_j from c_j^j, and we redistribute on the variables of the end, that need to be increased, starting redistributing from index k to m. Let us take an example:

Let us consider a vector $\omega = (2, 3, 4, 4, 7)$ and $n = 7$. Then, let $c^* = (2, 3, 2, 0, 0)$ and $c = (0, 1, 3, 1, 2)$. Here are the steps following the method to get c from c^*:

- Step 0: $c^0 = c^* = (2, 3, 2, 0, 0)$
- Step 1: $c^1 = (0, 3, 3, 1, 0)$, we remove 2 from c_1^0 and we add 1 to c_3^0 and 1 to c_4^0.
- Step 2: $c^2 = c = (0, 1, 3, 1, 2)$, we remove 2 from c_2^1 and we add 2 to c_5^1.

In a general way, we have:

- Step 0: $c^0 = c^* = (\omega_1, ..., \omega_{k-1}, c_k^*, 0, ..., 0)$.
- ...
- Step j: $c^j = (\omega_1 - d_1, ..., \omega_j - d_j, \omega_{j+1}, ..., \omega_{k-1}, c_k^* + u_k, u_{k+1}, ..., u_{q-1}, r_q, 0, ..., 0)$.
- ...
- Step k-1 : $c^{k-1} = c = (\omega_1 - d_1, ..., \omega_{k-1} - d_{k-1}, c_k^* + u_k, u_{k+1}, ..., u_m)$.

with $r_q \in \{1, ..., u_q\}$ (or $\{x_k^*, ..., x_k^* + u_k\}$, if $q = k$), which is the residue for a particular index q. This index does not necessarily increase from one step j to the following one, as d_j may be smaller than $u_q - r_q$.

The objective, now, is to prove that $f(c^j) \leq f(c^{j+1}), \forall j \in \{1, ..., k-2\}$. We have:

$$c^j = (\omega_1 - d_1, ..., \omega_j - d_j, \omega_{j+1}, ..., \omega_{k-1}, c_k^* + u_k, u_{k+1}, ..., u_{q_a - 1}, r_{q_a}, 0, ..., 0)$$

and

$$c^{j+1} = (\omega_1 - d_1, ..., \omega_{j+1} - d_{j+1}, \omega_{j+2} ..., \omega_{k-1}, c_k^* + u_k, u_{k+1}, ..., u_{q_b - 1}, r_{q_b}, 0, ..., 0)$$

with $q_b \geq q_a$.

Now, we study $\frac{f(c^j)}{f(c^{j+1})}$.

First case: $q_a < q_b$

$$\begin{aligned}\frac{f(c^j)}{f(c^{j+1})} &= \frac{\omega_{j+1}!}{\frac{\omega_{j+1}!}{d_{j+1}!}} \cdot \frac{\frac{\omega_{q_a}!}{(\omega_{q_a}-r_{q_a})!}}{\frac{\omega_{q_a}!}{(\omega_{q_a}-u_{q_a})!}} \cdot \frac{1}{\frac{\omega_{q_b}!}{(\omega_{q_b}-r_{q_b})!}} \\ &= d_{j+1}! \cdot \frac{(\omega_{q_a}-u_{q_a})!}{(\omega_{q_a}-r_{q_a})!} \cdot \frac{(\omega_{q_b}-r_{q_b})!}{\omega_{q_b}!} \\ &= \frac{(d_{j+1}-r_{q_b})! \cdot (d_{j+1}-r_{q_b}+1) \cdot \ldots \cdot d_{j+1}}{(\omega_{q_a}-r_{q_a}) \cdot \ldots \cdot (\omega_{q_a}-u_{q_a}+1) \cdot \omega_{q_b} \cdot \ldots \cdot (\omega_{q_b}-r_{q_b}+1)}\end{aligned}$$

And, $(d_{j+1}-r_{q_b}+1) \cdot \ldots \cdot d_{j+1}$ and $\omega_{q_b} \cdot \ldots \cdot (\omega_{q_b}-r_{q_b}+1)$ are products of r_{q_b} consecutive terms. Moreover, $\omega_{q_b} \geq \omega_{j+1} \geq d_{j+1}$, then:

$$\frac{(d_{j+1}-r_{q_b}+1) \cdot \ldots \cdot d_{j+1}}{\omega_{q_b} \cdot \ldots \cdot (\omega_{q_b}-r_{q_b}+1)} \leq 1$$

We can notice that, in the case where redistributing d_{j+1} makes the index of the residue grow, $d_{j+1} = (u_{q_a}-r_{q_a})+r_{q_b}$. In other words, what has been removed has been re-added. Then $(d_{j+1}-r_{q_b})!$ is the product of $u_{q_a}-r_{q_a}$ consecutive terms, the same as $(\omega_{q_a}-r_{q_a}) \cdot \ldots \cdot (\omega_{q_a}-u_{q_a}+1)$. We also know that $\omega_{q_a} \geq u_{q_a}$, then $d_{j+1}-r_{q_b} = u_{q_a}-r_{q_a} \leq \omega_{q_a}-r_{q_a}$, then:

$$\frac{(d_{j+1}-r_{q_b})!}{(\omega_{q_a}-r_{q_a}) \cdot \ldots \cdot (\omega_{q_a}-u_{q_a}+1)} \leq 1$$

Then,

$$\frac{f(c^j)}{f(c^{j+1})} \leq 1$$

Second case: $q_a = q_b$ We consider here the case where the index of the residue remains unchanged, but this does not mean that the residue is the same for c^j and c^{j+1}. Then we have $q_b = q_a$ and $r_{q_b} \geq r_{q_a}$. But we have $\omega_{q_a} = \omega_{q_b}$.

$$\begin{aligned}\frac{f(c^j)}{f(c^{j+1})} &= d_{j+1}! \cdot \frac{\frac{\omega_{q_a}!}{(\omega_{q_a}-r_{q_a})!}}{\frac{\omega_{q_b}!}{(\omega_{q_b}-r_{q_b})!}} \\ &= d_{j+1}! \cdot \frac{(\omega_{q_b}-r_{q_b})!}{(\omega_{q_a}-r_{q_a})!} \\ &= \frac{d_{j+1}!}{(\omega_{q_a}-r_{q_a}) \cdot \ldots \cdot (\omega_{q_a}-r_{q_b}+1)}\end{aligned}$$

What has been removed, has also been re-added: $d_{j+1} = r_{q_b}-r_{q_a}$ then $d_{j+1}!$ is a product of $r_{q_b}-r_{q_a}$ consecutive terms, like the product $(\omega_{q_a}-r_{q_a}) \cdot \ldots \cdot (\omega_{q_a}-r_{q_b}+1)$. Moreover, $d_{j+1} = r_{q_b}-r_{q_a} \leq \omega_{q_a}-r_{q_a}$. Thus we have,

$$\frac{f(c^j)}{f(c^{j+1})} \leq 1$$

We have proved that, $\forall j \in \{1, \ldots, k-2\}$, $f(c^j) \leq f(c^{j+1})$. Then,

$$f(c^*) = f(c^0) \leq f(c^1) \leq \ldots \leq f(c^{k-1}) = f(c)$$

There remains one case to deal with. We have proved that $f(c^*) \leq f(c)$, for every c such that $c_k \geq c^*_k$. In the case where $c_k \leq c^*_k$, we need to adapt the method described above. In the previous case, we started removing d_j for j going from 1 to $k-1$ in that order. In this case, we first remove d_k and only, then we remove d_j for j going from 1 to $k-1$, as before. We will not detail the calculations for this case as they are very similar.

It follows that $f(c^*) \leq f(c)$ for every c such that $\sum_{j=1}^{m} c_j = n$ and $\forall j \in \{1, ..., m\}, 0 \leq c_j \leq \omega_j$. Then c^* is a minimum of our problem. □

From this proposition, we can deduce a method to get an upper bound of $\mathcal{I}$. We first sort the occurrences vector ω, then we can compute the vector c^* and compute $\min_{\iota \in \mathcal{I}}(\prod_{j=1}^{m} A^{\omega_j}_{c_j(\iota)}) = \prod_{j=1}^{m} A^{\omega_j}_{c^*_j}$. Then we get an upper bound of $|\mathcal{I}|$.

Proposition 3.5. *If $\omega_1 \leq ... \leq \omega_m$, then*

$$|\mathcal{I}| \leq UB^{IP}(X, \omega) = \frac{\Phi(G(X, \omega))}{n'! \cdot \prod_{j=1}^{m} A^{\omega_j}_{c^*_j}} \tag{3.8}$$

We have now corrected the method presented in Section 3.1. We can use this new result to upper bound correctly the Residual Upper Bound Graphs. This new upper bound requires sorting ω and in practice, we will use the Bregman-Minc or Liang-Bai upper bound instead of computing $\Phi(G(X, \omega))$. We also consider that we can compute the factorial function in constant time. This bound is then polynomially computable. A time-complexity study is given in subsection 3.3.2.

3.3.2 Computation and Complexity

In this subsection we present Algorithm 4, which details how to proceed to compute UB^{IP}. Then a time-complexity study is given.

Algorithm 4: *computeUB*()

Data: X, D, Y, ω
Result: an upper bound of $\Phi(G(X, \omega))$

```
1  n' ← Σ_{j=1}^{m} ω_j − n;
2  ω_sorted ← ω.sortAsc();
3  c* ← zeros(m);
4  count ← n;
5  idx ← 0;
6  while count > 0 do
7      if count > ω_sorted(idx) then
8          count ← count − ω_sorted(idx);
9          c*(idx) ← ω_sorted(idx);
10     else
11         c*(idx) ← ω_sorted(idx) − count;
12         count ← 0;
13     end
14     idx ← idx + 1;
15 end
16 nbPM ← min(UB^BM(G(X, ω)), UB^LB(G(X, ω)));
17 return nbPM/(n'! · Π_{j=1}^{m} A^{ω_j}_{c*_j});
```

At Line 1 we compute n' the number of fake variables. At Line 2 we sort the ω vector in ascending order. We have a function $zeros(m)$ that builds a vector of size m filled with 0 in constant time. From Line 4 to Line 15 we simply fill c^*, such as it is defined. At Line 16 we compute an Bregman-Minc upper bound thanks to a function, that can compute it in $\mathcal{O}(n+n')$ operations. Finally at Line 17 we return $UB^{IP}(X,\omega)$. We consider that we can compute the number of arrangements and the factorial function in a constant time. We can conclude on the time-complexity for the computation of UB^{IP}:

Proposition 3.6. *The computation of UB^{IP} has a time-complexity in*

$$\mathcal{O}(n+n'+m\cdot\log(m))$$

Proof. Computing n' and $\prod_{j=1}^{m} A_{c_j^*}^{\omega_j}$ requires $\mathcal{O}(m)$ operations. Also filling c^* requires $\mathcal{O}(m)$ operations (the worst case being when we have to fill c^* entirely). Sorting the ω vector requires $\mathcal{O}(m\cdot\log(m))$. Computing $nbPM$ requires to compute a product over $n+n'$ factors, one for each variable node in $G(X,\omega)$, then it requires $\mathcal{O}(n+n')$ operations. Thus, we have a time-complexity in $\mathcal{O}(n+n'+m\cdot\log(m))$ □

Computing UB^{IP} is linear in the number of variables and quasilinear in the number of values. In Section 3.4, we test the efficiency of the corrected bound within Counting-Based search strategy: maxSD (Pesant et al. 2012).

3.4 Evaluation of the Upper Bound within `maxSD` Strategy

Even though the bound proposed by Pesant et al. (2012) is incorrect it still provides some estimate of the number of solutions for a *gcc* instance and may therefore still be used in a counting-based search heuristic. For instance, the *maxSD* heuristic may make the same choices using the corrected upper bound and the previous wrong one. In this section, we first compare the variable/value ordering given by these two estimators and, then, we compare their efficiency within the *maxSD* heuristic.

3.4.1 Impact of the New Upper Bound on the Variable/Value Ordering

In this subsection, we compare the instantiations order given by the former bound (the PQZ bound) and its correction. In Example 3.6, we compute, for a given instance of a *gcc*, the density of solutions for each possible instantiation, for the PQZ bound, its correction and the exact computation. We show that changing the way to estimate the number of solutions has a big influence on the variable/value ordering.

Example 3.6. *For this example, we randomly pick an instance of gcc, whose Value Graph is represented in Figure 3.12, with $n = 10$ variables, $m = 10$ values and an edge density $p = 0,3$, which means that each edge between a variable and a value has a probability $p = 0,3$ to exist. For each value, we also pick uniformly randomly a possible interval $l_j - u_j$, considering the number of neighbors of each value node in the Value Graph. These intervals are represented on the right of each corresponding value node.*

For each non-instantiated variable x_i, and each value $y_j \in D_i$, we propagate the instantiation $x_i \rightarrow y_j$ and we compute an estimation of the number of remaining solutions, with the previous bound (PQZ bound) and the corrected one (we also compute the exact number of remaining solutions). From these estimations, we can compute the solution density for each instantiation.

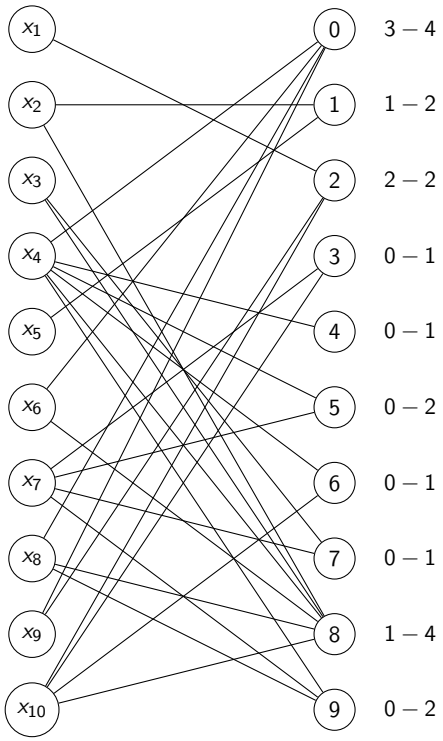


FIGURE 3.12: A random instance of *gcc*

The heuristic maxSD chooses first the instantiation associated with the highest solution density. With any of the estimators, the maxSD heuristic would choose first the instantiation $x_6 \rightarrow 0$. *We sort the instantiations by their solution density in descending order for the three estimators. If two instantiations have the same solution density, then we sort it with the lexicographic order.*

- *PQZ bound:* ($x_6 \rightarrow 0$, $x_8 \rightarrow 0$, $x_4 \rightarrow 0$, $x_{10} \rightarrow 2$, $x_9 \rightarrow 0$, $x_3 \rightarrow 8$, $x_2 \rightarrow 8$, $x_7 \rightarrow 9$, $x_2 \rightarrow 1$, $x_3 \rightarrow 7$, $x_9 \rightarrow 2$, $x_7 \rightarrow 3$, $x_7 \rightarrow 5$, $x_{10} \rightarrow 6$, $x_{10} \rightarrow 8$, $x_{10} \rightarrow 3$, $x_8 \rightarrow 9$, $x_8 \rightarrow 8$, $x_4 \rightarrow 4$, $x_4 \rightarrow 9$, $x_4 \rightarrow 6$, $x_4 \rightarrow 5$, $x_6 \rightarrow 8$, $x_4 \rightarrow 8$, $x_7 \rightarrow 7$)
- *Corrected bound:* ($x_6 \rightarrow 0$, $x_8 \rightarrow 0$, $x_{10} \rightarrow 2$, $x_4 \rightarrow 0$, $x_9 \rightarrow 0$, $x_3 \rightarrow 8$, $x_2 \rightarrow 8$, $x_7 \rightarrow 3$, $x_7 \rightarrow 5$, $x_2 \rightarrow 1$, $x_3 \rightarrow 7$, $x_7 \rightarrow 9$, $x_7 \rightarrow 7$, $x_9 \rightarrow 2$, $x_4 \rightarrow 8$, $x_{10} \rightarrow 8$, $x_4 \rightarrow 4$, $x_4 \rightarrow 5$, $x_4 \rightarrow 6$, $x_4 \rightarrow 9$, $x_{10} \rightarrow 6$, $x_8 \rightarrow 8$, $x_6 \rightarrow 8$, $x_{10} \rightarrow 3$, $x_8 \rightarrow 9$)
- *Exact computation:* ($x_6 \rightarrow 0$, $x_8 \rightarrow 0$, $x_{10} \rightarrow 2$, $x_9 \rightarrow 0$, $x_3 \rightarrow 8$, $x_4 \rightarrow 0$, $x_2 \rightarrow 8$, $x_2 \rightarrow 1$, $x_3 \rightarrow 7$, $x_7 \rightarrow 9$, $x_7 \rightarrow 5$, $x_7 \rightarrow 3$, $x_9 \rightarrow 2$, $x_7 \rightarrow 7$, $x_8 \rightarrow 8$, $x_6 \rightarrow 8$, $x_4 \rightarrow 8$, $x_{10} \rightarrow 8$, $x_4 \rightarrow 6$, $x_8 \rightarrow 9$, $x_4 \rightarrow 5$, $x_4 \rightarrow 4$, $x_{10} \rightarrow 6$, $x_4 \rightarrow 9$, $x_{10} \rightarrow 3$)

We can see that the variable/value ordering is different depending on the estimator. The two first instantiations are identical. The length of the common prefix of these three ordering is 2. In the following, we will formalize this measure of the (dis)similarity of variable/value ordering, as a way to distinguish heuristics.

In order to measure how different is the behavior of these estimators within *maxSD*, we will use two measures of the similarity between two variable/value orderings:

- $m_{LP}(l_1, l_2)$: the length of the common prefix between the variable/value ordering l_1 and l_2. The first instantiations are more important. We want to measure

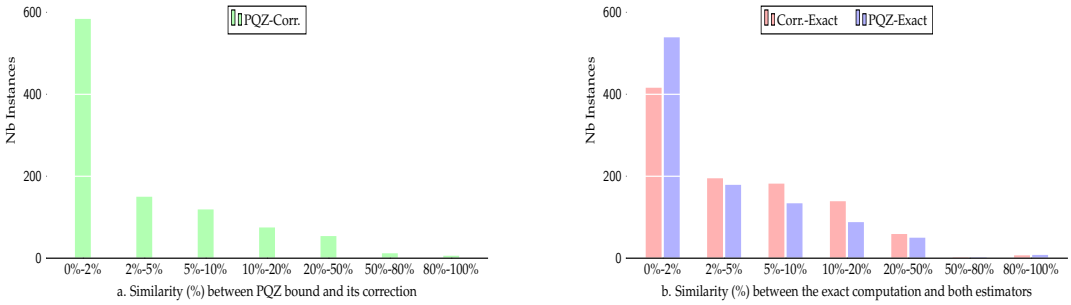


FIGURE 3.13: Proportion of instances per percentage of similarity according to m_{LP}

how many choices will be common in a row at the beginning. This measure is then normalized, dividing it by n.

- $m_{WS}(I_1, I_2)$: the weighted sum of common elements between the variable/value ordering I_1 and I_2. It measures the number of common elements and favor the common elements that appear in the beginning . The weight of the k^{th} element among p instantiations is $p - k + 1$.

  Let $I_1 = (\iota_1^1, \dots, \iota_p^1)$, $I_2 = (\iota_1^2, \dots, \iota_p^2)$ and χ, the function such that $\chi(u, v) = 1$ if $u = v$ and $\chi(u, v) = 0$ otherwise. Then:

$$m_{WS}(I_1, I_2) = \frac{\sum_{k=1}^{p}(p - k + 1) \cdot \chi(\iota_k^1, \iota_k^2)}{\frac{1}{2} \cdot p \cdot (p + 1)}$$

  The denominator is here to normalize the measure, so we can make the comparison between the two estimators on different instances of *gcc*.

Now that we have seen how to compare variable/value orderings, we generate randomly 1000 instances of feasible *gcc* and we will compare the orderings given by the PQZ estimator, its correction and the exact computation. To generate those instances, we will apply the same method we used for the generation of the instance presented in Example 3.6 with the same parameters, $n = 10$, $m = 10$ and $p = 0,3$.

Figure 3.13a shows the number of instances as a function of the percentage of similarity between the resulting variable/value orderings given by the PQZ bound and its correction, according to the length of common prefix measure. We can notice that almost 600 among the 1000 generated instances, lead to less than 2% similarity. More than 90% of the instances are less than 20% similar. The *PQZ* bound and its correction very often lead to different orderings. Figure 3.13b shows the number of instances in function of the percentage of similarity between the resulting orderings given by the exact computation and the two other estimators, according to the length of common prefix measure. The corrected bound is similar at less than 2% to the exact bound for 58.9% of all instances, and the PQZ bound for only 51.6%. In general, the corrected bound gives more accurate variable/value ordering.

In Figure 3.14a, we represent the number of instances as a function of the percentage of similarity according to the weighted sum of common elements measure. With the m_{WS} measure, only about 100 instances have more than 50% similarity between the PQZ bound and its correction. The variable/value orderings remain quite different.

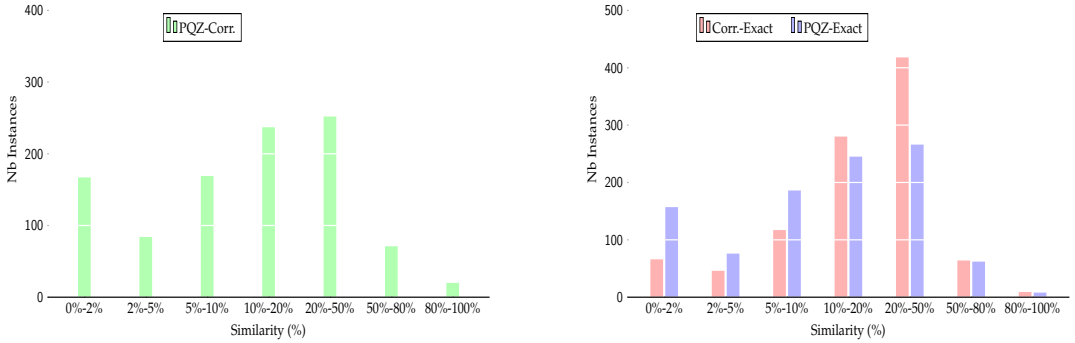


FIGURE 3.14: Proportion of instances per percentage of similarity according to m_{WS}

In Figure 3.14b is represented the similarity between the exact computation and the two estimators according to the weighted sum of common elements measure. Though the m_{WS} is a much more flexible measure, there are still more than 100 instances that remain below 50% of similarity for both estimators. The correction seems to give closer variable/value orderings than the PQZ estimator, as there are about 100 instances more that are less than 2% similar for the PQZ estimator and about 150 instances more that are between 20% and 50% similar for its correction.

Also, *maxSD* is a strategy that chooses the assignment with the highest density. We notice that, for 42.5% of these 1000 instances the PQZ estimator and its correction would choose the same first decision. For 59.3% of the instances, the exact computation and the corrected estimator would choose the same first decision against 47.0% for the exact computation and the PQZ estimator. And for 32.6% of the instances, the three estimators would choose the same first decision.

In this subsection, we have shown how to compare the influence of different number of solutions estimators within *maxSD* on the ordering, and we have shown that the PQZ bound and its correction behave in a different way.

3.4.2 Performance Analysis on Generated Instances

In this subsection we analyze the performance of each estimator within the *maxSD* strategy. We generate random instances of varying degree of difficulty, on which we run the two variants of the *maxSD* strategy. We first explain the generation process of such random instances and then we analyze the results obtained.

Generation of random CSPs. Let n be the number of variables, m the number of values and p, the edge density of the Value Graph. Each value $y_j \in Y$ have a probability p to be in D_i, for each variable x_i. Then, we add n_C global cardinality constraints to this CSP. The scope of each *gcc* is chosen randomly: each variable has a 50% chance to be picked. We are also given a tightness τ, which defines the length of every occurrence interval of each value. Let $\{x_{i_1}, \dots, x_{i_q}\}$ be the scope of one *gcc*, then the length of every occurrence interval of this *gcc* is $\tau \cdot q$, rounded to the nearest integer. Once the lenght is set, the interval is chosen randomly among every possible interval with such a length. The generated *gcc* are not trivially unsatisfiable, as we also ensure that the generated occurrence intervals, $[l_j, u_j]$, are such that:

$$\sum_{j=1}^{m} l_j \leq q \leq \sum_{j=1}^{m} u_j$$

$p \backslash n_C$	2	3	4	5	6	7	8	9	10
$p = 1.0$	492/**493**	481/**482**	**455**/446	**433**/408	**424**/414	**389**/369	**383**/362	**400**/380	**413**/394
$p = 0.66$	500/500	495/495	**491**/489	487/**489**	475/**477**	468/**474**	460/**472**	466/**479**	462/**479**
$p = 0.33$	500/500	500/500	500/500	499/**500**	499/**500**	500/500	500/500	500/500	499/**500**

TABLE 3.1: Number of solved instances for both estimators (PQZ/Corr.) as a function of n_C for $p \in \{0.33, 0.66, 1.0\}$

$p \backslash n_C$	2	3	4	5	6	7	8	9	10
$p = 1.0$	492/**493**	481/**482**	**455**/446	**433**/408	**424**/414	**389**/369	**383**/362	**400**/380	**413**/394
$p = 0.66$	500/500	495/495	**491**/489	487/**489**	475/**477**	468/**474**	460/**472**	466/**479**	462/**479**
$p = 0.33$	500/500	500/500	500/500	499/**500**	499/**500**	500/500	500/500	500/500	499/**500**

TABLE 3.2: Number of solved instances for both estimators (PQZ/Corr.) as a function of n_C for $p \in \{0.33, 0.66, 1.0\}$

We also ensure that any scope of a *gcc* is not included in another *gcc*. Indeed, two such *gcc* can be considered as one *gcc*, in which each occurrence interval is the intersection of two occurrence interval. We have thus exactly n_C disjoint global cardinality constraints in the generated *CSP*. This is an important point as we want the number of *gcc* and the tightness to be the two parameters that will directly affect the difficulty of the random instance.

Results Analysis. We generated several random CSP, with the method described above with the following parameters : $n = 20, m = 20, \tau \in [0.1, 1]$ by steps of 0.1, an edge density $p \in \{0.33, 0.66, 1.0\}$ and $n_C \in [2, 10]$ by steps of 1. For each couple (τ, n_C), we generate 50 random instances for a total of 4500 instances. We solve each instance with two different strategies: *maxSD* with the PQZ bound and *maxSD* with the corrected bound. The instances and the strategies are implemented in CHOCO solver (Prud'homme, Fages, and Lorca 2016) and we set, for each resolution, the time limit to 1 min and run on a 2.2GHz Intel Core i7 with 2.048GB.

The *maxSD* heuristic proposes to compute an estimator of the solution density, for each possible remaining instantiation, each time a fixed point is reached. This is a very costly strategy. Here, we will only compute and store the solution densities in the beginning of the search and each time we reach a fixed point, we will refer to them and choose the instantiation that led to the highest solution density at the beginning of the search and that is still available. We also thought about updating the solution densities each time the size of the domains have decreased enough (have reached a certain threshold), as proposed by Gagnon et al. (2018). This strategy is not effective on our generated problem, so we have preferred not to run it to gain more time.

Table 3.1 (resp. Table 3.2) shows the number of solved instances for both estimators for $\tau \in \{0.1, \dots, 1.0\}$ (resp. $n_C \in \{2 \dots, 10\}$) and $p \in \{0.33, 0.66, 1.0\}$. The hardest instances seems to be for $n_C = 8$ and $\tau = 0.9$. For $p = 1.0$, the PQZ estimator solved more instances: 3870, against 3748 for its correction. For $p = 0.66$, the corrected bound get better results: 4354, against 4304 for PQZ. As for the edge density $p = 0.33$, the corrected bound solved every instance, while the PQZ estimator missed 3 of them. When the edge density is very low ($p \leq 0.33$), the domains are sparser, so there are much fewer instantiations to test during the computation of the estimators. This is why almost all these instances are solved in less than 1min. One estimator does not perform better than the other in general. To conclude on these tables, when the Value Graph is complete, PQZ estimator gives slightly better results, and when the domains are not uniform, the corrected bound solves more instances.

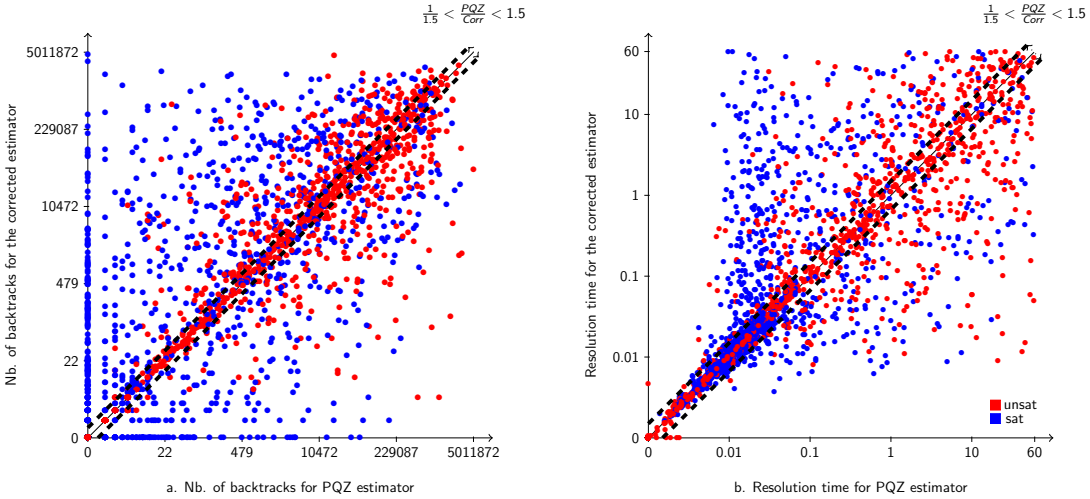


FIGURE 3.15: Comparison of the required number of backtracks and resolution time with p=1.0

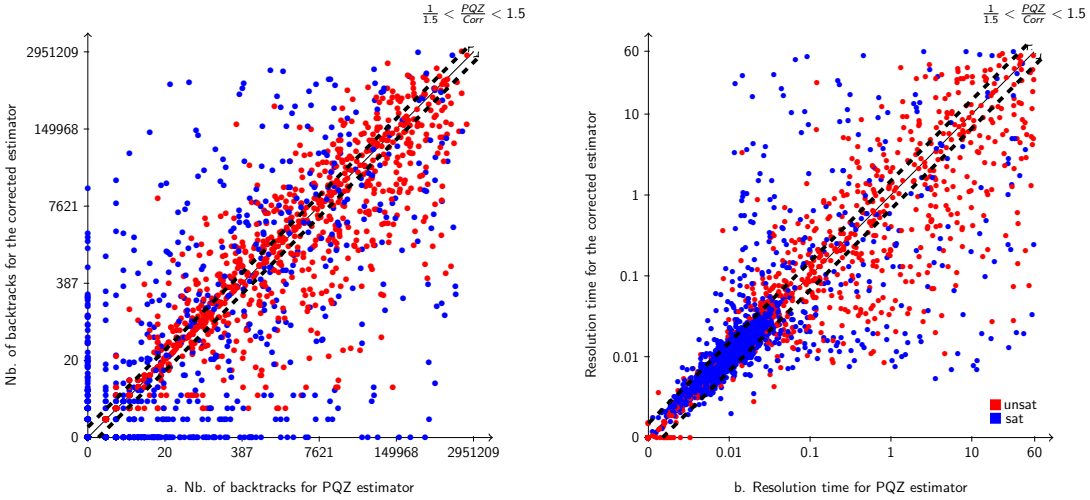


FIGURE 3.16: Comparison of the required number of backtracks and resolution time with p=0.66

It seems that the correction is able to catch more the difference among the domains than its previous version.

We now focus only on the instances that have been solved by both estimators. In Figure 3.15, Figure 3.16 and 3.17, we compare the required number of backtracks and the resolution time for each instance that have been solved (proved satisfiable or unsatisfiable) with both estimators for the edge density $p = 1.0$, $p = 0.66$ and $p = 0.33$. Blue dots represent the satisfiable instances and red ones represent the unsatisfiable instances. The area delimited by the black dotted lines represent the instances for which the ratio between the required number of backtracks (or resolution time) for PQZ and its correction is between $\frac{1}{1.5}$ and 1.5. In other words, we consider that, for the instances inside this area, both estimators have very comparable performances. If a point is below this area, it means that the corrected estimator has better performance, and on the opposite, if a point is above the area, the PQZ estimator has better performance.

For $p = 1.0$, 80.2% of the instances have been solved by both estimator. On Figure 3.15a, 60.1% of solved instances by both estimators are inside the dotted area.

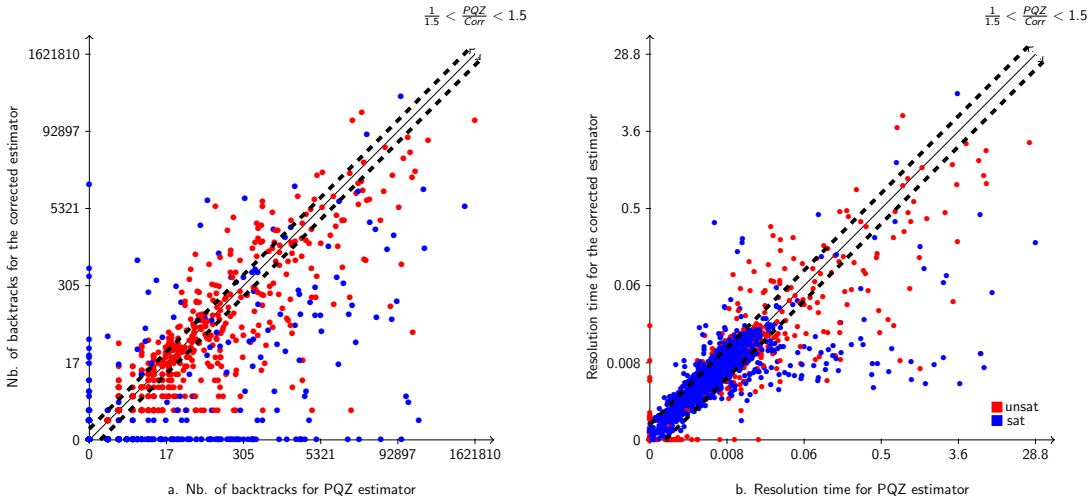


FIGURE 3.17: Comparison of the required number of backtracks and resolution time with p=0.33

The performance, in terms of number of backtracks, of both estimators on theses instances are very similar. There are 16.1% points below the area and 23.8% points above. The PQZ estimator seems slightly better on these instances. More precisely, 67.6% of unsatisfiable instances are inside the dotted area, 15.4% below and 17.1% above. As for satisfied instances, there are 54.1% blue points inside the dotted area, 16.8% below and 29.2% above. For unsatisfiable instances, the two estimators have equivalent performance, but the PQZ estimator seems to have better performance on these satisfiable instances. If we observe Figure 3.15b, 70.6% of red points and 70.4% of blue points are inside the dotted area. Also, 10.1% of blue points are below the area and 19.3% are above and 14.2% of red points are below and 15.4% are above. In terms of resolution time, both estimators have almost equivalent performance, but again, the PQZ estimator is slightly better on satisfiable instances.

For $p = 0.66$, 94.9% of the instances have been solved by both estimator. On Figure 3.16a, 71.2% of solved instances by both estimators are inside the dotted area. The performance, in terms of number of backtracks, of both estimators on theses instances are again very similar, even more for the edge density. There 16.1% points below the area and 12.7% points above. Here, the corrected estimator seems to have slightly better performance. There is no clear difference here in the behaviour of unsatisfiable and satisfiable instances: 17.0% of blue points and 15.2% of red points below the dotted area and 13.3% of blue points and 12.0% of red points above. On Figure 3.16b, 77.7% of red points and 78.1% of blue points are inside the dotted area. Also, 10.1% of the instances are above the area and 11.8% are below. In terms of resolution time, both estimators have very similar performance.

For $p = 0.33$, 99.9% of the instances have been solved by both estimators. On Figure 3.17a, 86.7% of solved instances by both estimators are inside the dotted area. For most instances, the performance of both estimators is more similar as the density gets higher. There are 9.7% of the instances below the area and 3.5% above. The correction performs a little better on this density too. On Figure 3.17b, we observe that there are a little more points in the bottom part of the plot. It seems like the lower the edge density, the better are the performance of the correction over PQZ estimator.

Our conclusions on this benchmark are the following. As a preliminary remark, both heuristics are meant to tune the search toward areas with many solutions (or a

high solution density for the *gcc* constraints). In practice, we observe on the benchmark that both heuristics roughly behave the same on unsatisfiable instances, with both a number of backtracks and a resolution time with a similar ratio. In general, we observe that the PQZ heuristic is slightly better than the corrected heuristic for the edge density $p = 1.0$, and for lower density, the corrected heuristic performs a bit better. Our assumption is that the corrected estimator is a little more able to catch the heterogeneity of the domains. As proved in the previous sections, the PQZ is not based on a bound, but our experiments show that in practice it can be considered as an estimator of the solution density. In practice, on our benchmark, this estimator seems to be useful to guide the search.

In conclusion, on all these experiments, there is no clear dominance of the PQZ estimator over our upper bound, when they are used inside a *maxSD* heuristic, in terms of solving efficiency. The number of solved instances are quite distributed around the bisector. Nevertheless, we observe that there is a clear difference between them, when they are used to find an variable/value ordering. In addition, we also observed that the bug in the PQZ bound indeed happens, in which case the presumed upper bound gives a value below the exact number of solutions. In the end, the benchmark is conclusive on this precise matter: the PQZ estimator and our Corrected bound are not equivalent, although using them in *maxSD* does not reveal this on the solving time.

3.5 Conclusion

This chapter revisited the solution-counting strategies for the well-known global cardinality constraint. It first highlights that in the computation of the initial result provided by Pesant et al. (2012), a solution can be counted several times and, thus, the alleged upper bound is not an upper bound. We then provide a correct calculation of an upper bound of the number of solutions for a global cardinality constraint. We build a benchmark for the `gcc` constraint with an original method, which had not been done in the previous work, and we compare heuristics based on both quantities (the previous calculation by Pesant et al. and our upper bound) on this benchmark. We show that the two estimators lead to different choices within counting-based strategies. Finally, we solve a number of instances with both heuristics and compare the results. In practice, neither heuristic dominates the other: the counting-based heuristics used with the estimator from Pesant et al. (2012) performs slightly better on instances with a high density of edges, *i.e.* instances where the domains are rather large and of comparable sizes. Our bound performs slightly better on instances with a lower density of edges, *i.e.* instances with more heterogeneous domains.

Though the former result is not an upper bound, it can now be seen as an estimator that can be used to guide the search, as an alternative way. That is probably why the error was difficult to spot. Our assumption is the following: when an estimator/bound is used inside a counting-based heuristic, the correlation between the estimator and the actual number of solutions is much more important that the estimator accuracy. Now that we do have an upper-bound of the number of solutions, new questions can be investigated: how do different estimators compare to this upper bound? Is it possible to use our methodology (both on the calculation techniques and the benchmark generation) to introduce average-case estimators, or estimators based on probabilistic approaches? Some of these questions will be investigated in the next chapters.

In Chapter 4, we tackle the problem of counting solutions on global cardinality constraints through the scope of Group Theory.

Chapter 4

A Group Theory Approach for the Global Cardinality Constraint

In Chapter 3, we have seen that several matchings in the Residual Upper Bound Graph can lead to a same instantiation of the variables, because of the duplicated value nodes. The difficulty comes from the fact that, for two different instantiations, the number of corresponding perfect matchings might be different. We need to quantify this redundancy in order to evaluate properly the real number of instantiations. We proposed a direct mathematical approach to upper bound the number of instantiation in Section 3.3. In this chapter, we study this problem from the point of view of Group Theory and we propose a second upper bound of the number of solutions of a `gcc` instance.

The Burnside's lemma (Burnside 2012) is a result from Group Theory used when counting elements of a set considering symmetries. It can be used, for example, to enumerate every combination of a Rubik's Cube. Here, we use this lemma to count every instantiation, considering the duplicated values symmetry and the fake variables symmetry in $\omega-$multiplied graph.

4.1 Group Theory Material

In this section, several notations and definitions are introduced and we present the Burnside's lemma. Unless specified, $\mathcal{A}$ is a set and $\mathcal{G}$ is a group acting on $\mathcal{A}$. The following definitions can be found in *Groups and Geometry* (1994).

Definition 4.1. *Let $a \in \mathcal{A}$. The orbit of a under the action of $\mathcal{G}$ is the set $\mathcal{O}_{\mathcal{G}}(a) = \{g \cdot a | g \in \mathcal{G}\}$. We note $\mathcal{A}/\mathcal{G}$, the set of orbits of $\mathcal{A}$ under the action of $\mathcal{G}$.*

The orbit of an element $a \in \mathcal{A}$ under the action of $\mathcal{G}$ is not to be mistaken with the orbit of a under the action of $g \in \mathcal{G}$, which is defined as follows:

Definition 4.2. *Let $a \in \mathcal{A}$. The orbit of a under the action of $g \in \mathcal{G}$ is the set $\mathcal{O}_g(a) = \{a, g \cdot a, g^2 \cdot a, ...\} = \{g^q \cdot a | \forall q \in \mathbb{N}\}$. When $\mathcal{A}$ is finite, $\mathcal{O}_g(a)$ necessarily describes a cycle, and $g^{|\mathcal{O}_g(a)|} \cdot a = a$. The order of this cycle is $|\mathcal{O}_g(a)|$.*

Notation. *If a cycle in $g \in \mathcal{G}$ has an order equal to r, it is noted a r-cycle and $cyc_r(g)$ is the number of $r-$cycles in g.*

Definition 4.3. *Let $g \in \mathcal{G}$, we note $\mathcal{A}^g = \{a \in \mathcal{A} | g \cdot a = a\}$ the invariant set of $\mathcal{A}$ by g, which is the set of elements of $\mathcal{A}$ that are fixed by g.*

In the following, the term "invariant" will only refer to its group theory meaning. Burnside's Lemma states that the number of orbits in $\mathcal{A}/\mathcal{G}$ is equal to the average of the cardinal of the invariant sets over all the elements of $\mathcal{G}$:

Lemma 2 (Burnside's Lemma, Burnside 2012)**.**

$$|\mathcal{A}/\mathcal{G}| = \frac{1}{|\mathcal{G}|}\sum_{g\in\mathcal{G}}|\mathcal{A}^g| \tag{4.1}$$

4.2 Using Burnside's Lemma for Counting Solutions on `gcc`

In our problem, the set $\mathcal{A}$ is the set of perfect matchings and $\mathcal{G}$ is the group of symmetries that transform a perfect matching into another perfect matching leading to the same instantiation. Those symmetries are introduced by the permutations over the fake variables, and the permutations among duplicates of a value which leave instantiations over X unchanged. We use the same notations introduced in Section 3.2. Now, let us introduce formally this symmetry group.

Definition 4.4 (Group of permutations over the fake variables)**.** *Let $\mathfrak{S}_{\overline{X}}$ be the subgroup of the permutation group over $X \cup \overline{X} = \{x_1, \dots, x_{n+n'}\}$, which leave the elements of X invariant: $\forall\sigma \in \mathfrak{S}_{\overline{X}}$, we have $\forall x_i \in X, \sigma(x_i) = x_i$ and $\forall x_i \in \overline{X}, \sigma(x_i) \in \overline{X}$.*

These symmetries thus only act on the part of the graph corresponding to the fake variables, and leave the true variables unchanged.

Definition 4.5 (Group of permutation over the duplicated values)**.** *Let $y_j \in Y$, we note $\mathfrak{S}_{y_j}$, the group of permutations over $\{y_j^1, \dots, y_j^{\omega_j}\}$. Let $\mathcal{S}_{Y_\omega} = \{\tau = \sigma_1 \circ \dots \circ \sigma_m | \forall y_j \in Y, \sigma_j \in \mathfrak{S}_{y_j}\}$, the set of compositions of elements of $\mathfrak{S}_{y_j}$.*

Similarly, each $\mathfrak{S}_{y_j}$ only acts on the duplicate values for the corresponding value y_j. $\mathcal{S}_{Y_\omega}$ is also a group as each permutation acts on a separate set. By extension, we consider that $\mathcal{S}_{Y_\omega}$ acts directly on Y_ω, instead of the product $\{y_1^1, \dots, y_1^{\omega_1}\} \times \dots \times \{y_m^1, \dots, y_m^{\omega_m}\}$.

$\mathfrak{S}_{\overline{X}}$ represents the symmetries on the variables part of the $\omega-$multiplied value graph and $\mathcal{S}_{D_\omega}$ represents the symmetries on the values part. Figure 4.1a illustrates an assignment problem with a permutation from $\mathfrak{S}_{\overline{X}}$ acting on the variables part and a permutation from $\mathcal{S}_{Y_\omega}$ acting on the duplicated values part. Now, we can define the symmetry group for our problem:

Definition 4.6 (Symmetry group)**.** *We call $\mathcal{S}(X, \omega) = \mathfrak{S}_{\overline{X}} \times \mathcal{S}_{Y_\omega}$ the symmetry group of the ω-multiplied graph $G(X, \omega)$.*

Intuitively, we define the symmetries (*i.e.* permutations) on each vertex (fake variable and duplicated value) independently, and we consider their combinations in the symmetry group. For simplification, $S(X, \omega)$ will be referred to, in the following, as $\mathcal{S}$. $\mathcal{S}$ is actually a set of applications acting on the edges of $G(X, \omega)$. We say that $\mathcal{S}$ acts on E_ω. We define the action of $\mathcal{S}$ on E_ω as follows:

$$\phi_{\mathcal{S}\to E_\omega} =: \begin{cases} \mathcal{S} \times E_\omega \to E_\omega \\ (s, e) \to s(e) = (\sigma(x_i), \tau(y_j^k)), \text{with } s = (\sigma, \tau) \text{ and } e = (x_i, y_j^k) \end{cases}$$

And, by extension, we define the following action of $\mathcal{S}$ on $\mathcal{PM}$:

$$\phi_{\mathcal{S}\to\mathcal{PM}} =: \begin{cases} \mathcal{S} \times \mathcal{PM} \to \mathcal{PM} \\ (s, \mu) \to s(\mu) = \{s(e_1), \dots, s(e_{n+n'})\}, \text{with } \mu = \{e_1, \dots, e_{n+n'}\} \end{cases}$$

The symmetry group $\mathcal{S}$ has actually been constructed such that $\forall\mu \in \mathcal{PM}, \forall s \in \mathcal{S}$, $s(\mu)$ and μ leads to the same instantiation: $\pi(\mu) = \pi(s(\mu))$. In other words,

every perfect matching from a same orbit under the action of $\mathcal{S}$, leads to the same instantiation. There are as many orbits of $\mathcal{PM}$ under the action of $\mathcal{S}$ as instantiations in $\mathcal{I}$:

Proposition 4.1. *The number of instantiations over X is equal to the number of orbits of $\mathcal{PM}$ under the action of $\mathcal{S}$:*

$$|\mathcal{I}| = |\mathcal{PM}/\mathcal{S}| \tag{4.2}$$

Proof. Consider ι_1 and ι_2 two different instantiations in $\mathcal{I}$. Let μ_1 and μ_2 in $\mathcal{PM}$ such that $\pi(\mu_1) = \iota_1$ and $\pi(\mu_2) = \iota_2$. We want to show that μ_1 and μ_2 are in two different orbits.

We can write $\mu_1 = \{(x_1, y_{j_1}^{k_1}), \dots, (x_n, y_{j_n}^{k_n}), e_n, \dots, e_{n+n'})\}$ with $(y_{j_1}^{k_1}, \dots, y_{j_n}^{k_n}) \in (Y_\omega)^n$ and $(e_{n+1}, \dots, e_{n+n'}) \in (E_\omega)^{n'}$.

Let $s = (\sigma, \tau) \in \mathcal{S}$. We have $s(\mu_1) = \{(x_1, \tau(y_{j_1}^{k_1})),$
$\dots, (x_n, \tau(y_{j_n}^{k_n})), s(e_{n+1}), \dots, s(e_{n+n'}))\}$. By definition of $\mathcal{S}_{Y_\omega}$, $\forall i \in \{1, \dots, n\}$, $\exists k_i'$ such that $\tau(y_{j_i}^{k_i}) = y_{j_i}^{k_i'}$, and $s(\mu_1) = \{(x_1, y_{j_1}^{k_1'}), \dots, (x_n, y_{j_n}^{k_n'}), s(e_{n+1}), \dots, s(e_{n+n'}))\}$. Thus we have $\pi(s(\mu_1)) = \pi(\mu_1) = \iota_1 \neq \iota_2 = \pi(\mu_2)$ and finally $s(\mu_1) \neq \mu_2$. This implies that μ_1 cannot be reached from μ_2 by $s \in \mathcal{S}$, which shows that $\forall s \in \mathcal{S}$, if $s(\mu_1) \neq \mu_2$, then μ_1 and μ_2 are not in the same orbit.

We will now show the reverse implication: if two perfect matchings lead to the same instantiation, then they are in the same orbit. Consider an instantiation $\iota = (y_{j_1}, \dots, y_{j_n}) \in \mathcal{I}$ and μ_1 and μ_2 from $\mathcal{PM}$, such that ι: $\pi(\mu_1) = \pi(\mu_2) = \iota$. We have:

$$\mu_1 = \{(x_1, y_{j_1}^{k_1}), \dots, (x_n, y_{j_n}^{k_n}), (x_{n+1}, y_{j_{n+1}}^{k_{n+1}}), \dots, (x_{n+n'}, y_{j_{n+n'}}^{k_{n+n'}})\}$$

and

$$\mu_2 = \{(x_1, y_{j_1}^{k_1'}), \dots, (x_n, y_{j_n}^{k_n'}), (x_{n+1}, y_{j'_{n+1}}^{k'_{n+1}}), \dots, (x_{n+n'}, y_{j'_{n+n'}}^{k'_{n+n'}})\}$$

For $i \in \{1, \dots, n\}$, x_i must be instantiated to a duplicated value of y_{j_i} but not necessarily the same. In μ_1, x_i is assigned to $y_{j_i}^{k_i}$, while in μ_2, it is assigned to $y_{j_i}^{k_i'}$. As for $i \in \{n+1, \dots, n+n'\}$, x_i instantiate any value.

We know that $\pi(\mu_1) = \pi(\mu_2) = \iota$, thus, $\forall j \in \{1, \dots, m\}$, if there are c_j occurrences of value y_j in ι, then there are c_j true variables, that are assigned to duplicated value nodes of y_j in both μ_1 and μ_2. Equivalently, there are $\omega_j - c_j$ variables that are assigned to duplicated value nodes of y_j in both μ_1 and μ_2. Then, for each $i \in \{n+1, \dots, n+n'\}$, we can find, in μ_2, $\omega_{j_i} - c_{j_i}$ fake variables that are assigned to a duplicated value node of y_{j_i}. Since the subgraph of $G(X, \omega)$ induced by $\overline{X}$ is complete, there is a $\sigma \in \mathfrak{S}_{\overline{X}}$ such that:

$$\mu_2 = \{(x_1, y_{j_1}^{k_1'}), \dots, (x_n, y_{j_n}^{k_n'}), (\sigma^{-1}(x_{n+1}), y_{j_{n+1}}^{k^\Delta_{n+1}}), \dots, (\sigma^{-1}(x_{n+n'}), y_{j_{n+n'}}^{k^\Delta_{n+n'}})\}$$

Let $\tau \in \mathcal{S}_{Y_\omega}$, such that:

$$\tau(y_{j_i}^{k_i}) = \begin{cases} y_{j_i}^{k_i'} \text{ if } i \in \{1, \dots, n\} \\ y_{j_i}^{k_i^\Delta} \text{ otherwise} \end{cases} \tag{4.3}$$

and let $s = (\sigma, \tau) \in \mathcal{S}$. We have $\mu_1 = s(\mu_2)$. We just showed that, if two perfect matchings lead to the same instantiation, then they are in the same orbits.

There are then as many orbits of $\mathcal{PM}$ under the action of $\mathcal{S}$ as instantiations in $\mathcal{I}$. □

The following proposition is the main result of this chapter. It states that the number of instantiations is the average of the size of the invariant sets over the symmetry group:

Proposition 4.2. *Let $\mathcal{PM}^s$ be the invariant set of $\mathcal{PM}$ by a symmetry $s \in \mathcal{S}$. We have:*

$$|\mathcal{I}| = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |\mathcal{PM}^s| \tag{4.4}$$

Proof. In Proposition 4.1, we showed that $|\mathcal{I}| = |\mathcal{PM}/\mathcal{S}|$. According to Burnside's Lemma, it follows that $|\mathcal{I}| = |\mathcal{PM}/\mathcal{S}| = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |\mathcal{PM}^s|$. □

In Section 4.3 we evaluate $|\mathcal{PM}^s|$, the number of perfect matchings that remain invariant under symmetry s.

4.3 Study of the Number of Perfect Matchings Fixed by a Symmetry

In this section, we suggest a characterization of symmetries, for which there are invariant perfect matchings, that allows us to evaluate the size of the invariant set of $\mathcal{PM}$ by a given symmetry s.

4.3.1 Necessary and Sufficient Condition on Symmetries that Allows Invariant Perfect Matchings

We first study Example 4.1, that illustrates two conditions for a symmetry to admit invariant perfect matchings.

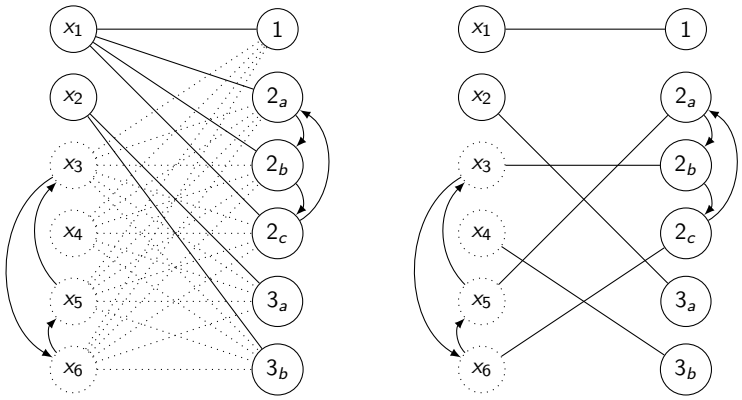
Example 4.1. *Let $X = \{x_1, x_2\}$, $D_1 = \{1, 2\}$ and $D_2 = \{3\}$, $\omega_1 = 1$, $\omega_2 = 3$ and $\omega_3 = 2$. We want to count how many instantiations of X there are. We need to add 4 fake variables, $\overline{X} = \{x_3, x_4, x_5, x_6\}$. The ω-multiplied value graph associated to this problem is shown in Figure 4.1a.*

We consider a symmetry $s = (\sigma, \tau) \in \mathcal{S}$, such that σ is the rotation $(x_3\ x_5\ x_6)$ and τ is the rotation $(2_b\ 2_a\ 2_c)$. This symmetry is also represented in Figure 4.1a.

We now want to count how many perfect matchings are invariant by s for this problem. Let us consider the following perfect matching $\mu = \{(x_1, 1), (x_2, 3_a), (x_3, 2_b), (x_4, 3_b), (x_5, 2_a), (x_6, 2_c)\}$, represented in 4.1b, which is invariant by s. We have indeed $s((x_1, 1)) = (x_1, 1)$, $s((x_2, 3_a)) = (x_2, 3_a)$ and $s((x_4, 3_b)) = (x_4, 3_b)$. Each variable that is invariant by σ matches with a value that is also invariant by τ. As for the three other variables, we have $s((x_3, 2_b)) = (x_6, 2_c)$, $s((x_5, 2_a)) = (x_3, 2_b)$ and $s((x_6, 2_c)) = (x_5, 2_b)$, then $s(\mu) = \mu$.

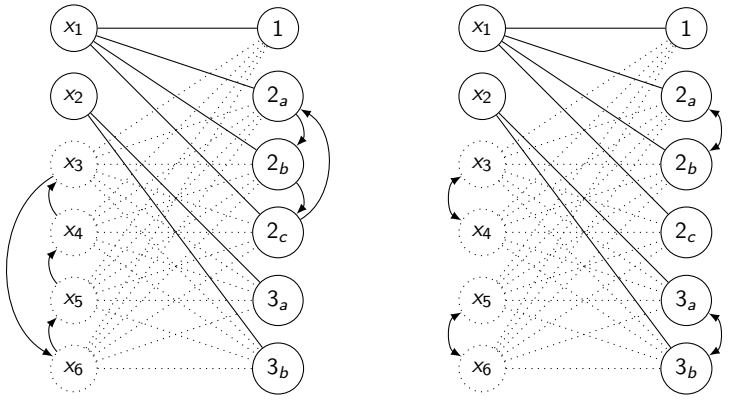
Given the symmetry s, we have found a perfect matching μ that is invariant by s. It is important to note that it is not possible to find such a perfect matching for each symmetry.

Figure 4.2a shows a symmetry for which there is no invariant perfect matching because, on the left we have one 4-cycle and two invariant variables and on the right, a 3-cycle and three invariant value nodes. It is then impossible that each invariant value node matches with an invariant variable. More generally, it is necessary to have the same number of cycles



(a) (b)

FIGURE 4.1: $\omega-$multiplied Value Graph of the problem described in Example 4.1 (a) and an invariant perfect matching μ (b)



(a) (b)

FIGURE 4.2: Examples of symmetry for which there is no invariant perfect matching

of the same order on both sides. Every variable from a cycle must match a value node from a same cycle.

Figure 4.2b shows a symmetry that checks the previous necessary conditions. There are two 2-cycles and four invariants nodes on both sides. However, we cannot find an invariant perfect matching for this symmetry. In fact, x_2, which remains invariant, has to match with 3_a or 3_b, which are not invariant. If we have the same number of same order cycles on both sides, then there are at least n invariant value nodes on the right side, as true variables are invariant, but this does not guarantee that these true variables and invariant value nodes can match.

The first condition is that symmetries allowing invariant perfect matchings are such that there are as many cycles of the same order on both sides. Those symmetries are called parallel:

Definition 4.7 (Parallel symmetry)**.** *Let $s = (\sigma, \tau) \in \mathcal{S}$, s is said to be parallel if and only if σ and τ have the same number of cycles of same order. We note $\mathcal{S}^{\parallel}$, the set of parallel symmetries.*

$$s \in \mathcal{S}^{\parallel} \Leftrightarrow \forall r \in \{1, \dots, \max_j(\omega_j)\}, cyc_r(\sigma) = cyc_r(\tau) \tag{4.5}$$

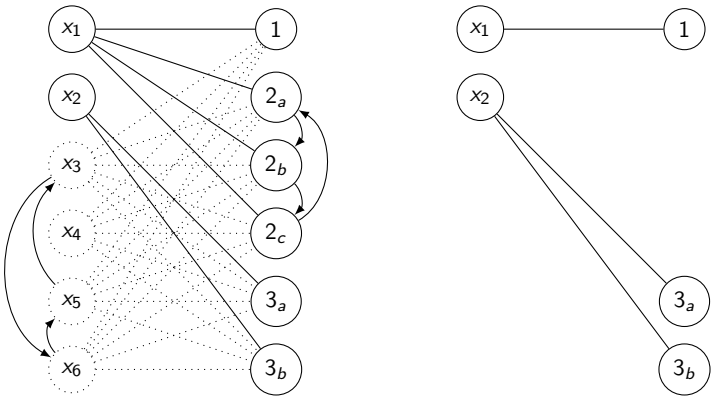


FIGURE 4.3: A symmetry on a $\omega-$multiplied value graph (a) and its induced $\omega-$multiplied value subgraph (b)

Remark 4.1. *If $r > \max_j(\omega_j)$, there cannot be a $r-$cycle in τ. This is why we do not check if $cyc_r(\sigma) = cyc_r(\tau)$ in this case.*

The following proposition gives a necessary condition for a symmetry to admit an invariant perfect matching:

Proposition 4.3. *Let $s \in \mathcal{S}$, if there is $\mu \in \mathcal{PM}$ such that μ is invariant by s, then s is parallel.*

Proof. Let $s = (\sigma, \tau) \in \mathcal{S}$, $\mu \in \mathcal{PM}$ and let us assume that $\mu = s(\mu)$. Let $e = (x_i, y_j^k) \in \mu$. Since $\mu = s(\mu)$, we have : $s(e) \in \mu$, $s(s(e)) \in \mu$, etc ... and $\forall e \in \mu$, $\mathcal{O}_s(e) \subseteq \mu$. And by definition, $\mathcal{O}_s(e) = \{(\sigma^q(x_i), \tau^q(y_j^k)) \in E_\omega | \forall q \in \mathbb{N}\} \subseteq \mathcal{O}_\sigma(x_i) \times \mathcal{O}_\tau(y_j^k)$. Since μ is a perfect matching, every variable and duplicated value are taken once and there is no q_1 and q_2 such that $q_1 < q_2 \leq |\mathcal{O}_s(e)|$ and $\big(\sigma^{q_1}(x_i) = \sigma^{q_2}(x_i)$ or $\tau^{q_1}(y_j^k) = \tau^{q_2}(y_j^k)\big)$. Then, $\forall q < |\mathcal{O}_\sigma(x_i)|$, $\sigma^q(x_i)$ is represented once in $\mathcal{O}_s(e)$, and $|\mathcal{O}_s(e)| = |\mathcal{O}_\sigma(x_i)|$. Similarly, we also have $|\mathcal{O}_s(e)| = |\mathcal{O}_\tau(y_j^k)|$. Thus there are the same number of cycles of same order in σ and τ, and s is parallel. □

We have seen, in Example 4.1, that having a parallel symmetry does not guarantee to have an invariant perfect matching. True variables must match invariant value nodes. We define the ω-multiplied subgraph induced by a symmetry:

Definition 4.8 ($\omega-$multiplied value subgraph induced by a symmetry)**.** *Let $s = (\sigma, \tau) \in \mathcal{S}^{\|}$, $\omega \in \mathbb{N}^m$, X, the set of variables and D_X, their domains and $G(X, \omega)$, the $\omega-$multiplied value graph associated to the problem.*

We define $G_s(X, \omega) = (X \cup \{y_j^k \in D_\omega | \tau(y_j^k) = y_j^k\}, \{(x_i, y_j^k) \in E_\omega | \tau(y_j^k) = y_j^k\})$ the $\omega-$multiplied value subgraph induced by s, as the subgraph of $G(X, \omega)$ containing only the true variables and the value nodes that are invariant by τ.

Example 4.2. *The $\omega-$multiplied value subgraph induced by the symmetry on Figure 4.3a is such as Figure on 4.3b. There are two maximum matchings covering the real variables on the subgraph: there are two possibilities to match true variables with invariant value nodes.*

$G_s(X, \omega)$ is such that if there is a matching covering X, then s admit an invariant perfect matching and there are $\Phi(G_s(X, \omega))$ possibilities to match true variables with invariant value nodes. In order to compute $\Phi(G_s(X, \omega))$, we add fake nodes

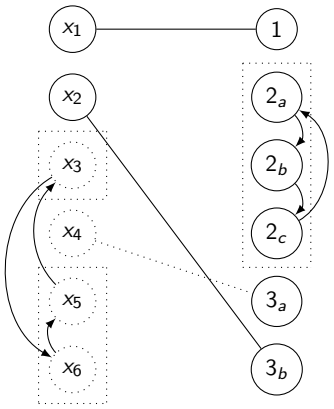


FIGURE 4.4: Cycles assignment

to balance the graph (if necessary), we compute the permanent (or an upper bound) of its biadjacency matrix and we remove the symmetry induced by the fake nodes, that is dividing by the factorial of the number of fake nodes. This is the method proposed by Pesant et al. (2012) (recalled in Section 1.4). It works here because there are no duplicated values symmetry in $G_s(X, \omega)$ to consider.

We can now deduce a necessary and sufficient condition for a symmetry s to have invariant perfect matching:

Proposition 4.4. *Let $s \in \mathcal{S}$ and $G_s(X, \omega)$, the $\omega-$multiplied value graph induced by s,*

$$\exists \mu \in \mathcal{PM}, s(\mu) = \mu \Leftrightarrow s \text{ is parallel and } \Phi(G_s(X, \omega)) > 0 \tag{4.6}$$

In next subsection, we count the number of invariant perfect matchings for symmetries that check the necessary and sufficient conditions of Proposition 4.4.

4.3.2 Enumerating Invariant Perfect Matchings for Parallel Symmetries

Once the true variables are assigned to invariant value nodes, we need to count how many possibilities there are to assign the fake variables to the remaining value nodes. We know that the subgraph containing only the fake variables is complete and that each variable node from a same cycle of σ must match with a duplicated value node from a same cycle in τ. Actually, we must assign each cycle from σ to a cycle of same order in τ and then, assign each variable from the first cycle to a duplicated value node from the second cycle. We want to count how many cycle's assignments there are and for each of them, how many variables assignments there are.

Example 4.3. *Figure 4.4 presents a possibility of cycle's assignment after assigning x_1 to 1 and x_2 to 3_b. Actually there is only one possibility here, we assign the set of variables $\{x_3, x_5, x_6\}$ with $\{2_a, 2_b, 2_c\}$, as there are both the unique 3-cycles on each side, and we assign x_4 to 3_a, as there are the unique remaining 1-cycle. For each cycle assignment, we now want to assign variables to value nodes. It is obvious that we must have x_4 assigned to 3_a. Concerning the 3-cycles assignment, there are several possibilities. Figure 4.5 shows the three possibilities for the 3-cycles assignment to match each variable with a value node. Let us consider Figure 4.5a, x_3 is assigned to 2_a, then the variable that comes next x_3 (considering the symmetry), x_6 must be assigned to the value nodes that comes next 2_a, which is 2_b, and so on... Actually the whole matching is determined by the first assignment. As there are three possibilities for the assignment of the first variable of the 3-cycle, then there are three possibilities to match every variable of the 3-cycle to value nodes from the corresponding 3-cycle. Considering that we have already instantiated x_1 to 1 and x_2 to 3_b, there are three*

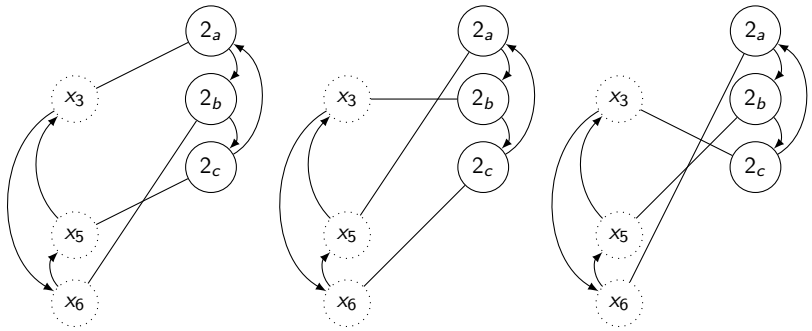


FIGURE 4.5: Every possible node's assignment inside the 3-cycle assignment

perfect matchings that are invariant by the symmetry. We also could have instantiated x_2 to 3_a, there would have been three invariant perfect matchings too. We do not need to know the exact assignment of true variables, as it does not change the number of remaining 1-cycles and the subgraph containing only the fake variables is complete. Finally, on this example, considering every true variables assignment, there are six invariant perfect matchings.

The next lemma is about the number of invariant matchings inside a r-cycle match (illustrated on Figure 4.5).

Lemma 3. *There are r invariant matchings inside a r-cycle match.*

Proof. If $r = 1$, then a match between two 1-cycles is equivalent to a match between the node in those 1-cycles and this matching is invariant.

For $r > 1$, let us consider a match between two r-cycles, with on one side the variable $\{x_1, \dots, x_r\}$ (those variables are fake, as there are only 1-cycles among true variables) and on the other side, the duplicated values $\{y_j^1, \dots, y_j^r\}$ and let $s = (\sigma, \tau) \in \mathcal{S}^{\parallel}$. Assume, wlog, that x_1 is assigned to y_j^k, with $k < r$. Then $\sigma(x_1)$ is assigned to $\tau(y_j^k)$, because the matching is invariant by s) and more generally, for every $q < r$, $\sigma^q(x_1)$ is assigned to $\tau^q(y_j^k)$. Then, assigning the first variable to any duplicated value determines the whole matching, if the matching is invariant. Thus, there are as many possibilities of invariant matchings as there are of fake variables, *i.e.* r. □

Notation. *Let $s = (\sigma, \tau) \in \mathcal{S}$, if s is parallel, we note $cyc_r(s) = cyc_r(\sigma) = cyc_r(\tau)$, the number of r-cycles in both σ and τ*

The next proposition gives a general formula for the number of invariant perfect matching for a given symmetry.

Proposition 4.5. *Let $s \in \mathcal{S}$, if s is parallel then*

$$|\mathcal{PM}^s| = \Phi(G_s(X, \omega)) \cdot (cyc_1(s) - n)! \prod_{r=2}^{\max_j(\omega_j)} r^{cyc_r(s)} \cdot cyc_r(s)! \tag{4.7}$$

Proof. Let $s = (\sigma, \tau) \in \mathcal{S}^{\parallel}$, $\omega \in \mathbb{N}^m$, and a problem of instantiation with X, the set of true variables and $G(X, \omega)$, the $\omega-$multiplied value graph. We want to count the number of perfect matchings of $G(X, \omega)$ that are invariant by s.

First we consider the true variables possible assignments. We have seen that there are $\Phi(G_s(X,\omega))$ ways to choose such an assignment. Then, for each $r \in \mathbb{N}$ we count the number of assignments for each r-cycles of both sides. We first consider 1-cycles. Some of those cycles have already been assigned, when assigning true variables. As fake variables can match every value node, there are $(cyc_1(s) - n)!$ ways to choose a matching among these 1-cycles.

Secondly, we consider r-cycles for $r > 1$. There are $cyc_r(s)$ r-cycles on both sides that have not been assigned yet and each of them can be assigned to any other, as we are only dealing with fake variables. Thus, for each $r \in \mathbb{N}$ there are $cyc_r(s)!$ ways to choose a matching among r-cycles.

Moreover, there are r ways to assign each variable from a r-cycle to a value node from the corresponding r-cycles by Lemma 3. We can finally write:

$$|\mathcal{PM}^s| = \Phi(G_s(X,\omega)) \cdot (cyc_1(s) - n)! \prod_{r=2}^{\max_j(\omega_j)} r^{cyc_r(s)} \cdot cyc_r(s)!$$

□

We now have a formula to compute $|\mathcal{PM}^s|$ for given parallel symmetry $s \in \mathcal{S}^{\|}$. In section 4.4, we question the possibility to deal with the exponential number of parallel symmetries.

4.4 Decomposition of the Symmetry Group

Replacing the term $|\mathcal{PM}^s|$ by the formula in Proposition 4.5 is not enough since we need to deal with a sum over the set of symmetries $\mathcal{S}$, wich is an exponential-size set. In this section, we will gather symmetries by the number of 1-cycles. For each kind of symmetry we suggest an upper bound of the invariant set size. We thus transform the equality in Proposition 4.2 to an upper bound that is polynomially computable. The choice of this decomposition is arbitrary. We could choose a decomposition of the symmetry group based on other criteria.

Definition 4.9 (Set of η-parallel symmetries)**.** *Let $\eta \in \mathbb{N}$, we note*

$$\mathcal{S}^{\|}_{\eta} = \{s \in \mathcal{S}^{\|} | cyc_1(s) = \eta\}$$

the set of parallel symmetries containing exactly η 1-cycles.

We know that, if $\eta < n, \forall s \in \mathcal{S}^{\|}_{\eta}$, $\mathcal{PM}^s = \emptyset$. Then we are only interested in the case $\eta \geq n$. We will find an upper bound of $|\mathcal{PM}^s|$, for each $\eta \geq n$, instead of for each $s \in \mathcal{S}^{\|}$ to dispose of the exponential number of terms in the sum.

Let $\eta \geq n$. We need to identify the kind of symmetries in $\mathcal{S}^{\|}_{\eta}$ which leads to the biggest number of possible assignment. Is it symmetries with a lot of small cycles or, instead, symmetries with a few big cycles ?

Figure 4.6 presents two symmetries from $\mathcal{S}^{\|}_3$ on a same bipartite graph. Edges are not represented and invariant variables have already been assigned. Now we want to know what is the best configuration to maximize matching possibilities. Let s_a be the symmetry represented in Figure 4.6a and s_b, the symmetry represented in Figure 4.6b. We have then:

- $|\mathcal{PM}^{s_a}| = 2^3 \cdot 3! = 48$

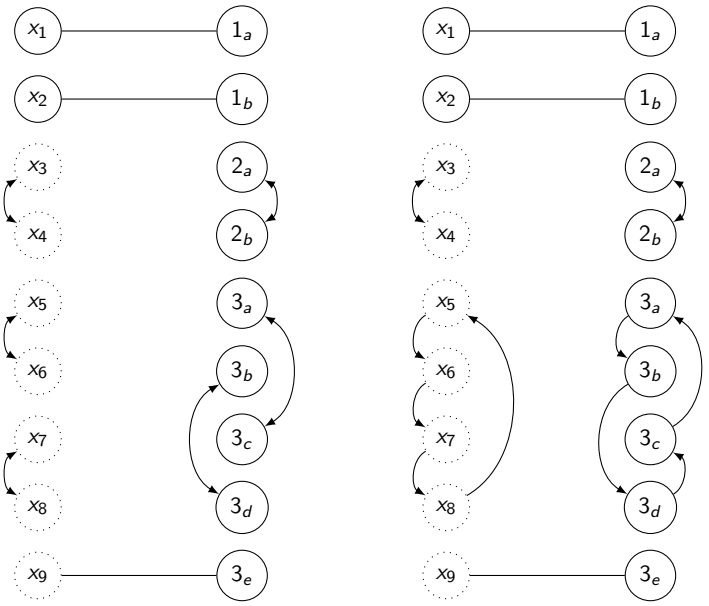


FIGURE 4.6: Two symmetries from $\mathcal{S}_3^{\|}$: one with three 2-cycles (a) and one with one 2-cycle and one 4-cycle (b)

- $|\mathcal{PM}^{s_b}| = 2^1 \cdot 1! \cdot 4^1 \cdot 1! = 8$

This example illustrates the following intuition : symmetries with a lot of small cycles lead to more matching possibilities than symmetries with a few big cycles. If a symmetry has a big cycle (with an order greater or equal to 4), we can increase matching possibilities by transforming this cycle into smaller 2-cycles or 3-cycles.

The next proposition suggests a bound for the number of invariant perfect matchings by a symmetry in $\mathcal{S}_\eta^{\|}$ and this bound is reached for symmetries composed of only 2-cycles (and η invariant nodes).

Proposition 4.6. *Let $\eta \geq n$, $s \in \mathcal{S}_\eta^{\|}$ and let β be defined as follow:*

$$\beta(z) = 2^{\lceil \frac{z}{2} \rceil} \cdot \lceil \frac{z}{2} \rceil ! \tag{4.8}$$

Then,

$$|\mathcal{PM}^s| \leq \beta(n + n' - \eta) \cdot \max_{s' \in \mathcal{S}_\eta^{\|}} (\Phi(G_{s'}(X, \omega))) \cdot (\eta - n)! \tag{4.9}$$

Proof. Let $s \in \mathcal{S}_\eta^{\|}$. We have:

- $(cyc_1(s) - n)! = (\eta - n)!$
- $\Phi(G_s(X, \omega)) \leq \max_{s' \in \mathcal{S}_\eta^{\|}} (\Phi(G_{s'}(X, \omega)))$

Now we want to upper bound $f(s) = \prod_{r=2}^{\max_j(\omega_j)} r^{cyc_r(s)} \cdot cyc_r(s)!$. We aim at finding a symmetry in $\mathcal{S}_\eta^{\|}$ that maximizes that quantity. In other words, we want to know what is the distribution of cycles, knowing that we have η 1-cycles, that maximizes

f. We can prove that symmetries that maximize f do not contain r-cycles, with $r > 3$: Let $s \in \mathcal{S}_\eta^{\|}$ and $\hat{s} \in \mathcal{S}_\eta^{\|}$ be the symmetry s for which each r-cycles have been transformed into 2-cycles and 3-cycles if $r > 3$. For each $r > 3$, we write $r = 2 \cdot a_r + 3 \cdot b_r$, where a_r (resp. b_r) are the number of 2-cycles (resp. 3-cycles) that compose the r-cycles in s. As $r > 3$, we necessarily have $a_r \geq 1$ or $b_r \geq 1$.

In the following, we note $M = \max_j(\omega_j)$. We have:

- $cyc_2(\hat{s}) = cyc_2(s) + \sum_{r=4}^{M} a_r \cdot cyc_r(s)$
- $cyc_3(\hat{s}) = cyc_3(s) + \sum_{r=4}^{M} b_r \cdot cyc_r(s)$

Now, we compare $f(s)$ and $f(\hat{s})$:

$$\frac{f(\hat{s})}{f(s)} = g_1(s) \cdot g_2(s)$$

with

$$g_1(s) = \frac{2^{\left(\sum_{r=4}^{M} a_r \cdot cyc_r(s)\right)} \cdot 3^{\left(\sum_{r=4}^{M} b_r \cdot cyc_r(s)\right)}}{\prod_{r=4}^{M} (2 \cdot a_r + 3 \cdot b_r)^{cyc_r(s)}}$$

and

$$g_2(s) = \frac{\left(cyc_2(s) + \sum_{r=4}^{M} a_r \cdot cyc_r(s)\right)! \cdot \left(cyc_3(s) + \sum_{r=4}^{M} b_r \cdot cyc_r(s)\right)!}{\prod_{r=2}^{M} cyc_r(s)!}$$

We now study g_1 and g_2. We can rewrite g_1:

$$g_1(s) = \prod_{r=4}^{M} \left(\frac{2^{a_r} \cdot 3^{b_r}}{2 \cdot a_r + 3 \cdot b_r}\right)^{cyc_r(s)}$$

We notice that $\forall r > 3$, we have:

$$2^{a_r} \cdot 3^{b_r} \geq r = 2 \cdot a_r + 3 \cdot b_r$$

Then $g_1(s) \geq 1$. We now study g_2. Since $\forall r > 3$, $a_r \geq 1$ or $b_r \geq 1$, we can write:

$$\forall r > 3, (a_r \cdot cyc_r(s))! \cdot (b_r \cdot cyc_r(s))! \geq cyc_r(s)!$$

Then

$$\prod_{r=4}^{M} (a_r \cdot cyc_r(s))! \cdot (b_r \cdot cyc_r(s))! \geq \prod_{r=4}^{M} cyc_r(s)!$$

$$cyc_2(s)! \cdot cyc_3(s)! \cdot \prod_{r=4}^{M} (a_r \cdot cyc_r(s))! \cdot (b_r \cdot cyc_r(s))! \geq \prod_{r=2}^{M} cyc_r(s)!$$

Also,

$$(cyc_2(s) + \sum_{r=4}^{M} a_r \cdot cyc_r(s))! \geq cyc_2(s)! \cdot \prod_{r=4}^{M} (a_r \cdot cyc_r(s))!$$
$$\text{and } (cyc_3(s) + \sum_{r=4}^{M} b_r \cdot cyc_r(s))! \geq cyc_3(s)! \cdot \prod_{r=4}^{M} (b_r \cdot cyc_r(s))!$$

Then

$$(cyc_2(s) + \sum_{r=4}^{M} a_r \cdot cyc_r(s))! \cdot (cyc_3(s) + \sum_{r=4}^{M} b_r \cdot cyc_r(s))! \geq \prod_{r=2}^{M} cyc_r(s)!$$

Then $g_2(s) \geq 1$. Finally we can conclude that $f(\hat{s}) \geq f(s)$, which means that the symmetry in $\mathcal{S}_\eta^{\parallel}$ maximizing f is only composed of η 1-cycles, 2-cycles and 3-cycles. The next and final question is then: how many 2-cycles and 3-cycles should we have to maximize f ? Actually, the intuition remains the same: the smaller the cycles are, the greater f is. In the following, we note $Rem = n + n' - \eta$ to simplify notations. Then Rem is the number of remaining variables (or duplicated values) to match after dealing with the η invariant nodes. We consider two cases, depending on whether Rem is even or not.

First case: Rem is even We consider now that s is only composed of 2-cycles and 3-cycles. Let a and b, be resp. the number of 2-cycles and 3-cycles in s. We want to show that:

$$2^a \cdot a! \cdot 3^b \cdot b! \leq 2^{\frac{Rem}{2}} \cdot \frac{Rem}{2}! \tag{4.10}$$

We know that $2a + 3b = Rem$ and Rem is even, thus necessarily, there is an even number of 3-cycles. We note $b = 2p$ and we can rewrite (4.10):

$$2^{\frac{Rem}{2}-3p} \cdot (\frac{Rem}{2} - 3p)! \cdot 3^{2p} \cdot (2p)! \leq 2^{\frac{Rem}{2}} \cdot \frac{Rem}{2}!$$
$$\Leftrightarrow \left(\frac{9}{8}\right)^p \leq \frac{\frac{Rem}{2}!}{(\frac{Rem}{2} - 3p)! \cdot (2p)!}$$

Ans $\frac{\frac{Rem}{2}!}{(\frac{Rem}{2}-3p)!} \geq (3p)!$ because the factorial of the sum is greater than the product of factorials. For $p = 0$, $\frac{(3p)!}{(2p)!} = 1 = \left(\frac{9}{8}\right)^p$ and for $p \geq 1$,

$$\begin{aligned}\frac{(3p)!}{(2p)!} &= (3p) \times ... \times (2p+1) \\ &\geq (2p+1)^p \\ &\geq 3^p \\ &\geq \left(\frac{9}{8}\right)^p\end{aligned}$$

It follows that $\forall p \geq 0$:

$$\frac{\frac{Rem}{2}!}{(\frac{Rem}{2} - 3p)! \cdot (2p)!} \geq \frac{(3p)!}{(2p)!} \geq \left(\frac{9}{8}\right)^p$$

Second case: *Rem* **is odd** We now want to show that:

$$2^a \cdot a! \cdot 3^b \cdot b! \leq 2^{\frac{Rem+1}{2}} \cdot \frac{Rem+1}{2}! \tag{4.11}$$

We know that $2a + 3b = Rem$ and *Rem* is odd, thus necessarily there is an odd number of 3-cycles. We note $b = 2p + 1$ and we can rewrite (4.11):

$$\begin{aligned} 2^{\frac{Rem-3}{2}-3p} \cdot (\frac{Rem-3}{2} - 3p)! \cdot 3^{2p+1} \cdot (2p+1)! &\leq 2^{\frac{Rem+1}{2}} \cdot \frac{Rem+1}{2}! \\ \Leftrightarrow \left(\frac{9}{8}\right)^p \cdot \frac{3}{4} &\leq \frac{\frac{Rem+1}{2}!}{(\frac{Rem-3}{2} - 3p)! \cdot (2p+1)!} \end{aligned}$$

Because *Rem* is odd and we do not consider the case $Rem = 1$ (as it is impossible to choose $\eta = n + n' - 1$, because the remaining invariant nodes must also be in a 1-cycle), we have $Rem \geq 3$. Then, for $p = 0$, we have:

$$\frac{\frac{Rem+1}{2}!}{(\frac{Rem-3}{2} - 3p)! \cdot (2p+1)!} = \frac{Rem+1}{2} \cdot \frac{Rem-1}{2} \geq 2 \geq \frac{3}{2} \cdot \left(\frac{9}{8}\right)^0$$

And for $p \geq 1$,

$$\begin{aligned} \left(\frac{8}{9}\right)^p \cdot \frac{\frac{Rem+1}{2}!}{(\frac{Rem-3}{2} - 3p)! \cdot (2p+1)!} &\geq \left(\frac{8}{9}\right)^p \cdot \frac{(3p+2)!}{(2p+1)!} \\ &\geq \left(\frac{8}{9}\right)^p \cdot (2p+2)^{p+1} \\ &\geq \left(\frac{32}{9}\right)^p \cdot 4 \\ &\geq \frac{3}{4} \end{aligned}$$

It follows that $\forall p \geq 0$:

$$\frac{\frac{Rem+1}{2}!}{(\frac{Rem-3}{2} - 3p)! \cdot (2p+1)!} \geq \frac{3}{4} \cdot \left(\frac{9}{8}\right)^p$$

Finally, we showed that, in order to maximize $\prod_{r=2}^{M} r^{cyc_r(s)} \cdot cyc_r(s)!$, s must be composed with only 2-cycles and 3-cycles (and 1-cycles for the already considered invariant nodes). And if s is only composed of 2-cycles and 3-cycles, we have:

$$\prod_{r=2}^{M} r^{cyc_r(s)} \cdot cyc_r(s)! \leq \beta(Rem) = 2^{\lceil \frac{Rem}{2} \rceil} \cdot \lceil \frac{Rem}{2} \rceil!$$

We can finally write:

$$|\mathcal{PM}^s| \leq \beta(n+n'-\eta) \cdot \max_{s' \in \mathcal{S}_\eta^{\|}} (\Phi(G_{s'}(X,\omega))) \cdot (\eta - n)!$$

□

From Proposition 4.2 and Proposition 4.6, we can deduce the following bound on the number of instantiations over X:

Proposition 4.7. *Let $\zeta(n,n',\eta) = A_{n+n'-\eta}^{n+n'} \cdot (\eta-n)! \cdot \beta(n+n'-\eta)$, with $A_{n+n'-\eta}^{n+n'}$ being the number of arrangements of $n+n'-\eta$ items among $n+n'$. Then:*

$$|\mathcal{I}| \leq UB_B(X,\omega) = \frac{1}{n'! \cdot \prod_{j=1}^{m} \omega_j!} \sum_{\eta=n}^{n+n'} \zeta(n,n',\eta) \cdot \max_{s' \in \mathcal{S}_\eta^{\|}} (\Phi(G_{s'}(X,\omega))) \tag{4.12}$$

Proof. Proposition 4.2 states that:

$$|\mathcal{I}| = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} |\mathcal{PM}^s|$$

We know that $\mathcal{S} = \mathfrak{S}_{\overline{X}} \times \mathcal{S}_{D_\omega}$, then:

$$|\mathcal{S}| = n'! \cdot \prod_{j=1}^{m} \omega_j!$$

And, according to Proposition 4.6, we can deduce:

$$\begin{aligned}
\sum_{s \in \mathcal{S}} |\mathcal{PM}^s| &= \sum_{s \in \mathcal{S}^{\|}} |\mathcal{PM}^s| \\
&= \sum_{\eta=n}^{n+n'} \sum_{s \in \mathcal{S}_{\equiv}^{\|}} |\mathcal{PM}^s| \\
&\leq \sum_{\eta=n}^{n+n'} |S_\eta^{\|}| \cdot \beta(n+n'-\eta) \cdot \max_{s' \in \mathcal{S}_\eta^{\|}} (\Phi(G_{s'}(X,\omega))) \cdot (\eta-n)!
\end{aligned}$$

Also, $S_\eta^{\|}$ is the set of parallel symmetries with η 1-cycle. In order to build a parallel symmetry, we must choose η invariant nodes among $n+n'$ nodes. Note that some of these combinations are impossible: for example, considering an instance for which a value y_j is duplicated twice, then we cannot choose only one of these two duplicated values to be invariant. Computing the exact value of $|S_\eta^{\|}|$ is hard, however, taking the number of combinations as an upper bound is sufficient for the result. This leads to:

$$|S_\eta^{\|}| \leq \binom{n+n'}{\eta} \cdot (n+n'-\eta)! = A_{n'+n-\eta}^{n+n'}$$

then, we conclude:

$$|\mathcal{I}| \leq UB_B(X,\omega) = \frac{1}{n'! \cdot \prod_{j=1}^{m} \omega_j!} \sum_{\eta=n}^{n+n'} \zeta(n,n',\eta) \cdot \max_{s' \in \mathcal{S}_\eta^{\|}} (\Phi(G_{s'}(X,\omega)))$$

□

In practice, to compute $\max_{s' \in \mathcal{S}^{\parallel}_{\eta}}(\Phi(G_{s'}(X,\omega)))$, we choose the η nodes that would maximize the Bregman-Minc bound or the Liang-Bai bound. We proceed as described by Pesant et al. (2012), when choosing the K variables in the Upper Bound Residual graph: we compute the factors in the considered bound associated with each node and choose the η bigger nodes. This has to be done n' times. We consider that we can compute the factorial function in constant time (or approximate it with a good accuracy). This bound is then polynomially computable. In Section 4.5, we compare the quality of UB_B to the quality of the upper bound UB_{IP} presented in chapter 3.

4.5 Comparison between Upper Bounds UB_{IP} and UB_B

For the group theory approach, we used Burnside's lemma and decomposed the number of possible instantiations into a sum of a lot of terms. Then we found a bound for each of those terms. Unfortunately, an error is accumulated on each term of the sum. Proposition 4.8 states that the upper bound proposed in Proposition 3.5 is closer to the actual number of solutions of a `gcc` instance than the upper bound deduced from the Burnside's lemma.

Proposition 4.8. *UB_{IP} dominates UB_B.*

Proof. Let X, the set of variables and ω, the vector of occurrences of each value,

$$\frac{UB_B(X,\omega)}{UB_{IP}(X,\omega)} = \frac{n'! \cdot \prod_{j=1}^{m} A_{c_j^*}^{\omega_j} \cdot \sum_{\eta=n}^{n+n'} \zeta(n,n',\eta) \cdot \max_{s \in \mathcal{S}^{\parallel}_{\eta}}(\Phi(G_s(X,\omega)))}{n'! \cdot \prod_{j=1}^{m} \omega_j! \cdot \Phi(G(X,\omega))}$$

where c^* is the vector minimizing $\prod_{j=1}^{m} A_{c_j}^{\omega_j}$ such that $\sum_{j=1}^{m} c_j = n$.

Then,

$$\frac{UB_B(X,\omega)}{UB_{IP}(X,\omega)} = \prod_{j=1}^{m} \binom{\omega_j}{c_j^*} \cdot \frac{\sum_{\eta=n}^{n+n'} \zeta(n,n',\eta) \cdot \max_{s \in \mathcal{S}^{\parallel}_{\eta}}(\Phi(G_s(X,\omega)))}{\Phi(G(X,\omega))}$$

And, for $\eta = n + n'$, $\zeta(n,n',\eta) = n'!$ and $\mathcal{S}^{\parallel}_{\eta}$ only contains the identity and $\Phi(G_{id}(X,\omega)) = \Phi(G(X,\omega))$. Then,

$$\frac{UB_B(X,\omega)}{UB_{IP}(X,\omega)} = \prod_{j=1}^{m} \binom{\omega_j}{c_j^*} \cdot \left(\frac{\sum_{\eta=n}^{n+n'-1} \zeta(n,n',\eta) \cdot \max_{s \in \mathcal{S}^{\parallel}_{\eta}}(\Phi(G_s(X,\omega)))}{\Phi(G(X,\omega))} + n'! \right)$$

$$\geq 1$$

□

We notice in the proof that UB_B is very large compared to UB_{LP}. This wide dominance comes from our decomposition of the set of parallel symmetries $\mathcal{S}^{\parallel}$ in Section 4. The sizes of the invariant sets are too different among symmetries from a same

subgroup $\mathcal{S}_\eta^{\|}$. The upper bound on the size of the invariant set for a subgroup is far from being sharp for every symmetry inside it. Using Burnside's lemma appears as an original way to deal with solution counting on constraints, as it properly considers symmetric solutions. Unfortunately, this reasoning ends with a too large upper bound. A possible improvement would be to find another decomposition of the parallel symmetry group.

Part III

Probabilistic Models for Cardinality Constraints

Chapter 5

Random Graph Models Applied to Alldifferent Constraint

Random graphs are graphs that are generated following a probabilistic distribution. We present in this chapter, the random graphs model of Erdős et al. (1963), who were pioneers in this field. Random graph models allow the study in average of some properties: existence of cycles, number of cliques, connection... More specifically, we are interested in studying perfect matchings inside random bipartite graphs. This average-case study will give us information on the number of solutions and the satisfiability of cardinality constraints. In this chapter, we will only focus on the *alldifferent* constraint, as we think the structure of this constraint is well suited for the use of random graphs model for cardinality constraints.

In Section 5.1, we present several probabilistic distributions for *alldifferent* graph models. We develop a method to compute an estimate of the number of solutions based on the edge density of an instance of *alldifferent* in Section 5.2 and we develop another estimate based on the domains sizes distribution in Section 5.4. These estimates are based on a probabilistic reasoning, unlike the estimates presented in Chapter 3 and Chapter 4, which are upper bounds of the number of solutions.

In this thesis, we are mostly concerned about estimating the number of solutions of cardinality constraints and in this chapter we develop random models to compute such estimates. Introducing random graph models is also the occasion to present another possible use of this probabilistic reasoning for CP. In Section 5.3, we study the asymptotic behaviour of the satisfiability of the constraint *alldifferent*. We highlight a phase transition for the existence of solutions for this constraint. A phase transition describes the fact that a system passes from one state to another. It has been generally used to described a physical system state of matter that changes when gaining or losing energy. Here the system is an instance of *alldifferent* and we study its two possible states : satisfiable and unsatisfiable.

In Section 5.5, we conclude with a comparative analysis between the two probabilistic estimates of the number of solutions presented in this chapter.

5.1 Random Value Graph Models

In this section, we present three random graph models that will be used to study the behaviour of the constraint *alldifferent*. Each of these models rely on different indicators on the state of the constraint. We will use the same notation as before in this thesis: let $X = \{x_1, \dots, x_n\}$ be the variables, $D = \{D_1, \dots, D_n\}$, the domains of these variables and $Y = \{y_1, \dots, y_m\} = \bigcup_{i=1}^{n} D_i$ the union of the domains.

5.1.1 Erdős-Renyi Model

Erdős et al. (1963) studied the existence and the number of perfect matchings on random graphs. They introduce two similar probabilistic models to pick up such graphs randomly. In the first model, the number of edges is fixed: given $n \in \mathbb{N}$ and $N \in \mathbb{N}$, the idea is to pick randomly uniformly one graph among the set of every graph with n vertices and N edges. Every graph of this set has then a probability $\binom{n^2}{N}^{-1}$ to be chosen.

In the second model, we do not reason on the set of every graph with n vertices and N edges, but we focus on each edge directly. Given a graph with n vertices, we pick up randomly and independently the existence of an edge with a probability $p = \frac{N}{n^2}$ between each pair of vertices. In this model, the number of edges is not fixed, instead N edges are expected. The number of edges actually follows a binomial distribution of parameter $p = \frac{N}{n^2}$.

This second model, which will be referred to as ER model, is the one we choose to exploit to elaborate random value graphs. The idea is to randomize the domain of each variable such that: for all $x_i \in X$ and for all $y_j \in Y$, the event $\{y_j \in D_i\}$ happens with a predefined probability $p \in [0, 1]$ and all such events are **independent**:

$$\mathbb{P}\left(\{y_j \in D_i\}\right) = p \in [0, 1] \tag{5.1}$$

This random model will allow the estimation of the number of solutions on an `alldifferent` instance, based on the edge density in the value graph (Section 5.2). It will also allow the study of existence of a solution (Section 5.3).

5.1.2 Fixed Domain Size Model

Estimating the number of solutions or studying their existence based only on the edge density in the value graph seems not very accurate. Indeed, the distribution of the edges inside the graph might have a great impact on the number of solutions. We hope to get a more accurate estimate by taking into account the domains size, and not only the edge density.

Let $n \in \mathbb{N}$, $m \in \mathbb{N}$ and $(d_1, \dots, d_n) \in \mathbb{N}^n$. In this random model, we pick randomly uniformly a value graph with $|X| = n$ variables, $|Y| = m$ values such that, for each variable $x_i \in X$, the corresponding domain size is $|D_i| = d_i$. We define $\mathcal{G}_{X,Y}(d_1, \dots, d_n)$, the set of such value graphs:

$$\mathcal{G}_{X,Y}(d_1, \dots, d_n) = \{G_{X,Y} \mid \forall x_i \in X, |D_i| = d_i\}$$

If we pick randomly uniformly an instance from $\mathcal{G}_{X,Y}(d_1, \dots, d_n)$, then we can evaluate the probability that a value y_j is in the domain D_i:

$$\begin{aligned}\mathbb{P}(\{y_j \in D_i\}) &= \frac{\#\text{number of configurations including } y_j}{\#\text{number of possible configurations}} \\ &= \frac{\binom{m-1}{d_i-1}}{\binom{m}{d_i}} = \frac{d_i}{m}\end{aligned}$$

It is important to notice that for two different values y_{j_1} and y_{j_2} and for a same domain D_i, the events $\{y_{j_1} \in D_i\}$ and $\{y_{j_2} \in D_i\}$ are not independent, unlike what happens in the Erdős-Renyi model. However, for two different domains D_{i_1} and D_{i_2}

and for any pair of values y_{j_1} and y_{j_2} (possibly the same), the events $\{y_{j_1} \in D_{i_1}\}$ and $\{y_{j_2} \in D_{i_2}\}$ are independent.

We call this model the Fixed Domain Size (FDS) model. It will allow the prediction of existence of solutions and an estimation of their number, relying on the domain's size of each variable.

5.1.3 Fixed Degrees Model

The Erdős-Renyi model is based on the edge density and the FDS model is based on the degree of the variables nodes. Another information that would be interesting to precise the edge distribution is the degree of the values' node. This last random value graph model is called the Fixed Degrees model.

Let $n \in \mathbb{N}$ and $m \in \mathbb{N}$, we note $D_j^{-1} = \{x_i \mid y_j \in D_i\}$ the set of variables that can take value y_j. We call it the inverted domains of y_j. Let $(d_1, \dots, d_n) \in \mathbb{N}^n$ and $(d_1^{-1}, \dots, d_m^{-1}) \in \mathbb{N}^m$. We pick randomly uniformly a value graph with $|X| = n$ variables, $|Y| = m$ values and such that, for each variable $x_i \in X$, the corresponding domain size is $|D_i| = d_i$ and for each value $y_j \in Y$, the corresponding inverted domain size is $|D_j^{-1}| = d_j^{-1}$. We define $\mathcal{G}_{X,Y}(d_1, \dots, d_n, d_1^{-1}, \dots, d_m^{-1})$, the set of such value graphs:

$$\mathcal{G}_{X,Y}(d_1, \dots, d_n, d_1^{-1}, \dots, d_m^{-1}) = \{G_{X,Y} \mid \forall x_i \in X, |D_i| = d_i \text{ and } \forall y_j \in Y, |D_j^{-1}| = d_j^{-1}\}$$

Intuitively, considering the degrees of each node in the value graph should give a better estimate of the number of solutions. Unfortunately, picking uniformly randomly such a value graph is a very difficult problem. Indeed, the existence of every edge in such a graph is inter-dependent, which makes the estimation of the number of perfect matchings much more difficult than the other models presented above.

This random model raises the question of the estimation of the permanent of a square binary matrix knowing only the sum over each row and each column. This question has not been solved in the general case. Some studies focus on the counting matchings in the specific case of regular and biregular bipartite graphs, that are bipartite graphs, in which the degrees of the nodes of one partition are all equal (Perarnau and Petridis 2013; Frieze and Melsted 2012). Yet, forcing value graphs to be regular or biregular is too restrictive for our purposes. The Fixed Degrees model is difficult to use in the study of perfect matchings in general bipartite graphs. In this chapter, we will only focus on the Erdős-Renyi model and the FDS model.

5.2 Estimating the Number of Solutions with Erdős-Renyi Model

In this section, we detail how we can compute an estimation of the number of solutions for an instance of `alldifferent` from the edge density of the value graph.

Proposition 5.1. *Let X be the set of variables and Y be the set of values with $|X| = n$ and $|Y| = m$. We take p the edge density in the value graph $G_{X,Y}$. According to the Erdős-Renyi model, the number of allowed tuples of $\texttt{alldifferent}(X)$ is expected to be:*

$$\mathbb{E}\left(\#\texttt{alldifferent}(X)\right) = \frac{m!}{(m-n)!}p^n \tag{5.2}$$

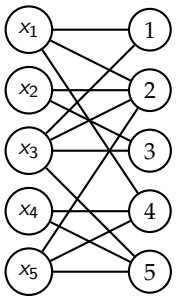


FIGURE 5.1: Value graph for Example 5.1

Proof. To simplify notation, we note $B = \mathcal{B}(G_{X,Y})$, the biadjacency matrix of the random value graph and, similarly, $B^b = \mathcal{B}(G^b_{X,Y})$, the biadjacency matrix of the value graph after balancing it. The matrix B can be seen as a random matrix, for which each element is a random 0-1 variable B_{ij} such that $P(\{B_{ij} = 1\}) = p$. According to Proposition 2.2 and Proposition 1.4, we have:

$$\begin{aligned}
\mathbb{E}\left(\#\texttt{alldifferent}\,(X)\right) &= \mathbb{E}\left(\frac{\Phi(G^b_{X,Y})}{(m-n)!}\right) \\
&= \mathbb{E}\left(\frac{Perm(B^b)}{(m-n)!}\right), \text{ by Proposition 1.1} \\
&= \mathbb{E}\left(\frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{m} B^b_{i\sigma(i)}\right) \\
&= \mathbb{E}\left(\frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n} B_{i\sigma(i)}\right),
\end{aligned}$$

because $\forall i > n, B^b_{ij} = 1$ and $\forall i \leq n, B^b_{ij} = B_{ij}$. The operator $\mathbb{E}(.)$ is linear and $\forall\sigma \in \mathfrak{S}_m, \forall i \in \{1, \dots, n\}$ the random variables $B_{i\sigma(i)}$ are independent, thus:

$$\begin{aligned}
\mathbb{E}(\#\texttt{alldifferent}\,(X)) &= \frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n}\mathbb{E}\left(B_{i\sigma(i)}\right) \\
&= \frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n} p \\
&= \frac{m!}{(m-n)!}p^n
\end{aligned}$$

□

When $n = m$, the expectancy of the number of solution is $n! \cdot p^n$. This result have been shown in Erdős et al. 1963. Proposition 5.1 extends it to unbalanced bipartite graph. We show practically how this result can be used in Example 5.1.

Example 5.1. *Let* $X = \{x_1, x_2, x_3, x_4, x_5\}$ *with* $D_1 = \{1, 2, 4\}$, $D_2 = \{2, 3\}$, $D_3 = \{1, 2, 3, 5\}$, $D_4 = \{4, 5\}$ *et* $D_5 = \{2, 4, 5\}$. *We obtain the value graph such as on Figure 5.1. The edge density in this graph is* $p = \frac{14}{25}$ *and* $n = m = 5$. *Then, according to the Erdős-Renyi, we can expect* $5! \cdot \left(\frac{14}{25}\right)^5 \simeq 6.61$ *solutions in* $\texttt{alldifferent}(X)$. *There are actually 8 solutions for this instance.*

We have shown in this section how to estimate the number of solutions for an instance of *alldifferent* based on the Erdős-Renyi model. In next section, we question the existence of one solution for such random instances.

5.3 Asymptotic Behaviour for the Existence of Solutions

The Erdős-Renyi model allows an estimation of the number of solution for an instance of *alldifferent*. In this section, we exploit a second result given by Erdős et al. (1963). We explain how we can detect if an instance of *alldifferent* has at least one solution. Erdős and Renyi gave the following result about the asymptotical behaviour of random balanced bipartite graph.

Proposition 5.2 (Erdős and Renyi, 1963)**.** *Let $n \in \mathbb{N}$ and we note $P_n(p)$, the probability that a random balanced bipartite graph of size n, generated with the parameter of edge density p, has at least one perfect matching. If we define the parameter in function of n as follows*[1]*:*

$$p_n = \frac{\ln n + c}{n} + o\left(\frac{1}{n}\right) \tag{5.3}$$

with $c \in \mathbb{R}$, then,

$$\lim_{n \to +\infty} P_n(p_n) = e^{-2e^{-c}} \tag{5.4}$$

This theoretical result is quite difficult to capture. It states that for a "big" balanced bipartite graph of size n, if the edge density is almost equal to $\frac{\ln n}{n}$, then we can estimate the probability that this graph has a perfect matching with Equation 5.4.

This result characterizes a phase transition. If the parameter p_n is very small compared to $\frac{\ln n}{n}$, then we are almost sure that the bipartite graph has not any perfect matching. And, on the opposite, if p_n is much bigger than $\frac{\ln n}{n}$, we are almost sure that there exists at least one perfect matching in the bipartite graph. Between these two extreme cases, there is an interval in which it is hard to decide whether the bipartite graph has a perfect matching or not. The transition is happening in this interval. Specifically, when n is bigger, this interval size decreases: the transition is quicker. For really big instances of bipartite graphs, it is easier to decide whether there is at least one perfect matching or not, as the interval is smaller. Figure 5.2 represents the shape of this phase transition for the existence of solutions for the *alldifferent* constraint.

We propose now a method to detect if an instance of *alldifferent* have at least one solution, or no. Let $X = \{x_1, \dots, x_n\}$, be the set of variables and $Y = \{y_1, \dots, y_n\}$, be the set of values of an instance *alldifferent*(X). Proposition 5.2 is stated for random balanced bipartite graph, this is why we focus on the case $|X| = |Y|$. We note δ the edge density in the value graph $G_{X,Y}$. The edge density is the ratio $\frac{N}{n^2}$ between the number of edges N and the maximum number of edges in a balanced bipartite graph of size n. Note that, in the case of *alldifferent*, the number of edges is the sum of the domains sizes. If n is big enough, we can estimate the probability that *alldifferent*(X) has at least one solution:

[1]The natural logarithm is noted ln.

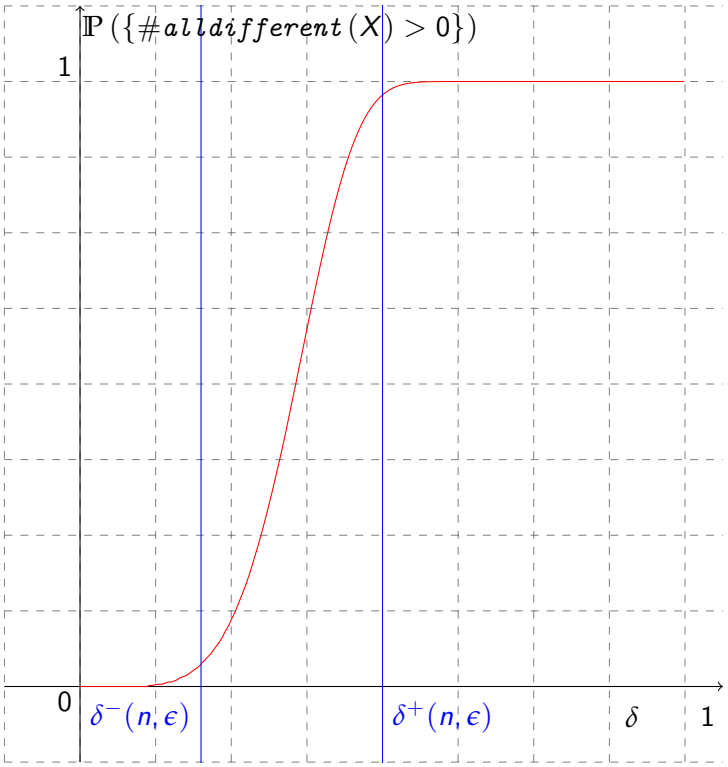


FIGURE 5.2: Phase transition for the existence of solutions in an *alldifferent* instance

$$\mathbb{P}(\{\#alldifferent(X) > 0\}) = e^{-2e^{-c}} \tag{5.5}$$
$$\text{with } c = n\delta - \ln n \tag{5.6}$$

We propose now to compute the maximal edge density $\delta^-(n,\epsilon)$ for which the probability of existence of at least one solution is below a certain value ϵ:

$$\begin{aligned}
\mathbb{P}(\{\#alldifferent(X) > 0\}) &< \epsilon \\
e^{-2e^{-c}} &< \epsilon \\
e^{-2e^{\ln n - n\delta}} &< \epsilon \\
e^{\ln n - n\delta} &> \frac{-\ln \epsilon}{2} \\
\delta < \delta^-(n,\epsilon) &= \frac{1}{n}\left(\ln n - \ln\left(\frac{-\ln \epsilon}{2}\right)\right)
\end{aligned}$$

We define similarly, the minimal edge density $\delta^+(n,\epsilon)$ for which the probability of existence of at least one solution is above a certain value $1-\epsilon$:

$$\delta^+(n,\epsilon) = \frac{1}{n}\left(\ln n - \ln\left(\frac{-\ln(1-\epsilon)}{2}\right)\right)$$

We represent in Figure 5.2, the interval $[\delta^-(n,\epsilon), \delta^+(n,\epsilon)]$, in which there is a transition, in which it is hard to decide if an instance with a given density $\delta \in [\delta^-(n,\epsilon), \delta^+(n,\epsilon)]$ has a solution or not.

n	10	20	50	100	200
$\delta^-(n,\epsilon)$	19.0%	13.0%	7.0%	4.2%	2.4%
$\delta^+(n,\epsilon)$	59.7%	33.3%	15.2%	8.3%	4.5%
false satisfiable	1	3	4	1	3
false unsatisfiable	1	0	0	1	0

TABLE 5.1: Transition interval and number of false predictions for 10000 instances of size $n \in \{10, 20, 50, 100, 200\}$

It is interesting to note that the size of this interval decreases and tends to zero while the size of the instance n grows:

$$\lim_{n\to+\infty}(\delta^+(n,\epsilon)-\delta^-(n,\epsilon)) = \lim_{n\to+\infty}\frac{1}{n}\ln\left(\frac{\ln\epsilon}{\ln(1-\epsilon)}\right) = 0$$

In other words, if an instance of `alldifferent` is big enough, then the interval of transition is very small: the interval of incertitude (for the existence of solutions) is very small. For big instances of `alldifferent`, most of the time—if the edge density is not inside the transition interval—we can decide whether there is a solution or not with a certitude parametrized by ϵ. The question is now to define how big the size of an instance n must be to apply these results.

We have run some experiments to test these theoretical results in practice. We want to test how accurate this method can predict if an `alldifferent` instance has solutions or not. For different parameter $n \in \{10, 20, 50, 100, 200\}$, we construct randomly 10000 instances according to the Erdós-Renyi process with a parameter $p \in]0, 1[$ that we choose randomly for each instance. For each instance, we compute its actual edge density δ and if $\delta > \delta^+(n,\epsilon)$, we predict that the instance is satisfiable and if $\delta < \delta^-(n,\epsilon)$, we predict that it is not satisfiable. Finally, if $\delta \in [\delta^-(n,\epsilon), \delta^+(n,\epsilon)]$, we conclude that we cannot predict the satisfiability of this instance. For each instance, we compare the prediction to the actual satisfiability and we count the number of false positive and false negative. In Table 5.1, we report these results for a confidence range $\epsilon = 5\%$:

We note that this method is very accurate for $\epsilon = 95\%$, as it does only a very few false predictions on the satisfiability of the instances. Moreover, it does not require the instance to be very large: for $n = 10$, the predictions are accurate.

In this section, we show how to use the asymptotic behaviour of random bipartite graphs described by Erdős et al. (1963) in order to predict the satisfiability of an `alldifferent` instance. We do not study practical applications of this result in solvers but we have some ideas that are leads to be explored. Following the fail-first principle, it might be interesting to first propagate constraints that we are almost sure that they do not have solution. Also, concerning instances for which it is hard to predict if they are satisfiable or not, we could branch on pairs variable/value that would make the instance toggle into the satisfiable or the unsatisfiable part (on the left or on the right part of the transition phase). Although, propagating on `alldifferent` is not difficult in practice (even for big instances), we think that it is interesting to keep these ideas in mind if similar results can be exploited for other cardinality constraints.

One of the main limitation of this method is that it is designed for balanced bipartite graphs, which corresponds to `alldifferent` instances where the set of variables and the set of values are the same size. Unlike the computation of the permanent for which we complete the rectangular matrix with lines full of "1" to make it square,

here balancing the bipartite graph adds a bias. The added edges are not picked randomly following the Erdős-Renyi distribution and Proposition 5.2 does not apply in that case. Proposition 5.2 should be extended to fit the unbalanced bipartite graph case.

The asymptotic behaviour of bipartite graphs according to the distribution of the degrees have not been studied in this thesis. We do not have similar results for the FDS model.

5.4 Estimating the Number of Solutions with Fixed Domain Size Model

In this section, we present a study of the number of solutions based on the size of each variable's domain. As for Section 5.2, we first develop an estimate of the number of solutions for a random instance of `alldifferent` and show a practical example of how to use such an estimate. We propose comparative analysis of the accuracy of the first estimate based on the Erdős-Renyi model and the estimate based on the Fixed Domain Size model in Section 5.5.

Proposition 5.3. *Let X be the set of variables and Y be the set of values with $|X| = n$ and $|Y| = m$ and $\forall x_i \in X$, we note $d_i = |D_i|$. According to the FDS model, the number of allowed tuples of $\texttt{alldifferent}(X)$ is expected to be:*

$$\mathbb{E}(\#\texttt{alldifferent}(X)) = \frac{m!}{(m-n)! \cdot m^n} \cdot \prod_{i=1}^{n} d_i \tag{5.7}$$

Proof. The proof is very similar to the proof of Proposition 5.1. We note $B = \mathcal{B}(G_{X,Y})$, the biadjacency matrix of the random value graph and $B^b = \mathcal{B}(G^b_{X,Y})$, the biadjacency matrix of the value graph after balancing it. The matrix B can be seen as a random matrix, for which each element is a random 0-1 variable B_{ij} such that $P(\{B_{ij} = 1\}) = \frac{d_i}{m}$. According to Proposition 2.2 and Proposition 1.4, we have:

$$\begin{aligned}\mathbb{E}\left(\#\texttt{alldifferent}\left(X\right)\right) &= \mathbb{E}\left(\frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n} B_{i\sigma(i)}\right)\\ &= \frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\mathbb{E}\left(\prod_{i=1}^{n} B_{i\sigma(i)}\right), \text{ by linearity of } \mathbb{E}\left(.\right)\end{aligned}$$

Here again, due to the independence hypothesis, $\mathbb{E}\left(\prod_{i=1}^{n} B_{i\sigma(i)}\right)$ is equal to $\prod_{i=1}^{n}\mathbb{E}\left(B_{i\sigma(i)}\right)$. In FDS model, two coefficients B_{ij_1} and B_{ij_2} on a same row are not independent, by opposition of the Erdős-Renyi model. However, the product $\prod_{i=1}^{n} B_{i\sigma(i)}$ is over a set of coefficients that are on different rows and are, thus, all independent. Then, we have:

$$\begin{aligned}\mathbb{E}\left(\#\mathtt{alldifferent}\left(X\right)\right) &= \frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n}\mathbb{E}\left(B_{i\sigma(i)}\right)\\ &= \frac{1}{(m-n)!}\sum_{\sigma\in\mathfrak{S}_m}\prod_{i=1}^{n}\frac{d_i}{m}\\ &= \frac{m!}{(m-n)!\cdot m^n}\cdot\prod_{i=1}^{n}d_i\end{aligned}$$

□

In Example 5.2, we show how practically can use this last estimate:

Example 5.2. *We take the same instance as given in Example 5.1. Let* $X = \{x_1, x_2, x_3, x_4, x_5\}$ *with* $D_1 = \{1, 2, 4\}$, $D_2 = \{2, 3\}$, $D_3 = \{1, 2, 3, 5\}$, $D_4 = \{4, 5\}$ *et* $D_5 = \{2, 4, 5\}$. *We obtain the value graph such as on Figure 5.1. We have then* $d_1 = 3, d_2 = 2, d_3 = 4, d_4 = 2$ *and* $d_5 = 3$. *According to the Erdős-Renyi, we can expect* $\frac{5!}{5^5}\cdot 3\cdot 2\cdot 4\cdot 2\cdot 3 \simeq 5,53$ *solutions in* $\mathtt{alldifferent}(X)$. *There are actually 8 solutions for this instance.*

For the instance presented in Example 5.1, the estimate based on the Erdős-Renyi model is actually more accurate than the FDS model. We will see in the final section of this chapter that it is not always the case.

5.5 Comparative Analysis between Erdős-Renyi Model and FDS Model

We first present a qualitative analysis of the estimators of the number of solutions based on the ER and FDS models and the PQZ estimator developed by Pesant et al. (2012). Then, we adapt the `maxSD` strategy (Pesant et al. 2012), presented in Section 2.3.2, so that it is guided by the estimates derived from ER model and FDS model.

5.5.1 Qualitative analysis on `alldifferent` instances

For this qualitative analysis, we have generated randomly and uniformly 10000 instances of `alldifferent`, which present at least one solution, with $n = 10$ variables and $m = 10$ values. We choose randomly a parameter p and we generate an instance according to the Erdős-Renyi model. For each of those instances, we have computed the three estimators of the number of solutions and we compare them to the real number of solutions. Figure 5.3 shows the percentage of instances for different ranges of relative gap between the value of the estimators and the real number of solutions. The relative gap is computed this way : $\Delta = \frac{|est-real|}{real}$ where *est* is the value of the estimator and *real* is the value of the true number of solutions.

We notice that the third estimator, the expectancy according to the FDS model, seems more accurate as, for about 40% of instances, the relative gap is less than or equal to 0.05 and, for less than 10% of instances, the relative gap is superior to 1. The expectancy according to the Erdős-Renyi model is less accurate. As for the upper bound UB^{PQZ}, it is very far from being accurate.

These results can be explained by the fact that the two last estimators correspond to the expected number of solutions according to Erdős-Renyi model and FDS model, whereas the first estimator is an upper bound. The FDS estimator is more accurate than the ER estimator, as it considers the distribution of domains size

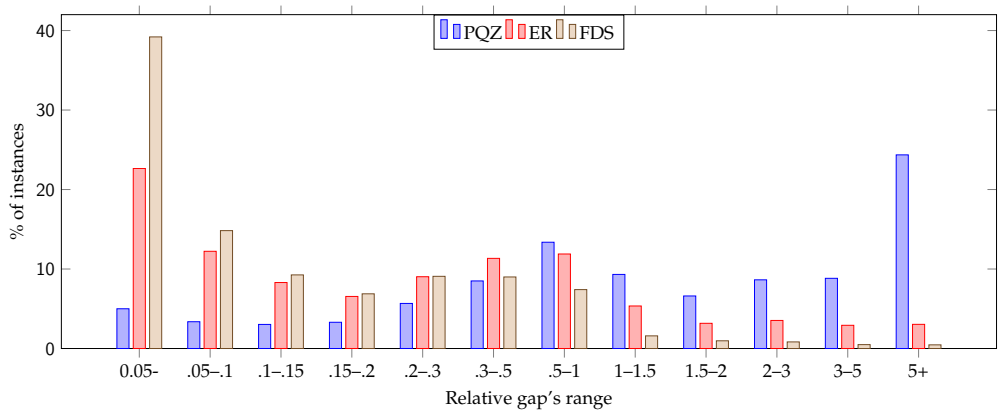


FIGURE 5.3: Percentage of instances per relative gap for each estimator

and not only the density of edges. For search strategies, the correlation between the estimator and the real number of solutions is more important than the quality of the estimator. In next subsection, we compare the efficiency of Counting-Based search for the three estimators.

5.5.2 Estimators' efficiency within Counting-Based Search

We have adapted the *maxSD* heuristic presented in Pesant et al. 2012, originally designed with the PQZ estimator, such that the solution densities are computed from the ER estimator and the FDS estimator. We run this strategy with the different estimators on 40 hard instances of the Latin Square Problem.

A Latin Square problem is defined by a $n * n$ grid whose squares each contain an integer from 1 to n such that each integer appears exactly once per row and column[2]. The model uses a matrix of integer variables and an `alldifferent` constraint for each row and each column. We tested on the 40 hard instances used by Pesant et al. (2012) with n = 30 and 42% of holes (corresponding to the phase transition), generated following C. Gomes and Shmoys (2002).

To solve these instances, we have implemented each version of *maxSD* in the solver Choco v4.0.9 (Prud'homme et al. 2016). Figure 5.4 shows the evolution of the number of solved instances with the number of backtracks and required number of backtracks.

On this benchmark, none of the estimators seems to outperform the others. With the FDS estimator, we solved 70% instances against 77.5% instances for the two other estimators. The `maxSD` strategy with the PQZ estimator performed slightly better than the two other probabilistic estimators, as half of the solved instances required less backtracks and time.

We can conclude on these experiments that the quality of an estimator is not highly correlated to its efficiency within the `maxSD` strategy. Indeed, we showed that the FDS estimator is much more accurate than the ER estimator, but the results given by this benchmark are very comparable for both probabilistic estimators.

This benchmark only considers one class of problem. Its purpose is overall to show that it is possible to integrate these new probabilistic estimators in a solver and that the results can be very comparable to the state of the art. The limitation of this

[2]Gilles Pesant. *CSPLib Problem 067: Quasigroup Completion*. Ed. by Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, and Ian P. Gent. http://www.csplib.org/Problems/prob067

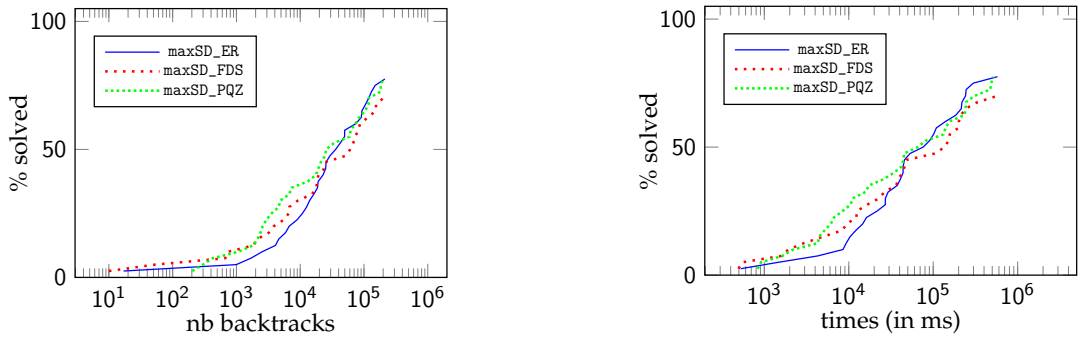


FIGURE 5.4: Performances of `maxSD_ER` and `maxSD_FDS` on 40 hard Latin Square instances, in number of backtracks (left) and time (right)

benchmark also comes from the lack of existing benchmark on the `maxSD` strategy in general.

Chapter 6

Estimating the Number of Solutions with Cardinality Constraints Decompositions

In this chapter, we aim at estimating the number of solutions on the following cardinality constraints, all presented in 2.2.3: *alldifferent*, *nvalue*, *atmost_nvalue*, *atleast_nvalue*, *occurrence*, *atmost*, *atleast*, *among*, *uses*, *disjoint*. These estimates are all derived from the Erdős-Renyi Model, presented in chapter 5. We propose here a systematic method to estimate the number of solutions of most of the cardinality constraints. This work have been published in the proceedings of the 25th International Conference CP 2019 (Lo Bianco, Lorca, and Truchet 2019).

Pesant et al. (2012) showed that counting or estimating the number of solutions required dedicated models for each constraint. If we could decompose every cardinality constraints into smaller identical constraints, then having a counting techniques for this smaller constraint would help counting solutions on the initial cardinality constraint.

Bessiere, Hebrard, Hnich, Kiziltan, and Walsh (2009) introduce two new global constraints *range* and *roots*, that can be used to specify many cardinality constraints. In other words, for almost every cardinality constraint, there is an equivalent model using only the more primitive *range* and *roots* constraints (and some arithmetic constraints).

We show how to use the *range* and *roots* decomposition for counting solutions. More precisely, we develop a probabilistic approach, based on the Erdős-Renyi Model, to estimate the number of solutions on a *range* and on a *roots* constraint. We derive from it a systematic method to estimate the number of solutions on cardinality constraints. Compared to Pesant et al. (2012), we obtain an estimation instead of an upper bound, and we propose a method that can be generalized to a large set of cardinality constraints without redesigning a dedicated model.

6.1 Introduction to *range* and *roots* Constraints

The *range* and *roots* constraints (Bessiere et al. 2009) are two auxiliary constraints that can help decomposing a lot of cardinality constraints. In this study, we will use these decomposition to count solutions on cardinality constraints. As Bessiere et al. (2009) wrote, "*range* captures the notion of image of a function and *roots* captures the notion of domain". We use alternative definitions for these constraints, equivalent to those of Bessiere et al. (2009) and better suited to our needs.

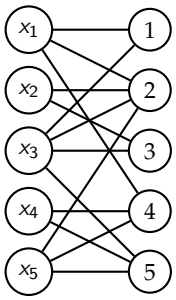


FIGURE 6.1: A value graph $G_{X,Y}$

Definition 6.1 (`range`). *Let $X' \subseteq X$ and $Y' \subseteq Y$. The constraint $\texttt{range}(X, X', Y')$ holds if the values assigned to variables of X' covers **exactly** Y' and not more. Formally:*

$$\mathcal{S}_{\texttt{range}(X,X',Y')} = \{(v_1, \dots, v_n) \in \mathcal{D} \mid \{v_i | x_i \in X'\} = Y'\} \tag{6.1}$$

Definition 6.2 (`roots`). *Let $X' \subseteq X$ and $Y' \subseteq Y$. The constraint $\texttt{roots}(X, X', Y')$ holds if the variables that are assigned to values of Y' covers **exactly** X' and not more. Formally:*

$$\mathcal{S}_{\texttt{roots}(X,X',Y')} = \{(v_1, \dots, v_n) \in \mathcal{D} \mid \{x_i | v_i \in Y'\} = X'\} \tag{6.2}$$

Example 6.1. *Let's take the value graph given in Figure 6.1.*

- *The tuple $(2, 2, 3, 4, 5)$ is allowed by the constraint $\texttt{range}(X, \{x_1, x_2, x_3\}, \{2, 3\})$.*
- *The tuple $(2, 2, 5, 4, 5)$ is not allowed by the constraint $\texttt{range}(X, \{x_1, x_2, x_3\}, \{2, 3\})$ but it is allowed by $\texttt{range}(X, \{x_1, x_2, x_3\}, \{2, 5\})$.*
- *The tuple $(2, 2, 3, 4, 5)$ is allowed by the constraint $\texttt{roots}(X, \{x_1, x_2, x_3\}, \{2, 3\})$.*
- *The tuple $(2, 2, 3, 4, 2)$ is not allowed by the constraint $\texttt{roots}(X, \{x_1, x_2, x_3\}, \{2, 3\})$, but it is allowed by $\texttt{roots}(X, \{x_1, x_2, x_3, x_5\}, \{2, 3\})$.*
- *The tuple $(2, 2, 2, 4, 5)$ is allowed by the constraint $\texttt{range}(X, \{x_1, x_2, x_3\}, \{2\})$ but not by $\texttt{range}(X, \{x_1, x_2, x_3\}, \{2, 3\})$, as 3 is not assigned to x_1, x_2 nor x_3. However, the tuple $(2, 2, 2, 4, 5)$ is allowed by $\texttt{roots}(X, \{x_1, x_2, x_3\}, \{2, 3\})$. Indeed, 3 is not assigned to any variable, so X' is only defined by the variables that are assigned to 2.*

Note that `range` and `roots` are not exactly reciprocal because every variable must be assigned to a value, but a value is not necessarily assigned to a variable.

6.2 Counting Solutions on `range` and `roots`

As developed by Pesant et al. (2012), counting solutions on cardinality constraints requires dedicated counting algorithm for each constraint. In this section we are interested by computing the number of solutions on the `range` and the `roots` constraints. The idea is then to only use the decomposition of cardinality constraints into these more primitive constraints and to reuse the counting method on `range` and `roots` to count solutions on cardinality constraints.

6.2.1 Exact solutions counting on range and roots

In this subsection, we are interested by exactly computing the number of allowed tuples for a range constraint and a roots constraint. We will use the notation that we used all along this thesis: let $X = \{x_1, \dots, x_n\}$ be the variables, $D = \{D_1, \dots, D_n\}$, the domains of these variables and $Y = \{y_1, \dots, y_m\} = \bigcup_{i=1}^{n} D_i$ the union of the domains. We note $d_i = |D_i|$, the size of the domain D_i.

Proposition 6.1. *Let $X' \subseteq X$ and $Y' \subseteq Y$. We note $\overline{X'}$, the complement of X' in X, such that $\overline{X'} \cup X' = X$ and $\overline{X'} \cap X' = \emptyset$. Then, the number of tuples allowed by $\mathtt{range}(X, X', Y')$ is*

$$\#\mathtt{range}(X, X', Y') = \#\mathtt{range}(X', X', Y') \cdot \prod_{x_i \in \overline{X'}} d_i \tag{6.3}$$

Proof. On one side, we must consider every possible assignment for the variables of $\overline{X'}$ that are not constrained: $\prod_{x_i \in \overline{X'}} d_i$. And on the other side, we must count every tuples allowed for variables of X', that are constrained, that is simply $\#\mathtt{range}(X', X', Y')$. The number of tuples is thus the product of these quantities. □ □

Proposition 6.1 reduces the problem of counting allowed tuples for every variable in X to only counting tuples for the constrained variables X'. We thus have reduced the problem to counting the number of allowed tuples in the case where every variable and value is constrained.

Proposition 6.2.

$$\#\mathtt{range}(X, X, Y) = \prod_{x_i \in X} d_i - \sum_{Y' \subsetneq Y} \#\mathtt{range}(X, X, Y') \tag{6.4}$$

Proof. Inside $G_{X,Y}$, we must count every possible assignment of variables of X such that every value of Y is covered. To do that, we first count the number of every possible assignment of variables of X in $G_{X,Y}$ (without considering the range constraint):

$$\prod_{x_i \in X} d_i$$

And then, we withdraw, one by one, the assignment of X such that Y is not fully covered, that is, for every subset $Y' \subsetneq Y$, the solutions of $\mathtt{range}(X, X, Y')$:

$$\sum_{Y' \subsetneq Y'} \#\mathtt{range}(X, X, Y')$$

Indeed, for two different subsets $Y'_1 \neq Y'_2 \subsetneq Y$, the sets of allowed tuples $\mathcal{S}_{\mathtt{range}(X,X,Y'_1)}$ and $\mathcal{S}_{\mathtt{range}(X,X,Y'_2)}$ are necessarily disjoint: there is a value $y_j \in Y$ such that $y_j \in Y'_1$ and $y_j \notin Y'_2$ (or $y_j \in Y'_2$ and $y_j \notin Y'_1$), so the value y_j must be assigned to one of the variable of X to satisfy $\mathtt{range}(X, X, Y'_1)$ but none of the variable of X must be assigned to y_j to satisfy $\mathtt{range}(X, X, Y'_2)$ (or vice-versa). A solution of $\mathtt{range}(X, X, Y'_1)$ cannot be a solution of $\mathtt{range}(X, X, Y'_2)$ and vice-versa. No solution are counted twice in $\sum_{Y' \subsetneq Y} \#\mathtt{range}(X, X, Y')$. We have:

$$\#\mathtt{range}(X, X, Y) = \prod_{x_i \in X} d_i - \sum_{Y' \subsetneq Y} \#\mathtt{range}(X, X, Y')$$

□

Remark 6.1. *Proposition 6.2 can be used in Proposition 6.1 and we obtain:*

$$\#\texttt{range}(X, X', Y') = \prod_{x_i \in \overline{X'}} d_i \cdot \left(\prod_{x_i \in X'} d_i(Y') - \sum_{Y'' \subsetneq Y'} \#\texttt{range}(X', X', Y'') \right)$$

This formulae requires to recursively sum and evaluate terms over a exponential-size set and is not tractable in practice (we believe that it is a $\#P-$complete problem). In next subsection, we will give an approximation which is much faster to compute.

We now deal with the `roots` constraint. In the following, we note $d_i(Y') = |D_i \cap Y'|$, the size of the domains D_i reduced to the subset Y'.

Proposition 6.3. *Let $X' \subseteq X$ and $Y' \subseteq Y$. We note $\overline{X'}$, the complement of X' in X and $\overline{Y'}$ the complement of Y' in Y. Then, the number of tuples allowed by $\texttt{roots}(X, X', Y')$ is*

$$\#\texttt{roots}(X, X', Y') = \prod_{x_i \in X'} d_i(Y') \cdot \prod_{x_i \in \overline{X'}} d_i(\overline{Y'}) \tag{6.5}$$

Proof. In order to satisfy $\texttt{roots}(X, X', Y')$, every variable from X' must take a value in Y' and no value from Y' must be assigned to a variable from $\overline{X'}$, that is every variable from $\overline{X'}$ must be assigned to values from $\overline{Y'}$:

- $\prod_{x_i \in X'} d_i(Y')$ represents the number of ways of assigning every variable of X'
- $\prod_{x_i \in \overline{X'}} d_i(\overline{Y'})$ represents the number of ways of assigning every variable of $\overline{X'}$

□

The formula given by Proposition 6.3 is polynomial to compute. In practice, the formula depends on the subsets X' and Y'. Applying the Erdos-Renyi model on `roots` allows the estimation of $\#\textit{roots}(X, X', Y')$ using only the sizes of X' and Y', with a linear complexity.

In section 6.3, we compose these constraints to count solutions on other cardinality constraints.

6.2.2 Probabilistic model applied to `range` and `roots`

This subsection presents a probabilistic model for cardinality constraints based on the work of Erdős and Renyi In Erdős et al. (1963). The idea is to randomize the domain of the variables. Then, we use this model to get a computable estimation of the number of solutions on `range` and `roots`.

We recall here the Erdős-Renyi Model introduced in Chapter 5. The idea is to randomize the domain of each variable such that: for all $x_i \in X$ and for all $y_j \in Y$, the event $\{y_j \in D_i\}$ happens with a predefined probability $p \in [0, 1]$ and all such events are **independent**:

$$\mathbb{P}(\{y_j \in D_i\}) = p \in [0, 1] \tag{6.6}$$

Erdős-Renyi Model applied to range constraint. We study the expectancy of the number of solutions of a range constraint within these random graphs. In the case where every variable of X and every value of Y are constrained, the expectancy of $\#range(X, X, Y)$ is a function of n, m and p (as a reminder, $|X| = n$ and $|Y| = m$). More precisely:

Proposition 6.4. *In the case where every variable of X and every value of Y are constrained, there exists a coefficient $a_{n,m}$ such that:*

$$\mathbb{E}(\#range(X, X, Y)) = a_{n,m} \cdot p^n \tag{6.7}$$

where $\mathbb{E}(\#range(X, X, Y))$ is the expectancy of $\#range(X, X, Y)$ under the hypothesis of the Erdős-Renyi Model.

Proof. To prove this result, we simply reason with a mathematical induction on $|Y| = m$. Let $|X| = n \in \mathbb{N}$.

Base case: Let $Y = \{y\}$ be a singleton. In this particular case, an instance $range(X, X, Y)$ have one allowed tuple, if y is inside every domain D_i, and have zero allowed tuple otherwise. Then,

$$\begin{aligned}
\mathbb{E}(\#range(X, X, \{y\})) &= 0 * \mathbb{P}(\{range(X, X, \{y\}) \text{ have no solution}\}) \\
&\quad + 1 * \mathbb{P}(\{range(X, X, \{y\}) \text{ have one solution}\}) \\
&= \mathbb{P}(\{range(X, X, \{y\}) \text{ have one solution}\}) \\
&= \mathbb{P}(\{\forall x_i \in X, y \in D_i\}) \\
&= \prod_{i=1}^{n} \mathbb{P}(\{y \in D_i\}), \text{ by hypothesis of independence} \\
&= p^n
\end{aligned}$$

We thus set $a_{n,1} = 1$, which proves the result.

Inductive step. We assume that the property is true for all $|Y| = k \in \{1, \dots, m-1\}$: $\forall Y$, such that $1 \leq |Y| = k \leq m-1$, $\exists a_{n,k} \in \mathbb{N}$,

$$\mathbb{E}(\#range(X, X, Y)) = a_{n,k} \cdot p^n$$

We want to prove that, under this assumption, for a set Y with $|Y| = m$, there exists $a_{n,m}$ such that $\mathbb{E}(\#range(X, X, Y)) = a_{n,m} \cdot p^n$

According to Proposition 6.2, we have:

$$
\begin{aligned}
&\mathbb{E}\left(\#range\left(X, X, Y\right)\right) \\
&= \mathbb{E}\left(\prod_{x_i \in X} d_i\right) - \sum_{Y' \subset Y} \mathbb{E}\left(\#range\left(X, X, Y'\right)\right), \text{ by linearity of the operator } \mathbb{E}\left(.\right) \\
&= \mathbb{E}\left(\prod_{x_i \in X} d_i\right) - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k} \cdot p^n, \text{ by hypothesis of induction.} \\
&= \prod_{x_i \in X} \mathbb{E}\left(d_i\right) - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k} \cdot p^n, \text{ by hypothesis of independence} \\
&= (mp)^n - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k} \cdot p^n, \text{ because } \forall x_i \in X, \mathbb{E}\left(d_i\right) = mp \\
&= \left(m^n - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k}\right) \cdot p^n
\end{aligned}
$$

We have identified the coefficient $a_{n,m}$:

$$a_{n,m} = m^n - \sum_{k=1}^{m-1} \binom{m}{k} a_{n,k} \tag{6.8}$$

□

Remarking that $\binom{m}{m} = 1$, we can rewrite 6.8 as follows:

$$m^n = \sum_{k=1}^{m} \binom{m}{k} a_{n,k} \tag{6.9}$$

Also, $\forall n \in \mathbb{N}^+, a_{n,1} = 1$. These coefficients are referenced as the "triangles of numbers" in OEIS.[1] The coefficients $a_{n,m}$ corresponds to the number of possible surjections from a set of cardinal n into a set of cardinal m. The coefficients $a_{n,m}$ are actually equal to $m! \cdot S_2(n, m)$, where $S_2(n, m)$ is the stirling number of second kind. More information about it can be found in Section 1.9 of *Enumerative Combinatorics: Volume 1*.

There is a non-recursive formula to compute these coefficients. The following results is admitted here. An intuition of the proof is that this results is an application of the inclusion-exclusion principle (see Section 1.9. The Twelvefold Way of *Enumerative Combinatorics: Volume 1*).

Proposition 6.5. *For* $0 < m \leq n$,

$$a_{n,m} = \sum_{k=0}^{m} (-1)^k \binom{m}{k} (m-k)^n \tag{6.10}$$

Proposition 6.6 is a property of triangle of numbers and will be used to make some simplifications for future mathematical developments.

Proposition 6.6.

$$a_{n,n} = n! \tag{6.11}$$

[1] https://oeis.org/A019538

Proof. $a_{n,n}$ is the number possible surjections from a set of cardinality n into a set of cardinality n, which is actually the number of bijections in that specific case. □

We can now extend Proposition 6.4 to the case where the `range` constraint only concerns subsets $X' \subseteq X$ and $Y' \subseteq Y$:

Proposition 6.7. *Let $X' \subseteq X$ and $Y' \subseteq Y$. We note $|X'| = n'$ and $|Y'| = m'$.*

$$\mathbb{E}\left(\#\texttt{range}(X, X', Y')\right) = a_{n',m'} \cdot m^{n-n'} \cdot p^n \tag{6.12}$$

Proof. According to Propositions 6.1 and 6.4 and by hypothesis of independence:

$$\begin{aligned}
\mathbb{E}\left(\#\texttt{range}\,(X, X', Y')\right) &= \mathbb{E}\left(\#\texttt{range}\,(X', X', Y')\right) \cdot \mathbb{E}\left(\prod_{x_i \in \overline{X'}} d_i\right) \\
&= a_{n',m'} \cdot p^{n'} \cdot \prod_{x_i \in \overline{X'}} \mathbb{E}\left(d_i\right) \\
&= a_{n',m'} \cdot p^{n'} \cdot (mp)^{n-n'} \\
&= a_{n',m'} \cdot m^{n-n'} \cdot p^n
\end{aligned}$$

□

Erdős-Renyi Model applied to `roots` constraint. We study now the expectancy of the number of solutions of a `roots` constraint.

Proposition 6.8. *Let $X' \subseteq X$ and $Y' \subseteq Y$. We note $|X'| = n'$ and $|Y'| = m'$.*

$$\mathbb{E}\left(\#\texttt{roots}(X, X', Y')\right) = m'^{n'} \cdot (m - m')^{n-n'} \cdot p^n \tag{6.13}$$

Proof. According to Proposition 6.3 and by hypothesis of independence:

$$\begin{aligned}
\mathbb{E}\left(\#\texttt{roots}\,(X, X', Y')\right) &= \mathbb{E}\left(\prod_{x_i \in X'} d_i(Y')\right) \cdot \mathbb{E}\left(\prod_{x_i \in \overline{X'}} d_i(\overline{Y'})\right) \\
&= \prod_{x_i \in X'} \mathbb{E}\left(d_i(Y')\right) \cdot \prod_{x_i \in \overline{X'}} \mathbb{E}\left(d_i(\overline{Y'})\right) \\
&= (m'p)^{n'} \cdot \left((m - m')p\right)^{n-n'} \\
&= m'^{n'} \cdot (m - m')^{n-n'} \cdot p^n
\end{aligned}$$

□

The parameter p corresponds to the density of edges in the value graph. To use the estimators in practice, we need to estimate p: we will later set p to the division of the sum of domains size by the total number of possible edges: $n \cdot m$.

6.3 Generalisation to other Cardinality Constraints

This section details, in a systematic way, how to count solutions for many cardinality constraints thanks to their `range` and `roots` decompositions. Each subsection

first recalls the definitions of the considered constraint, then details its decomposition as extracted from Bessiere et al. (2009) and finally provides the formula for the expectancy of its number of solution in our model.

6.3.1 $alldifferent$

Definition 6.3 (Régin 1994)**.** *A constraint* $alldifferent(X)$ *is satisfied iff each variable* $x_i \in X$ *is instantiated to a value of its domain* D_i *and each value* $y_j \in Y$ *is chosen at most once. We define formally the set of allowed tuples:*

$$\mathcal{S}_{alldifferent(X)} = \{(v_1, \dots, v_n) \in \mathcal{D} \mid \forall i, j \in \{1, \dots, n\}, i \neq j \Leftrightarrow v_i \neq v_j\} \tag{6.14}$$

A decomposition of $alldifferent$ with a $range$ constraint is given by the following:

$$alldifferent(X) \Leftrightarrow range(X, X, Y') \wedge |Y'| = n$$

From this decomposition, we can deduce a formula for the expectancy of the number solutions on an $alldifferent$ constraint, within the Erdős-Renyi Model.

Proposition 6.9.

$$\mathbb{E}(\#alldifferent(X)) = \frac{m!}{(m-n)!} \cdot p^n \tag{6.15}$$

Proof. According to the decomposition of $alldifferent$

$$\#alldifferent(X) = \sum_{Y' \subseteq Y, |Y'|=n|} \#range(X, X, Y')$$

Then,

$$\begin{aligned}\mathbb{E}(\#alldifferent(X)) &= \sum_{Y' \subseteq Y, |Y'|=n|} \mathbb{E}\left(\#range(X, X, Y')\right) \\ &= \binom{m}{n} \cdot a_{n,n} \cdot p^n = \frac{m!}{(m-n)!} \cdot p^n\end{aligned}$$

□

We find again the estimate we developed in Section 5.2, which is comforting since the two methods are based on the same probabilistic model. This method (through constraint decomposition) can be seen as a second way to prove this result.

6.3.2 $nvalue$

Definition 6.4 (Pachet et al. 1999)**.** *The constraint* $nvalue(X, N)$ *holds if exactly* N *values from* Y *are assigned to the variables. Formally:*

$$\mathcal{S}_{nvalue(X,N)} = \{(v_1, \dots, v_n) \in \mathcal{D} |\ N = |\{y_j \in Y | \exists i \in \{1, \dots, n\}, v_i = y_j\}|\} \tag{6.16}$$

A decomposition of $nvalue$ with a $range$ constraint is given by the following:

$$nvalue(X, N) \Leftrightarrow range(X, X, Y') \ \&\ |Y'| = N$$

From this decomposition, we can deduce a formula to estimate solutions on a $nvalue$ constraint, within the Erdős-Renyi Model.

Proposition 6.10. *Let $N \in \mathbb{N}$,*

$$\mathbb{E}\left(\#nvalue(X, N)\right) = \binom{m}{N} \cdot a_{n,N} \cdot p^n \tag{6.17}$$

Proof. The proof is the same as Proposition 6.9 □

We can generalize Proposition 6.9 to the case where N is a variable. The set of solutions for two different values of N are disjoints, then we can simply sum this estimates on the domain of N to compute an estimate in the general case.

6.3.3 `atmost_nvalue` and `atleast_nvalue`

From Proposition 6.10, we can deduce the expectancy of the number of solutions on `atmost_nvalue` and `atleast_nvalue`.

Definition 6.5 (`atmost_nvalue`, Bessière et al. 2005b)**.** *The constraint* $\texttt{atmost\_nvalue}(X, N)$ *holds if at most N values from Y are assigned to the variables. Formally:*

$$\mathcal{S}_{\texttt{atmost\_nvalue}(X,N)} = \{(v_1, \dots, v_n) \in D |\ N \geq |\{y_j \in Y | \exists x_i \in X, v_i = y_j\}|\} \tag{6.18}$$

Definition 6.6 (`atleast_nvalue`, Bessière et al. 2005b)**.** *The constraint* $\texttt{atleast\_nvalue}(X, N)$ *holds if at least N values from Y are assigned to the variables. Formally:*

$$\mathcal{S}_{\texttt{atleast\_nvalue}(X,N)} = \{(v_1, \dots, v_n) \in D |\ N \leq |\{y_j \in Y | \exists x_i \in X, v_i = y_j\}|\} \tag{6.19}$$

We can decompose `atleast_nvalue` and `atmost_nvalue` directly with the constraint `nvalue`:

$$\texttt{atmost\_nvalue}(X, N) \Leftrightarrow \texttt{nvalue}(X, k)\ \&\ k \leq N$$

$$\texttt{atleast\_nvalue}(X, N) \Leftrightarrow \texttt{nvalue}(X, k)\ \&\ k \geq N$$

It follows that:

Proposition 6.11.

$$\mathbb{E}\left(\#\texttt{atmost\_nvalue}(X, N)\right) = \sum_{k=1}^{N} \binom{m}{k} a_{n,k} \cdot p^n \tag{6.20}$$

$$\mathbb{E}\left(\#\texttt{atleast\_nvalue}(X, N)\right) = \sum_{k=N}^{n} \binom{m}{k} a_{n,k} \cdot p^n \tag{6.21}$$

6.3.4 *among*

Definition 6.7 (Beldiceanu and Contejean 1994)**.** *Let $Y' \subseteq Y$. The constraint* $among(X, Y', N)$ *holds iff exactly N variables are assigned to value from Y'.*

$$\mathcal{S}_{among(X,Y',N)} = \{(v_1, \dots, v_n) | N = |\{x_i | v_i \in Y'\}|\}$$

The decomposition of *among* is given by the following equivalence:

$$among(X, Y', N) \Leftrightarrow roots(X, X', Y') \wedge |X'| = N$$

Proposition 6.12. *Let $m' = |Y'|$ and $N \in \mathbb{N}$,*

$$\mathbb{E}\left(\#among(X, Y', N)\right) = \binom{n}{N} m'^{N}(m - m')^{n-N} \cdot p^n \tag{6.22}$$

Proof. According to the decomposition of *among*, we can write:

$$\#among(X, Y', N) = \sum_{X' \subseteq X, |X'|=N} \#roots(X, X', Y')$$

Indeed, for two different subsets $X'_1, X'_2 \subseteq X$, the sets of solutions of $roots(X, X'_1, Y')$ and $roots(X, X'_2, Y')$ have an empty intersection, then no solution is counted twice. And:

$$\begin{aligned}
\mathbb{E}\left(\#among(X, Y', N)\right) &= \sum_{X' \subseteq X, |X'|=N} \mathbb{E}\left(\#roots(X, X', Y')\right) \\
&= \sum_{X' \subseteq X, |X'|=N} m'^{m'}(m - m')^{n-|X'|} \cdot p^n, \text{ by Proposition 6.8} \\
&= \binom{n}{N} m'^{m'}(m - m')^{n-N} \cdot p^n
\end{aligned}$$

□

In the same way as for *nvalue*, we can generalize Proposition 6.12 to the case where N is a variable.

6.3.5 *occurrence*

Definition 6.8 (Carlsson et al. 2014)**.** *Let $y \in Y$, the constraint $occurrence(X, y, N)$ holds iff exactly N variables are assigned to value y.*

$$\mathcal{S}_{occurrence(X,y,N)} = \{(v_1, \dots, v_n) | N = |\{x_i | v_i = y\}|\}$$

The decomposition of *occurrence* is given by the following equivalence:

$$occurrence(X, y, N) \Leftrightarrow roots(X, X', \{y\}) \wedge |X'| = N$$

Proposition 6.13. *Let $N \in \mathbb{N}$,*

$$\mathbb{E}\left(\#occurrence(X, y, N)\right) = \binom{n}{N} (m - 1)^{n-N} \cdot p^n \tag{6.23}$$

Proof. The proof is the same as Proposition 6.12 in the case where $Y' = \{y\}$ is a singleton. □ □

Proposition 6.13 can also be generalized to the case where N is a variable.

6.3.6 `atmost` and `atleast`

Definition 6.9 (`atmost`, Dincbas et al. 1988)**.** *Let $y \in Y$, the constraint $\texttt{atmost}(X, y, N)$ holds iff at most N variables are instantiated to the value y. Formally:*

$$\mathcal{S}_{\texttt{atmost}(X,y,N)} = \{(v_1, \dots, v_n) \in D |\ N \geq |\{x_i \in X | v_i = y\}|\} \tag{6.24}$$

Definition 6.10 (`atleast`, Dincbas et al. 1988)**.** *Let $y \in Y$, the constraint $\texttt{atleast}(X, y, N)$ holds iff at least N variables are instantiated to the value y. Formally:*

$$\mathcal{S}_{\texttt{atmost}(X,y,N)} = \{(v_1, \dots, v_n) \in D |\ N \leq |\{x_i \in X | v_i = y\}|\} \tag{6.25}$$

The decomposition of the constraints `atmost` and `atleast` in `range` et `roots` are:

$$\texttt{atmost}(X, y, N) \Leftrightarrow \texttt{roots}(X, X', \{y\}) \ \&\ |X'| \leq N$$

$$\texttt{atleast}(X, y, N) \Leftrightarrow \texttt{roots}(X, X', \{y\}) \ \&\ |X'| \geq N$$

Proposition 6.14.

$$\mathbb{E}(\#\texttt{atmost}(X, y, N)) = \sum_{k=1}^{N} \binom{n}{k} (m-1)^{n-k} \cdot p^n \tag{6.26}$$

$$\mathbb{E}(\#\texttt{atleast}(X, y, N)) = \sum_{k=N}^{n} \binom{n}{k} (m-1)^{n-k} \cdot p^n \tag{6.27}$$

Proof. We develop the proof only for the `atmost` constraint. On a:

$$\#\texttt{atmost}(X, y, N) = \sum_{X' \subseteq X, |X'| \leq N} \#\texttt{roots}(X, X', \{y\})$$

Indeed, for two different subsets $X'_1 \neq X'_2 \subseteq X$, the set of solutions of $\texttt{roots}(X, X'_1, \{y\})$ and $\texttt{roots}(X, X'_2, \{y\})$ have an empty intersection. No solution are counted twice. And:

$$\begin{aligned}
\mathbb{E}(\#\texttt{atmost}(X, y, N)) &= \sum_{X' \subseteq X, |X'| \leq N} \mathbb{E}(\#\texttt{roots}(X, X', \{y\})) \\
&= \sum_{X' \subseteq X, |X'| \leq N} (m-1)^{n-|X'|} \cdot p^n, \text{ by Proposition 6.8} \\
&= \sum_{k=1}^{N} \binom{n}{k} (m-1)^{n-k} \cdot p^n
\end{aligned}$$

□

6.3.7 `uses`

Definition 6.11 (`uses`, Bessière et al. 2005c)**.** *Let $X_1 \subseteq X$ and $X_2 \subseteq X$ be two subsets of variables. The constraint $\texttt{uses}(X, X_1, X_2)$ holds iff the set of values taken by variables of X_2 is a subset of the set of values taken by variables of X_1. Formally:*

$$\mathcal{S}_{\texttt{uses}(X,X_1,X_2)} = \{(v_1, \dots, v_n) \in D |\ \{y_j | \exists x_i \in X_2, v_i = y_j\} \subseteq \{y_j | \exists x_i \in X_1, v_i = y_j\}\} \tag{6.28}$$

The constraint `uses` can be expressed with a `range` and a `roots` constraints:

$$\texttt{uses}(X, X_1, X_2) \Leftrightarrow \texttt{range}(X, X_1, Y') \;\&\; \texttt{roots}(X, X_2, Y')$$

Proposition 6.15. *Let $X_1, X_2 \subseteq X$ such that $X_1 \cap X_2 = \emptyset$. We note $n_1 = |X_1|$ and $n_2 = |X_2|$.*

$$\mathbb{E}\left(\#\texttt{uses}(X, X_1, X_2)\right) = m^{n-n_1-n_2} \cdot \sum_{k=1}^{m} \binom{m}{k} a_{n_1,k} k^{n_2} \cdot p^n \tag{6.29}$$

Proof. From the decomposition of the constraint `uses`, we can write:

$$\#\texttt{uses}(X, X_1, X_2) = \prod_{x_i \in X \setminus \{X_1 \cup X_2\}} |D_i| \cdot \sum_{Y' \subseteq Y} \#\texttt{range}(X_1, X_1, Y') \cdot \#\texttt{roots}(X_2, X_2, Y')$$

The first factor of the product corresponds to the number of possible combinations for the unconstrained variables. Then, for each subset of values $Y' \subseteq Y$, we count the number of possible instantiations for the constrained variables of X_1 and multiply it by the number of possible instantiations for the constrained variables of X_2.

By assumption of independence, we have:

$$\mathbb{E}\left(\prod_{x_i \in X \setminus \{X_1 \cup X_2\}} |D_i|\right) = (mp)^{n-n_1-n_2}$$

Also, according to Proposition 6.7 and Proposition 6.8, we can write:

$$\mathbb{E}\left(\#\texttt{range}(X_1, X_1, Y')\right) = a_{n_1,|Y'|} p^{n_1}$$

And,

$$\mathbb{E}\left(\#\texttt{roots}(X_2, X_2, Y')\right) = (|Y'| \cdot p)^{n_2}$$

Finally,

$$\begin{aligned}\mathbb{E}\left(\sum_{Y' \subseteq Y} \#\texttt{range}(X_1, X_1, Y') \cdot \#\texttt{roots}(X_2, X_2, Y')\right) &= \sum_{Y' \subseteq Y} a_{n_1,|Y'|} p^{n_1} \cdot (|Y'| \cdot p)^{n_2} \\ &= \sum_{k=1}^{m} \binom{m}{k} a_{n_1,k} k^{n_2} \cdot p^{n_1+n_2}\end{aligned}$$

Then, we can conclude:

$$\mathbb{E}\left(\#\texttt{uses}(X, X_1, X_2)\right) = m^{n-n_1-n_2} \cdot \sum_{k=1}^{m} \binom{m}{k} a_{n_1,k} k^{n_2} \cdot p^n$$

□

6.3.8 `disjoint`

Definition 6.12 (`disjoint`, Nicolas Beldiceanu 2001)**.** *Let $X_1 \subset X$ and $X_2 \subset X$, such that $X_1 \cap X_2 = \emptyset$, $\texttt{disjoint}(X, X_1, X_2)$ holds iff none of the variables from X_1 is assigned to a value that one of the variable from X_2 takes. Formally:*

$$\mathcal{S}_{\texttt{disjoint}(X,X_1,X_2)} = \{(v_1, \dots, v_n) \in D |\ \{v_i | x_i \in X_1\} \cap \{v_i | x_i \in X_2\} = \emptyset\} \tag{6.30}$$

Bessière et al. (2005b) give a decomposition of $\mathit{disjoint}$ with two constraints range:

$$\mathit{disjoint}(X, X_1, X_2) \Leftrightarrow \\ \mathit{range}(X, X_1, Y_1)\ \&\ \mathit{range}(X, X_2, Y_2)\&\ Y_1 \subset Y\ \&\ Y_2 \subset Y\ \&\ Y_1 \cap Y_2 = \emptyset$$

We deduce, under the hypothesises of the Erdős-Renyi Model, the following formulae to estimate the number of solutions on $\mathit{disjoint}$.

Proposition 6.16. *Let $X_1, X_2 \in X$ with $|X_1| = n_1$ and $|X_2| = n_2$ and $X_1 \cap X_2 = \emptyset$*

$$\mathbb{E}((\#\mathit{disjoint}(X, X_1, X_2)) = \\ p^n \cdot m^{n-n_1-n_2} \cdot \sum_{k=1}^{min(n_1,m)} \sum_{l=1}^{min(n_2,m-k)} \binom{m}{k}\binom{m-k}{l} a_{n_1,k} a_{n_2,l}$$

Proof. From the decomposition of the $\mathit{disjoint}$ constraint, we have:

$$\#\mathit{disjoint}(X, X_1, X_2) = \\ \left(\prod_{x_i \in X \setminus \{X_1 \cup X_2\}} d_i\right) \cdot \left(\sum_{Y_1 \cap Y_2 = \emptyset} \#\mathit{range}(X_1, X_1, Y_1) \cdot \#\mathit{range}(X_2, X_2, Y_2)\right)$$

The first factor of the product is the total number of combinations for the unconstrained variables. Once these variables are instantiated, we have to choose two subsets Y_1 and Y_2 such that $Y_1 \cap Y_2 = \emptyset$ and compute the number of solutions of $\mathit{range}(X_1, X_1, Y_1)$ and $\mathit{range}(X_2, X_2, Y_2)$. For two different pairs of subsets (Y_1^A, Y_2^A) and (Y_1^B, Y_2^B), the set of solutions of $\{\mathit{range}(X_1, X_1, Y_1^A)\&\mathit{range}(X_2, X_2, Y_2^A)\}$ and $\{\mathit{range}(X_1, X_1, Y_1^B)\&\mathit{range}(X_2, X_2, Y_2^B)\}$ are disjoints. Thus, no solution is counted twiced.

Also, we have:

$$\mathbb{E}\left(\prod_{x_i \in X \setminus \{X_1 \cup X_2\}} d_i\right) = \prod_{x_i \in X \setminus \{X_1 \cup X_2\}} \mathbb{E}(d_i) = (mp)^{n-n_1-n_2}$$

And by Proposition 6.4 and by the independence assumption:

$$\mathbb{E}(\sum_{Y_1 \cap Y_2 = \emptyset} \#\mathit{range}(X_1, X_1, Y_1) \cdot \#\mathit{range}(X_2, X_2, Y_2)) \\ = \sum_{Y_1 \cap Y_2 = \emptyset} a_{n_1,|Y_1|} p^{n_1} \cdot a_{n_2,|Y_2|} p^{n_2}$$

Each pair (Y_1, Y_2) with fixed sizes $|Y_1| = k$ and $|Y_2| = l$ ends to a same formuale, then we choose first a subset of values Y_1 of size $k \in \{1, \dots, min(n_1, m)\}$ and then a subset of values Y_2 of size $l \in \{1, \dots, min(n_2, m-k)\}$:

$$\sum_{Y_1 \cap Y_2 = \emptyset} a_{n_1,|Y_1|} \cdot a_{n_2,|Y_2|} = \sum_{k=1}^{min(n_1,m)} \sum_{l=1}^{min(n_2,m-k)} \binom{m}{k}\binom{m-k}{l} a_{n_1,k} a_{n_2,l}$$

Multiplying all these factors, we deduce Proposition 6.16 □

Constraint	Formula with $\lvert X\rvert = n$, $\lvert X_1\rvert = n_1$, $\lvert X_2\rvert = n_2$, $\lvert Y\rvert = m$ and $\lvert Y'\rvert = m'$
$alldifferent(X)$	$\frac{m!}{(m-n)!} \cdot p^n$
$among(X, Y', N)$	$\binom{n}{N} m'^N (m-m')^{n-N} \cdot p^n$
$nvalue(X, N)$	$\binom{m}{N} \cdot a_{n,N} \cdot p^n$
$atmost\_nvalue(X, N)$	$\sum_{k=1}^{N} \binom{m}{k} a_{n,k} \cdot p^n$
$atleast\_nvalue(X, N)$	$\sum_{k=N}^{n} \binom{m}{k} a_{n,k} \cdot p^n$
$occurrence(X, y, N)$	$\binom{n}{N} (m-1)^{n-N} \cdot p^n$
$atmost(X, y, N)$	$\sum_{k=1}^{N} \binom{n}{k} (m-1)^{n-k} \cdot p^n$
$atleast(X, y, N)$	$\sum_{k=N}^{n} \binom{n}{k} (m-1)^{n-k} \cdot p^n$
$uses(X, X_1, X_2)$	$m^{n-n_1-n_2} \cdot \sum_{k=1}^{m} \binom{m}{k} a_{n_1,k} k^{n_2} \cdot p^n$
$disjoint(X, X_1, X_2)$	$m^{n-n_1-n_2} \cdot \sum_{k=1}^{min(n_1,m)} \sum_{l=1}^{min(n_2,m-k)} \binom{m}{k} \binom{m-k}{l} a_{n_1,k} a_{n_2,l} \cdot p^n$

TABLE 6.1: Counting formulae extracted from *range* and *roots* reformulation

6.3.9 Synthesis

We report the estimators of the number of solutions in Table 3 for several cardinality constraints. We observe a pattern in all these formulae: the estimation of the number of allowed tuples is always p^n multiplied by the number of tuples allowed by the constraint if every domain were equal to the set of values Y (if the value graph were complete). This remark leads to the following Proposition.

Proposition 6.17. *Let C be a constraint over X with $\lvert X\rvert = n$, Y be the union of the domains and p the edge density in the value graph $G_{X,Y}$, then:*

$$\mathbb{E}(\#C) = \#C^* \cdot p^n \tag{6.31}$$

with $\#C^*$ the number of allowed tuples if $G_{X,Y}$ were complete.

Proof. Let $\mathcal{S}_{C^*}$ be the set of allowed tuples if $G_{X,Y}$ were complete. For each $s \in \mathcal{S}_{C^*}$, let Z_s be the random variable such that, $Z_s = 1$ if s is in the set of allowed tuples $\mathcal{S}_C$ of C, and $Z_s = 0$ otherwise. A solution s is an instantiation of every variable, then, in the Erdős-Renyi Model, $\mathbb{P}(\{Z_s = 1\}) = \mathbb{E}(Z_s) = p^n$. Then,

$$\mathbb{E}(\#C) = \mathbb{E}\left(\sum_{s \in \mathcal{S}_{C^*}} Z_s\right) = \sum_{s \in \mathcal{S}_{C^*}} \mathbb{E}(Z_s) = \#C^* \cdot p^n$$

□

In Section II, we have shown how to count solutions on a *range* and a *roots* constraints and in Section 6.3, how to use the *range* and *roots* / decomposition to estimate the number of solutions on many cardinality constraints. Proposition 6.17 highlights a general pattern for such estimates. In Section 6.4, we experiment these probabilistic estimators within counting-based heuristics on some problems using cardinality constraints.

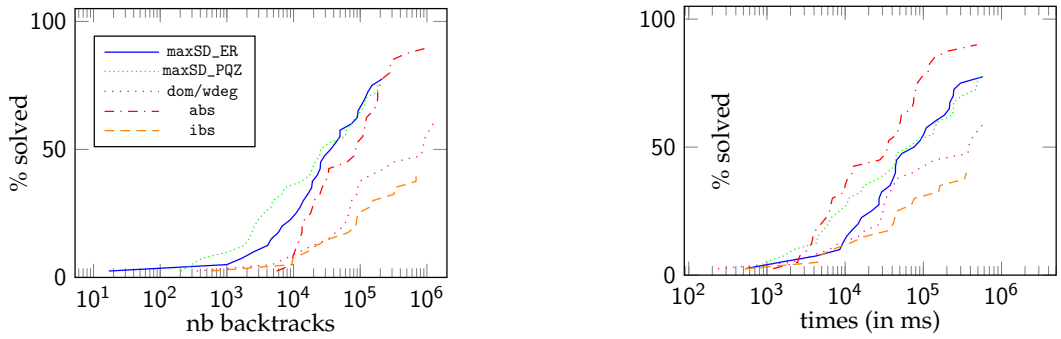


FIGURE 6.2: Performances of `maxSD_ER`, `maxSD_PQZ`, `dom/wdeg`, `ibs` and `abs` on 40 hard Latin Square instances, in number of backtracks (left) and time (right).

6.4 Experimental Analysis

In this section, we present two problems, on which we have run different heuristics: `maxSD` (Pesant et al. 2012), `dom/wdeg` (Boussemart, Hemery, Lecoutre, and Sais 2004b), `abs` (activity-based search) (Michel et al. 2012) and `ibs` (impact-based search) (Refalo 2004). This benchmark has been chosen by taking the problems in XSCP, CSPLib, MiniZinc which matched our testing needs: no COP, with cardinality constraints at the core of the problem but no gcc. Also, the lack of knowledge on how to use `maxSD` on problems with several constraints restricts a lot the practical use of the heuristic. These conditions restricted our benchmark to Latin Squares and Sports Tournament Scheduling.

We actually run `maxSD` as suggested by Gagnon et al. (2018): we re-compute the ordering of the variables only when the product of the domains size have decreased enough. Here, we set a threshold at 20%. Also, the coefficients $a_{n,m}$, the binomial coefficients and the factorials are computed in advance. The computation of the approximations is thus made in linear time in n.

We first introduce the problem and the cardinality constraints that are used in the model and then compare their efficiency in terms of solving time and number of required backtracks. The instances and the strategies are implemented in Choco solver (Prud'homme et al. 2016) and we run them on a 2.2GHz Intel Core i7 with 2.048GB.

6.4.1 Latin Square Problem

A Latin Square problem is defined by a $n * n$ grid whose squares each contain an integer from 1 to n such that each integer appears exactly once per row and column [2]. The model uses a matrix of integer variables and an *alldifferent* constraint for each row and each column. We tested on the 40 hard instances used by Pesant et al. (2012) with n = 30 and 42% of holes (corresponding to the phase transition), generated following C. Gomes et al. (2002). For these instances, we also compare our probabilistic estimator (`maxSD_ER`) with the estimator that is proposed by Pesant et al. (2012) (`maxSD_PQZ`) for *alldifferent*. We set a time limit to 10min.

Figures 6.2 represent the percentage of solved instances in function of the number of required backtracks, and of the solving time. The strategies `maxSD` (for both

[2] Gilles Pesant. *CSPLib Problem 067: Quasigroup Completion*. Ed. by Christopher Jefferson et al. `http://www.csplib.org/Problems/prob067`

n	n=6	n=8	n=10	n=12	n=14
maxSD_ER	60 (0.239)	**10** (0.707)	**1056** (3.587)	**74168 (92.396)**	**37883 (128.272)**
ibs	**3 (0.172)**	214 (**0.648**)	1232 (**1.865**)	TO	TO
abs	101 (0.077)	3081 (0.692)	246767 (24.207)	TO	TO
dom/wdeg	89380 (3.829)	TO	TO	TO	TO

TABLE 6.2: Number of backtracks (time in s) for different n

estimators maxSD_ER and maxSD_PQZ) and abs performed better than dom/wdeg and ibs. abs solved more instances than the two versions of maxSD, but required more backtracks. maxSD seems to perform better on the easiest instances (in term of number of backtracks). maxSD_PQZ has slightly better performances than maxSD_ER on the medium instances and have very comparable performances on the hardest ones.

6.4.2 Sports Tournament Scheduling Problem

The sports tournament scheduling problem [3] consists in scheduling a tournament of n teams over $n-1$ weeks, with each week divided into $n/2$ periods, and each period divided into two slots. A tournament must satisfy the following three constraints: every team plays once a week; every team plays at most twice in the same period over the tournament; every team plays every other team. The first and the third constraint are modeled with an *alldifferent* constraint and the second one is modeled with an *atmost* constraints. We run this problem with the different settings: $n \in \{6, 8, 10, 12, 14\}$.

In table 6.2, we report the number of backtracks required (and the time required) to solve the problem for different values of n with four different heuristics. Here maxSD_PQZ cannot be used as there is no estimator for *atmost* in the previous work of Pesant et al. (2012). Consequently, we only focused on our approach maxSD_ER. We fixed a time limit to 5min. We observe that maxSD_ER outperforms abs and dom/wdeg. For $n \in \{6, 8, 10\}$, maxSD_ER and ibs have similar performances but ibs could not find a solution in less than 5min for $n = 12$ and $n = 14$.

We have shown that our probabilistic estimator for *alldifferent* gives very comparable result than the estimator given by Pesant et al. (2012) on the Latin Square instances. Also our estimators within maxSD_ER gives better results than ibs, abs and dom/wdeg on the Sport Tournament Scheduling problem.

6.5 Conclusion

In this chapter, we have presented a method to estimate the number of solutions of the *range* and *roots* constraints with a probabilistic Erdős-Renyi Model. We detailed the method to estimate the number of solutions of ten cardinality constraints using their *range* and *roots* decompositions. We highlighted a general formula to compute such an estimation on cardinality constraints. We have implemented the heuristic maxSD_ER with these new probabilistic estimators and compare their efficiency to dom/wdeg, abs, and ibs.

We think that the main asset of this approach is its systematic nature. We have shown here an application of counting solutions for counting based search. Such an

[3] Toby Walsh. *CSPLib Problem 026: Sports Tournament Scheduling*. Ed. by Christopher Jefferson, Ian Miguel, Brahim Hnich, Toby Walsh, and Ian P. Gent. http://www.csplib.org/Problems/prob026

approach could also be used, for example, for uniform random instances generation, probabilistic reasoning or search space structure analysis.

We did not study the `gcc` constraint, as its decomposition involves several non-disjoint subsets of the variables. Further research includes extending our approach to the case where several `range` and `roots` constraints may apply to a common set of variables. This will lead us to estimators of the number of solutions for conjunctions of cardinality constraints, or `gcc` constraints.

Conclusion

Counting Solutions on Global Cardinality Constraint

The global cardinality constraint is a very difficult constraint to tackle regarding its filtering algorithms or counting techniques. In Chapter 3, we have developed a counting method that computes an upper bound of the number of solutions on *gcc* and that fixes the first result established by Pesant et al. (2012). We have implemented the former result and its correction within the `maxSD` strategy and proposed an experimental protocol to compare their efficiency as estimators of the true number of solutions on *gcc*. The experimental analysis showed that both estimators behave in a very similar way within the `maxSD` strategy. It demonstrates that, for the use of counting-based strategies, the correlation between an estimator and the actual number of solutions prevails over its accuracy.

In Chapter 4, we have investigated an original way to count solutions on the global cardinality constraint under a group theory point of view. We have shown how to properly consider solutions symmetries in *gcc* by using the Burnside's lemma. We have established another upper bound of the number of solutions on *gcc*. Although this approach seems perfectly appropriate for this counting problem, this latter upper bound is largely dominated by the upper bound we developed in Chapter 3. Yet, we think that this group theory approach is not to be set aside. The choice we have made in Section 4.3 for the decomposition of the symmetry group probably deteriorates a lot the quality of the upper bound. This arbitrary decomposition appeared as the most natural for us and we address as future work to find a better decomposition. This application of the Burnside's lemma on the global cardinality constraint might also be used as an inspiration for counting solutions on other constraints, in particular if there are non-trivial symmetries to deal with.

Probabilistic Models for Cardinality Constraint

The second main contribution of this thesis is to study cardinality constraints under a probabilistic point of view and to perform an average-case analysis of their behaviour. In Chapter 5, we have introduced probabilistic models, inspired by random graph models, for the *alldifferent* constraint. We have studied the number of solutions on *alldifferent* with two different random graph models. From the Erdős-Renyi model, we have developed an estimator of the number of solutions based on the edge density of the value graph. The Fixed Domain Size model gives an estimator based on the distribution of the domains sizes, which is more accurate. We have shown that both estimators can be implemented within counting-based heuristics and that it gives very comparable results to the state of the art on some classes of problems. We notice that the accuracy of our second estimator, based on the domains sizes distribution, does not improve the efficiency of the counting-based strategy compared to the performance of our first estimator, based on the edge density. This result tends to confirm the hypothesis that the correlation between an

estimator and the true number of solutions is much more important than its accuracy for these counting-based heuristics.

We have shown how to apply the asymptotic result on random graphs from Erdős et al. (1963) in the case of the *alldifferent* constraint to study the satisfiability of big instances. Predicting the satisfiability of an *alldifferent* instance or, more generally, of an instance of any kind of constraint can be very beneficial to CP solvers. We can think of, for example, propagating first on the constraints that are likely unsatisfiable, as a way to implement the fail-first principle. We can also think of designing search strategies based on the number of constraints that remain likely satisfiable or not. Applying average-case analysis on constraints might be very valuable for CP solvers and we think that Chapter 5 is a great introduction to such probabilistic reasoning on constraints.

In Chapter 6, we have extended our probabilistic reasoning on solutions counting to most of the cardinality constraints. The technique we have developed is based on the *range* and *roots* decomposition. We have shown how to count exactly the number of solutions on the *range* and the *roots* constraints. Then we have developed computable estimators, based on the edge density, by applying the Erdős-Renyi random graph model on these two constraints. We deduce the exact number of solutions and an estimate of it for many cardinality constraints, by taking advantage of the *range* and *roots* decomposition. The main asset of this counting method is that it does not require to develop a specific counting model for each cardinality constraint. We have highlighted a general pattern for the estimators based on Erdős-Renyi model. We have proved that these estimates can be directly deduced by counting solutions on the constraint considering that the value graph is complete, which is decreasing the combinatorial complexity a lot. However, we could not establish any probabilistic estimator for the global cardinality constraint. Unlike the other cardinality constraints, the *range* and *roots* decomposition of *gcc* carries on non-independent sets of variables. Our method does not fit well to this case. Actually, this issue is the same as one of the main limitation of counting-based search: it is very hard to count solutions on a conjunction of constraints in the general case.

Further Research

This thesis presents many counting techniques and probabilistic-based reasonings for constraint programming and there are still a lot to explore in that direction. In Section 5.3, we have presented an asymptotic study of the satisfiability for the *alldifferent* constraint and highlighted a phase transition. It would be interesting to extend this asymptotic behaviour study and highlight other phase transitions for other cardinality constraints. Also, this asymptotic study of the satisfiability has been done regarding the edge density in the value graph. We could get a finer analysis on cardinality constraints if we consider the domains distribution instead of the edge density only. This is a much more difficult task to achieve, as the domains distribution is a n-dimensional vector and not just a real parameter like the edge density.

We decided to focus on cardinality constraints because they can all be modeled in a similar way, with bipartite graphs, and there were a lot of combinatoric tools on graphs that we could adapt to these constraints. Another interesting challenge would be to extend this study to other constraints. Counting and performing average-case analysis on other constraints require other combinatorial materials. In the field of enumerative combinatorics, that studies the existence and properties of

patterns, in its general meaning, there are many results that could be used for sequence constraints or time series constraints for instance. For the constraint `regular`, that deals with the recognition of words in an automaton, there might be interesting results to find around Markov chain theory.

The main limitation of counting-based search is that it focuses on each constraint separately. It would be much more efficient to reason on conjunction of constraints. Counting the number of solutions on a conjunction of constraints is very hard in the general case because it implies to consider the overriding of several complex structures. Here again, there might be average-case analysis tools that could help us capturing this complexity. A first question would be, given a set of n variables and k constraints of the same type (all $alldifferent$ for example) over these variables, how many solutions are there in average considering the conjunction of the k constraints and the domain size of the variables. A second concern is the identification of critical sets of constraints that, put together, constrained the problem much more than the other constraints. Identifying these conjunctions of constraints would allow a better understanding of the problem and why it is difficult. It would also make possible the automatic configuration of a strategy by first focusing on these critical sets of constraints.

An other idea of application is the generation of random solutions. Given a problem, we want to construct randomly a solution, uniformly chosen among the solution set. There already exists random heuristics in CP, that selects randomly and uniformly a variable and a value of its domain, but it does not generate uniformly a solution of the problem in the end. However, if we add weights, in the random selection, to pairs variable/value proportionally to the corresponding number of remaining solutions after the instantiation, the random generation would be uniform. This is an interesting tool to develop for the generation of benchmarks when it is important to sample a wide range of different instances of problems in order to cover many combinatorial specificities.

This thesis has been the occasion to explore a lot of combinatoric tools and to raise many theoretical questions, in particular concerning random graphs and the computation of the permanent. The permanent is a very fascinating mathematical tool and its computation belongs to this class of problems that can be stated very simply but require exponential-time methods to be solved. This is all the more surprising regarding the similarity between the permanent and the determinant definitions, the latter one being computable in polynomial time. The permanent is central to many combinatoric problems and there are many studies that propose estimations or upper bounds. An interesting question is how to estimate the permanent of a matrix from the sums over its rows and columns. This corresponds to the fact that we know the degrees of each node in our value graph (variables nodes and values nodes). This question, which appeared as a side-effect during our work on random graph models, should be investigated, in particular in the case of constraints studies.

Some other theoretical questions on random graphs that interested us are: given a predefined distribution of degrees, how do we pick up uniformly randomly such a graph and how many such graphs are there? These questions are very challenging and answering it could help a lot, for example, developing the Fixed Degrees random model presented in Section 5.1.

One of the main difficulty of these questions is that they are overriding with some topics from the analytic combinatorics field, in which the research is often focus on more theoretical matters such as the asymptotic behaviour on specific structures. Asymptotic study on constraints can give information on the behaviour of

biggest instances (see Section 5.3) but it is not always easy to find answers from this discipline to questions that matter more to the CP community, in particular to estimate finite quantities. The work we have developed shows that tools from graph theory or group theory can provide useful insight on practical questions in combinatorial optimization. There are many more subjects to investigate for the average-case analysis of constraints, which may also raise interests in the analytic combinatorics community.

Bibliography

[1] Paul R. Halmos and Herbert E. Vaughan. "The Marriage Problem". In: *American Journal of Mathematics* 72.1 (1950), pp. 214–215. ISSN: 00029327, 10806377. URL: http://www.jstor.org/stable/2372148.

[2] Claude Berge. "Two Theorems in Graph Teory". In: *Proceedings of the National Academy of Sciences* 43.9 (1957), pp. 842–844. ISSN: 0027-8424. DOI: 10.1073/pnas.43.9.842. eprint: https://www.pnas.org/content/43/9/842.full.pdf. URL: https://www.pnas.org/content/43/9/842.

[3] D. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton, NJ, USA: Princeton University Press, 1962. ISBN: 0691146675, 9780691146676.

[4] P. Erdős and A. Renyi. "On random matrices". In: *Publication of the Mathematical Institute of the Hungarian Academy of Science* (1963).

[5] C. Berge. *Graphes et hypergraphes*. Dunod Université. Dunod, 1973. URL: https://books.google.fr/books?id=Gu7uAAAAMAAJ.

[6] L. M. Brégman. "Some Properties of Nonnegative Matrices and their Permanents." In: *Soviet Mathematics Doklady* 14(4) (1973), pp. 945–949.

[7] Jean-Louis Laurière. "A Language and a Program for Stating and Solving Combinatorial Problems". In: *Artif. Intell.* 10.1 (1978), pp. 29–127. DOI: 10.1016/0004-3702(78)90029-2. URL: https://doi.org/10.1016/0004-3702(78)90029-2.

[8] Leslie G. Valiant. "The Complexity of Computing the Permanent". In: *Theor. Comput. Sci.* 8 (1979), pp. 189–201. DOI: 10.1016/0304-3975(79)90044-6. URL: https://doi.org/10.1016/0304-3975(79)90044-6.

[9] C. D. Godsil and I. Gutman. "On the theory of the matching polynomial". In: *Journal of Graph Theory* 5.2 (1981), pp. 137–144. DOI: 10.1002/jgt.3190050203. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jgt.3190050203. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/jgt.3190050203.

[10] Mehmet Dincbas et al. "The Constraint Logic Programming Language CHIP". In: *FGCS*. 1988, pp. 693–702.

[11] Abderrahmane Aggoun and Nicolas Beldiceanu. "Extending CHIP in order to solve complex scheduling and placement problems". In: *JFPL'92, 1ères Journées Francophones de Programmation Logique, 25-27 Mai 1992, Lille, France*. Ed. by Jean-Paul Delahaye et al. 1992, p. 51.

[12] David G. Mitchell, Bart Selman, and Hector J. Levesque. "Hard and Easy Distributions of SAT Problems". In: *Proceedings of the 10th National Conference on Artificial Intelligence, San Jose, CA, USA, July 12-16, 1992.* Ed. by William R. Swartout. AAAI Press / The MIT Press, 1992, pp. 459–465. ISBN: 0-262-51063-4. URL: http://www.aaai.org/Library/AAAI/1992/aaai92-071.php.

[13] N Beldiceanu and E Contejean. "Introducing global constraints in CHIP". In: *Mathematical and Computer Modelling* 20.12 (1994), pp. 97–123. ISSN: 0895-7177. DOI: https://doi.org/10.1016/0895-7177(94)90127-9. URL: http://www.sciencedirect.com/science/article/pii/0895717794901279.

[14] Peter M. Neumann, Gabrielle A. Stoy, and Edward C. Thompson. *Groups and Geometry*. Oxford University Press, 1994. ISBN: 0198534515.

[15] W.P.M. Nuijten. "Time and resource constrained scheduling : a constraint satisfaction approach". English. PhD thesis. TUE : Department of Mathematics and Computer Science, 1994. ISBN: 90-386-0224-3. DOI: 10.6100/IR431902.

[16] Jean-Charles Régin. "A Filtering Algorithm for Constraints of Difference in CSPs". In: 1994, pp. 362–367.

[17] Jean-Charles Régin. "Generalized Arc Consistency for Global Cardinality Constraint". In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1*. 1996, pp. 209–215.

[18] Ehud Friedgut, An, and Jean Bourgain. "Sharp thresholds of graph properties, and the k-sat problem". In: *J. Amer. Math. Soc* 12 (1999), pp. 1017–1054.

[19] François Pachet and Pierre Roy. "Automatic Generation of Music Programs". In: *Principles and Practice of Constraint Programming - CP'99, 5th International Conference, Alexandria, Virginia, USA, October 11-14, 1999, Proceedings*. Ed. by Joxan Jaffar. Vol. 1713. Lecture Notes in Computer Science. Springer, 1999, pp. 331–345. ISBN: 3-540-66626-5. DOI: 10.1007/978-3-540-48085-3_24. URL: https://doi.org/10.1007/978-3-540-48085-3%5C_24.

[20] Nicolas Beldiceanu. "Pruning for the Minimum Constraint Family and for the Number of Distinct Values Constraint Family". In: *Principles and Practice of Constraint Programming - CP 2001, 7th International Conference, CP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*. Ed. by Toby Walsh. Vol. 2239. Lecture Notes in Computer Science. Springer, 2001, pp. 211–224. ISBN: 3-540-42863-1. DOI: 10.1007/3-540-45578-7_15. URL: https://doi.org/10.1007/3-540-45578-7%5C_15.

[21] Nicolas Beldiceanu and Mats Carlsson. "Revisiting the Cardinality Operator and Introducing the Cardinality-Path Constraint Family". In: *Logic Programming, 17th International Conference, ICLP 2001, Paphos, Cyprus, November 26 - December 1, 2001, Proceedings*. Ed. by Philippe Codognet. Vol. 2237. Lecture Notes in Computer Science. Springer, 2001, pp. 59–73. ISBN: 3-540-42935-2. DOI: 10.1007/3-540-45635-X_12. URL: https://doi.org/10.1007/3-540-45635-X%5C_12.

[22] Willem Jan van Hoeve. "The alldifferent Constraint: A Survey". In: *CoRR* (2001). URL: http://arxiv.org/abs/cs.PL/0105015.

[23] Carla Gomes and David Shmoys. "Completing Quasigroups or Latin Squares: A Structured Graph Coloring Problem". In: (Jan. 2002).

[24] Frédéric Boussemart et al. "Boosting Systematic Search by Weighting Constraints". In: *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, Valencia, Spain, August 22-27, 2004*. IOS Press, 2004, pp. 146–150.

[25] Frédéric Boussemart et al. "Boosting Systematic Search by Weighting Constraints". In: *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*. Ed. by Ramón López de Mántaras and Lorenza Saitta. IOS Press, 2004, pp. 146–150. ISBN: 1-58603-452-9.

[26] Mark Jerrum, Alistair Sinclair, and Eric Vigoda. "A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries". In: *J. ACM* 51.4 (2004), pp. 671–697. DOI: 10.1145/1008731.1008738. URL: https://doi.org/10.1145/1008731.1008738.

[27] Heng Liang and Fengshan Bai. "An upper bound for the permanent of (0,1)-matrices". In: *Linear Algebra and its Applications* 377 (2004), pp. 291–295.

[28] Gilles Pesant. "A Regular Language Membership Constraint for Finite Sequences of Variables". In: *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*. Ed. by Mark Wallace. Vol. 3258. Lecture Notes in Computer Science. Springer, 2004, pp. 482–495. ISBN: 3-540-23241-9. DOI: 10.1007/978-3-540-30201-8_36. URL: https://doi.org/10.1007/978-3-540-30201-8%5C_36.

[29] Philippe Refalo. "Impact-Based Search Strategies for Constraint Programming". In: *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004*. Vol. 3258. Lecture Notes in Computer Science. Springer, 2004, pp. 557–571.

[30] Roman Barták and Michela Milano, eds. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings*. Vol. 3524. Lecture Notes in Computer Science. Springer, 2005. ISBN: 3-540-26152-4. DOI: 10.1007/b136920. URL: https://doi.org/10.1007/b136920.

[31] Nicolas Beldiceanu, Pierre Flener, and Xavier Lorca. "The tree Constraint". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings*. Ed. by Roman Barták and Michela Milano. Vol. 3524. Lecture Notes in Computer Science. Springer, 2005, pp. 64–78. ISBN: 3-540-26152-4. DOI: 10.1007/11493853_7. URL: https://doi.org/10.1007/11493853%5C_7.

[32] Christian Bessière et al. "Among, Common and Disjoint Constraints". In: *Recent Advances in Constraints, Joint ERCIM/CoLogNET International Workshop on Constraint Solving and Constraint Logic Programming, CSCLP 2005, Uppsala, Sweden, June 20-22, 2005, Revised Selected and Invited Papers*. Ed. by Brahim Hnich et al. Vol. 3978. Lecture Notes in Computer Science. Springer, 2005, pp. 29–43. ISBN: 3-540-34215-X. DOI: 10.1007/11754602_3. URL: https://doi.org/10.1007/11754602%5C_3.

[33] Christian Bessière et al. "Filtering Algorithms for the NValue Constraint". In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 30 - June 1, 2005, Proceedings*. Ed. by Roman Barták and Michela Milano. Vol. 3524. Lecture Notes in Computer Science. Springer, 2005, pp. 79–93. ISBN: 3-540-26152-4. DOI: 10.1007/11493853_8. URL: https://doi.org/10.1007/11493853%5C_8.

[34] Christian Bessière et al. "The Range and Roots Constraints: Specifying Counting and Occurrence Problems". In: *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh, Scotland, UK, July 30 - August 5, 2005*. Ed. by Leslie Pack Kaelbling and Alessandro Saffiotti. Professional Book Center, 2005, pp. 60–65. ISBN: 0938075934. URL: http://ijcai.org/Proceedings/05/Papers/0735.pdf.

[35] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. New York, NY, USA: Elsevier Science Inc., 2006. ISBN: 0444527265.

[36] David L. Applegate et al. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007. ISBN: 0691129932, 9780691129938.

[37] Carla P. Gomes et al. "From Sampling to Model Counting". In: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. 2007, pp. 2293–2299.

[38] Shmuel Friedland. "An upper bound for the number of perfect matchings in graphs". In: *http://arxiv.org/abs/0803.0864* (2008).

[39] Christian Bessiere et al. "Range and Roots: Two common patterns for specifying and propagating counting and occurrence constraints". In: *Artif. Intell.* 173.11 (2009), pp. 1054–1078. DOI: 10.1016/j.artint.2009.03.001. URL: https://doi.org/10.1016/j.artint.2009.03.001.

[40] László Lovasz and Michael D. Plummer. *Matching Theory*. American Mathematical Society, 2009.

[41] Richard P. Stanley. *Enumerative Combinatorics: Volume 1*. 2nd. New York, NY, USA: Cambridge University Press, 2011. ISBN: 1107602629, 9781107602625.

[42] William Burnside. *Theory of Groups of Finite Order*. Cambridge Library Collection - Mathematics. 2012. DOI: 10.1017/CBO9781139237253.

[43] Alan M. Frieze and Páll Melsted. "Maximum matchings in random bipartite graphs and the space utilization of Cuckoo Hash tables". In: *Random Struct. Algorithms* 41.3 (2012), pp. 334–364. DOI: 10.1002/rsa.20427. URL: https://doi.org/10.1002/rsa.20427.

[44] Laurent Michel and Pascal Van Hentenryck. "Activity-Based Search for Black-Box Constraint Programming Solvers". In: *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems - 9th International Conference, CPAIOR 2012, Nantes, France, May 28 - June1, 2012*. Vol. 7298. Lecture Notes in Computer Science. Springer, 2012, pp. 228–243.

[45] Gilles Pesant, Claude-Guy Quimper, and Alessandro Zanarini. "Counting-Based Search: Branching Heuristics for Constraint Satisfaction Problems". In: *J. Artif. Intell. Res.* 43 (2012), pp. 173–210. DOI: 10.1613/jair.3463. URL: https://doi.org/10.1613/jair.3463.

[46] Jérémie Du Boisberranger et al. "When is it worthwhile to propagate a constraint? A probabilistic analysis of AllDifferent". In: *Proceedings of the 10th Meeting on Analytic Algorithmics and Combinatorics, ANALCO 2013, New Orleans, Louisiana, USA, January 6, 2013*. Ed. by Markus E. Nebel and Wojciech Szpankowski. SIAM, 2013, pp. 80–90. ISBN: 978-1-61197-254-2. DOI: 10.1137/1.9781611973037.10. URL: https://doi.org/10.1137/1.9781611973037.10.

[47] Guillem Perarnau and Giorgis Petridis. "Matchings in Random Biregular Bipartite Graphs". In: *Electr. J. Comb.* 20.1 (2013), P60.

[48] Mats Carlsson and Thom Fruehwirth. *Sicstus PROLOG User's Manual 4.3*. Books On Demand - Proquest, 2014. ISBN: 3735737447, 9783735737441.

[49] Kuldeep S. Meel et al. "Constrained Sampling and Counting: Universal Hashing Meets SAT Solving". In: *CoRR* abs/1512.06633 (2015). arXiv: 1512.06633. URL: http://arxiv.org/abs/1512.06633.

[50] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. *Choco Solver Documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. 2016. URL: http://www.choco-solver.org.

[51] Lior Eldar and Saeed Mehraban. "Approximating the Permanent of a Random Matrix with Vanishing Mean". In: *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. Ed. by Mikkel Thorup. IEEE Computer Society, 2018, pp. 23–34. ISBN: 978-1-5386-4230-6. DOI: 10.1109/FOCS.2018.00012. URL: https://doi.org/10.1109/FOCS.2018.00012.

[52] Samuel Gagnon and Gilles Pesant. "Accelerating Counting-Based Search". In: *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings*. Ed. by Willem Jan van Hoeve. Vol. 10848. Lecture Notes in Computer Science. Springer, 2018, pp. 245–253. ISBN: 978-3-319-93030-5. DOI: 10.1007/978-3-319-93031-2_17. URL: https://doi.org/10.1007/978-3-319-93031-2%5C_17.

[53] Giovanni Lo Bianco, Xavier Lorca, and Charlotte Truchet. "Estimating the Number of Solutions of Cardinality Constraints through `range` and `roots` Decompositions". In: *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019*. 2019.

[54] Giovanni Lo Bianco, Xavier Lorca, Charlotte Truchet, and Gilles Pesant. "Revisiting Counting Solutions for the Global Cardinality Constraint". In: *Journal of Artificial Intelligence Research* 66 (2019).

[55] Gilles Pesant. "From Support Propagation to Belief Propagation in Constraint Programming". In: *J. Artif. Intell. Res.* 66 (2019), pp. 123–150. DOI: 10.1613/jair.1.11487. URL: https://doi.org/10.1613/jair.1.11487.

[56] Gilles Pesant. *CSPLib Problem 067: Quasigroup Completion*. Ed. by Christopher Jefferson et al. http://www.csplib.org/Problems/prob067.

[57] Toby Walsh. *CSPLib Problem 026: Sports Tournament Scheduling*. Ed. by Christopher Jefferson et al. http://www.csplib.org/Problems/prob026.

Résumé long

Contexte

Dans cette thèse, nous nous intéressons à la Programmation par Contraintes (PPC), un paradigme permettant d'exprimer de manière déclarative et de résoudre des problèmes combinatoires. Un problème combinatoire peut être énoncé comme suit. Un ensemble de variables représente les inconnues du problème. Ces variables peuvent prendre leurs valeurs dans un domaine fini. Le problème est ensuite exprimé en ajoutant des contraintes, qui sont des relations logiques et mathématiques sur les variables. L'objectif est enfin de trouver des affectations des variables aux valeurs des domaines, de sorte que toutes les contraintes soient satisfaites.

Les solveurs de contraintes explorent l'espace de recherche de manière récursive en alternant deux étapes. Tout d'abord, on raisonne sur chaque contrainte une à une pour filtrer les valeurs inconsistante, qui ne peuvent apparaître dans une solution. Chaque fois qu'une variable est instanciée ou que son domaine est réduit, le solveur de contraintes appelle récursivement les algorithmes de filtrage de chaque contrainte qui porte sur cette variable. Pour chaque contrainte, il existe un ou plusieurs algorithmes de filtrage. Cette étape est appelée la propagation. Une fois que nous ne pouvons plus réduire les domaines, nous avons atteint le point fixe et nous devons sélectionner une variable à instancier, pour poursuivre l'exploration de l'espace de recherche. Si un domaine est vide après l'application d'un algorithme de filtrage, nous avons trouvé une contradiction et les choix précédents la dernière instanciation doivent être révisés. En revanche, si chaque variable est instanciée, nous avons trouvé une solution.

L'un des principaux atouts de la programmation par contraintes est le large éventail de contraintes pouvant être utilisées pour modéliser les problèmes combinatoires et les différents algorithmes de filtrage qui proviennent de nombreux domaines tels que la recherche opérationnelle, l'intelligence artificielle et la théorie des graphes. Les algorithmes de filtrage peuvent être paramétrés pour être plus ou moins consistent. Il y a un équilibre à trouver entre l'intelligence de ces algorithmes et leur temps de calcul. De plus, la stratégie d'exploration de l'espace de recherche, qui sélectionne les variables à instancier une fois le point fixe atteint, doit être paramétrée. La configuration d'un solveur de contraintes dépend la plupart du temps du problème et nécessite une connaissance approfondie de la PPC. Par conséquent, la programmation par contraintes est loin de pouvoir être utilisée comme une boîte noire. Idéalement, les solveurs de contraintes seraient automatiquement configurés pour s'adapter à la structure du problème.

Dans cette thèse, nous développons des modèles mathématiques pour calculer le nombre de solutions autorisées par une contrainte sans la résoudre, et pour mieux comprendre la structure de l'espace de recherche. Le dénombrement a de nombreuses applications telles que le raisonnement probabiliste et le machine learning. Le dénombrement de solutions pour les contraintes nous fournit des informations importantes sur leur structure et facilite la configuration des solveurs de contraintes. Nous nous concentrons sur le dénombrement de solutions pour les contraintes de

cardinalité. Les contraintes de cardinalité sont le plus souvent utilisées pour modéliser les problèmes d'affectation et peuvent toutes être représentées comme des problèmes de couplage. La plupart des algorithmes de filtrage pour les contraintes de cardinalité viennent de la théorie des graphes et de la théorie des couplages. La théorie des graphes est un outil puissant de modélisation, pour lequel il existe également de nombreuses applications de dénombrement, que nous utilisons pour compter les solutions sur ces contraintes de cardinalité. La première contribution de cette thèse est l'étude du nombre de solutions dans une contrainte de cardinalité globale et la seconde contributions portent sur l'évaluation probabiliste du nombre de solutions pour toutes les contraintes de cardinalité.

Dénombrement de solutions dans une contrainte de cardinalité globale

La contrainte de cardinalité globale *gcc* porte sur le nombre d'occurrences de chaque valeur dans la solution et peut être représentée sous la forme d'un problème de flot dans un graphe biparti. C'est la contrainte de cardinalité la plus générale: la plupart des contraintes de cardinalité peuvent être exprimées comme un cas particulier de *gcc*. Elle requiert des algorithmes de filtrage très complexes et Pesant et al. (2012) ont montré qu'il est aussi difficile de développer des algorithmes de dénombrement de solution pour *gcc* que de développer ses algorithmes de filtrage. Les problèmes qui consistent à dénombrer des solutions dans de telles structures combinatoires appartiennent à la classe de complexité #P-complet. Les algorithmes de dénombrement exact de solutions sur *gcc* se font en temps exponentiel. Par conséquent, Pesant et al. n'ont pas développé de méthodes de comptage exact mais des estimations du nombre réel de solutions. Ils ont établi une borne supérieure du nombre de solutions sur *gcc* et ont intégré ce résultat dans la stratégie `maxSD` qui explore en premier les régions de l'espace de recherche qui promettent le plus de solutions.

Nous montrons, grâce à un contre-exemple, dans le premier chapitre de cette thèse que la borne supérieure développée par Pesant et al. n'en est pas une. Dénombrer les solutions d'une contrainte de cardinalité globale implique de casser des symétries dans la structure de la contrainte. Une erreur de raisonnement avait été commise et le nombre de symétries avait été sur-évalué, conduisant ainsi à une borne supérieure erronée. Nous montrons comment compter correctement ces symétries et proposons une nouvelle borne supérieure du nombre de solutions pour une instance de *gcc*.

Nous implémentons ensuite cette borne au sein de la stratégie d'exploration `maxSD` et nous comparons son efficacité avec la borne erronée. Dans un premier temps, nous montrons que l'ordre des variables instanciées est différent selon que la stratégie `maxSD` est guidée par la borne corrigée ou bien par l'ancienne borne erronée. Puis, nous évaluons les performances de ces deux estimateurs sur des problèmes générés aléatoirement comportant un nombre de différent de contraintes *gcc* à difficulté variable. Nous montrons qu'aucune des deux versions de `maxSD` n'est plus performante que l'autre. Cela signifie qu'en pratique la borne erronée et sa correction peuvent servir à guider l'exploration. Cela signifie aussi que, dans le cadre de `maxSD`, l'écart de précision entre l'estimateur et le nombre exacte de solutions est moins importante que la corrélation entre l'estimateur et le nombre exacte de solutions. Cette étude du nombre de solutions dans une instance de *gcc* a été publié dans Journal of Artificial Intelligence Research.

Dans le Chapitre 4, nous proposons une seconde borne supérieure grâce à une nouvelle approche basée sur la théorie des groupes. Nous considérons l'ensemble des symétries comme un groupe agissant sur l'ensemble des solutions autorisées par la contrainte et nous utilisons le lemme de Burnside qui permet, dans cette configuration, de dénombrer le nombre de solutions en considérant ces symétries. Ce lemme a déjà été utilisé pour compter le nombre de configurations d'un Rubik's cube. Cette approche résulte en une nouvelle borne supérieure pour le nombre de solutions d'une instance de *gcc*. Malheureusement cette nouvelle borne est non seulement largement dominée par celle développée dans le chapitre 3, mais aussi plus coûteuse à évaluer. Néanmoins, l'approche est originale et nous pensons qu'elle peut inspirer des méthodes de dénombrement pour d'autres contraintes, surtout si celles-ci font intervenir des symétries.

Dénombrement probabiliste de solutions dans une contrainte de cardinalité

Nous avons montré dans la première partie que la corrélation entre l'estimateur et le nombre réel de solutions est plus important que la précision de l'estimateur, et donc il n'est pas nécessaire de construire des bornes supérieures: une estimation probabiliste est suffisante pour guider `maxSD`. Dans la deuxième partie de cette thèse, nous présentons des modèles probabilistes pour compter le nombre de solutions dans les contraintes de cardinalité. Ces modèles probabilistes sont basés sur des graphes aléatoires. Les graphes aléatoires sont générés en suivant une certaine distribution probabiliste. Nous présentons dans le chapitre 5 de cette thèse le modèle des graphes aléatoires de Erdős et Renyi, pionniers dans ce domaine. Les modèles de graphes aléatoires permettent d'étudier en moyenne certaines propriétés: existence de cycles, nombre de cliques, connexité ... Plus précisément, nous nous intéressons aux couplages au sein de graphes bipartis aléatoires. Cette analyse en moyenne nous donne des informations sur le nombre de solutions et la satisfiabilité des contraintes de cardinalité. Dans un premier temps, nous ne nous concentrerons que sur la contrainte *alldifferent*, car sa structure est bien adaptée à l'utilisation de modèle de graphes aléatoires. Nous présentons deux modèles probabilistes pour *alldifferent*: l'un basé sur le modèle d'Erdős-Renyi (ER) et l'autre qu'on appelle le modèle FDS (Fixed Domain Size). Le premier modèle nous permet d'estimer le nombre de solutions en se basant uniquement sur la densité d'arêtes dans le graphe biparti représentant la contrainte et l'autre en se basant sur la taille des domaines de chaque variable. Nous comparons ces deux estimateurs en évaluant leur écart au nombre réel de solutions et nous montrons, sur plusieurs milliers d'instance de *alldifferent* généré aléatoirement, que l'estimateur FDS est bien plus précis que l'estimateur ER. Cependant, dans le cadre de la stratégie `maxSD` sur certaines instances difficiles de carrés latins, l'estimateur ER est un peu plus performant que l'estimateur FDS. Cela confirme l'intuition que la corrélation entre l'estimateur et le vrai nombre de solutions est plus important que la précision de cet estimateur dans le cadre de `maxSD`.

Nous étudions aussi, dans le cadre du modèle ER, l'existence d'une solutions pour une instance de *alldifferent* donnée. Nous montrons qu'il est possible de réutiliser les résultats asymptotiques qu'avaient développés Erdős et Renyi dans le cadre de la programmation par contrainte. L'intuition est la suivante: pour une taille de problème donné (ici le nombre de variable), plus la densité d'arête dans le graphe biparti représentant la contrainte est grande, plus il est probable que l'instance soit satisfiable et, à l'inverse, plus cette densité est faible, plus il est probable que l'instance

soit insatisfiable. Il y a une zone d'incertitude lorsque la densité n'est ni trop faible ni trop élevée. Erdős et Renyi ont prouvé que plus la taille du problème est élevée, plus cette zone d'incertitude est petite, et donc le problème est plus décidable. Cela est intéressant car, pour de grandes instances de *alldifferent*, on peut dire s'il existe ou non une solution en se basant uniquement sur la densité d'arête dans le graphe. Cela pourrait être utile à la création d'heuristiques de recherche par exemple.

Dans le sixième et dernier chapitre de cette thèse, nous montrons comment estimer le nombre de solutions sur les contraintes de cardinalité suivantes: *alldifferent*, *nvalue*, *atmost_nvalue*, *atleast_nvalue*, *occurrence*, *atmost*, *atleast*, *among*, *uses*et *disjoint*. Ces estimateurs sont tous dérivés du modèle Erdős-Renyi. Nous proposons ici une méthode systématique pour estimer le nombre de solutions de la plupart des contraintes de cardinalité. Ces travaux ont été présentés à CP 2019.

Pesant et al. ont montré que compter ou estimer le nombre de solutions requiert des méthodes dédiées à chaque contrainte. Si nous pouvions décomposer chaque contrainte de cardinalité grâce à une contraintes primitive, alors disposer d'une technique de dénombrement pour cette contrainte primitive aiderait à compter les solutions sur les contraintes de cardinalité initiales. Bessiere, Hebrard, Hnich, Kiziltan et Walsh (2009) ont développé deux nouvelles contraintes globales, *range* et *roots*, qui peuvent être utilisées pour spécifier de nombreuses contraintes de cardinalité. En d'autres termes, pour presque chaque contrainte de cardinalité, il existe un modèle équivalent utilisant uniquement les contraintes *range* et *roots* (et quelques contraintes arithmétiques).

Nous montrons comment utiliser les décompositions *range* et *roots* pour dénombrer les solutions. Plus précisément, nous développons une approche probabiliste, basée sur le modèle d'Erdős-Renyi, pour estimer le nombre de solutions sur la contrainte *range* et sur la contrainte *roots*. Nous en déduisons une méthode systématique pour estimer le nombre de solutions sur les contraintes de cardinalité. En comparaison à Pesant et al., nous obtenons une estimation au lieu d'une borne supérieure et nous proposons une méthode qui peut être généralisée à un grand nombre de contraintes de cardinalité sans avoir à recréer un modèle dédié. Dans le tableau suivant, nous listons tous les estimateurs, basés sur le modèle d'Erdős-Renyi, que nous avons pu déduire avec notre méthode. L'ensemble des variables est noté X, l'ensemble des valeurs est noté Y (l'union de tous les domaines) et p est la densité d'arêtes dans le graphe variables/valeurs.

Nous remarquons que tous les estimateurs ont la même forme: un coefficient dépendant de la taille du problème multiplié par p^n. Nous montrons que, dans le cadre du modèle d'Erdös-Renyi, ce coefficient est en fait le nombre de solutions que la contrainte aurait si toutes les variables pouvaient prendre toutes les valeurs (si le graphe variable/valeur était complet).

Nous avons implémenté ces estimateurs au sein de la stratégie d'exploration `maxSD` et avons comparé l'efficacité de cette stratégie avec d'autres heuristiques génériques classiques tels que `dom/wdeg`, `abs` et `ibs`. Avec ces estimateurs probabilistes, on montre que les performances de `maxSD` sont très comparables aux autres heuristiques génériques sur des instances difficiles de carrés latins, et surpassent même ces heuristiques sur certains problèmes comme le Sport Scheduling Problem.

Constraint	Formula with $\|X\| = n$, $\|X_1\| = n_1$, $\|X_2\| = n_2$, $\|Y\| = m$ and $\|Y'\| = m'$
$alldifferent(X)$	$\frac{m!}{(m-n)!} \cdot p^n$
$among(X, Y', N)$	$\binom{n}{N} m'^N (m - m')^{n-N} \cdot p^n$
$nvalue(X, N)$	$\binom{m}{N} \cdot a_{n,N} \cdot p^n$
$atmost\_nvalue(X, N)$	$\sum_{k=1}^{N} \binom{m}{k} a_{n,k} \cdot p^n$
$atleast\_nvalue(X, N)$	$\sum_{k=N}^{n} \binom{m}{k} a_{n,k} \cdot p^n$
$occurrence(X, y, N)$	$\binom{n}{N} (m-1)^{n-N} \cdot p^n$
$atmost(X, y, N)$	$\sum_{k=1}^{N} \binom{n}{k} (m-1)^{n-k} \cdot p^n$
$atleast(X, y, N)$	$\sum_{k=N}^{n} \binom{n}{k} (m-1)^{n-k} \cdot p^n$
$uses(X, X_1, X_2)$	$m^{n-n_1-n_2} \cdot \sum_{k=1}^{m} \binom{m}{k} a_{n_1,k} k^{n_2} \cdot p^n$
$disjoint(X, X_1, X_2)$	$m^{n-n_1-n_2} \cdot \sum_{k=1}^{min(n_1,m)} \sum_{l=1}^{min(n_2,m-k)} \binom{m}{k} \binom{m-k}{l} a_{n_1,k} a_{n_2,l} \cdot p^n$

TABLE 3: Counting formulae extracted from *range* and *roots* reformulation

Conclusion

Dans la première partie de cette thèse, nous avons développé une borne supérieure au nombre de solutions d'une contrainte *gcc*, qui corrige le premier résultat établi par Pesant et al. Nous avons implémenté le résultat précédent et sa correction dans la stratégie maxSD et nous avons proposé un protocole expérimental pour comparer leur efficacité en tant qu'estimateurs du nombre réel de solutions sur *gcc*. L'analyse expérimentale a montré que les deux estimateurs se comportent de manière très similaire dans la stratégie maxSDCela démontre que la corrélation entre un estimateur et le nombre réel de solutions prévaut sur sa précision. Ensuite nous avons développé une méthode originale pour dénombrer le nombre de solutions dans *gcc* en se basant sur la théorie des groupes. Malheureusement, cette approche n'améliore pas la première correction.

Dans la deuxième partie de cette thèse, nous avons développé des méthodes probabilistes pour dénombrer les solutions sur *alldifferent*, puis sur toutes les contraintes de cardinalité. Nous avons mené une étude probabiliste de *alldifferent* en ce qui concerne le nombre de solutions d'une instance tirée aléatoirement selon le modèle d'Erdős-Renyi ou le modèle FDS, mais aussi l'existence ou non d'une solution. Nous avons développé une méthode systématique pour estimer le nombre de solutions sur un grand nombre de contraintes de cardinalité grâce à leur décomposition *range* et *roots* sans avoir besoin de redéfinir une méthode de dénombrement dédiée à chaque contrainte. Nous avons validé ces estimateurs probabilistes en les intégrant dans la stratégie maxSD en comparant cette stratégie à d'autres stratégies génériques.

Malheureusement, nous ne sommes pas parvenus à développer un estimateur probabiliste pour *gcc*. Celle-ci échappe toujours la méthode systématique développée, car contrairement aux autres contraintes de cardinalité, sa décomposition en contraintes *range* et *roots* a une structure très complexe. Compter le nombre de solutions d'une contrainte *gcc* en se basant sur sa décomposition *range* et *roots* revient à compter le nombre de solutions sur une conjonction de contraintes portant sur des ensembles de variables non indépendants. Ce dernier point est d'ailleurs

le principal frein à l'efficacité de la stratégie `maxSD`: nous savons compter les solutions sur une contrainte donnée, mais il est très difficile de compter les solutions sur n'importe quelle conjonction de contraintes.

Nous espérons pouvoir trouver d'autres outils dans les domaines de la Combinatoire et de l'Analyse en Moyenne pour répondre à ces interrogations et venir enrichir la programmation par contraintes.

UNIVERSITE BRETAGNE LOIRE / MATHSTIC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : ESTIMER LE NOMBRE DE SOLUTIONS SUR LES CONTRAINTES DE CARDINALITÉ

Mots clés : programmation par contraintes, dénombrement, graphes aléatoires

Résumé : La richesse de la programmation par contraintes repose sur la très large variété des algorithmes qu'elle utilise en puisant dans les grands domaines de l'Intelligence Artificielle, de la Programmation Logique et de la Recherche Opérationnelle. Cependant, cette richesse, qui offre aux spécialistes une palette quasi-illimitée de configurations possibles pour attaquer des problèmes combinatoires, devient une frein à la diffusion plus large du paradigme, car les outils actuels sont très loin d'une boîte noire, et leur utilisation suppose une bonne connaissance du domaine, notamment en ce qui concerne leur paramétrage.

Dans cette thèse, nous proposons d'analyser le comportement des contraintes de cardinalité avec des modèles probabilistes et des outils de dénombrement, pour paramétrer automatiquement les solveurs de contraintes: heuristiques de choix de variables et de choix de valeurs et stratégies de recherche.

Title : ESTIMATING THE NUMBER OF SOLUTIONS ON CARDINALITY CONSTRAINTS

Keywords : constraint programming, counting, random graphs

Abstract : The main asset of constraint programming is its wide variety of algorithms that comes from the major areas of artificial intelligence, logic programming and operational research. It offers specialists a limitless range of possible configurations to tackle combinatorial problems, but it becomes an obstacle to the wider diffusion of the paradigm. The current tools are very far from being used as a black-box tool, and it assumes a good knowledge of the field, in particular regarding the parametrization of solvers.

In this thesis, we propose to analyze the behavior of cardinality constraints with probabilistic models and counting tools, to automatically parameterize constraint solvers: heuristics of choice of variables and choice of values and search strategies.