The *Path* to Satisfaction:

Polynomial Algorithms for *SAT*

Daniel J Hulme

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Engineering Doctorate

of the

University of London.

Department of Computer Science

University College London

2008

I, Daniel J Hulme, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

Twenty years ago the bottleneck preventing us from solving many practical problems lay in the limited capacity and performance of computers. However, modern computers are now capable of storing a huge amount of data and processing it at very high-speed. The bottleneck nowadays is in the efficiency and effectiveness of the algorithms manipulating the data. Two fields of Computer Science related to this issue - Constraint Satisfaction (CSP) and Propositional Satisfiability (*SAT*) - have developed relatively independently over the past half century, it is only in recent years that the mutual benefits of these fields have started to be explored.

One method used to explore the relationship between these fields is to study the mappings between the two core problem domains. After introducing the relevant background and providing a detailed survey of *SAT* and CSP encodings, my analysis of these encodings uncovers three categories, providing a framework for current and future *SAT* and CSP encodings to be developed. As a result of this framework I define a new encoding and I demonstrate that in certain circumstances this encoding may be more advantageous than other encodings. This new encoding also opens up the potential for additional CSP to *SAT* encodings.

After categorising *SAT* and CSP encodings the focus of this research shifts to characterising them. Some empirical work on comparing algorithmic performance on instances with different *solution-density* has been performed, and it has been shown that this feature is correlated with the solubility of a problem instance and can be used to choose between using stochastic and branching algorithms. I provide a characterisation of current *SAT* to CSP encodings and I prove that some encodings result in problems with varying *solution-densities*. Since it has been shown that *solution-density* is an important factor in determining the solubility of an instance, my work provides a guide to assist in choosing one encoding over another.

A large amount of theoretical analysis has been published comparing stochastic and branching algorithmic techniques on *SAT* and CSP encodings, however, very little research has compared the effect of enforcing local-consistency in each of these domains. After introducing the main algorithmic techniques from the *SAT* and CSP communities I use the graph-theoretic framework to reconfirm and strengthen the equivalence between the *SAT* and CSP-based proof methods. I provide a comprehensive comparison

between local-consistency techniques on each of the *SAT* to CSP encodings. One result of this work has direct practical implications regarding the choice of encoding; that enforcing local-consistency on some types of encoded problems does more work than when using other type of encodings. I also prove that enforcing certain levels of local-consistency on some types of encoded *SAT* instances does zero work, which is supported by the empirical results presented in this thesis.

In addition, I bridge further the CSP and *SAT* techniques by introducing the concept of Extended-*Consistency*, thus providing a more complete picture of the extended proof-systems. Extended proof-systems allow the introduction of auxiliary variables and are some of the most powerful proof-systems known. Relatively little practical work has been done on extended proof-systems. Typically *SAT* and CSP algorithms that employ additional variables tend to apply them in an 'ad-hoc' manner. The motivation of defining Extended-*Consistency* is to allow synergies between extended proof-systems to be cultivated and explored.

I identify two key problematic aspects of empirical studies of encodings. First, typically one type of problem is used as a benchmark to compare encodings. Second, either stochastic or backtracking algorithms are applied to the encoded problems. Clearly both of these choices may bias the results, since an encoding/algorithm may 'favour' a particular problem. I address these issues by applying a local-consistency algorithm to a wide variety of unsatisfiable problems. These experiments support the theoretical analysis and strongly indicate that the choice of encoding has a dramatic effect on reducing the search-space when enforcing local-consistency. I demonstrate that enforcing a small local-level of consistency on problems using a certain encoding does not solve any of the *SAT* benchmarks, which is in stark contrast to problems encoded using a different method. I show empirically that enforcing such a low-level of consistency can solve a large number of 'hard' *SAT* benchmark families. In particular, converting *SAT* instances to CSP and applying local-consistency can not only solve many 'hard' *SAT* instances, but this technique can even compete with state-of-the-art *SAT*-Solvers.

The results presented in this thesis are part of a research program aimed at bridging the gap between Propositional Satisfiability and Constraint Satisfaction. Hence, the main goal of this thesis is to strengthen this relationship and to capitalise on synergies between these two fields. The broader aim is to develop a better understanding of Computer Science, which aims to benefit the scientific and industrial communities by increasing the theoretical understanding of the complexity and tractability of natural problems, thus improving practical algorithms that can be applied it to pertinent scientific and industrial problems.

Acknowledgements

I wish to acknowledge the support of the EPSRC Engineering Doctorate from the EngD Programme at UCL. The EngD is an excellent programme hosted by a fantastic university, it is a four-year postgraduate award intended for the UK's leading research engineers. With a taught component spanning both Engineering and electives from an MBA at the London Business School, the EngD programme is better suited to the development of translational research, and provides a more vocationally oriented Doctorate in Engineering.

I would like to thank Beau Lotto for providing me with a diverse environment, as well as the freedom to pursue this research. I thank Bernard Buxton - my supervisor and mentor - for his continuous encouragement and advice, which has been invaluable throughout my academic and personal development. I am indebted to Robin Hirsch for his help with this work and for his guidance and patience over the years.

I sincerely thank the members of the LottoLab and the IOO for a pleasant working environment and our various stimulating discussions, including Martina Wicklein, Udi Schlessinger, David Malkin, David Corney, Erwan Le Martelot and Steve Dakin, and in particular Rich Clarke for making the past four years some of my most enjoyable. I thank Pete Jeavons, Dave Cohen, Ian Gent and the members of Oxford's, Holloway's and St Andrews Constraints group for our brief yet valuable conversations. Special appreciation goes to my examiners Barbara Smith and Dave Cohen for their advice about how to greatly strengthen this work.

I thank my family and friends for their continuous encouragement, enthusiasm and support; especially David Bradshaw for doing a great job of proof-reading the first version of this work.

Finally, and most importantly, a very special thank you goes to Nana, Grampy and Shona; to whom this thesis is dedicated — it would be impossible to have done it without them.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction and Motivation

In the seminal paper by Hartmanis & Stearns (1965) Computational Complexity was born. Informally, Computational Complexity is the discipline of Computer Science concerned with the number of algorithmic operations required to solve a well-defined problem. In fact, Sipser (1992) points out that several papers around this time proposed and developed the notion of measuring the complexity of a problem by the number of steps required to solve it with an algorithm.

More generally, Complexity Theory is the study of the level of resource required to solve a mathematically posed problem[1] (i.e. time or space), and over the past 40 years research in this field has made tremendous inroads to industry. We are unable to go from A to B without algorithms solving a plethora of optimisation problems along the way — indeed, sometimes even the process of going from A to B efficiently is an optimisation problem.

Twenty years ago the bottleneck preventing us from solving many practical problems lay in the limited capacity and performance of computers. However, modern computers are now capable of storing a huge amount of data and processing it at very high-speed. The bottleneck nowadays is in the efficiency and effectiveness of the algorithms manipulating the data, which is why Complexity Theory has become such an important and popular field.

Certainly, progress in this field continues to increase as witnessed by the growing number of Complexity-related papers published in journals and presented at major conferences. However, it may be safe to say that no other article in Computer Science has stimulated more discussion than the **P** vs **NP** problem[2], which has now been with us for over three decades.

1.1 P vs NP Problem

The **P** vs **NP** problem, formulated independently by Stephen Cook (1971) and Leonid Levin (Trakhtenbrot (1984)), is arguably one of the most important scientific questions posed to date. Simply stated, the **P** vs **NP** question asks if there exists a polynomial solution to any of the problems shown

[1] FOLDOC: Free On-line Dictionary of Computing. http://foldoc.org
[2] **P**olynomial (**P**) and **N**on-deterministic **P**olynomial (**NP**), defined in Section 2.1

to be **NP**-complete. Cook showed that the SATISFIABILITY PROBLEM is **NP**-complete, whilst Levin proved **NP**-completeness for a variant of the TILING PROBLEM[3]. Cook's Theorem states that any problem that can be solved in polynomial-time by a non-deterministic Turing machine can be reduced (in polynomial-time) to the problem of determining whether a Boolean formula is satisfiable.

Over the past several decades researchers have been trying to (dis)prove the $\mathbf{P} \neq \mathbf{NP}$ conjecture by determining whether or not there is a polynomial solution to any of the known **NP**-complete problems, many of which are described in Garey & Johnson (1990) Computers and Intractability: A Guide to the Theory of **NP**-completeness. If indeed we were to discover that one of these **NP**-complete problems was in **P**, then by the process of polynomial-reduction we will have managed to solve every **NP** problem polynomially. Cook (2000) states that although a practical algorithm for solving an **NP**-complete problem would have devastating consequences for cryptography, it would also have stunning implications of a more positive nature,

> "for example, it would transform mathematics by allowing a computer to find a formal proof of any theorem which has a proof of reasonable length, since formal proofs can easily be recognized in polynomial time. Example theorems may well include all of the Clay Mathematical Institute prize problems. Although the formal proofs may not be initially intelligible to humans, the problem of finding intelligible proofs would be reduced to that of finding a recognition algorithm for intelligible proofs."

Similar remarks apply to diverse creative human endeavours, such as architecture, creating physical theories, composing music or even automating intelligent behaviour. Though still unproven, the general consensus from the Gasarch (2002) poll is that $\mathbf{P} \neq \mathbf{NP}$[4]. Undoubtedly, even if $\mathbf{P} \neq \mathbf{NP}$ were true the consequences of showing that every **NP** problem important to science and industry is 'susceptible' to a polynomial-time algorithm is difficult to imagine, since this might yield many of the practical benefits that could be expected in a world in which $\mathbf{P} \approx \mathbf{NP}$ (Cook (2003)). Clearly, the significance of developing practical techniques to improve the solubility of **NP**-complete problems is huge, and is one of the key motivations for the work in this thesis.

Aside from the algorithms that are being applied to problems discussed in this thesis, there are also many restricted versions of **NP**-complete problems that can be solved using polynomial-time algorithms. Markedly, Constraint Satisfaction research has provided various conditions that have been shown to be sufficient to ensure tractability, most notably the works of Cohen and Jeavons (see Cooper et al. (1994); Jeavons & Cooper (1995); Jeavons et al. (1996, 1997, 1998); Cohen et al. (2000)).

---

[3] See Fortnow & Homer (2002) and Sipser (1992) for excellent introductions to the history of Computational Complexity and the **P** vs. **NP** problem.

[4] meaning that no polynomial-time algorithm exists that can solve a class of **NP**-complete problems.

1.2 Constraint Satisfaction

Pioneered by Montanari, Waltz and Mackworth in the 1970's, the Constraint Satisfaction Problem (CSP) describes a general framework for problems in which values must be assigned to a set of variables subject to specific constraints (Mackworth (1975)). It is one of the most prominent research areas in Theoretical Computer Science and Artificial Intelligence.

Widely studied in academia for several decades, research in this field is successfully beginning to penetrate mainstream industry (Wallace (1996)). With the advent of the modern computer contributing to the significant growth of this field, Constraint Satisfaction research is being applied to numerous Operational Research (OR) problems, including timetabling [5], location, scheduling, car sequencing, cutting stock, vehicle routing and rostering (see Brailsford et al. (1999)). In particular, one type of CSP, Propositional Satisfiability (*SAT*), has been adopted by the semiconductor industry as a methodology (Formal Verification) for testing the design of Integrated Circuits (Chips).

1.2.1 Formal Verification

In the context of hardware and software systems, Formal Verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property. Over the past decade (though much more so within the last few years) formal-methods have been successfully integrated into Electronic Design Automation (EDA) tools as a technique to detect 'bugs' in Chip designs (Prasad et al. (2005)). Roughly half of the costs for any design project are on verifying the design. About 70% of first fabricated silicon chips come back with errors, not because of problems with yield or timing, but because of functional problems that were missed during testing [6]. This, coupled with the increasing complexity of Chip design, is posing semiconductor companies with escalating costly delays, potential recall of faulty chips and huge reputational risks.

With around 50% of the entire design effort currently focused on Verification this increasing bottleneck is lengthening the design cycle, delaying time-to-market and eating into profits. Consequently, Chip manufacturers are applying pressure to EDA tool vendors to deliver more effective and robust Verification algorithms[7].

These algorithms, typically *SAT*-based, have been embraced by industry because they help ease a real 'customer pain', namely the Chip testing bottleneck. A lack of market opportunity could be one reason why Constraint-based approaches to more common OR problems (such as timetabling) have been adopted with less fervour in comparison (Broadfoot & Broadfoot (2003)) — besides the typical 'barriers' cited as the cost of technology/methodology transfer, and the level of knowledge required (Bowen & Hinchey (1997)).

---

[5] See Schaerf (1999) for an excellent survey of algorithmic techniques to solve the TIMETABLING PROBLEM.
[6] FPGA Explosion Will Test EDA - http://www.elecdesign.com/Articles/Index.cfm?ArticleID=15910&bypass=1
[7] Fragmentation of the IC Verification Process - http://www.edat.com/NEA21.htm

As a potential consequence of this 'market force' *SAT*-algorithms have evolved relatively independently from the field of Constraint Satisfaction. Nevertheless, in recent years cross-fertilisation has begun between these two research areas, particularly the techniques concerning *Resolution* (*SAT*) and *Consistency* (CSP) (see Walsh (2000a); Rish & Dechter (2000); Mitchell (2003); Kullmann (2004); Bennaceur (2004)).

1.3 Propositional Satisfiability

The SATISFIABILITY PROBLEM in propositional logic (*SAT*) is the quintessential **NP**-complete problem, and is a particularly important type of CSP. Over the past decade dramatic improvements in *Resolution*-based algorithms have given rise to *SAT*-Solvers (such as Moskewicz et al. (2001) ZCHAFF, Goldberg & Novikov (2002) BERKMIN and Een & Sorensson (2003) MINISAT) that can solve instances with thousands and even millions of variables (Hoos & Stützle (2000)).

Given a propositional formula on a set of Boolean variables, *SAT* asks whether or not there exists an assignment to the set of variables such that the formula evaluates to *True*. More specifically, *SAT*, in Conjunctive Normal Form (CNF), consists of the conjunction of a number of *clauses*, where a clause is a disjunction of a number of propositions or their negations. Given a set of clauses $C_0, C_1, \ldots, C_{m-1}$ on the propositions $x_0, x_1, \ldots, x_{n-1}$, the problem is to determine whether the formula $F = \bigwedge_{j<m} C_j$ has an assignment of values to the propositions such that it evaluates to *True*.

1.4 Motivation and Thesis Contribution

Two fields of Computer Science - Constraint Satisfaction and Propositional Satisfiability - have developed relatively independently over the past half century, it is only in recent years that the mutual benefits of these fields have started to be explored. As the performance of *SAT*-Solvers begins to plateau the necessity to look towards other fields for inspiration to continue progression has become more prevalent. Since the *SAT* problem is a restricted type of CSP and many CSPs can be represented as *SAT* instances it seems sensible to explore and strengthen the relationship between Constraint Satisfaction and Propositional Satisfiability research and to capitalise on synergies between these two fields.

The results presented in this thesis are part of a research program aimed at bridging the gap between Propositional Satisfiability and Constraint Satisfaction. One of the main challenges is to combine the inherent efficiencies of *SAT*-Solvers, operating on uniform encodings, with the much more sophisticated propagation techniques of CSP formalisms. The aim is to develop a better understanding of Computer Science as part of a wider goal, which aims to benefit the scientific and industrial communities in three ways:

1. Increase the theoretical understanding of the complexity and tractability of natural problems.

2. Provide practical tools and algorithms that will benefit both CSP and *SAT* research.

3. Demonstrate the value of this research by applying it to pertinent scientific and industrial problems.

Whilst most current research focuses on how *SAT* techniques can be utilised by the Constraint Satisfaction community, this thesis addresses the opposite, asking what CSP techniques can aid the *SAT* community. I have attempted to make the content of this thesis 'broad' enough so that it is interesting to the wider audience, but 'deep' enough so as to provide important and valuable theoretical *and* practical insights into the field. Although the general consensus agrees that bridging the two fields of Propositional Satisfiability and Constraint Satisfaction is mutually beneficial only a handful of researchers have crossed the chasm.

1.4.1 Theoretical Results

After introducing the relevant background and providing a detailed survey of *SAT* and CSP encodings, my in-depth analysis of these encodings uncovers three categories. These encoding categories are inspired by observing how the CSP *micro-structure* is constructed and expressed. Utilising the ideas presented in this thesis, and cross-fertilising these two fields, I provide a framework for current and future *SAT* and CSP encodings to be developed. As a result of this framework I define a new encoding and demonstrate that in certain circumstances this encoding is preferable to other encodings. This new encoding also opens up the potential for additional CSP to *SAT* encodings.

After categorising *SAT* and CSP encodings the focus of this research shifts to characterising them. Previous empirical work by others comparing the algorithmic performance on instances with different *solution-density* has been performed, and it has been shown that this measure can be used as a guide for choosing between using stochastic and branching algorithms. I provide a comprehensive characterisation of current *SAT* to CSP encodings and I prove that some encodings result in problems with a higher number of solutions than others. Since it has been shown in the literature that *solution-density* is an important factor in determining the solubility of an instance the results in this thesis provide a guide to assist in choosing one encoding over another.

Similarly, a significant amount of theoretical analysis has been published comparing stochastic and branching *SAT* and CSP algorithmic techniques on *SAT* and CSP encodings, however, very little research has compared the effect of enforcing local-consistency algorithms in each of these domains. After introducing the main algorithmic techniques from the *SAT* and Constraint Satisfaction communities, I use a graph-theoretic framework to reconfirm and strengthen the equivalence between the *SAT* and CSP-based proof methods. I provide a comparison of local-consistency techniques on each of the *SAT* to CSP encodings. One result of this work has direct practical implications regarding the choice of encoding; that enforcing local-consistency on one type of encoded problem does more work than on other encoded instances. I also prove that enforcing certain levels of local-consistency on some types of encoded *SAT*

instances does zero work if each clause has distinct literals, which is supported by the empirical results presented in this thesis.

In addition, using this framework I bridge further between the CSP and *SAT* techniques by introducing the concept of Extended-*Consistency*, thus providing a more complete picture of extended proof-systems. Extended proof-systems are some of the most powerful systems known, by allowing the introduction of auxiliary variables to maintain a constant arity. Relatively little practical work has been done on extended proof-systems. *SAT* and CSP algorithms that employ additional variables tend to apply them in an 'ad-hoc' manner. This motivates the definition of Extended-*Consistency*, a new proof-system that may make it easier for synergies between extended proof-systems to be cultivated and explored.

1.4.2 Empirical Results

Reviewing previous empirical studies of encodings highlights two problematic issues that call into question the validity of the wider implications of the results. First, typically one type of problem is used as a benchmark to compare encodings. Second, either stochastic or backtracking algorithms are applied to the encoded problems. Clearly both of these choices may bias the results, since an encoding/algorithm may 'favour' a particular problem. Although each author may proclaim the benefits of their encoding, a rigorous empirical and theoretical investigation remains to be performed to better determine their advantages and disadvantages. However, to perform a comprehensive and rigorous survey of these encodings is arguably a mammoth task.

I address these two issues by applying a local-consistency algorithm to a wide variety of unsatisfiable problems. As a result of applying this algorithm to CSP encoded *SAT* instances the experimental results strongly indicate that the choice of encoding has a dramatic effect on reducing the search-space. These empirical results are consistent with my theoretical results. I demonstrate that enforcing a small local-level of consistency on problems using a certain encoding does not solve any of the *SAT* benchmarks[8]. This is in stark contrast to problems encoded using another encoding, where I show that enforcing such a low-level of consistency can solve a large number of 'hard' *SAT* benchmark families. In particular I show that converting *SAT* instances to CSP and applying local-consistency can not only solve many 'hard' *SAT* instances, but this technique can even compete with state-of-the-art *SAT*-Solvers. Branching algorithms can prove both satisfiability and unsatisfiability, whereas stochastic algorithms can only prove satisfiability. Proving unsatisfiability has been shown to be more difficult for branching algorithms, so I suggest that future research is focused on the development of sophisticated local-consistency algorithms, which might be a viable approach to redress this balance.

[8] They do not prove unsatisfiability.

1.4.3 Publications

1.4.3.1 Conference Papers: Refereed

- Daniel J Hulme, Robin Hirsch, Bernard Buxton, and R.Beau Lotto. A new reduction from *3-SAT* to n-Partite Graphs. *In FOCI07: IEEE Symposium on Foundations of Computational Intelligence, IEEE Symposium on Computational Intelligence, Hawaii, USA, April 2007*.

  Abstract. The Constraint Satisfaction Problem (CSP) is one of the most prominent problems in artificial intelligence, logic, theoretical computer science, engineering and many other areas in science and industry. One instance of a CSP, the satisfiability problem in propositional logic (*SAT*), has become increasingly popular and has illuminated important insights into our understanding of the fundamentals of computation. Though the concept of representing propositional formulae as n-partite graphs is certainly not novel, in this paper we introduce a new polynomial reduction from *3-SAT* to $G\frac{n}{7}$ graphs and demonstrate that this framework has advantages over the standard representation. More specifically, after presenting the reduction we show that many hard *3-SAT* instances represented in this framework can be solved using a basic path-consistency algorithm, and finally we discuss the potential advantages and implications of using such a representation.

1.4.3.2 Papers in Preparation

In addition to the publications listed in this section, it is expected that at least two more papers will be submitted post-completion of this thesis:

1. Solving *SAT* using a Polynomial Consistency Algorithm.

   - This paper is constructed from Chapters 2, 3 and 6. It is an extended version of Hulme et al. (2007) containing a more detailed definition of the encodings, experiments and results.

2. Categorising *SAT* and CSP Encodings.

   - The results from Chapters 4 and 5 will be used for the content of this paper. More specifically, it will survey *SAT* and CSP encodings, describe how they can be characterised according to their mapping and introduce the INVERSE encoding.

3. Characterising *SAT* to CSP Encodings.

   - This paper will be constructed from Chapters 5 and 6. In this paper I will show how to characterise encodings according to their *solution-density*, detail the theoretical results of comparing *SAT* and *Consistency* techniques and provide the empirical evidence to support these claims.

1.5 Thesis Organisation

Here I give an overview of each chapter.

Chapter 1: Introduction and Motivation

In this chapter I introduce the context as well as the major topics covered in this thesis. I highlight the contributions to the field of Computer Science and give a content overview of each chapter.

Chapter 2: Definitions and Background

In Chapter 2 I introduce the necessary definitions and background information required for the rest of the thesis. I define the CSP and *SAT* problems, as well as a number of basic Graph Theory concepts, and show how CSPs are represented graphically. The aim of this chapter is to lay the foundation for the remainder of the thesis, where I provide a comprehensive and detailed theoretical analysis of how CSP and *SAT* problems are mapped using a graph-theoretic framework. This framework is used as an effective bridge between each of the two problem domains and is the primary mechanism used throughout this thesis to understand the similarities and differences between *SAT* and CSP techniques.

Chapter 3: Literature Review

In the first part of the Chapter 3 I review the encoding literature, providing a detailed survey of CSP and *SAT* encodings as well as critically assessing the major theoretical and empirical studies. The focus of the latter part of this chapter shifts towards the proof-complexity of polynomial algorithms and problems that are hard. This review highlights a number of gaps in the literature that are addressed in this thesis.

Chapter 4: Categorising Encodings

In Chapter 4 I demonstrate that all of the *SAT* to CSP encodings defined in this thesis (and vice-versa) can be categorised as one of three types of mappings. Categorising the encodings in this way highlights scope for several new CSP to *SAT* encodings, one of which I formally define and demonstrate its relative advantages over some of the other encodings; highlighting one major benefit of examining the relationship between *SAT* and CSP research.

Chapter 5: Characterising *SAT* to CSP Encodings

In Chapter 5 I demonstrate that the graph-theoretic approach is a useful framework to explore the similarities and differences between *Consistency* and *Resolution* techniques, and use it to show that *Consistency* and *Resolution* are the same procedure. I separate *SAT* to CSP encodings according to their *solution-density* and prove that some encodings will result in CSPs with a higher number of solutions than others.

The second part of this chapter addresses a major gap in theoretical research, which is comparing the performance of polynomial preprocessors on the various encodings. Here I provide a theoretical comparison of *Resolution* and *Consistency*-based techniques on the *SAT* to CSP encodings and I discuss the practical implications. For instance, I show that enforcing a low-level of local-consistency on the

DUAL encoded CSP does much more work in comparison to using the LITERAL encoding [9]. I also prove that enforcing *path*-consistency on LITERAL encoded *3-SAT* instances does zero work if each clause has distinct literals, which is consistent with the empirical results presented in Chapter 6.

Towards the end of this chapter I focus again on the relationship between the *Consistency* and *Resolution* proof-systems. Inspired by the work of Tseitin, Baker and Mitchell, I build upon this relationship by defining Extended-*Consistency* - the generalisation of Extended-*Resolution* - therefore allowing synergies between these two extended proof-systems to be explored. Using an example I demonstrate that the use of auxiliary variables can be automated to maintain constraint arity.

Chapter 6: Empirical Analysis of the LITERAL and DUAL Encodings

One of the main results of Chapter 5 is that enforcing local-consistency on DUAL encoded problems does more work than on LITERAL encoded instances. In Chapter 6 I address the two experimental issues raised in Chapter 3 by enforcing local-consistency on a wide variety of 'hard' unsatisfiable problems. Here I compare the performance of enforcing *path*-consistency on LITERAL and DUAL encoding problems, and demonstrate that it does not solve any of the *SAT* benchmarks when LITERAL encoded, which is in stark contrast to problems encoded using the DUAL encoding. I show that enforcing a low-level of consistency on the DUAL encoding can not only solve a surprising number of what are considered 'hard' *SAT* benchmarks, but can also compete with state-of-the-art *SAT*-Solvers.

Chapter 7: Discussion, Future Work and Exploitation

Finally, in Chapter 7 I provide a summary of the thesis contribution and I discuss the wider context and implications of this research. I introduce other areas of study that complement this research, highlight future work and discuss what I think are the next major areas of study relating these two fields.

As a requirement of the Engineering Doctorate, Research Engineers are asked to discuss the dissemination and exploitation aspects of their research. I provide an overview of the use of *SAT* algorithms in EDA and highlight many of the prominent issues facing the Chip design industry. I discuss the direct benefits that improvements in *SAT* technology can have to the Formal Verification aspect of the semi-conductor industry and beyond.

[9] See Sections 3.3.1 and 3.3.2 for definitions of the LITERAL and DUAL encodings respectively.

Chapter 2

Definitions and Background

In this chapter I introduce the necessary definitions and background information required for the rest of the thesis. I start by introducing **P**, **NP**, and the concept of polynomial reduction. After formally defining the two problems domains of Constraint Satisfaction and Propositional Satisfiability, I introduce several basic graph-theoretic concepts and demonstrate one way that CSPs can be visualised. Finally I introduce the algorithmic techniques that are applied to both the CSP and *SAT* problems that are discussed in later chapters.

This chapter lays the foundation for the remainder of this thesis, where I provide a comprehensive and detailed theoretical analysis of how CSP and *SAT* problems are encoded and mapped using a graph-theoretic framework. Whilst the use of Graph Theory is a fundamental aspect of CSP research and although representing CSPs as graphs is certainly not novel, as far as I am aware, this is the first major attempt using this approach to show how these fields are related. This framework will be used as an effective bridge between each of the two problem domains and is the primary tool used throughout this thesis to understand the similarities and differences between *SAT* and CSP techniques.

2.1 P and NP

The following definitions are adapted from Urquhart (1995).

Let Σ be a finite alphabet. Σ^* is the set of all finite strings over Σ, and a *language* is defined as a subset of Σ^*; that is, a set of strings over a fixed alphabet Σ. Let $\mathcal{L}$ be the class of polynomial-time computable ('feasible') functions.

Definition 2.1.1. *If Σ_1 and Σ_2 are finite alphabets, a function f from Σ_1^* into Σ_2^* is in $\mathcal{L}$ if it can be computed by a deterministic Turing machine in time bounded by a polynomial in the length of the input.*

Definition 2.1.2 (Proof-system)**.** *If $L \subseteq \Sigma^*$, a* proof system *for L is a function $f : \Sigma_1^* \to L$ for some alphabet Σ_1, where $f \in \mathcal{L}$ and f is onto. A proof system is* polynomially bounded *if there is a polynomial $p(n)$ such that for all $y \in L$, there is an $x \in \Sigma_1^*$ such that $y = f(x)$ and $|x| \leq p(|y|)$.*

A set of strings is in the class **P** (**NP**) if it is recognised by a deterministic (non-deterministic) Turing machine in time bounded by a polynomial in the length of the input. More specifically, a set S of strings is in **P** if its characteristic function is in $\mathcal{L}$, while it is in **NP** if the condition $y \in S$ can be expressed in the form $\exists x(|x| \leq p(|y|) \wedge R(x, y))$, where p is a polynomial and R is a polynomial-time computable relation (Urquhart (1995)).

2.1.1 Polynomial Reduction

The method of showing that a problem is **NP**-complete by polynomial reduction is one of the most elegant and productive in Computational Complexity (Adleman & Manders (1977)). It is a means of providing compelling evidence that a problem in **NP** is not in **P**.

Cook (2000) defines the following:

Definition 2.1.3. *Suppose that L_i is a language over $\Sigma_i, i = 1, 2$. Then $L_1 \leq_p L_2$ (L_1 is polynomially reducible to L_2) iff there is a polynomial-time computable function $f : \Sigma_1 \rightarrow \Sigma_2$ such that $x \in L_1 \Leftrightarrow f(x) \in L_2$, for all $x \in \Sigma_1$.*

Definition 2.1.4. *A language L is **NP**-complete iff L is in **NP**, and $L' \leq_p L$ for every language L' in **NP**.*

Proposition 2.1.1. *Given any two languages, L_1 and L_2:*

1. *If $L_1 \leq_p L_2$ and $L_2 \in$ **P** then $L_1 \in$ **P**.*
2. *If L_1 is **NP**-complete, $L_2 \in$ **NP**, and $L_1 \leq_p L_2$ then L_2 is **NP**-complete.*
3. *If $L \in$ **P** and L is **NP**-complete, then **P** = **NP**.*

2.1.2 NP and co-NP

A set of strings is in the class **co-NP** if it is the complement of a language in **NP**. One of the most important questions in Theoretical Computer Science lies in the result of Cook & Reckhow (1979):

Proposition 2.1.2. ***NP** = **co-NP** iff there is a polynomial-bounded proof-system for the classical tautologies.*

Since the complexity class **P** is closed under complementation, it follows that if **NP** = **co-NP** then **P** = **NP**.

2.2 Constraint Satisfaction

Here I formally define the Constraint Satisfaction Problem and relevant notations, which have been adapted from Green (2005) and Rossi et al. (2006).

Definition 2.2.1 (r-ary relation)**.** *A r-ary **relation** over $\mathcal{D}$ is a subset of $\mathcal{D}^r$, where $\mathcal{D}$ is any set. A **tuple** is an element of an r-ary relation.*

Definition 2.2.2 (Constraint Satisfaction Problem)**.** *A CSP is triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where:*

- *$\mathcal{V}$ is a finite set of **variables**, $\{V_0, \ldots, V_{n-1}\}$.*
- *$\mathcal{D}$ is a set, $\{D_0, \ldots, D_{n-1}\}$, where each $D_i \in \mathcal{D}$ is the set of values that V_i can take, called the **domain**.*
- *$\mathcal{C}$ is a finite set of **constraints**, $\{C_0, \ldots, C_{m-1}\}$, where each constraint $C_i \in \mathcal{C}$ is a pair $\langle s_i, R_i \rangle$, such that*
  - *s_i is an ordered tuple of variables, called the **constraint scope***
  - *R_i is an $|s_i|$-ary relation over $\mathcal{D}$, called the **constraint relation**. $|s_i|$ is the **arity** of the constraint.*

  *The CSP asks if there is a **solution**, that is, a mapping θ of all variables to domain values such that for each $\langle s, R \rangle \in \mathcal{C}$, $\theta(s_i) \in R$.*

The constraint scope is a list of variables over which the constraint acts, and the constraint relation defines the allowed combinations of values for this list of variables.

Definition 2.2.3 (partial and full assignments)**.** *Given any CSP = $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, a **partial assignment** A is a subset of variables $V \subseteq \mathcal{V}$ with some mapping V to domain values. A is a **full assignment** if $|A| = |\mathcal{V}|$. If A is a* partial assignment *we may denote this as $\{x_0^{a_0}, x_1^{a_1}, \ldots, x_{n-1}^{a_{n-1}}\}$, where $\{x_0, x_1, \ldots, x_{n-1}\}$ is the domain of A and for each $i < n$, $a_i = A(x_i)$.*

Definition 2.2.4 (satisfying assignments)**.** *Given any CSP = $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, if the variables V, in the scope of a constraint $C \in \mathcal{C}$, are contained entirely by the variables in an assignment A, then A **covers** V. A **satisfies** C if V is covered by A and V is mapped to domain values allowed by C. A **partial satisfying assignment** is an assignment A to some subset of variables $V \subseteq \mathcal{V}$ such that every covered constraint in V is satisfied by A. If A is a* partial assignment *then I write $\gamma(A)$ to denote that A is a* partial satisfying assignment.

A full satisfying assignment *(one that covers all variables) is also referred to as a **solution-tuple** or **solution**. A CSP that has a satisfying assignment is **satisfiable**, otherwise it is **unsatisfiable***

Definition 2.2.5 (conflict)**.** *Given any CSP = $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, a **conflict** is a partial assignment A to some subset of variables $V \subseteq \mathcal{V}$ such that there is at least one covered constraint in V that is not satisfied by A. I write $\chi(A)$ to denote that A is a* conflict.

Definition 2.2.6 (nogood)**.** *Given any CSP = $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, a partial assignment, A, defined on k variables is a k-ary* ***nogood*** *iff it cannot be extended to a solution. In this case I write $\eta(A)$ to denote that A is a* nogood*. Any* conflict *is necessarily a* nogood*, but the converse is not true in general.*

In summary, if a partial assignment A is denoted as $\{x_0^{a_0}, x_1^{a_1}, \ldots, x_{n-1}^{a_{n-1}}\}$, then:

- if A is a *partial satisfying assignment* we say $\gamma\{x_0^{a_0}, \ldots, x_{n-1}^{a_n-1}\}$
- if A is a *nogood* we say $\eta\{x_0^{a_0}, \ldots, x_{n-1}^{a_n-1}\}$
- if A is a *conflict* we say $\chi\{x_0^{a_0}, \ldots, x_{n-1}^{a_n-1}\}$

2.2.1 Implicit and Explicit CSP Representations

When defining a CSP there are two typical representations for these relations:

1. **implicit** representation implies both the relation and scope by a definition.
2. **explicit** representation explicitly lists the tuples of the constraint relation.

Suppose we define a CSP that has variables x and y, over the domain $\{R, G, B\}$ and enforce the constraint, C, such that x and y must take different values.

Example 2.2.1 (Implicit Representation)**.** *The inequality relation allows pairs of values that are not equal and can be written as $\neq_X$, where X is the domain.*

The example constraint, C, above can be written as:

$$x \neq_{\{R,G,B\}} y$$

In this case, the set of variables in the definition form the scope.

Example 2.2.2 (Explicit Representation)**.** *C may be written explicitly as:*

$$\langle\langle x, y\rangle, \{\langle R, G\rangle, \langle R, B\rangle, \langle G, R\rangle, \langle G, B\rangle, \langle B, R\rangle, \langle B, G\rangle\}\rangle$$

This may also be written in complement notation as:

$$\langle\langle x, y\rangle, \backslash\{\langle R, R\rangle, \langle G, G\rangle, \langle B, B\rangle\}\rangle$$

In this thesis I often write the constraints using the superscript notation:

$$C_{x,y} = \{\eta\{x^R, y^R\}, \eta\{x^G, y^G\}, \eta\{x^B, y^B\}\}$$

Though the *implicit* representation is typically more compact than the *explicit* representation the majority of the literature bridging *SAT* and CSP techniques adopts the *explicit* approach since *SAT* clauses are explicit list of *nogoods*. However, it is important to note that for general CSPs the compactness of the *implicit* representation of a problem may be preferable since the successful application of an algorithm is usually a function of the size of the problems definition. For instance, it might be impractical to represent a set of explicit *nogoods* of a constraint in memory, whereas the implicit constraint could be trivially represented (Do & Kambhampati (2001)).

2.2.2 Graphs

One of the archetypal examples used to demonstrate CSPs is that of GRAPH 3-COLOURABILITY, originally shown to be **NP**-complete by Karp (1972). Before I define GRAPH K-COLOURABILITY first it is necessary to introduce some basic concepts from Graph Theory.

Definition 2.2.7 (Undirected graph)**.** *An undirected graph (V, E) consists of a set, V, of nodes together with a set E of edges where all edges have the form $\{x, y\}$ for some $x, y \in V$.*

All graphs are assumed to be undirected.

Definition 2.2.8 (Hypergraph)**.** *A **hypergraph** is a pair (V, H) where V is a set of nodes and H is a set of hyperedges. Each hyperedge h is a set of nodes from V ($h \subseteq V$). Hypergraphs are a generalisation of graphs, where each hyperedge may connect more than two nodes.*

A k-hypergraph is an hypergraph with all hyperedges having size k.

Definition 2.2.9 (GRAPH K-COLOURABILITY, Garey & Johnson (1990))**.** *Given a graph $G = (V, E)$ and positive integer $k \leq |V|$ does there exist a function $f : V \rightarrow \{1, 2 \ldots k\}$ such that $f(u) \neq f(v)$ whenever $\{u, v\} \in E$?*

Example 2.2.3 (GRAPH 3-COLOURABILITY as CSP)**.** *A simple instance of* GRAPH 3-COLOURABILITY *is shown in Figure 2.1. This instance has five regions $\{x_0, \ldots, x_4\}$ with a total of six neighboring pairs of regions. Each region can be coloured either Red (R), Green (G) or Blue (B).*

One possible formulation of this instance as a CSP is as follows:

- *$\mathcal{V}$ is the set of nodes to be coloured, $\{x_0, x_1, x_2, x_3, x_4\}$*
- *$\mathcal{D}$ is the domain over each variable, $\mathcal{D}_{x_0} = \mathcal{D}_{x_1} = \mathcal{D}_{x_2} = \mathcal{D}_{x_3} = \mathcal{D}_{x_4} = \{R, G, B\}$*
- *$\mathcal{C}$ is the set of restrictions (constraints) over sets of variables, where $\langle x_i, x_j \rangle$ means that x_i and x_j are adjacent nodes:*
  - $\langle \langle x_0, x_1 \rangle, \setminus \{\langle R, R \rangle, \langle G, G \rangle, \langle B, B \rangle\} \rangle$
  - $\langle \langle x_0, x_4 \rangle, \setminus \{\langle R, R \rangle, \langle G, G \rangle, \langle B, B \rangle\} \rangle$

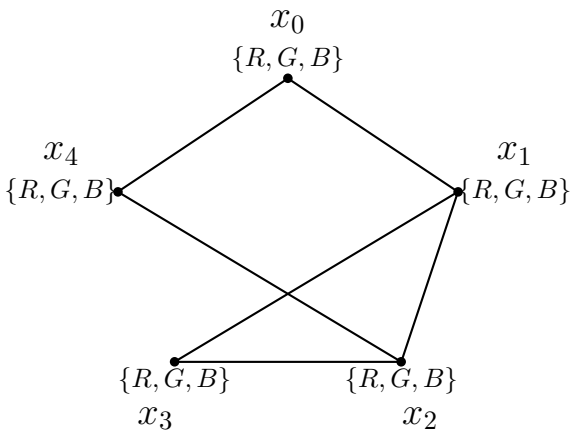

Figure 2.1: An instance of GRAPH 3-COLOURABILITY with five variables over the domain $\{R, G, B\}$ and six constraints.

- $\langle\langle x_1, x_2\rangle, \setminus\{\langle R, R\rangle, \langle G, G\rangle, \langle B, B\rangle\}\rangle$
- $\langle\langle x_1, x_3\rangle, \setminus\{\langle R, R\rangle, \langle G, G\rangle, \langle B, B\rangle\}\rangle$
- $\langle\langle x_2, x_3\rangle, \setminus\{\langle R, R\rangle, \langle G, G\rangle, \langle B, B\rangle\}\rangle$
- $\langle\langle x_2, x_4\rangle, \setminus\{\langle R, R\rangle, \langle G, G\rangle, \langle B, B\rangle\}\rangle$

In Example 2.2.3 a *partial satisfying assignment* is $\gamma\{x_1^B, x_2^G, x_3^R\}$ and a *conflict* is $\chi\{x_1^B, x_2^G, x_3^B\}$. A **full satisfying assignment** is $\gamma\{x_0^B, x_1^R, x_2^B, x_3^G, x_4^R\}$ as shown in Figure 2.2.

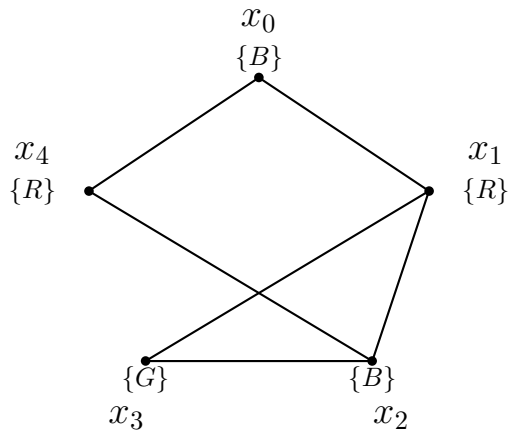

Figure 2.2: The $\gamma\{x_0^B, x_1^R, x_2^B, x_3^G, x_4^R\}$ solution to Example 2.2.3.

2.2.2.1 Representing CSPs Graphically

The work in this thesis draws heavily on graphical representations of CSPs. Dechter (1992a) describes three types of graphical representations of CSPs:

1. **Hypergraph**: where nodes represent the variables, and hyperedges (drawn as regions) group those variables that belong to the same constraint scope.

2. **Primal-graph**: represents variables by nodes and associates an edge with any two nodes residing in the same constraint scope.

3. **Dual-graph**: represents each constraint scope by a node and associates a labelled edge with any two nodes whose constraint scope share variables, where edges are labelled by the shared variables.

Figure 2.1 is the Primal-Graph of Example 2.2.3.

However, these do not represent the constraint relations (only the constraint scope) so only partially capture the CSP. *Primal-graphs* are binary, so they do not capture the true structure of a k-ary (for $k > 2$) CSP. Indeed, the generalisation of the *primal-graph* would be identical to the *hypergraph*, if one were to represent constraints as hyperedges. The *dual-graph* can also be confusing since it moves away from the simple graph representation, allowing the labelling of edges. In Chapter 4 I build upon this work and present a homogeneous framework that captures the various families of encodings.

The graphical method to represent CSPs I adopt is referred to as the CSP *micro-structure* (see Jegou (1993)). Unlike the graphical representation definitions above, the *micro-structure* captures the full structure of the CSP.

Definition 2.2.10 (micro-structure)**.** *The **micro-structure** of a binary CSP (the cardinality of the constraints are size two) is constructed from a CSP instance $P = (\mathcal{V}, \mathcal{D}, \mathcal{C})$. P is a graph with set of nodes $\mathcal{V} \times \mathcal{D}$ where each edge corresponds either to an assignment allowed by a specific constraint or to an assignment allowed because there is no constraint between the associated variables.*

Definition 2.2.11 (n-partite graphs)**.** *A graph is n-**partite** iff the nodes can be partitioned into n independent subsets (classes), such that every edge has its ends in different classes — i.e. nodes in the same partition class must not be adjacent.*

Definition 2.2.12 (clique)**.** *Given a graph G, a **clique** C is a subset of nodes of G such that every pair of distinct nodes in C are adjacent.*

A CSP is called k-ary if the maximum arity of any of its constraints is k. Unless otherwise stated, the CSPs in this thesis have variables with same domain cardinality (such as the GRAPH K-COLOURABILITY PROBLEM). The majority of CSPs in this thesis are represented graphically as a restricted type of n-partite graph called a G_k^n graph.

Definition 2.2.13 (G_k^n graph)**.** *A G_k^n **graph** is an n-partite graph, with each independent set containing* at most k *nodes. I refer to each set of k nodes as a **component set**.*

Definition 2.2.14 (The $\mathcal{G}_k^n$ Graph Problem)**.** *An instance of $\mathcal{G}_k^n$ is a G_k^n graph (for some n), and is a* satisfiable *instance if it contains an n-clique and is an* unsatisfiable *instance otherwise. Any n-clique of G_k^n contains at most one node from each component set.*

For convenience and clarity it is often easier to represent disallowed constraints in a graph G, called the *micro-structure complement* or $\bar{G}$. I denote a CSP that has i variables of arity j as CSP^i_j.

Definition 2.2.15 (micro-structure complement)**.** *The **micro-structure complement** of a binary CSP is constructed from a CSP instance $P = (\mathcal{V}, \mathcal{D}, \mathcal{C})$. P is a graph with set of nodes $\mathcal{V} \times \mathcal{D}$ where the edges joining pairs of nodes are disallowed by some constraint or are incompatible assignments for the same variable.*

Whereas an n-clique describes a solution in a CSP's *micro-structure* with n variables, an independent set of size n describes a solution in its *complement*. For instance, the $\bar{G^n_k}$ graph of Example 2.2.3 is shown in Figure 2.3. A full satisfying assignment to Example 2.2.3 is $\gamma\{x^B_0, x^R_1, x^B_2, x^G_3, x^R_4\}$. This independent set is highlighted in Figure 2.3, connecting a node from each of the five component sets by dotted lines.

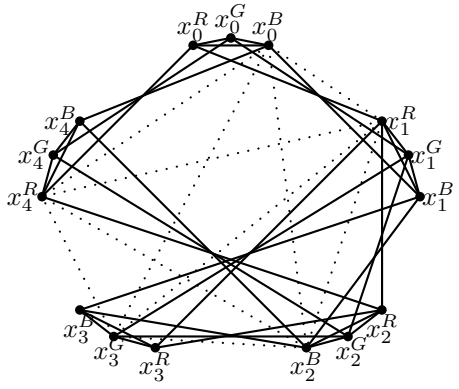

Figure 2.3: Example 2.2.3 as a $\bar{G^5_3}$ graph, with a solution-tuple, $\gamma\{x^B_0, x^R_1, x^B_2, x^G_3, x^R_4\}$, highlighted with dotted lines between the corresponding set of independent nodes.

So far these graph definitions have been restricted to binary CSPs, with edges in the *micro-structure complement* corresponding to disallowed constraints, and solutions mapping to independent sets between the component set nodes. These definitions extend naturally to non-binary cases. Hypergraphs (*micro-structure complements*) are constructed from CSPs with k-ary constraints. In this case, a set of nodes is a hyperedge of the *micro-structure complement* if it represents an assignment disallowed by a constraint, or else consists of a pair of incompatible assignments for the same variable.

2.3 CSP Algorithms

2.3.1 *Consistency*

Local-consistency conditions are properties of CSPs related to the consistency of subsets of variables or constraints. Mackworth (1975) defines three properties that characterise the local-consistency of

networks: *node*-, *arc*-, and *path*-consistent. Freuder (1978) later generalises this to k-consistent.

Definition 2.3.1 (k-consistent)**.** *A binary CSP = $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ is k-**consistent** if for every partial satisfying assignment P on any $k-1$ variables ($\subset \mathcal{V}$) for any other variable $v \in \mathcal{V}$ there exists a domain value d such that $P \cup \{v \mapsto d\}$ is also a satisfying partial assignment. If a CSP is k consistent for some k then we say that it is **locally consistent**. A 2-consistent binary CSP is called **arc-consistent**. A 3-consistent binary CSP is called **path-consistent**.*

A CSP that is k-consistent is said to have k-consistency. An algorithm that establishes a k-consistent CSP is said to enforce k-**consistency**. Local-consistency can be enforced via transformations of the problem called *constraint propagation*. Local-consistency conditions require that every consistent assignment can be consistently extended to a domain assignment in another variable. An algorithm that establishes the consistency of a problem is said to enforce local-consistency (also referred to as establishing a local-level of consistency).

Informally, a binary CSP is 1-consistent if every variable has a domain value that satisfies it. This is often referred to as enforcing *node*-consistency. Following from the definition above, a CSP is 2-consistent (or *arc*-consistent) if every pair of variables have domain assignments that can form a partial satisfying assignment. *Path*-consistent is the name given to a CSP such that every pair of partial satisfying assignments between two variables can form an extended partial satisfying assignment with every other variable.

Definition 2.3.2 ((i,j)-consistent)**.** *A CSP = $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ is (i,j)-**consistent** if for every satisfying partial assignment P on any i variables ($\subset \mathcal{V}$) there exists a satisfying assignment Q on every other set of j variables such that $P \cup Q$ is also a satisfying partial assignment. Note that $(k,1)$-consistent is the same as $(k+1)$-consistent.*

Enforcing a particular level of k-consistency does not necessarily determine the satisfiability of a problem, nor does it mean that the problem is j-consistent for any $j < k$. Tsang (1993) provides an excellent introduction to the foundations of the CSP field with many valuable examples to illustrate these points. A stronger notion of local-consistency is strong-k-consistency.

Definition 2.3.3 (strong-k-consistent, Tsang (1993))**.** *If a CSP is j-consistent, for all $j \leq k$, then it is **strong-k-consistent**. A CSP with n variables that is strong-n-consistent is called **globally consistent**.*

A large amount of work has been published on the development of efficient polynomial-time algorithms that enforce these various levels of consistency. This is because it is often the case that the CSP solution space can be significantly pruned using these techniques, which typically allows search-based algorithms to determine satisfiability more quickly.

2.3.2 Maintaining Arc-Consistency

Waltz Filtering Algorithm (Waltz (1975)), originally developed for computer vision, was probably the first *arc*-consistency algorithm (AC-1), which has a space-time complexity of $O(c + nd)$ and $O(d^3nc)$ respectively (where c is the number of constraints, n the number of variables, and d the size of the maximum domain). Twenty years later, and six iterations on, AC-6 was published by Bessiere & Cordier (1994) with a space-time complexity of $O(cd)$ and $O(d^2c)$. One of the most recent *arc*-consistency algorithms was published by Lecoutre et al. (2003), named AC-2001/3.3. This algorithm combines the properties of two previous AC methods, namely the simple implementation methods of AC-3 (Mackworth & Freuder (1985)) and the bi-directionality of AC-7 (Bessiére et al. (1999)).

Gaschnig (1974) suggests Maintaining Arc-Consistency (**MAC**) during backtracking search and gives the first explicit algorithm containing this idea. The **MAC** algorithm maintains *arc*-consistency on constraints with at least one uninstantiated variable. At each node of the search tree, an algorithm for enforcing *arc*-consistency is applied to the CSP. Since *arc*-consistency was enforced on the parent of a node, initially constraint propagation only needs to be enforced on the constraint that was posted by the branching strategy. In turn, this may lead to other constraints becoming arc inconsistent and constraint propagation continues until no more changes are made to the domains. If, as a result of constraint propagation, a domain becomes empty, the branch is a deadend and is rejected. If no domain is empty, the branch is accepted and the search continues to the next level.

2.3.3 Forward Checking

The Forward Checking algorithm (**FC**), introduced by Haralick & Elliott (1980), maintains *arc*-consistency on constraints with exactly one uninstantiated variable, which can be enforced in $O(d)$ time (where d is the size of the domain of the uninstantiated variable). **FC** is essentially a backtracking search algorithm that branches on variable labels until a *solution-tuple* is found or backtracks if another variable's domain is exhausted (often called 'wiped-out' in the literature).

FC analogous to the **DLL** procedure described in Section 2.5.5, and as with **DLL** many sophisticated improvements to the search procedure have been proposed. The focus of this thesis is on polynomial algorithms, but for the interested reader Bacchus & Grove (1995) provides an excellent introduction to **FC** and its variants, and Bessiere et al. (2002) published a more involved paper on solving non-binary CSPs using **FC**.

2.3.4 Stochastic Search for CSP

For dynamic vehicle routing where real-time performance is important, finding a fast 'good' solution is often much more valuable than finding the optimum too late. In the cases where speed is more important than accuracy *stochastic* methods are often preferable. As mentioned in Section 2.5.6, stochastic algorithms are a class of search that includes heuristics and an element of non-determinism to traverse

the search-space. The next move is partly determined by the outcome of the previous move, and these methods tend to be incomplete.

2.3.4.1 Min-Conflicts

Min-Conflicts (**MC**) is stochastic algorithm for CSPs in which a variable in a violated constraint is picked at random, and a value (in the domain of that variable) is assigned that most reduces the number of violated constraints.

2.4 Boolean Satisfiability

One of the original problems shown by Cook (1971) to be **NP**-complete, *SAT* is considered the one of the most important **NP** problems. The obvious difference cited between general CSPs and the SATISFIABILITY PROBLEM is that the former traditionally have non-binary domains with binary constraints, whereas the latter have binary domains with non-binary constraints.

Definition 2.4.1 (BOOLEAN SATISFIABILITY PROBLEM)**.** *The* SAT *problem in Conjunctive Normal Form (CNF) consists of the conjunction (*$\wedge$ *representing the Boolean* and *connective) of a number of* ***clauses****, where a clause is a disjunction (*$\vee$ *representing the Boolean* or *connective) of a number of propositions or their negations (****literals****).*

If x_i *represent propositions that can assume only the values* True *(*$\equiv 1 \equiv \top$*) or* False *(*$\equiv 0 \equiv \bot$*), then an example formula in CNF would be*

$$(x_0 \vee x_2 \vee \bar{x}_3) \wedge (x_3) \wedge (x_1 \vee \bar{x}_2)$$

where $\bar{x}_i$ *is the negation of* x_i*.*

Given a set of clauses $C_0, C_1, \ldots, C_{m-1}$ *on the propositions* $x_0, x_1, \ldots, x_{n-1}$*, the problem is to determine whether the formula* $F = \bigwedge_{j<m} C_j$ *has an assignment of truth values to the propositions such that it evaluates to* True*.*

Definition 2.4.2 (*k-SAT*)**.** *The* **k-SAT** ***problem*** *is a CNF formula with exactly* k *literals in each clause. For example, instances of* 3-SAT *are restricted to Boolean formulae in CNF with exactly three literals per clause.*

Formula 2.1 is a 3CNF formula with four variables and five clauses, and is a *satisfiable* instance of *3-SAT*; where one of the eight satisfying assignments (*solution-tuples*) is $\{x_0^1, x_1^1, x_2^0, x_3^0\}$:

$$(x_0 \vee x_1 \vee x_2) \wedge (\bar{x}_0 \vee x_1 \vee \bar{x}_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_0 \vee \bar{x}_1 \vee x_3) \wedge (\bar{x}_0 \vee \bar{x}_2 \vee \bar{x}_3) \tag{2.1}$$

I denote the set of *k-SAT* instances that has n variables and m clauses as *k-SAT*$_n^m$.

2.5 *SAT* Algorithms

2.5.1 *Resolution*

Propositional *Resolution* is a sound and complete proof-system for *SAT* defined by Robinson (1965).

Definition 2.5.1 (*Resolution*)**.** *As usual x denotes a Boolean variable with domain $\{0,1\}$. A literal over x is either x or $\bar{x}$, and a clause is a disjunction of literals. The main rule is the*

$$\textbf{\textit{Resolution Rule}}: \quad \frac{A \vee \bar{x} \qquad x \vee B}{A \vee B}$$

where $x \in \{x_0, x_1, \ldots, x_{n-1}\}$ and A and B are arbitrary clauses[1].

Resolution is a complete theorem proving method that 'searches' for a contradiction (i.e. the empty clause) by refutationally saturating a given clause set — that is, systematically and exhaustively applying all possible inferences using the *Resolution Rule* (Bachmair & Ganzinger (2001)).

Definition 2.5.2 (*Resolution* derivation)**.** *A Resolution* ***derivation*** *from a CNF formula Ψ is a sequence of clauses in which each clause is either in Ψ or derived from clauses in Ψ using the* Resolution Rule*. Given any unsatisfiable set of clauses, a contradiction, $\frac{\bar{x} \qquad x}{\bot}$, can always be derived using Resolution.*

Definition 2.5.3 (*k-Resolution*)**.** *A k-**Resolution** derivation on a CNF formula is a Resolution derivation of all of the possible clauses that contain at most $k-1$ literals.*

2.5.2 Extended-*Resolution*

Extended-*Resolution* proofs are exactly the same as standard *Resolution* proofs but with the addition of the *Extension Rule*. The *Extension Rule*, first suggested by Tseitin (1968), is a powerful addition to the *Resolution* system that allows the use of new literals as abbreviations for longer formulae. The length of an Extended-*Resolution* proof is the total number of different clauses in it.

Definition 2.5.4 (Extension Rule, Tseitin (1968))**.** *Given a CNF formula F, for arbitrary variables a, b, the **Extension Rule** introduces a **new** variable v (new relative to the CNF formula F) such that*

$$F \longrightarrow F \cup \{(\bar{v} \vee a \vee b) \wedge (v \vee \bar{a}) \wedge (v \vee \bar{b})\}.$$

The clause-set $\{(\bar{v} \vee a \vee b) \wedge (v \vee \bar{a}) \wedge (v \vee \bar{b})\}$ is the CNF representation of $v \leftrightarrow (a \vee b)$.

2.5.3 *NG-RES* Proof-System

de Kleer (1989) showed how to transform CSP instances into *SAT* and described a *Resolution* proof-system that is equivalent to enforcing local-consistency on the original CSP:

[1] Some authors often include the Weakening Rule for completeness, though it is not essential.

Weakening Rule: $\frac{A}{A \vee B}$

$$\frac{\begin{array}{c}(x_0 \vee x_1 \vee \cdots \vee x_{i-1}) \\ (\bar{x}_0 \vee X_0) \\ (\bar{x}_1 \vee X_1) \\ \vdots \\ (\bar{x}_{i-1} \vee X_{i-1})\end{array}}{(X_0 \vee X_1 \vee \cdots \vee X_{i-1})}$$

where X_j are clauses and x_j are literals, for $j < i$.

This idea of 'exhausting the domain' to derive implicit constraints was later defined by Baker (1995) who referred to it as 'a *Resolution* proof method for a CSP'. Recently Mitchell (2003) redefined it as *nogood-Resolution* (***NG-RES***).

Definition 2.5.5 (*nogood-Resolution*)**.** *Given a CSP* $= (\mathcal{V}, \mathcal{D}, \mathcal{C})$*. Given that the domain of a variable* x *is* $\{0, 1, \ldots, (d-1)\}$*, the* nogood Resolution Rule *allows one to infer a* nogood*, called the* resolvent*, from a set of* nogoods*, the* premises*, by resolving on* x*:*

$$\frac{\begin{array}{c}\eta\{\{x^0\} \cup X_0\} \\ \eta\{\{x^1\} \cup X_1\} \\ \vdots \\ \eta\{\{x^{d-1}\} \cup X_{d-1}\}\end{array}}{\eta\{X_0 \cup X_1 \cup \cdots \cup X_{d-1}\}}$$

where X_i *is a partial assignment (for* $i < d$*), and* $\eta\{\{x^0\} \cup X_0\}, \eta\{\{x^1\} \cup X_1\}, \ldots, \eta\{\{x^{d-1}\} \cup X_{d-1}\}$ *are* nogoods*.*

A nogood Resolution refutation *of a CSP is a derivation of the empty set, which means that there is a variable that cannot be part of a solution, hence there is no global solution.*

Mitchell showed that ***NG-RES*** is a sound and complete refutation system, that there is a ***NG-RES*** refutation of a CSP instance iff it is unsatisfiable.

2.5.4 Davis-Putnam Procedure

Although Robinson (1965) defined the *Resolution* proof-system, the basic *Resolution Rule* appeared several years earlier, most notably in an algorithm defined by Davis & Putnam (1960). This algorithm, the Davis-Putnam Procedure (**DP**), was the birth of the modern-day *SAT*-Solver. Besides the *Resolution Rule*, the **DP** algorithm introduces two more rules:

1. **unit literal rule**: this rule causes variable elimination and states that if there exists a clause in the CNF formula that contains only one literal, then the formula resulting from making that literal *True* has the same satisfiability as the original formula.

2. **pure literal rule**: if a literal either appears only positive or negative in the CNF formula then it is *pure*. This means that all clauses containing that literal can be deleted from the formula without altering its satisfiability.

The **DP** algorithm applies each of these rules until the CNF formula cannot be simplified further. The *Resolution Rule* is applied to eliminate one variable, and if successful applies each of the above rules again. This process continues until either the empty clause is generated (meaning that the formula is unsatisfiable) or there are no variables left to remove (hence it is satisfiable).

In the worst-case, each time a variable is eliminated the number of clauses may grow quadratically, therefore the **DP** algorithm may require an exponential amount of memory. Despite the **DP**'s 'legacy' it was quickly superseded by the Davis-Logemann-Loveland Procedure, which introduced a minor syntactic modification of the original algorithm to address this memory problem.

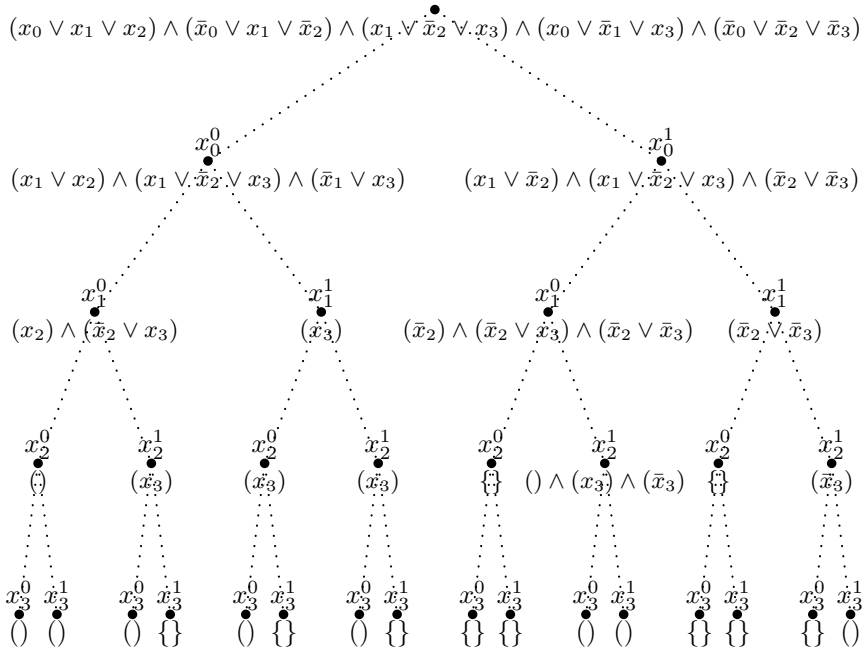

Figure 2.4: A binary search tree of Formula 2.1.

2.5.5 Davis-Logemann-Loveland Procedure

Davis et al. (1962) replaced the *Resolution Rule* in **DP** by a *splitting rule* in order to avoid the exponential memory explosion. This change resulted in the Davis-Logemann-Loveland Procedure (**DLL**), which forms the foundations of most *SAT*-Solvers used today. The memory requirement for **DLL** is polynomial and it can handle very large formulae without memory overflow. The **DLL** algorithm (Algorithm 1) is categorised as a 'branch and search' procedure[2], which are typically represented as binary trees. At the root of the tree is the CNF formula and each node represents the parent formula under a variable assignment (this step is referred to as *unit propagation*). If any of the leaf nodes is an empty set then the formula is satisfiable, otherwise if the procedure generates an empty clause then the algorithm backtracks and continues down a different route (referred to as the *conflict* and *splitting rule*). The **DLL** algorithm searches the entire tree unless an empty set is generated at a leaf. Figure 2.4 represents a binary search

[2]Also known as *branching*, and *branch and bound*

tree for Formula 2.1. Notice that each path (from route to leaf) terminating with an empty set ({}) corresponds to one of the eight satisfying assignments to the formula shown in Table 3.3.

If a clause has all but one of its literals evaluate to *False* then the remaining 'free' literal must evaluate to *True* to satisfy the clause. These clauses are called *unit clauses* and the 'free' literal is called the *unit literal*. The process of iteratively forcing *unit literals* to be assigned a value is called *Boolean Constraint Propagation* (BCP). BCP and backtracking constitute the core operations of the **DLL** algorithm.

Algorithm 1 DLL algorithm on a CNF formula F.

```
DLL(F)
if F is empty then
   return SAT
else if there is an empty clause in F then
   return UNSAT
else if there is a pure literal x in F then
   return DLL(F(x))
else if there is a unit clause (x) in F then
   return DLL(F(x))
else
   select a variable y mentioned in F
   if DLL(F(y))=SAT then
      return SAT
   else
      return DLL(F(¬y))
   end if
end if
```

The **DLL** procedure is a backtracking depth-first search through partial truth assignments augmented by the *unit clause* and *pure literal* rules. The '`select a variable` y `mentioned in` F' rule from Algorithm 1 may use sophisticated heuristics as described in the PhD thesis by Zhang (2003), one of the developers of CHAFF, which is probably one of the most influential *SAT*-Solvers developed in recent times (see Moskewicz et al. (2001)).

Since the CNF format is very low-level, *SAT*-Solvers are not very structure-aware, but this can have great advantages. *SAT*-Solvers are typically highly optimised to perform efficient deduction on CNF Boolean formulae, which can be compactly stored in memory with very good cache behaviour. Highly efficient deduction algorithms have been proposed to perform reasoning on clauses. Many branching heuristics and *conflict-driven* learning techniques also rely on the fact that the formula is in CNF. By combining these techniques, modern *SAT*-Solvers can routinely solve instances with hundreds of thousands of variables (Bordeaux et al. (2006)).

2.5.6 Stochastic Search for *SAT*

The **DP** and **DLL** algorithms (and the majority of their variants) are complete; given enough time and space they will always determine whether a CNF formula is satisfiable or not. Less so for *SAT* appli-

cations than for CSP is the important trade-off between accuracy and speed. In Formal Verification for example, it is critically important that *SAT*-Solvers produce the correct answer (usually) regardless of the amount of time it takes, since the incorrect functioning of hardware could have devastating consequences.

The GSAT algorithm, for instance, is stochastic (also known as *local-search*). As described in Section 2.3.4, stochastic algorithms are based on mathematical optimisation techniques, and although these classes of algorithms cannot prove a formula is unsatisfiable, given enough time they may find a satisfying solution if there is one. Typically these algorithms are applied to problems where there is likely to be a satisfying assignment, which I elaborate on more in later chapters.

2.5.6.1 GSAT

GSAT was proposed by Selman et al. (1992) and implements a greedy algorithm that attempts to minimise an objective function. The GSAT algorithm (Algorithm 2) begins by choosing a random variable assignment, if the instance is not satisfied under this assignment then the algorithm chooses a variable with the maximum score and negates it (also called *flipping*). The score of a variable is the difference between the current number of unsatisfied clauses and the number of unsatisfied clauses if the variable were negated. If some variables have the same score then one is chosen at random. If no variables have a positive score (or a maximum number of *flips* have been performed) then the algorithm will restart by choosing another random variable assignment. This process continues until a predefined number of restarts have been reached.

Algorithm 2 GSAT algorithm.

```
for i = 1 to MAXTRIES do
  T = a randomly generated assignment
  for j = 1 to MAXFLIPS do
    if no unsatisfied clause exists then
      return T
    else
      x = choose variable in T with max score
      if score of x < 0 then
        break
      else
        negate x
      end if
    end if
  end for
end for
return UNKNOWN
```

2.5.6.2 WALKSAT

Another popular stochastic search algorithm, called WALKSAT, was proposed by McAllester et al. (1997). This differs from GSAT mainly in the selection of the variables to be negated. The score given to the variables in WALKSAT is the number of clauses that will change from satisfied to unsatisfied if

that variable were to be *flipped*. If the instance is not satisfied then an unsatisfied clause is chosen at random. If this clause contains a variable with a score zero (i.e. flipping it will not make any unsatisfied clauses satisfied) then the variables will be negated. If this is not the case, then a variable from the clause is chosen probabilistically based on its score. This is referred to as *random walk* and tends to introduce sufficient 'noise' to allow the algorithm to escape local minima[3].

As with **DLL**, sophisticated enhancements have been made to these algorithms to guide the search and provide ways to escape local-minima, many of which can be found in the PhD thesis by Hoos (1999) who provides an excellent investigation into stochastic local search algorithms. Though slightly outdated Gu et al. (1997) provides a comprehensive and thorough survey of the algorithmic 'space' of the *SAT* problem.

2.6 Chapter Summary and Discussion

In this chapter I formally defined the CSP and *SAT* problems, as well as a number of basic Graph Theory concepts. I demonstrated how CSPs are represented graphically and introduced the main algorithmic techniques from each field. Although representing CSPs as graphs is certainly not novel, the aim of this chapter was to present the framework upon which I draw heavily in the remainder of this thesis.

More specifically, in the next chapter I provide an extensive survey of *SAT* and CSP encodings. Analysing the CSP *micro-structure* construction using this framework inspires a new type of CSP to *SAT* encoding in Chapter 4, as well as providing a general framework to categorise these encodings. In Chapter 5 I provide a new set of complexity analyses of these encodings with respect to *Resolution* and *Consistency* algorithmic techniques. I use this graph-theoretic approach to demonstrate an equivalence between *Resolution* and *Consistency*, which inspires the new concept of Extended-*Consistency*.

[3] areas of the search-space that algorithms may 'waste effort' exploring.

Chapter 3

Literature Review

As mentioned in Chapter 1, the fields of Propositional Satisfiability and Constraint Satisfaction have developed as two relatively independent threads of research, cross-fertilising occasionally. Only in the past few years have we developed a more intimate understanding of the similarities and differences between these two problem domains, providing us with the ability to capitalise on synergies between these areas.

One method used to explore the relationship between these fields is to study the mappings between the two core problem domains. Rossi et al. (1990) published an excellent paper on the equivalence of CSPs, though twenty years ago we were limited to only a few encodings. More specifically, Rossi et al. (1990) defined CSPs as being equivalent if they are "mutually reducable", and used this definition to prove formally that binary and non-binary CSPs are equivalent in this sense. They pointed out that such a result is useful because many efficient solution algorithms have been developed for binary CSPs, that it is important to be able to transform a non-binary into a binary CSP, solve it, and finally be able to go from that to the solution of the original problem. In recent years a wide range of encodings have appeared in the literature providing a more complete picture of this relationship.

The advantage to the *SAT* community of such mappings is that all algorithms and heuristics developed within this framework can be exploited, for instance, the structure of the *SAT* problem can be further analysed via its binary CSP expression with constraint techniques. The CSP community can in turn profit from the practical aspects of how to implement effective algorithmic techniques that can dramatically speed-up the search procedure. From a pragmatic perspective, problems that are hard for constraint-based algorithms can utilise the brute-force power of *SAT*-Solvers, whereas problems difficult for *SAT*-Solvers might be trivially solvable using constraints techniques.

In the first part of this chapter I review the encoding literature. I provide a thorough survey of CSP and *SAT* encodings and critically assess the major theoretical and empirical studies. This analysis highlights a number of gaps in the literature that this thesis addresses. The focus of the latter part of this chapter shifts towards the proof-complexity of polynomial algorithms and problems that are considered

computationally hard.

3.1 CSP to *SAT* Encodings

Encoding a CSP to a *SAT* instance is the process of taking a CSP and translating it to CNF. There are three common encodings, called DIRECT, SUPPORT, and LOG. In this section I define these encodings and review the major theoretical and empirical studies performed on them [1]. In the next chapter I show how these encodings can be categorised, which inspires a new type of CSP to *SAT* translation that I demonstrate has some advantages over current encodings.

3.1.1 DIRECT Encoding

The DIRECT encoding, coined by Walsh (2000a), first appeared in the late 80's in the seminal paper by de Kleer (1989).

Definition 3.1.1 (DIRECT encoding)**.** *Given a CSP* $= (\mathcal{V}, \mathcal{D}, \mathcal{C})$*. For this encoding each* SAT *variable* x_i^m *is defined as* True *iff the CSP variable* x_i *(*$\in \mathcal{V}$*) is assigned the domain value* m *(*$\in \mathcal{D}_{x_i}$*). The* SAT *instance is generated as a triple set of clauses* $\{positive, negative, constraint\}$ [2] *as follows:*

- positive: $(x_0^0 \vee x_0^1 \vee \cdots \vee x_0^{m-1}) \wedge (x_1^0 \vee x_1^1 \vee \cdots \vee x_1^{m-1}) \wedge \cdots \wedge (x_{n-1}^0 \vee x_{n-1}^1 \vee \cdots \vee x_{n-1}^{m-1})$
- negative: *for all* $x_i \in \mathcal{V}$*,* $(\bar{x}_i^p \vee \bar{x}_i^q)$ *such that* $p, q \in \mathcal{D}_{x_i}$ *and* $p < q$ *(also known as the* pairwise encoding*)*
- constraint: *every disallowed labelling in a constraint is encoded as a negated conjunction. For example,* $\eta\{x_a^p, x_b^q, x_c^r, \dots\}$ *is the negated conjunction* $\neg(x_a^p \wedge x_b^q \wedge x_c^r \wedge \dots)$*, which is* $(\bar{x}_a^p \vee \bar{x}_b^q \vee \bar{x}_c^r \vee \dots)$ *in disjunctive form.*

Recall Example 2.2.3:

- $\mathcal{V} = \{x_0, x_1, x_2, x_3, x_4\}$
- $\mathcal{D} = \{\mathcal{D}_{x_0}, \mathcal{D}_{x_1}, \mathcal{D}_{x_2}, \mathcal{D}_{x_3}, \mathcal{D}_{x_4}\}$, such that $\mathcal{D}_{x_0} = \mathcal{D}_{x_1} = \mathcal{D}_{x_2} = \mathcal{D}_{x_3} = \mathcal{D}_{x_4} = \{R, G, B\}$
- $\mathcal{C} = \{\mathcal{C}_{x_0,x_1}, \mathcal{C}_{x_0,x_4}, \mathcal{C}_{x_1,x_2}, \mathcal{C}_{x_1,x_3}, \mathcal{C}_{x_2,x_3}, \mathcal{C}_{x_2,x_4}\}$, where
  - $\mathcal{C}_{x_0,x_1} = \{\eta\{x_0^R, x_1^R\}, \eta\{x_0^G, x_1^G\}, \eta\{x_0^B, x_1^B\}\}$
  - $\mathcal{C}_{x_0,x_4} = \{\eta\{x_0^R, x_4^R\}, \eta\{x_0^G, x_4^G\}, \eta\{x_0^B, x_4^B\}\}$
  - $\mathcal{C}_{x_1,x_2} = \{\eta\{x_1^R, x_2^R\}, \eta\{x_1^G, x_2^G\}, \eta\{x_1^B, x_2^B\}\}$
  - $\mathcal{C}_{x_1,x_3} = \{\eta\{x_1^R, x_3^R\}, \eta\{x_1^G, x_3^G\}, \eta\{x_1^B, x_3^B\}\}$

[1] There are other 'less established' encodings that have not been mentioned. For instance, Roussel (2004) defines the *PHNF* encoding where the binary constraints between complementary *SAT* literals are mapped to CSP variables. Another *SAT* to CSP encoding (the *CGNF* encoding), described by Paris et al. (2006), aims to maintain the size and structure of the problem.

[2] *positive*, *negative* and *constraint* clauses are also referred to as *at-least-one*, *at-most-one* and *conflict* clauses respectively.

– $\mathcal{C}_{x_2,x_3} = \{\eta\{x_2^R, x_3^R\}, \eta\{x_2^G, x_3^G\}, \eta\{x_2^B, x_3^B\}\}$

– $\mathcal{C}_{x_2,x_4} = \{\eta\{x_2^R, x_4^R\}, \eta\{x_2^G, x_4^G\}, \eta\{x_2^B, x_4^B\}\}$

From this CSP the following *SAT* instance is constructed using the DIRECT encoding as the conjunction of this set of clauses:

- *positive*: $(x_0^R \vee x_0^G \vee x_0^B) \wedge (x_1^R \vee x_1^G \vee x_1^B) \wedge (x_2^R \vee x_2^G \vee x_2^B) \wedge (x_3^R \vee x_3^G \vee x_3^B) \wedge (x_4^R \vee x_4^G \vee x_4^B)$
- *negative*: $(\bar{x}_0^R \vee \bar{x}_0^G) \wedge (\bar{x}_0^R \vee \bar{x}_0^B) \wedge (\bar{x}_0^G \vee \bar{x}_0^B) \wedge (\bar{x}_1^R \vee \bar{x}_1^G) \wedge (\bar{x}_1^R \vee \bar{x}_1^B) \wedge (\bar{x}_1^G \vee \bar{x}_1^B) \wedge (\bar{x}_2^R \vee \bar{x}_2^G) \wedge (\bar{x}_2^R \vee \bar{x}_2^B) \wedge (\bar{x}_2^G \vee \bar{x}_2^B) \wedge (\bar{x}_3^R \vee \bar{x}_3^G) \wedge (\bar{x}_3^R \vee \bar{x}_3^B) \wedge (\bar{x}_3^G \vee \bar{x}_3^B) \wedge (\bar{x}_4^R \vee \bar{x}_4^G) \wedge (\bar{x}_4^R \vee \bar{x}_4^B) \wedge (\bar{x}_4^G \vee \bar{x}_4^B)$
- *constraint*: $(\bar{x}_0^R \vee \bar{x}_1^R) \wedge (\bar{x}_0^G \vee \bar{x}_1^G) \wedge (\bar{x}_0^B \vee \bar{x}_1^B) \wedge (\bar{x}_0^R \vee \bar{x}_4^R) \wedge (\bar{x}_0^G \vee \bar{x}_4^G) \wedge (\bar{x}_0^B \vee \bar{x}_4^B) \wedge (\bar{x}_1^R \vee \bar{x}_2^R) \wedge (\bar{x}_1^G \vee \bar{x}_2^G) \wedge (\bar{x}_1^B \vee \bar{x}_2^B) \wedge (\bar{x}_1^R \vee \bar{x}_3^R) \wedge (\bar{x}_1^G \vee \bar{x}_3^G) \wedge (\bar{x}_1^B \vee \bar{x}_3^B) \wedge (\bar{x}_2^R \vee \bar{x}_3^R) \wedge (\bar{x}_2^G \vee \bar{x}_3^G) \wedge (\bar{x}_2^B \vee \bar{x}_3^B) \wedge (\bar{x}_2^R \vee \bar{x}_4^R) \wedge (\bar{x}_2^G \vee \bar{x}_4^G) \wedge (\bar{x}_2^B \vee \bar{x}_4^B)$

Figure 3.1 shows the *micro-structure complement* of this instance; where *positive* clauses map to the component sets, *negative* clauses are represented by the edges between the nodes within the component sets, and *constraint* clauses represent edges between the nodes in different component sets.

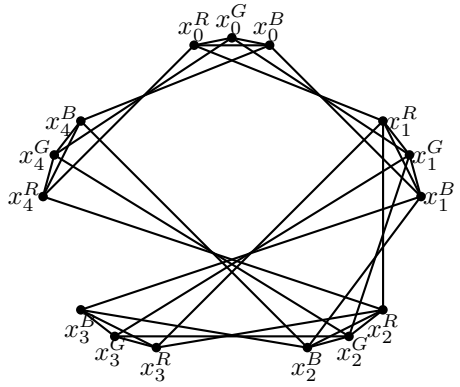

Figure 3.1: The DIRECT encoding of Example 2.2.3, represented as a $\bar{G}_3^5$ graph.

3.1.1.1 DIRECT Encoding Complexity

For simplicity, let us assume a CSP has n variables (each with a fixed domain size m) and q explicit *nogoods* (each with a fixed arity r). The DIRECT encoding will generate a *SAT* instance with nm variables and $n + n\binom{m}{2} + q$ clauses:

- n *positive* clauses of size m
- n sets of $\binom{m}{2}$ *negative* binary clauses
- q *constraint* clauses of size r

It is not always feasible to encode CSP problems as *SAT*. Take for instance the implicit constraint over variables a, b as $a = b + 1$. If each variable can take a value from the domain $\{1, 2, \ldots, 10000\}$, this will generate over 100 million clauses using the DIRECT encoding; most of them *negative* clauses.

SAT instances necessarily encode *nogoods* explicitly, however, it is sometimes the case that implicit constraints encode an exponential number of explicit *nogoods*, which makes encoding a CSP as *SAT* impossible. Take for instance the implicit constraint:

$$x^n \bmod y = z$$

which encodes $O(x^n)$ explicit *nogoods*.

Propositional Formulae do not have quantifiers (such as $\forall$ and $\exists$), however many problems are most naturally expressed when using first-order logic that include quantification over elements within the domain, such as

$$\forall i, j, k\ (\bar{i} \vee j) \wedge (j \vee \bar{k})$$

This can be expressed implicitly in a CSP, however converting this to *SAT* means expanding such quantifiers with the result increase size exponential in the number of quantifiers. Parkes (1999) was one of the early pieces of research to demonstrate that it can be practical to extend *SAT*-Solvers and apply them to Quantifies Boolean Formulae (*QBF*), which has become an increasing popular research area in recent years.

In the next chapter I describe a new encoding that can have a much more compact *SAT* representation in some circumstances. Whilst this new encoding may not completely alleviate this issue, it is an alternative encoding that makes it possible to transform some CSP instances to *SAT*.

3.1.2 SUPPORT Encoding

The SUPPORT encoding was first defined by Kasif (1990) then later by Gent (2002), and only differs from the DIRECT encoding by the definition of the *constraint* clauses. With this encoding each *constraint* clause in the DIRECT encoding is replaced by two *support* clauses.

The SUPPORT encoding can be thought of as 'preprocessed' version of the DIRECT encoding in the sense that each pair of *support* clauses are the *resolvents* of a binary *constraint* clause with two *positive* clauses.

Recall the clauses generated by the DIRECT encoding of Example 2.2.3 in Section 3.1.1. If we take the *constraint* clause $(\bar{x}_0^R \vee \bar{x}_1^R)$ and the two *positive* clauses $(x_0^R \vee x_0^G \vee x_0^B)$ and $(x_1^R \vee x_1^G \vee x_1^B)$ as an example, then two *support* clauses are a result of resolving:

$$\frac{x_0^R \vee x_0^G \vee x_0^B \qquad \bar{x}_0^R \vee \bar{x}_1^R}{\bar{x}_1^R \vee x_0^G \vee x_0^B}\ (1) \qquad \frac{x_1^R \vee x_1^G \vee x_1^B \qquad \bar{x}_0^R \vee \bar{x}_1^R}{\bar{x}_0^R \vee x_1^G \vee x_1^B}\ (2)$$

The *support* clauses *implicitly* encode the *constraints* in the DIRECT encoding. Figure 3.2 shows how a *constraint* clause is encoded using the two *support* clauses resolved in the example above.

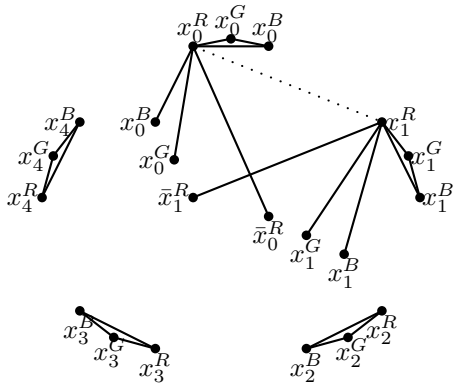

Figure 3.2: An example of how the *support* clauses *imply* the *constraint* clauses.

We can see in Figure 3.2 that by introducing these two constraints the edge $\{x_0^R, x_1^R\}$ (dotted line) is prevented from forming a solution with any node from either one of the *support* sets. A satisfying assignment must contain exactly one node from each set, however, since $\{x_0^R, x_1^R\}$ cannot form a local clique with either of the *support* clause sets then the edge cannot form part of a global solution. I will examine this inference method in more detail in Chapter 5. There are two *support* clauses for every *constraint* clause resulting from the DIRECT encoding.

3.1.2.1 SUPPORT Encoding Complexity

Again for simplicity, let us assume a CSP has n variables (each with a fixed domain size m) and q explicit *nogoods* (each with a fixed arity r). The SUPPORT encoding will generate a *SAT* instance with nm variables and $n + n\binom{m}{2} + 2q$ clauses:

- n *positive* clauses of size m
- n sets of $\binom{m}{2}$ *negative* clauses of size 2
- $2q$ *support* clauses of size m

Bessiere et al. (2003) generalises the SUPPORT encoding, calling it the *k-AC encoding*. The *k-AC encoding* differs from the SUPPORT encoding in two ways. It captures a larger family of consistencies, and it works for any constraint arity.

3.1.3 LOG Encoding

The LOG encoding, again coined by Walsh (2000b), appears in the literature under various names and is applied to a variety of problems. Again, if we assume a CSP has n variables (each with a fixed domain size m), the purpose of encoding a problem using this method is to reduce the number of propositional variables from nm to $n\lceil \log_2(m) \rceil$.

Definition 3.1.2 (LOG encoding[3])**.** *Given our simplified CSP =* $(\mathcal{V}, \mathcal{D}, \mathcal{C})$*,* v *unique variables are generated for each domain such that* $v = \lceil \log_2(m) \rceil$*. These* v *variables are binary encodings and are used to identify the* m^{th} *domain value. For example, if variable* x_i *has the domain* $\mathcal{D}_{x_i} = \{R, G, B, Y\}$*, then two* SAT *variables* $\{v_{x_i}[0], v_{x_i}[1]\}$ *would be generated, where* $v^j_{x_i}$ *indexes the* j^{th} *bit of the binary encoding for the domain of* x_i*. The set* v *encodes (in binary) each domain variable, i.e.* $(\bar{v}_{x_i}[0] \wedge \bar{v}_{x_i}[1]) \equiv x^R_i, (\bar{v}_{x_i}[0] \wedge v_{x_i}[1]) \equiv x^G_i, (v_{x_i}[0] \wedge \bar{v}_{x_i}[1]) \equiv x^B_i$*, and* $(v_{x_i}[0] \wedge v_{x_i}[1]) \equiv x^Y_i$*.*

The LOG *encoding generates two sets of clauses:*

- negative: *for domains that have fewer than* $\lceil \log_2(m) \rceil$ *elements we must disallow any redundant binary choices. For example, let us assume that* x_i *has the domain* $\mathcal{D}_{x_i} = \{R, G, B\}$*. We must arbitrarily disallow one of the redundant binary assignments, say* $\langle \langle v_{x_i}[0], v_{x_i}[1] \rangle, \setminus \{\langle 1, 1 \rangle\} \rangle$*, which in clausal form is* $(\bar{v}_{x_i}[0] \vee \bar{v}_{x_i}[1])$*.*

- constraint: *every constraint clause is simply the negated conjunction of the restricted sets of binary variables. For example, the constraint* $\langle \langle x_0, x_1 \rangle, \setminus \{\langle G, G \rangle\} \rangle$ *is the negated conjunction* $\neg(x^G_0 \wedge x^G_1)$*, which is* $(\bar{x}^G_0 \vee \bar{x}^G_1)$ *in disjunctive form. On the assumption that the variables* $\{v_{x_0}[0], v_{x_0}[1]\}$ *encode the domain for* x_0*, and* $\{v_{x_1}[0], v_{x_1}[1]\}$ *encode the domain for* x_1*, and* G *is assigned the binary values* $(0, 1)$*, then the resulting clause would be* $(v_{x_0}[0] \vee \bar{v}_{x_0}[1] \vee v_{x_1}[0] \vee \bar{v}_{x_1}[1])$*.*

Using the LOG encoding, the *SAT* instance of the GRAPH 3-COLOURABILITY Example 2.2.3 (on the assumption that the binary values map to $R = (0, 0), G = (0, 1)$ and $B = (1, 0)$) can be found in Appendix B.1.

Notice that in this instance the encoding generates non-binary constraints (4-tuples), which are difficult to represent graphically without introducing hyperedges. In Chapter 4 I categorise encodings by their structure, so understanding how instances are translated is important. So although not a strict *micro-structure* representation, Figure 3.3 is a graphical representation of the above *SAT* instance. The 4-ary constraints are represented as binary edges between component set nodes. Edges within the component set nodes represent the *negative* clauses. For example, the edge $\{\{v_{x_0}[0] = 0, v_{x_0}[1] = 0\}, \{v_{x_1}[0] = 0, v_{x_1}[1] = 0\}\}$ represents the clause $(v_{x_0}[0] \vee v_{x_0}[1] \vee v_{x_1}[0] \vee v_{x_1}[1])$.

Many of the encodings described in this chapter have appeared relatively recently in the literature, and variants are often published. For instance Frisch & Peugniez (2001) describe a variant to the LOG encoding (called the *binary transformation*) where the *negative* clauses are removed and redundant binary assignments are mapped to used binary assignments. As I will discuss later, there still remains a great deal of analysis to be done in order to determine the relative merits of each encoding.

[3] I introduce some additional notation for this encoding and hope it does not cause confusion. Here, the index $v[i]$ denotes the i^{th} bit. The Boolean assignments are associated to a variable with or without a *bar*, i.e. $v[i]$ means that this is assigned the value *True*, otherwise a $\bar{v}[i]$ means that it is assigned the value *False*.

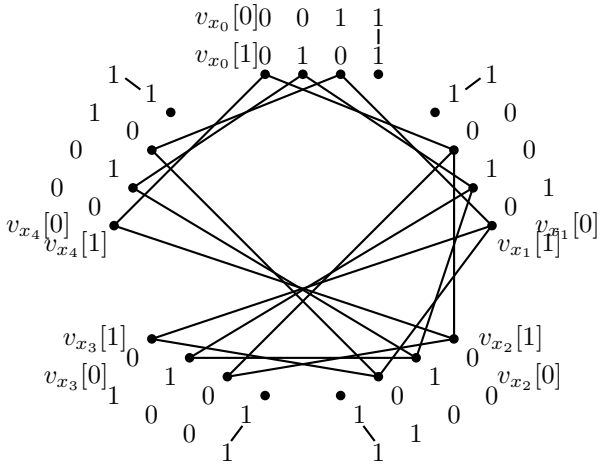

Figure 3.3: A graphical representation of the LOG encoding of Example 2.2.3. Binary edges between component set nodes correspond to the 4-ary constraints, whereas edges within the component set nodes correspond to the *negative* clauses.

3.1.3.1 LOG Encoding Complexity

On the assumption that a CSP has n variables (each with a fixed domain size m) and q explicit *nogoods* (each with a fixed arity r), the LOG encoding will generate a *SAT* instance with $n\lceil \log_2(m) \rceil$ variables, q *constraint* clauses of size $r\lceil \log_2(m) \rceil$ and $q(2^{\lceil \log_2(m) \rceil} - n)$ *negative* clauses of size $\lceil \log_2(m) \rceil$.

3.2 Analysis of CSP to *SAT* Encodings

In this section I introduce the major empirical and theoretical results published about CSP to *SAT* encodings.

3.2.1 Empirical Analysis

Representing problems as a CSP has an important advantage over *SAT*, which comes from its flexibility. A range of constraints not easily represented in CNF are often trivially definable in the CSP framework, such as the *cumulative*[4] and *alldiff*[5] constraints (Bordeaux et al. (2006)). Indeed, the *alldiff* constraint can make an exponential difference. Typically, search algorithms that do not use global information on a group of disequalities will entirely explore the search tree to prove inconsistency. Hence, simple *branch and bound* techniques are inadequate on problems such as the PIGEON-HOLE PROBLEM (defined in Section 3.6.3) and take exponential-time. Régin (1994) shows that it is sometimes possible to express these types of global constraints using a conjunction of logical constraints, but it is generally more efficient to make deductions using specialised CSP algorithms[6].

[4]For example, stating that the length, S, of a TRAVELLING SALESPERSON PROBLEM (TSP) journey must be less than some value z:

$$S = \sum_{i=0}^{j} x_i : S < z.$$

[5]For instance, the constraint $\forall i \forall j > i.\ x_i \neq x_j$.

[6]See Beldiceanu et al. (2005) for an exhaustive catalogue of global constraints.

Instances formulated as CNF very often do not directly express the problem, instead they are usually translated from the CSP definition, losing much of the problem structure during the translation stage. However, the main strength of formulating problems in CNF is that all effort can be focused on a single representation, resulting in highly optimised data-structures and efficient algorithms.

Although there are several ways to encode CSPs as *SAT*, there are few guidelines on how to choose amongst them. As Prestwich (2003) highlights, this aspect of problem modelling is currently more an art than a science, yet the choice of encoding can be as important as the choice of search algorithm. Prestwich provides an extensive empirical investigation into the performance of stochastic algorithms on GRAPH COLOURABILITY problems encoded as *SAT* with the aim of providing some guidelines about how to choose an encoding that might improve the likelihood of solubility. Prestwich confirmed that — in the case of MULTIVALUED vs. DIRECT — encodings with more solutions are typically easier to solve by local search.

Gent (2002) showed that a **DLL**-based algorithm performed better on the SUPPORT encoded randomly generated hard problems than the DIRECT encoding. Similarly, Gent's research showed that WALKSAT performs an order of magnitude faster on these randomly generated instances using the SUPPORT encoding than it did on the DIRECT encoding. However, Prestwich observed the opposite result, reporting that DIRECT encoded GRAPH COLOURABILITY problems were often solved much faster than when SUPPORT encoded. Van Gelder (2008) provided an extensive empirical survey of the performance of several *SAT*-Solvers on DIRECT and LOG encoded GRAPH COLOURABILITY problems. Van Gelder's work showed that in all cases but one, the DIRECT encoding was superior. Prestwich also found similar results.

Ansotegui & Manya (2004) evaluated several *SAT* encodings generated for a number of combinatorial problems (graph coloring, random binary CSPs, pigeon hole, and all interval series) using two leading *SAT*-Solvers. Their results provide empirical evidence that encoding combinatorial problems with different mappings can provide substantial performance improvements for complete *SAT*-Solvers.

However, as with much of the empirical research, all of these studies restrict themselves to either one type of problem or one type of algorithm, or both. It will not be until much more empirical research is performed on various encodings of a plethora of problems running a multitude of algorithms that we will develop a better idea about which encoding is likely to be 'best', and even then, the answer might still be unclear. Several *SAT* communities have appeared with the aim to stimulate research in this area.

3.2.1.1 SATLIB and *SAT* Competition

SATLIB (Hoos & Stützle (2000)) is an online resource for *SAT*-related research that was established in June 1998. SATLIB's core component is a benchmark suite of *SAT* instances. The aim of SATLIB is to facilitate empirical research on *SAT* by providing a uniform test-bed for *SAT*-Solvers, along with freely

available implementations of high-performing *SAT* algorithms. SATLIB offers four different types of problems:

1. Randomly generated native *SAT* instances.
2. *SAT*-encoded, randomly generated problem instances from other domains.
3. Instances from direct applications of *SAT*.
4. *SAT*-encoded instances from other application domains.

For most of these problem types there are instances of different sizes. For randomly generated instances, such as Random-*3*-*SAT*, SATLIB provides standardised test-sets sampled from the underlying distributions. For the most part, SATLIB collects problem instances that are intrinsically hard or difficult to solve for a broad range of algorithms and avoids instances which are known to be trivially solvable. While 'easy' instances can sometimes be useful for illustrating or investigating properties of specific algorithms (for example polynomially solvable instances which are hard for certain, otherwise high-performing algorithms), they are not used as general benchmark problems since this can easily lead to heavily biased evaluations and assessments of the usefulness of specific algorithms. Hence, SATLIB's benchmark collection is comprised mostly of instances that are known to be hard for a wide range of *SAT* algorithms.

Generally, benchmark sets should contain a large variety of different types of problem instances so that they can be used as a basis for evaluating different types of algorithms in an unbiased way. The most obvious - but also the most important - function of a benchmark library is to facilitate the use of the same set of problem instances across different studies and thus to enhance the comparability of the respective results. Furthermore, different types of studies will focus on problem instances with different properties, and a benchmark set becomes more useful if it can support a broader range of studies.

Almost every year since 2002 there has been a *SAT* Competition. Organised by Daniel Le Berre and Laurent Simon, the purpose of the competition is to identify new challenging benchmarks and to promote new solvers for Propositional Satisfiability as well as to compare them with state-of-the-art solvers. With thousands of instances this has now become the primary source of Industrial, Random and Crafted benchmarks. The *SAT* Competition provides access to the results of the world's leading *SAT*-Solvers on instances that range from having tens of clauses to several million.

3.2.2 Theoretical Analysis

To address many of the issues that the empirical studies raise, a great deal of complementary theoretical research has been carried out about the comparative algorithmic performances between the problem domains (though more-so for *SAT* to CSP encodings than for CSP to *SAT*). Table 3.1 summarises the theoretical analysis performed by Walsh (2000b), Gent (2002) and Bennaceur (2004) on CSP to *SAT*

encodings, comparing **DLL** to **MAC** and **FC**. For notation, let us consider approach X vs. Y. $X = Y$ denotes that X and Y have equivalent behaviour, and $X < Y$ denotes that Y is superior to X. We can see for instance that enforcing *arc*-consistency on the original problem does more work than *unit propagation* on the DIRECT encoding. That is, if *unit propagation* identifies unsatisfiability then enforcing *arc*-consistency on the DIRECT encoding also does, but there are problems which enforcing *arc*-consistency will show are insoluble that *unit propagation* will not. With equivalent branching heuristics, **DLL** applied to the DIRECT encoding explores the same size search tree as **FC** applied to the original problem.

Comparison	**DIRECT**	**SUPPORT**	**LOG**
Unit-Propagation vs. *arc*-consistency	$<$		$<$
DLL vs. **MAC**	$<$	$=$	
DLL vs. **FC**	$=$		$<$

Table 3.1: A comparison of algorithmic techniques on CSP to *SAT* encodings.

The notion of 'work' can be confusing. When discussing the 'work' done by branching or stochastic algorithms this typically is a reference to the amount of the search-space that is explored by the algorithm. When we say that **FC** does more work than **DLL** on a particular encoding we mean that **FC** explores more of the search-space that **DLL**. When discussing 'work' in the context of comparing local-consistency (or pruning) algorithms (such as *Resolution* and *Consistency*), we mean to say that one algorithm prunes the search-space more (or less) than the other. In Chapter 5 I compare the 'work' achieved by the same local-consistency algorithm applied to different encodings of a problem. For instance, applying a certain level of local-consistency on an encoded version of a problem may achieve a higher (or lower) level of local-consistency than the same algorithm applied to the original problem.

Notice that the work of constraint-based techniques are, for the most part, superior to *SAT*-based techniques on the CSP to *SAT* encodings. However, one of the main problems with this theoretical analysis is that modern-day *SAT*-Solvers are almost beyond comparison with the original **DLL** procedure, meaning that these studies will tell us very little about the actual comparative performance of today's CSP and *SAT* algorithms[7].

3.2.2.1 *NG-RES*

It seems that de Kleer (1989) was the first to show the link between the *Resolution* and *Consistency* proof-systems, stating that *nogood-Resolution* (***NG-RES***) is equivalent to establishing strong-k-consistency on DIRECT encoded instances. de Kleer (1989) described several inference rules (**H0**, **H3** and **H5** below) that described the ***NG-RES*** procedure on the CSP instance represented as *SAT* using the DIRECT encoding (C is the *SAT* instance clauses set):

[7] Unless the innovation trajectory of CSP and *SAT* algorithms are developing at the same rate, which, as far as this author is aware, they are not.

- **H0:** This rule removes subsumed clauses from C. If clause $a \in C$ is subsumed by some other clause $b \in C$, then a is removed from C.

- **H3:** The *unit resolution* rule:

$$\frac{\begin{array}{c}\bar{x}_i \\ x_i \vee x_0 \vee \cdots \vee x_{i-1}\end{array}}{x_0 \vee \cdots \vee x_{i-1}}$$

- **H5:** The main rule:

$$\frac{\begin{array}{c}(x_0 \vee x_1 \vee \cdots \vee x_{i-1}) \\ (\bar{x}_0 \vee X_0) \\ (\bar{x}_1 \vee X_1) \\ \vdots \\ (\bar{x}_{i-1} \vee X_{i-1})\end{array}}{(X_0 \vee X_1 \vee \cdots \vee X_{i-1})}$$

  where X_j are clauses and x_j are literals, for $j < i$.

The **H5** rule can generate a large number of clauses, de Kleer points out a restriction that can address this:

- **H5-*k*: H5** restricted to only infer clauses below size k.

Let $I = \{\mathbf{H0}, \mathbf{H3}, \mathbf{H5}\}$. de Kleer proposed the following algorithmic properties that hold for DIRECT *SAT* encodings of a CSP.

Proposition 3.2.1. *Given any subset of inference rules from I, any order of application will lead to the same resulting clause set as long as the clause set is closed under those rules.*

Proposition 3.2.2. *Any algorithm incorporating any subset of inference rules from I achieves* node-*consistency as long as the resulting clause set is closed under **H3**.*

Proposition 3.2.3. *Any algorithm incorporating any subset of inference rules from I achieves strong-k-consistency as long as the resulting clause set is closed under **H5**-**k**.*

If follows that any algorithm incorporating any subset of inference rules from I achieves *node* and *arc*-consistency as long as the resulting clause set is closed under **H3** and **H5-2**, and achieves node, arc and *path*-consistency as long as the resulting clause set is closed under **H3** and **H5-3**.

Interestingly, Mitchell (2002) proved that there is a super-polynomial separation between ***NG-RES*** and *constraint-Resolution* (***C-RES***). ***C-RES*** is simply standard *Resolution* applied to a CSP that has been transformed into CNF using the DIRECT encoding. Mitchell (2002) found that ***NG-RES*** takes super-polynomial time to solve a variant of the PIGEON-HOLE PROBLEM (see Section 3.6.3), whereas ***C-RES*** only takes quadratic-time. Hwang (2004) added to the super-polynomial separation result further by proving an exponential separation between the two proof methods.

3.2.2.2 Phase Transition

Many **NP**-complete problems display a rapid transition in solubility as the *constrainedness* (Gent et al. (1996)) of the problem increases (for randomly generated problem instances). This transition (referred to as the *phase transition*) is associated with problems that are hard for backtracking procedures to solve. It tends to be easy to solve problems that are either under-constrained (have many solutions) or over-constrained (have few or no solutions). The phase transition is the intermediate point where problems are *critically constrained*, i.e. out of a random sample of problems some will be soluble and some not, and it is usually hard to find a solution or to prove that one exists (Cheeseman et al. (1991)).

Experiments performed by Cook & Mitchell (1997) on random *3-SAT* instances show that the probability of an instance being satisfiable shifts with the ratio of clauses-to-variables from being almost 1 (with ratios much below 4) to being almost 0 (at ratios much above 5), and that the range of ratios over which this transition occurs becomes smaller as the number of variables increases.

A large amount of research has been carried out to examine the phase transition of CSP and *SAT* problems; most notably by Smith (1993); Smith et al. (1995); Gent & Walsh (1995); Gent et al. (1996); Smith & Dyer (1996); Prosser (1996); Mitchell (1998); MacIntyre et al. (1998); Achlioptas et al. (2005) for CSP, and Mitchell et al. (1992); Gent & Walsh (1994, 1996); Cook & Mitchell (1997); Istrate (2002) for *SAT*. It is only within the past few years that attention has been paid to the phase transition of encodings of *SAT* problems where only a specific number of literals must be satisfied (also known as *cardinality constraints*) (see Bailleux & Boufkhad (2003) and Sinz (2005)[8]), and even more recently Marques-Silva & Lynce (2007) published results indicating that some algorithms perform significantly better on *SAT* encodings involving these types of constraints.

3.2.2.3 The MULTIVALUED Encoding and *Solution Density*

As Prestwich (2003) points out in his excellent survey of CSP to *SAT* encodings, it is often the case that the *negative* clauses are omitted from the DIRECT encoding. Whilst omitting this set does not effect the *satisfiability* of an instance, there is an important difference with this variant. Prestwich calls the result the MULTIVALUED encoding, and mentions that MULTIVALUED encodings have a higher *solution-density* than their counterparts containing *negative* clauses.

Definition 3.2.1 (Solution Density)**.** *The* solution-density *of a* SAT *instance is defined as the number of solutions divided by* 2^n*, where* n *is the number of* SAT *variables.*

This definition can be generalised to any CSP, such that the number of full satisfying assignments is divided by the total number of possible full assignments. For the *micro-structure complement* the *solution-density* equals the number of independent sets divided by the total number of possible independent sets of size n.

[8] Sinz also showed that instead of introducing the quadratic set of *negative* clauses, auxiliary variables can be introduced that specify the same thing.

Remark 3.2.1. *The number of clauses generated by the* DIRECT *encoding is strictly greater than the number of clauses generated by the* MULTIVALUED *encoding.*

Table 3.2 shows a summary of the size-complexity of these encodings (including the MULTIVALUED) based on a CSP with n variables of domain m, and q r-ary constraints. The **propositions** column indicates the number of propositions generated, whereas **positive**, **negative** and **constraint** describes the number of clauses of each type. The notation $a : b$ denotes that a clauses are generated of arity b.

encoding	propositions	positive	negative	constraint
DIRECT	nm	$n : m$	$n\binom{m}{2} : 2$	$q : r$
SUPPORT	nm	$n : m$	$n\binom{m}{2} : 2$	$2q : m$
LOG	$n\lceil \log_2(m) \rceil$	0	$q(2^{\lceil \log_2(m) \rceil} - n) : \lceil \log_2(m) \rceil$	$q : r\lceil \log_2(m) \rceil$
MULTIVALUED	nm	$n : m$	0	$q : r$

Table 3.2: The CSP to *SAT* encoding size-complexity.

Notice that the key difference between the DIRECT and MULTIVALUED encodings is that the *SAT* instance resulting from the DIRECT contains negative clauses specifying that CSP variables cannot take two contradictory domain values, whereas this is not explicitly constrained by the MULTIVALUED encoding. MULTIVALUED encoded instance can have more solutions than DIRECT encoded instances. This relates to research concerning the X*SAT* problem. The X*SAT* version (also known as *exactly-1-SAT*) includes all *negative* clauses specifying that only one literal in each clause must be *True*. The 3-X*SAT* problem (also known as *1-in-3-SAT*) was originally shown to be **NP**-complete in the epic paper by Schaefer (1978) and this problem has had relatively little attention since[9]. Recently, algorithmic upper-bounds for the *1-in-3-SAT* problem have been defined by Porschen et al. (2002), Madsen & Rossmanith (2004) and Kulikov (2005). If we compare these recent algorithmic upper bounds for 3-X*SAT* with *3-SAT*, $O(1.112^n)$ versus $O(1.38^n)$ (Kulikov (2005) and Mitchell (2002) respectively), we can see that 3-X*SAT* appears to be easier to solve, though much more research remains to determine the effect of including *negative* clauses.

A number of studies have been made to analyse the relationship between the *solution-density* and solubility of a problem. Clark et al. (1996) found that the hardest problems have few solutions and usually occur at the soluble phase. They showed that this finding was robust across problem class and types of stochastic search procedure, though they determined that the number of solutions was not the only factor influencing problem hardness.

Yokoo (1997) analysed the instances of *3-SAT* and 3-COLOURING problems. Yokoo showed that as more constraints are added the number of solutions decreases and that the number of local-minima also

[9] Its phase transition has been analysed by Achlioptas et al. (2001)

decreases, and thus that the number of solution-reachable states increases. So, adding more constraints (removing solutions) to take a problem beyond the phase transition removes local-minima and made it easier to solve using stochastic algorithms. In Prestwich's experiments using stochastic algorithms on the GRAPH COLOURABILITY problem, he showed that the MULTIVALUED encoding was uniformly better than the DIRECT encoding, concurring with the results of Selman et al. (1992). This suggests that stochastic search performs better on instances with higher *solution-density*.

Given that *solution-density* is an important factor in problem solubility, in Chapter 5 I prove that several *SAT* to CSP encodings can be differentiated by the proportion of solutions in the resulting translation. This is significant because the estimated *solution-density* is one factor that can influence our decision about which encoding might be better to use in a particular situation.

3.3 *SAT* to CSP Encodings

Encoding a *SAT* instance as a CSP is the process of taking a propositional Boolean formula and translating it into a CSP. Naturally, a *SAT* instance is a restricted type of CSP, with the variables constrained to the Boolean domain, and a list of explicit constraints (clauses). There are five common encodings, called LITERAL, DUAL, NON-BINARY, PLACE and HIDDEN VARIABLE. In this section I define these encodings and review the major theoretical and empirical studies performed on them. In Chapter 4 I show how these encodings can be categorised, and in Chapter 5 demonstrate a number of ways in which each encoding can be distinguished from each of the others.

As with the CSP to *SAT* encodings, I illustrate the *SAT* to CSP encodings using an example. Recall Formula 2.1,

$$(x_0 \vee x_1 \vee x_2) \wedge (\bar{x}_0 \vee x_1 \vee \bar{x}_2) \wedge (x_1 \vee \bar{x}_2 \vee x_3) \wedge (x_0 \vee \bar{x}_1 \vee x_3) \wedge (\bar{x}_0 \vee \bar{x}_2 \vee \bar{x}_3).$$

This 3CNF formula is a *3-SAT* instance with four variables $\{x_0, x_1, x_2, x_3\}$ and five clauses $\{C_0, C_1, C_2, C_3, C_4\}$, and it has eight satisfying assignments (shown in Table 3.3).

$\{x_0^0, x_1^0, x_2^1, x_3^1\}$	$\{x_0^0, x_1^1, x_2^0, x_3^1\}$	$\{x_0^0, x_1^1, x_2^1, x_3^1\}$	$\{x_0^1, x_1^0, x_2^0, x_3^0\}$
$\{x_0^1, x_1^0, x_2^0, x_3^1\}$	$\{x_0^1, x_1^1, x_2^0, x_3^0\}$	$\{x_0^1, x_1^1, x_2^0, x_3^1\}$	$\{x_0^1, x_1^1, x_2^1, x_3^0\}$

Table 3.3: The eight assignments that satisfy Formula 2.1

3.3.1 LITERAL Encoding

The LITERAL encoding is attributed to Bennaceur (1996) but could arguably be traced back to the early seventies when Karp (1972) reduced *3-SAT* to the CLIQUE PROBLEM.

Definition 3.3.1 (LITERAL encoding)**.** *Every clause C_i is associated with a variable $c_i \in \mathcal{V}$. The*

domain of each variable is the set of literals in the corresponding clause. For example, given a clause $C_j = (a \vee \bar{b})$*, the CSP variable* c_j *has the domain* $\{1, 0\}$*. Binary constraints are posted between variables that have complementary literals.*

Here is the CSP result obtained when Formula 2.1 is encoded using the LITERAL encoding.

- $\mathcal{V} = \{c_0, c_1, c_2, c_3, c_4\}$
- $\mathcal{D} = \{\mathcal{D}_{c_0}, \mathcal{D}_{c_1}, \mathcal{D}_{c_2}, \mathcal{D}_{c_3}, \mathcal{D}_{c_4}\}$, where $\mathcal{D}_{c_0} = \{x_0^1, x_1^1, x_2^1\}, \mathcal{D}_{c_1} = \{x_0^0, x_1^1, x_2^0\}, \mathcal{D}_{c_2} = \{x_1^1, x_2^0, x_3^1\}, \mathcal{D}_{c_3} = \{x_0^1, x_1^0, x_3^1\}, \mathcal{D}_{c_4} = \{x_0^0, x_2^0, x_3^0\}$.
- $\mathcal{C} = \{\mathcal{C}_{c_0,c_1}, \mathcal{C}_{c_0,c_2}, \mathcal{C}_{c_0,c_3}, \mathcal{C}_{c_0,c_4}, \mathcal{C}_{c_1,c_2}, \mathcal{C}_{c_1,c_3}, \mathcal{C}_{c_1,c_4}, \mathcal{C}_{c_2,c_3}, \mathcal{C}_{c,c_4}, \mathcal{C}_{c_3,c_4}\}$, where
  - $\mathcal{C}_{c_0,c_1} = \{\eta\{c_0^{x_0^1}, c_1^{x_0^0}\}, \eta\{c_0^{x_2^1}, c_1^{x_2^0}\}\}$
  - $\mathcal{C}_{c_0,c_2} = \{\eta\{c_0^{x_2^1}, c_2^{x_2^0}\}\}$
  - $\mathcal{C}_{c_0,c_3} = \{\eta\{c_0^{x_1^1}, c_3^{x_1^0}\}\}$
  - $\mathcal{C}_{c_0,c_4} = \{\eta\{c_0^{x_0^1}, c_4^{x_0^0}\}, \eta\{c_0^{x_2^1}, c_4^{x_2^0}\}\}$
  - $\mathcal{C}_{c_1,c_3} = \{\eta\{c_1^{x_0^0}, c_3^{x_0^1}\}, \eta\{c_1^{x_1^1}, c_3^{x_1^0}\}\}$
  - $\mathcal{C}_{c_2,c_3} = \{\eta\{c_2^{x_1^1}, c_3^{x_1^0}\}\}$
  - $\mathcal{C}_{c_2,c_4} = \{\eta\{c_2^{x_3^1}, c_4^{x_3^0}\}\}$
  - $\mathcal{C}_{c_3,c_4} = \{\eta\{c_3^{x_0^1}, c_4^{x_0^0}\}, \eta\{c_3^{x_3^1}, c_4^{x_3^0}\}\}$

Figure 3.4 represents the corresponding $\bar{G_3^5}$ graph. Notice the *solution-tuple* $\gamma\{c_0^{x_2^1}, c_1^{x_0^0}, c_2^{x_3^1}, c_3^{x_1^0}, c_4^{x_0^0}\}$ represented as a dotted 5-clique, which corresponds to the satisfying assignment $\{x_0^0, x_1^0, x_2^1, x_3^1\}$.

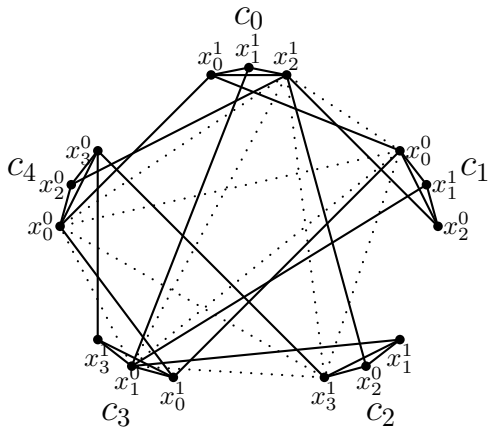

Figure 3.4: Formula 2.1 as a $\bar{G_3^5}$ graph using the LITERAL encoding.

3.3.1.1 LITERAL Encoding Complexity

For simplicity let us assume a CNF formula has n variables and m clauses of cardinality k. The LITERAL encoding of this *k-SAT* instance will generate a CSP with:

- m variables, each with a domain size k
- $O(k^2n^2)$ binary constraints.

3.3.2 DUAL Encoding

Definition 3.3.2 (DUAL encoding)**.** *As described by Walsh (2000b), with the* DUAL *encoding (Dechter (1992a)) each clause C_i is associated with a variable $c_i \in \mathcal{V}$. The domain of c_i is the set of satisfying assignments to the clause C_i. For instance if we had a clause $C_j = (a \vee \bar{b})$, then the domain of c_j would be $\{\{a^0, b^0\}, \{a^1, b^0\}, \{a^1, b^1\}\}$. The set of constraints $\mathcal{C}$ are binary and are posted between the CSP variables that have opposing proposition assignments.*

Figure 3.5 represents the (partial) $\bar{G}_7^5$ graph of Formula 2.1 when encoded as a CSP using the DUAL encoding. Although not shown in the figure, note that there is an edge between each pair of nodes in each of the sets C_0 and C_1,

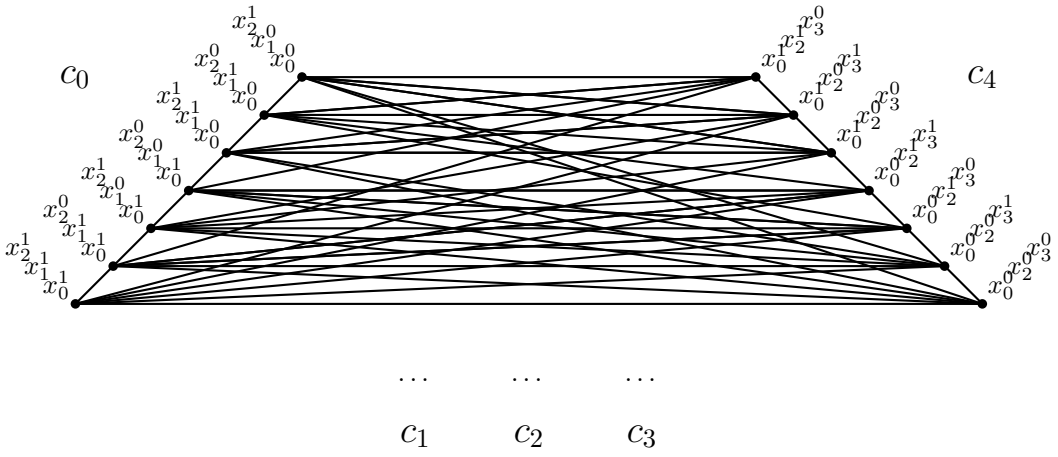

Figure 3.5: The (partial) $\bar{G}_7^5$ graph of Formula 2.1 mapped to CSP using the DUAL encoding.

3.3.2.1 DUAL Encoding Complexity

Assuming a CNF formula has n variables and m clauses of cardinality k, the DUAL encoding of this *k-SAT* instance will generate a CSP with:

- m variables
- each with a domain size $d = 2^k - 1$, and
- $O(d^2n^2)$ binary constraints.

Notice that the DUAL encoding typically produces larger CSP instances than the LITERAL encoding. Indeed, the DUAL encoding can be problematic since it requires an exponential number of CSP domain values per *SAT* clause. For example, if the *SAT* instance we are translating has a clause with 20 literals, this will generate a CSP variable with domain size 2^{20} (over 1 million values). To overcome this problem one can use De Morgan laws (see Table 6.2), and introduce new variables to reduce the size of the original clause by splitting it into lots of smaller clauses. I use this technique in my empirical study of the DUAL encoding in Chapter 6. For instance, given a clause of size 20, we can split this into 19 ternary clauses by introducing 17 new auxiliary variables.

Given that space is one resource that can limit the representation of a problem one might ask the question "why not use the most compact encoding?". The answer to this question is that identical algorithmic processes can perform more work on some encodings than others. That is, whilst one encoding might require more space than another, enforcing a particular level of consistency might require less time, or **DLL** might explore more branches. This begs the question of "which encoding is best?", about which little is currently know. In Chapters 5 and 6 I provide a theoretical and empirical study on the LITERAL and DUAL encodings that sheds light onto some of the answers to these questions. In particular, I show that although the DUAL encoding requires more space, enforcing local-consistency on DUAL encoded problems achieves more than when the problems are encoded using the LITERAL encoding. Moreover, I show that enforcing *path*-consistency on CSP encoded *SAT* instances using the DUAL encoding can dramatically increase the solubility of many 'hard' unsatisfiable *SAT* benchmarks, in stark contrast to none that are solved when represented LITERALLY.

3.3.3 NON-BINARY Encoding

The NON-BINARY encoding is the most compact and natural translation from *SAT* to CSP, it is simply a (non-binary) description of a *SAT* instance as a CSP.

Definition 3.3.3 (NON-BINARY encoding)**.** *Every CNF propositional variable is associated with a CSP variable x_j, each with the domain $\{0, 1\}$. The constraints are the partial assignments that fail to satisfy each clause. For instance, a clause $(x_0 \vee \bar{x}_1 \vee x_2)$ would generate the constraint $\langle\langle x_0, x_1, x_2\rangle, \backslash\{\langle 0, 1, 0\rangle\}\rangle$.*

The CSP result of Formula 2.1 when encoded using the NON-BINARY encoding is:

- $\mathcal{V} = \{x_0, x_1, x_2, x_3\}$
- $\mathcal{D} = \{\mathcal{D}_{x_0}, \mathcal{D}_{x_1}, \mathcal{D}_{x_2}, \mathcal{D}_{x_3}\}$, where $\mathcal{D}_{x_i} = \{0, 1\}, i < n$
- $\mathcal{C} = \{\mathcal{C}_{x_0,x_1,x_2}, \mathcal{C}_{x_0,x_2,x_3}, \mathcal{C}_{x_0,x_1,x_3}, \mathcal{C}_{x_1,x_2,x_3}\}$, where
  - $\mathcal{C}_{x_0,x_1,x_2} = \{\eta\{x_0^0, x_1^0, x_2^0\}, \eta\{x_0^1, x_1^0, x_2^1\}\}$

– $\mathcal{C}_{x_0,x_2,x_3} = \{\eta\{x_0^1, x_2^1, x_3^1\}\}$

– $\mathcal{C}_{x_0,x_1,x_3} = \{\eta\{x_0^0, x_1^1, x_3^0\}\}$

– $\mathcal{C}_{x_1,x_2,x_3} = \{\eta\{x_1^0, x_2^1, x_3^0\}\}$

Figure 3.6 represents the corresponding $\bar{G}_2^4$ 3-hypergraph. Though it is difficult to represent, note that each closed curve connects only three points, one for each constraint.

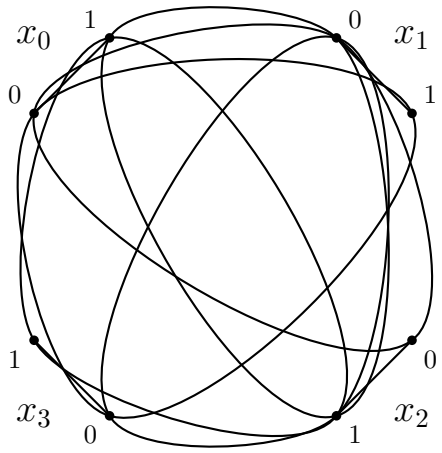

Figure 3.6: Formula 2.1 as a $\bar{G}_2^4$ 3-hypergraph using the NON-BINARY encoding. Variables are represented by the components sets, whereas the ternary clauses are represented as hyperedges that connect three nodes (i.e. the unsatisfying assignments to each clause).

3.3.3.1 NON-BINARY Encoding Complexity

Again on the assumption that a CNF formula has n variables and m clauses of cardinality k, the NON-BINARY encoding of this *k-SAT* instance will generate a CSP with:

- n variables
- each with a domain size 2, and
- m k-ary constraints.

3.3.4 PLACE Encoding

This encoding was defined independently by Gent et al. (2003) and Jarvisalo & Niemela (2004) who named it the EXTENDED LITERAL encoding and PLACE encoding respectively [10].

Definition 3.3.4 (PLACE encoding)**.** *The* PLACE *encoding has two sets of variables that make up $\mathcal{V}$:*

1. *every clause C_i in the CNF formula is associated with a variable c_i, and as with the* LITERAL *encoding its domain is the set of literals in its clause.*

[10] Although Jarvisalo & Niemela (2004) was published later than Gent et al. (2003) I adopt the name PLACE, so as not to confuse the the reader between the EXTENDED LITERAL encoding and Extended proof-systems.

2. *like the* NON-BINARY *encoding, every propositional variable in the CNF formula is associated with a variable* x_j *with the domain* $\{0, 1\}$.

Binary constraints are only posted between c_i *and* x_j *variables that have complementary literals.*

Figure 3.7 represents the *micro-structure complement* of Formula 2.1 transformed to CSP using the PLACE encoding. Note that this graph is a $\bar{G}_3^9$ since the maximum cardinality of any set is 3.

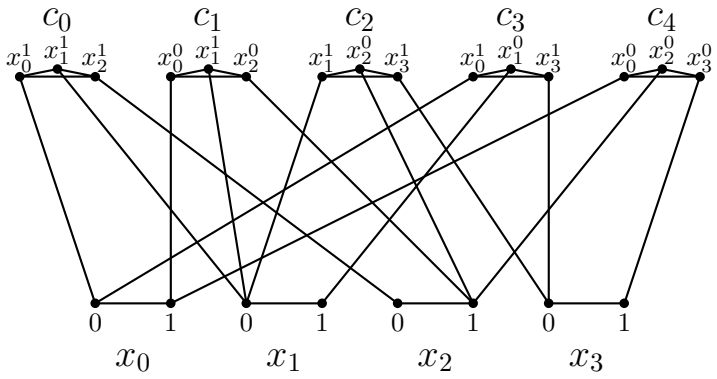

Figure 3.7: The $\bar{G}$ graph of Formula 2.1 transformed to a CSP using the PLACE encoding. The component sets $c = \{c_0, \ldots, c_4\}$ are the same as the LITERAL encoding, whereas the component sets $x = \{x_0, \ldots, x_3\}$ are the same as the nodes in the NON-BINARY encoding. Binary edges are posted between nodes in x and c that contain complementary literal assignments.

3.3.4.1 PLACE Encoding Complexity

For a *k-SAT* instance with n variables and m clauses the PLACE encoding will generate a CSP with:

- n variables with binary domains, plus
- m variables with a domain cardinality of k
- $O(kn)$ binary constraints.

In Chapter 4 (Figure 4.3) I demonstrate that PLACE encoding is simply a combination of the LITERAL and NON-BINARY encodings.

3.3.5 HIDDEN VARIABLE Encoding

Originally defined by Dechter (1990), the HIDDEN VARIABLE encoding is a binary CSP similar to the PLACE encoding described in Section 3.3.4.

Definition 3.3.5 (HIDDEN VARIABLE encoding)**.** *As with the* PLACE *encoding,* $\mathcal{V}$ *is constructed from two sets of variables:*

1. *every propositional variable in the CNF formula is associated with a variable* x_j *with the domain* $\{0, 1\}$.

2. *like the* DUAL *encoding, the set of satisfying assignments to each clause C_i in the CNF formula is the domain of each corresponding CSP c_i.*

Binary constraints are only posted between the complementary domain values of x_j and c_j.

To illustrate this, Figure 3.8 (partially) represents the *micro-structure complement* of encoding Formula 2.1 as a CSP using the HIDDEN VARIABLE encoding. Noticing that, although not drawn, there is an edge between each pair of nodes within the component sets C_0 and C_4.

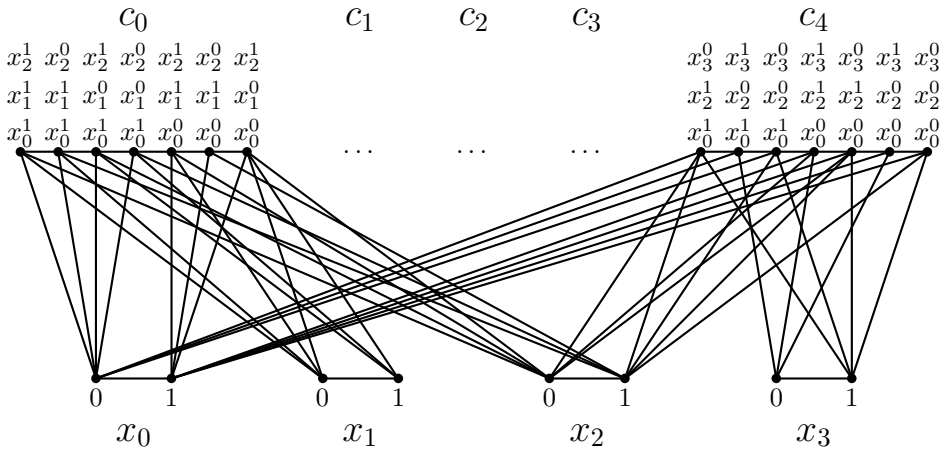

Figure 3.8: The $\bar{G}$ graph of Formula 2.1 encoded using the HIDDEN VARIABLE encoding. The components sets $c = \{c_0, \ldots, c_4\}$ are the same as the DUAL encoding, whereas the component sets $x = \{x_0, \ldots, x_3\}$ are the same as the nodes in the NON-BINARY encoding. Binary edges are posted between nodes in x and c that contain complementary literal assignments.

3.3.5.1 HIDDEN VARIABLE Encoding Complexity

For a *k-SAT* instance with n variables and m clauses the HIDDEN VARIABLE encoding will generate a CSP with:

- n variables with binary domains, plus
- m variables with a domain cardinality of $2^k - 1$
- $O(kn)$ binary constraints.

In a similar manner to that for the PLACE encoding, in Chapter 4 I illustrate that HIDDEN VARIABLE encoding is a combination of the DUAL and NON-BINARY encodings.

3.4 Analysis of *SAT* to CSP Encodings

As in Section 3.2 here I present the various empirical and theoretical results of studies performed on *SAT* to CSP encodings.

It is well-known that non-binary CSPs can be transformed into equivalent binary CSPs, and this work has generated a great deal of knowledge about the theory and practice of solving CSPs. The main

reason cited for translating non-binary into binary constraints is that more is known about how to solve binary CSPs; including better heuristics, known tractable cases and optimised algorithms (Bordeaux et al. (2006)). Another advantage of these encodings is that the structure of a *SAT* problem can be further analysed via its binary CSP expression using constraint techniques. However it is still largely unknown whether or not these techniques have any potential advantages, and surprising only very recently has work been done to examine the effectiveness of these encodings.

MAC and **FC** have been widely studied. In particular Grant & Smith (1995) published a rigorous empirical study of the performance of **MAC** and **FC** algorithms over a broad range of problem topologies and sizes, which highlighted many of the relative virtues of these algorithms with respect to the problem structure[11].

Table 3.4 summarises the CSP encoding complexity of a *k-SAT* instance with n propositional variables and m clauses. The **variables** column shows the number of CSP variables, and **domain** describes the size of the variable domain. The number of constraints are shown in **constraints**, and the arity denoted in **arity**. The $a : b$ notation in the **domain** column denotes a variables of domain size b.

encoding	variables	domain	constraints	arity
LITERAL	m	k	$O(k^2m^2)$	2
DUAL	m	$2^k - 1$	$O(2^k m^2)$	2
NON-BINARY	n	2	m	k
PLACE	$n + m$	$n : 2, m : k$	$O(kn)$	2
HIDDEN VARIABLE	$n + m$	$n : 2, m : 2^k - 1$	$O(kn)$	2

Table 3.4: The *SAT* to CSP encoding size-complexity.

Jarvisalo & Niemela (2004) pointed out that the PLACE encoding is the only encoding that is linear in all of these parameters; that this encoding is the first which is "propagation-optimal". That is, propagation using the standard search algorithm **MAC** in the CSP encoding performs the same search as **DLL** on the original *SAT* instance, and does so in the same worst-case time complexity. Gent et al. (2003) pointed out that this theoretical equality is unlikely to lead to acceptable performance in practice, since *SAT*-Solvers are highly-optimised algorithms[12]. Gent also showed that it is possible for CSP search to take exponentially longer than the *SAT* search when using the PLACE encoding. More specifically, if **MAC** sets some of the extra CSP variables before all *SAT* variables, it is possible for **DLL** to search exponentially fewer nodes. Representing CSPs as *SAT* may thus produce an exponential saving, though

[11] Having implemented several variants of **MAC** for the empirical studies in Chapter6, for my purposes I found that the overhead of queueing mechanisms could not compete with simply recursing through the data-structure.

[12] They suggested that *"we might compare the* SAT*-Solvers and CSP algorithms to a Formula-1 car and a family saloon. You would not want to pick up the kids from school and drop by the supermarket on the way home in the [Formula-1] car. However, the saloon car might benefit from Formula-1 technology. Constraint programming is general purpose, whereas the* SAT*-Solvers are specialised pieces of code."*

it is still unclear under what circumstances it is advisable to convert one problem type to another.

Table 3.5 summarises the theoretical analysis performed by Walsh (2000b), Gent et al. (2003) and Jarvisalo & Niemela (2004) on CSP to *SAT* encodings, comparing **DLL** to **MAC** and **FC**, as well as Min-Conflicts to the stochastic *SAT* search techniques WALKSAT and GSAT[13]. Again, we consider approaches X vs. Y and let $X = Y$ denote that X and Y have equivalent behaviour. $X > Y$ denote that X is superior to Y and let $X \neq Y$ mean that X and Y are incomparable with each other.

Comparison	**NON-BINARY**	**LITERAL**	**DUAL**	**HIDDEN VARIABLE**	**PLACE**
Unit-Propagation vs. *arc*-consistency	$<$	$=$	$<$	$=$	$=$
DLL vs. **MAC**		$>$	$\neq$	$=$	$=$
DLL vs. **FC**	$=$ (nFC0), $<$ (nFC1)	$>$	$>$	$=$	$=$
GSAT vs. **MC**	$\neq$			$\neq$	$\neq$
WALKSAT vs. **MC**	$=$			$<$	$<$

Table 3.5: A comparison of algorithmic techniques on *SAT* to CSP encodings.

Bacchus & van Beek (1998) showed that algorithms applied to problems represented using the DUAL encoding can be more efficient by orders of magnitude than HIDDEN VARIABLE encoded problems when the number of constraints is low relative to the number of variables and the constraints are restrictive. They suggested that although translating a non-binary CSP into *SAT* involves some overhead the number of satisfying assignments to a problem is perhaps the most important factor in determining the worth of a particular encoding.

3.4.1 DOUBLE Encoding

Stergiou & Walsh (1999) showed how the HIDDEN VARIABLE encoding can be transformed into the DUAL encoding, and introduced a new encoding that combined both the HIDDEN VARIABLE and DUAL encodings. They called this the DOUBLE encoding but it was not shown to have any advantages, possibly because the HIDDEN VARIABLE encoding already contains the DUAL *micro-structure* and hence it is superfluous to include it again. In fact, Stergiou & Walsh (1999) published a very nice survey on the performance of solving Golomb Ruler problems and Cross-Word Puzzle generation using a variety of encodings including the DOUBLE encoding. Interestingly, the results show that the time to generate Cross-Words encoded by means of the DOUBLE encoding appears to be approximately the sum of the time taken for the HIDDEN VARIABLE and DUAL encodings. Smith et al. (2000) also produced a thorough survey of various encodings of Golomb Ruler problems (including the DOUBLE encoding), though from their results I could identify no such linear correlation, only that the DOUBLE encoding performed

[13] Bessiere et al. (2002) describes $nFC0$ and $nFC1$, which refer to certain types of generalisations of **FC** for non-binary CSPs.

significantly worse than the HIDDEN VARIABLE and DUAL encodings alone. The DOUBLE encoding is a good example that highlights the lack of a formal framework to develop and compare encodings.

3.5 Preprocessing

Preprocessing is a recent area of research that has formed around CNF-formula transformation and simplification. The aim of preprocessing a problem is to reduce the search-space (not necessarily reduce the problem size), allowing stochastic and branching algorithms to find a solution more quickly. A smaller problem implies that there is less that the *SAT*-Solver needs to process, however, this does not necessarily imply it is easier to solve. Indeed, some of the hardest problems are those that have no 'redundant' information present in the problem.

In recent years a flurry of research has been published regarding the application and impact of preprocessing *SAT* instances prior to the use of a *SAT*-Solver. This suggests that as the performance of *SAT*-Solvers begins to plateau *SAT* researchers are exploring other methods to help improve performance. Preprocessing a formula before solving is now known as an important step (Lynce & Marques-Silva (2001)), and many preprocessors have already been proposed, several of which are described below. One of the first and simplest preprocessing algorithm, called 3-RESOLUTION, performed 4-*Resolution* — adding to the formula all resolvent clauses of size less than or equal to 3 — until saturation, however, this algorithm is often too slow and computationally expensive to be used in practice on large instances.

3.5.0.1 SIMPLIFY-2

Brafman (2004) proposed 2-SIMPLIFY, a less computationally heavy preprocessor than applying 4-*Resolution*, which was developed to better manage real-world benchmarks that often contain many binary clauses. Roughly, the idea is thus to use those binary clauses to construct an implication graph, from which unit clauses can be deduced by computing the transitive closure. Any unit clauses that have been obtained are propagated and this process is iterated until an exit point is reached.

3.5.0.2 HYPRE

HYPRE, developed by Bacchus & Winter (2003), employs a form of binary reasoning called 'hyper-binary resolution' in addition to the techniques found in 2-SIMPLIFY. 'Hyper-binary resolution' performs a resolution step involving more than two input clauses to generate binary clauses, using a method that is similar to (but more restricted than) ***NG-RES***.

3.5.0.3 NIVER

A weaker schema has been adopted by the NIVER procedure (Subbarayan & Pradhan (2004)), which stands for "Non-Increasing Variable Elimination Resolution". This technique attempts to overcome the size-explosion problem associated with variable elimination by only eliminating variables by resolution if this computation does not increase the number of literals of the CNF formula.

3.5.0.4 SATELITE

SATELITE, by Een & Biere (2005), is one of the most effective preprocessing techniques, so much so that it is currently integrated in many state-of-the-art *SAT*-Solvers. SATELITE improves on NIVER by combining binary clause resolution simplification with non-increasing variable-elimination, adding new resolution rules for clause subsumption. Clause subsumption proves to be useful for simplifying clauses resulting from variable elimination, enabling an efficient clause-variable simplification procedure which can be repeated until no more reductions are possible.

3.5.0.5 REVIVAL

Piette et al. (2008) points out that the main problem of these preprocessors is that it is difficult to measure the relevance of each added or eliminated clause with respect to the resolution step. It is possible that a preprocessor eliminates clauses but can derive a harder sub-formula. Similarly, adding new clauses may increase the space complexity without reducing the search-space. Piette et al. (2008) proposes a new preprocessing technique based on limited forms of resolution and conflict analysis, called REVIVAL. REVIVAL uses clause redundancy checking to produce sub-clauses and to add new relevant clauses. The aim is to substitute existing clauses by more constrained ones.

3.5.0.6 Preprocessor Results

Een & Biere (2005) published an excellent paper that definitively demonstrated that preprocessing can not only significantly improve the solubility of industrial instances, but that the time invested by the preprocessor is also worthwhile. More specifically, Een & Biere (2005) extended implementation aspects of NIVER, and demonstrated its performance with three of the world's leading *SAT*-Solvers (BERKMIN Goldberg & Novikov (2002), MINISAT Een & Sorensson (2003), and ZCHAFF Moskewicz et al. (2001)). Although the encoding and preprocessing of *SAT* problems is cited as having an important role, our understanding of how and when to use these techniques is still very limited. This is highlighted none more so than by the winners of this year's (2008) *SAT* Race, Een and Sorensson, who state that although the preprocessing of MINISAT 2.1 scales relatively well, there are still cases where it takes too much time or memory, so as a simple safe-guard measure preprocessing is deactivated if the problem has more than 4 million clauses.

Recently Condrat & Kalla (2007) applied the Gröbner basis engine (see Buchberger & Winkler (1998)) to many SATLIB benchmarks prior to using a state-of-the art *SAT*-Solver. On many instances the benefits of applying these preprocessing techniques far outweighed the time spent during preprocessing, but the processing varied greatly, with some problems benefiting from large numbers of clauses processed, and others very few. Also the time saved during solving varied from only marginal improvement to significant savings, however, in many cases the *SAT*-Solver could still find solutions in less time. Condrat & Kalla (2007) also combined the Gröbner basis engine with SATELITE but the results were

mixed.

However, it appears that combining several preprocessors often produces even better improvements. Indeed, a combination of SATELITE and REVIVAL produced good results in *SAT*-Race 2008. Anbulagan & Slaney. (2006) proposed a multiple preprocessing technique (using the preprocessors described above) to boost the performance of systematic *SAT*-Solvers, and argued that applying multiple preprocessors prior to the systematic search process can improve overall performance because each preprocessor takes different strategy to simplify clause sets. One finding of Anbulagan & Slaney. (2006) was that the use of multiple preprocessors one after the other can be much more effective than using any one of them alone, but that the order in which they are applied is significant. Their empirical study of the effects of several recently proposed *SAT* preprocessors prior to applying a two leading *SAT*-Solvers highlighted several outcomes:

1. *SAT*-Solvers benefit greatly from preprocessing. Improvements of four orders of magnitude in runtimes are not uncommon.

2. It is unlikely to equip a *SAT*-Solver with just one preprocessor of the kind considered. Very different preprocessing techniques are appropriate to different problem classes.

3. There are frequently benefits to be gained from running two or more preprocessors in series on the same problem instance.

4. *SAT*-Solvers can also benefit greatly from resolution between longer clauses, as in the 3-RESOLUTION preprocessor, but the effects are far from uniform.

Relatively little study has been carried out on enforcing a local-level of consistency prior to applying complete or stochastic algorithms such as **DLL** and GSAT, owing to the likely reason that making a problem more than strong-3-consistent can be very computationally expensive. However, Kask & Dechter (1995) published a paper that focused on the problem of how enforcing a slight variant of *path*-consistency improves the performance of GSAT. In particular, they investigated the effect of this preprocessing step on two different classes of problems; random uniform problems that do not have any structure, and random structured problems. Though this study was performed on random problem types, they found that the effect of local consistency is sharply different on these two classes of problems. When problems do not have any special structure, enforcing local-consistency does not have a significant effect on the performance of GSAT. However, on certain classes of structured problems, local-consistency can significantly improve the performance of GSAT. Their experiments showed that enforcing local-consistency can make these problems almost trivial for GSAT, and that the overhead associated with this preprocessing is much less than the computation needed to solve the problem without it. Kask & Dechter (1995) do not state what encoding of *SAT* to CSP was used to allow local-consistency

to be enforced, but from my theoretical and empirical analysis (see Chapters 5 and 6) it appears that the choice of encoding can dramatically effect the performance of the preprocessor.

The motivation for applying the preprocessing algorithms is to reduce (or even totally eliminate) the number of backtracks required to identify the solution. It is now well acknowledged that the performances of *SAT*-Solvers is usually greatly improved by preprocessing, up to the point where SATELITE is now often used by *SAT* competitors. Preprocessing approaches have traditionally concentrated on reducing the overhead, and techniques such as HYPRE, NIVER, REVIVAL and SATELITE reduce this overhead through resolution-based preprocessing. Whilst some empirical studies have been undertaken to determine the effect of enforcing a local-level of consistency prior to applying a backtrack or stochastic algorithm, little theoretical analysis has been performed. de Kleer (1989) shows the equivalence for *Consistency* and *Resolution* algorithms on DIRECT encodings of CSPs and in Chapter 5 I provide a comprehensive study of the equivalence between these two algorithms on each of the *SAT* to CSP encodings. I demonstrate these results empirically in Chapter 6 where I also show that enforcing a low-level of local-consistency on DUAL encoded *SAT* instances can solve many 'hard' *SAT* instances.

3.6 Proof Complexity

Resolution is the most studied propositional proof-system owing to its simplicity and relation to automated theorem proving algorithms used in industry. Whereas *Resolution* is a proof-system for *SAT*, *Consistency* is a proof-system for CSP. Although this is well-known (implicitly) in the Constraint Satisfaction community it is only in the past few years that it has been explicitly stated. As far as I am aware this was first 'completely' defined by Atserias et al. (2004), in their paper titled '*Constraint Propagation as a Proof System*', who argued the importance of 'mapping' the proof-system space. Atserias et al. (2004) pointed out that *"viewing constraint propagation as a proof system lifts proof complexity from propositional logic to all constraint-satisfaction problems"*, and suggested that this could lead to CSP-Solvers that deal directly with the CSP instances, avoiding the need to translate to CNF and applying a *SAT*-Solver. In this 'spirit', in Section 5.3, I extend the proof-system of *Consistency* inspired by the synergies with *Resolution* techniques described throughout this thesis.

The complexity of a *Resolution* proof is the number of clauses generated during the course of the proof (also known as the proof *length* and *size*). This 'resource' (*size*) has been related to the maximum *width* of the proof (see Clegg et al. (1996); Beame & Pitassi (1996)) where the *width* of a problem is simply any clause with the maximum number of literals generated by the proof (Galil (1977)).

Definition 3.6.1 (Proof *width*)**.** *The width of a clause is the number of literals appearing in it. The width of a set of clauses is the maximal width of a clause in the set. The width needed for the Resolution of an unsatisfiable CNF formula is the minimal width needed over its Resolution refutations.*

3.6.1 The *Width-Size* Relation

This relation is extremely important for proving *size* lower bounds, since thanks to it, it is sufficient to prove *width* lower bounds[14]. Since the SATISFIABILITY PROBLEM is an **NP**-complete problem, if *Resolution* could always give proofs that are bounded polynomially in *length* ($\propto$ *width*), then **co-NP** would equal **NP**. Two decades after Robinson (1965) defined *Resolution*, Haken (1985) proved the first exponential bounds for *Resolution* in his seminal thesis on the PIGEON-HOLE PROBLEM.

Haken was followed shortly by Urquhart (1987) who proved exponential *Resolution* bounds for Tseitin Graphs. Ben-Sasson (2001) developed a general strategy for proving *width* based on Haken's original proof method. Ben-Sasson & Wigderson (1999) used this 'simplified' method to reaffirm proof lower-bounds for the PIGEON-HOLE PROBLEM, Tseitin Graphs and Random CNFs, as well as new lower bounds for two variants of the PIGEON-HOLE PROBLEM. The basic relation between the complexity measure of resolution *size* and *width* is:

Theorem 3.6.1 (*width-Size* relation, Ben-Sasson & Wigderson (1999))**.** *For an unsatisfiable formula F in CNF with n variables, if F has a Resolution refutation of size S, then it has a refutation of maximal width, $W(F)$, of $O(\sqrt{n \log S(F)})$. If F has a maximal width $W(F)$, then its size, $S(F)$ is $\exp^{\Omega(\frac{W(F)^2}{n})}$.*

Ben-Sasson & Wigderson (1999) describe an immediate consequence of this *width-size* relation which is a procedure for refuting unsatisfiable formulae, described by Algorithm 3.

Algorithm 3 *k-RES width* algorithm.

```
Given F
for k = 1 to W(F) do
  if k-RES(F) ⊢ ⊥ then
    return ⊥
  end if
end for
return ⊤
```

It is easy to see that this algorithm runs in time $n^{O(W(F))}$, where n is the number of variables in formula F (with maximal *width* $W(F)$). This algorithm was originally investigated by Galil (1977). However, Ben-Sasson & Wigderson (1999) proved that this dynamic algorithm never performs much worse than the standard **DP** procedure (described in Section 2.5.4) and provided a family of formulae for which Algorithm 3 performs exponentially faster than **DP**.

3.6.2 *Local* and *Global* Consistency

Though the importance of *width* did not appear in the *SAT* literature until the early nineties, it was already becoming established in the Constraint Satisfaction community over ten years earlier in the seminal paper by Freuder (1982). This research not only introduced the importance of the *width-size* relation,

[14] *Space* is another important 'resource' that was recently introduced by Esteban & Torán (2001) and Ben-Sasson (2001). The *space* of a *Resolution* refutation is the number of clauses that have to be kept in memory simultaneously to infer a contradiction. See Atserias & Dalmau (2003) and Toran (2004) for an excellent introduction to this research.

but also described a procedure to solve CSPs similar to Algorithm 3; again highlighting how little the *SAT* and CSP communities have communicated over the years. Later, Dechter (1992b) provided another extremely important *width-size* relation for CSPs, proving that a strong-$(d(r-1)+1)$-consistent CSP (with domain cardinality d and maximal arity r) is *globally consistent*. It is well-known that problems with a guaranteed fixed width (not a function of n) can be solved in polynomial-time. For instance, *2-SAT* has a fixed *width* of size 2, i.e. it is impossible to infer clauses with more than two literals.

Theorem 3.6.2 (strong-$(d(r-1)+1)$-consistency ensures *global* consistency, Dechter (1992b))**.** *Any d-valued r-ary constraint network that is strong-$(d(r-1)+1)$-consistent is globally consistent. In particular, any d-valued binary constraint network that is strong-$(d+1)$-consistent is globally consistent.*

van Beek & Dechter (1997) extended this work by identifying two new complementary properties on the restrictiveness of the constraints in a network called *constraint tightness* and *constraint looseness*.

Definition 3.6.2 (m-tight)**.** *A constraint relation R of arity k is called m-tight if for any variable x_i constrained by R and any instantiation $\bar{a}$ of the remaining k variables constrained by R either there are at most m extensions of $\bar{a}$ to x_i that satisfy R or there are exactly $|D_i|$ such extensions, where D_i is the domain of x_i.*

Definition 3.6.3 (m-loose)**.** *A constraint relation R of arity k is called m-loose if for any variable x_i constrained by R and any instantiation $\bar{a}$ of the remaining k variables constrained by R there are at least m extensions of $\bar{a}$ to x_i that satisfy R.*

Their results can be viewed as an improvement on Dechter's theorem, in the sense that the tightness of the constraints specify a level of strong-consistency that is less than or equal to the level of strong-consistency required by Dechter's theorem. They showed that these measures can be used to estimate the level of local-consistency needed to ensure *global* consistency. In addition, they presented a sufficient condition based on *constraint tightness* and the level of local-consistency that guarantees that a solution can be found in a backtrack-free manner.

Theorem 3.6.3 (van Beek & Dechter (1994))**.** *If a binary constraint network, R, is m-tight and if the network is strong-$(m+2)$-consistent, then the network is globally consistent.*

The relation to CSP is that r (maximal arity) is analogous to clause *width*, so in general CSPs with $O(n)$ arity have exponential lower-bounds refutational complexity for strong-consistency. One example widely studied in the *Resolution* proof complexity field is the PIGEON-HOLE PROBLEM (*PHP*).

3.6.3 Pigeon-Hole Problem

Definition 3.6.4 (Pigeon-Hole Problem)**.** *The* PIGEON-HOLE PROBLEM*, with p pigeons and h holes, states that there is no one-one mapping from p to h when $p > h$. This is also referred to as PHP_h^p.*

A CSP formulation of PHP_h^p is a triple $(\mathcal{V}, \mathcal{D}, \mathcal{C})$, where

- $\mathcal{V}$ *is a finite set of pigeons* $\{x_0, x_1 \ldots x_{p-1}\}$
- $\mathcal{D}$ *is a the domain is holes* $\{y_0, y_1 \ldots y_{h-1}\}$
- $\mathcal{C}$ *is the set of constraints stating that only one pigeon can be in a hole:*
  - $x_i^{y_k} \neq x_j^{y_k}$, *for* $0 \leq i < j < p, 0 \leq k < h$

As we have seen, this can be formulated as a CNF formula on $p \times h$ variables x_{ik}, $0 \leq i < p$, $0 \leq k < h$, where x_{ik}^1 means that pigeon i is in hole k. The typical way of encoding *PHP* into CNF is by the MULTIVALUED encoding described in Section 3.2.2.3:

$$positive: \quad \bigwedge_{0 \leq i < p} \bigvee_{0 \leq k < h} x_{ik} \tag{3.1}$$

$$constraint: \quad \bigwedge_{0 \leq i < j < p} \bigwedge_{0 \leq k < h} \bar{x}_{ik} \vee \bar{x}_{jk} \tag{3.2}$$

When there are more pigeons than holes the problem is unsatisfiable. Consider PHP_n^{n+1}. This instance is unsatisfiable with $O(n^2)$ variables and $O(n^2)$ clauses, where each positive clause has a *width* of size n. With a domain cardinality n and binary constraints, according to Dechter (1992b), PHP_n^{n+1} requires strong-$(n(2-1)+1)$-consistency (strong-$(n+1)$-consistency), which is the maximum level of consistency possible for this problem. Deduction techniques used in *constraint*-solvers are not very effective when the difference constraints are considered independently of each other and a classical search algorithm (*Resolution*) which does not use *global* information will explore the entire search tree of size $(n-1)!$ to prove inconsistency. Search-based solvers have intrinsic limitations, and global constraints have been a means, in the Constraint Satisfaction field, to overcome them. For instance *alldiff* constraints (mentioned in the Introduction) can be propagated very efficiently using graph-based algorithms to solve *PHP*.

The difficulty of *PHP* for *Resolution* should not be underestimated. For instance, it may sound easier to determine unsatisfiability if the number of pigeons was much greater than the number of holes, but it is just as hard. Proving exponential bounds for the *PHP* is still an active area of research.

Theorem 3.6.4 (Haken (1985))**.** $S(PHP_n^{n+1}) = 2^{\Omega(n)}$.

Buss & Turan (1988) generalised Haken's lower-bound where the number of pigeons is much larger than the number of holes.

Theorem 3.6.5 (Buss & Turan (1988))**.** $S(PHP_h^p) = 2^{\Omega(\frac{h^2}{p \log p})}$.

Razborov (2001) improved on the findings of Raz (2001) showing that when $p > h$:

Theorem 3.6.6 (Razborov (2001))**.** $S(PHP_h^p) = 2^{\Omega(h^{1/3})}$.

3.6.4 Extended Proof-Systems

Resolution is the most theoretically studied propositional proof-systems and though one of the simplest, it proved extremely challenging to determine its first exponential complexity bounds. Given that the **DLL** algorithm produces a search space that is *tree-like* a great deal of research has been carried out on the complexity of *tree-like Resolution*[15] (also called *General Resolution* in which identical clauses can be derived more than once).

However, there are proof-systems that produce polynomial proofs for *PHP*, namely Frege-systems and Extended-*Resolution* (Cook (1976)).

Definition 3.6.5 (Frege proof-system)**.** *A Frege system F consists of a language, a finite number of axiom schemes and inference rules, which are sound and complete. For instance, a particular language may contain the constants $\{0, 1\}$, connectives $\{\neg, \wedge, \vee\}$ and some atoms $\{a, b, \dots\}$. An axiom is any substitution instance of an axiom scheme. For example, F often has several axiom schemes and only one inference rule (e.g. the modus ponens).*

Analysis of Frege proof-system is beyond the scope of this thesis, but for the interested reader the literature of Buss (1987), Ajtai (1994) and Bonet et al. (1994) is highly recommended. Extended-*Resolution* (a less restricted version of *Resolution*) is one of the most powerful proof-systems available (equal to Extended-Frege[16]) which allows the introduction of auxiliary variables to maintain a constant arity (clause width). Remarkably, no known problems exist that demand exponential Extended-*Resolution* (or Extended-Frege) proofs.

It has been shown that the Frege proof-system is strictly more powerful than *Resolution* proofs, and that Extended-*Resolution* has the same power as Extended-Frege (one of the most powerful proof systems at our disposal) (Krajicek & Pudlak (1989)). In fact, there are no proof-systems known to be stronger than these extended proof-systems, and there are no known classes of problem that demand exponential size proofs for them. However, the fact that a proof-system is strong does not mean that it works well in practice. Very little is known about how to implement Extended-*Resolution* for the simple reason that virtually nothing is known about how to select new variables so as to shorten proof length.

Although auxiliary variables have been used in the Constraint Satisfaction community for some time, their application is still very much an art. For instance Smith et al. (2000) performed an extensive study of the use of auxiliary variables and implied constraints in modelling a class of non-binary CSPs called *problems of distance*. Their experiments show that the introduction of auxiliary variables and implied constraints significantly reduced the size of the search-space.

[15] Analysis of restricted versions of *Resolution* are beyond the scope of this thesis, but the interested reader may want to start with Mitchell (2003) and Ben-Sasson & Wigderson (1999).

[16] Extended-Frege is a simply a Frege proof-system extended in the same way that *Resolution* is extended to Extended-*Resolution* with the *Extension Rule*.

Only in recent years have papers begun to emerge in which these extended techniques have been applied to *SAT*, namely Sinz & Biere (2006) and Jussila et al. (2006). This work evaluates a practical method to obtain Extended-*Resolution* proofs for conjoining Binary Decision Diagrams (BDDs [17]) in *SAT* solving. Their results enable the use of BDDs for these purposes instead (or in combination with) already established methods based on **DLL** with clause learning.

3.7 Chapter Summary and Discussion

In this chapter I provided an extensive survey of the main *SAT* and CSP encodings, as well as a detailed review of the major theoretical and empirical investigations performed on them. This investigation highlights several questions and areas of research that this thesis addresses:

- Some encodings that have been published that are useful, whilst others are not. Is there a common framework to allow researchers from both the *SAT* and CSP communities to assess existing encodings and to develop and assess new encodings?
- It has been shown that *solution-density* can effect the performance of stochastic and branching algorithms. Is there a distinction between the various encodings according to their *solution-density*?
- A large amount of theoretical research has been published comparing the performance of *SAT* and CSP-based stochastic and branching algorithmic techniques on various encodings. Is there a way to compare the relative theoretical performance of preprocessing (local-consistency) techniques used in these two fields?
- Several empirical results have been published demonstrating the performance of stochastic and branching algorithmic techniques on various encodings, yet relatively little on the effect of applying incremental levels of local-consistency. Do local-consistency algorithms perform better with some encodings than others?
- *Width* is in important characteristic that correlates with the complexity of problems. Some theoretical and empirical work has been published that looks at '*width management*' techniques, such as Extended-*Resolution* and Extended-Frege proof-systems. Can the *Consistency* proof-system be extended in a way that might facilitate synergies between these lines of research, and provide a complete landscape of these three proof-systems?

3.7.1 Theoretical Studies Summary

I have highlighted several situations where research from the *SAT* and CSP communities has been repeated by the other. The aim of Chapter 4 is to provide a framework within which current and future *SAT*

[17] Briefly, a Binary Decision Diagram is a data structure that is used to represent a Boolean function as a rooted, directed, acyclic graph, which consists of decision nodes and two terminal nodes called 0-terminal and 1-terminal. Each decision node is labelled by a Boolean variable and has two child nodes called low child and high child. The edge from a node to a low (high) child represents an assignment of the variable to 0 (1).

and CSP encodings can be developed. This framework results in a new type of CSP to *SAT* encoding that I show can have advantages over previous encodings.

There has been a large amount of theoretical analysis comparing stochastic and branching *SAT* and CSP algorithmic techniques. However, very little research has been done comparing the effect of enforcing local-consistency algorithms for each approach, probably because these techniques have previously been considered incomparable. This is true when comparing *Resolution* and *Consistency* on many types of encodings, but with the introduction of ***NG-RES*** a theoretical comparison is now possible, and in Chapter 5 I provide a comparative study between *Resolution* and *Consistency* on each of the *SAT* to CSP encodings. The main result of my theoretical analysis is that enforcing local-consistency on DUAL encoded problems does more work than on LITERAL encoded instances. Although this might not be totally surprising (since DUAL encoded problems are typically larger than those that are LITERAL encoded) the empirical results on each of these encodings in Chapter 6 are surprising.

In addition, some empirical work on comparing algorithmic performance on instances with different *solution-density* has been performed, and it has been shown that this measure can be used to choose between stochastic and branching algorithms. In Chapter 5 I demonstrate a new way of characterising *SAT* to CSP encodings based on the *solution-density*. In particular, I show that DUAL encoded instances can have a lower proportion of solutions than the LITERAL encoded instances, for example. Since it has been shown that *solution-density* is an important factor in determining the solubility of an instance, my work provides a guide to assist in choosing one encoding over another. Also in Chapter 5 I use the graph-theoretic framework to reconfirm and strengthen the equivalence between the *Resolution*-based and *Consistency*-based proof methods, and introducing the concept of Extended-*Consistency*, thus providing a complete picture of the Frege, *SAT* and CSP proof-systems.

3.7.2 Empirical Studies Summary

Two key problematic aspects of previous empirical studies of *SAT* and CSP encodings often arise that call into question the wider implications of the results. First, typically one type of problem is used as a benchmark to compare encodings. Second, either stochastic or backtracking algorithms are applied to the encoded problems. Clearly both of these choices may bias the results, since an encoding/algorithm may 'favour' a particular problem. Although each author may proclaim the benefits of their encoding, a rigorous empirical and theoretical investigation remains to be performed to definitively determine their advantages and disadvantages. However, to perform such a comprehensive and rigorous survey of these encodings is a mammoth task.

Whilst much current research focuses on how *SAT* techniques can be utilised by the Constraint Satisfaction community, this thesis addresses the opposite, asking what CSP techniques can aid *SAT*. In particular I show in Chapter 6 that converting *SAT* instances to CSP and applying local-consistency

can solve many 'hard' *SAT* instances and even compete with state-of-the-art *SAT*-Solvers. Currently there are two families of algorithm adopted by the *SAT* community, stochastic and branching. Branching algorithms can prove both satisfiability and unsatisfiability, whereas stochastic algorithms can only prove the former. Sophisticated local-consistency algorithms might be a viable approach to redress this balance, unable to determine whether an instance is satisfiable but capable of proving unsatisfiability.

Although the general consensus is that bridging the two fields of Propositional Satisfiability and Constraint Satisfaction is mutually beneficial, only a handful of researchers have crossed the chasm. As the performance of *SAT*-Solvers begins to plateau, the necessity to look towards other fields for improved solutions has become more important. Bennaceur (2004) suggests that comparative analysis between the methods of these two frameworks might lead to the conception of hybrid methods for the *SAT* problem.

Chapter 4

Categorising Encodings

As discussed in the previous chapter, problems encoded using the DOUBLE encoding (Section 3.4.1) unnecessarily repeats aspects of its own data-structure. This instance highlights one reason why it is important to categorise the encoding landscape. In this chapter I demonstrate how all of the *SAT* to CSP encodings (and vice-versa) can be categorised as one of three types of mappings (DOMAIN, CONSTRAINT and COMBINED). Categorising the encodings in this general way is useful because it highlights gaps that can be filled by several new CSP and *SAT* encodings, one of which (the INVERSE encoding) I formally define and demonstrate its relative advantages over some of the other encodings.

4.1 Mapping Categories

Analysis of the CSP and *SAT* encodings highlight three categories of CSP mappings that I call:

1. DOMAIN
2. CONSTRAINT
3. COMBINED

These categories are inspired by observing how the CSP *micro-structure* is constructed and expressed.

4.1.1 DOMAIN Mapping

DOMAIN mappings map the variable domains of a CSP to the variables of the resulting CSP. The constraints of the original CSP map to constraints in the resulting CSP. Recall the definition of the *SAT* to CSP NON-BINARY encoding:

Definition 4.1.1 (NON-BINARY encoding)**.** *Every CNF propositional variable is associated with a CSP variable x_j, each with the domain $\{0, 1\}$. The constraints are the partial assignments that fail to satisfy each clause.*

The CSP resulting from the NON-BINARY encoding of Formula 2.1 on page 56 is:

- $\mathcal{V} = \{x_0, x_1, x_2, x_3\}$

- $\mathcal{D} = \{\mathcal{D}_{x_0}, \mathcal{D}_{x_1}, \mathcal{D}_{x_2}, \mathcal{D}_{x_3}\}$, where $\mathcal{D}_{x_i} = \{0, 1\}, i < n$
- $\mathcal{C} = \{\mathcal{C}_{x_0,x_1,x_2}, \mathcal{C}_{x_0,x_2,x_3}, \mathcal{C}_{x_0,x_1,x_3}, \mathcal{C}_{x_1,x_2,x_3}\}$, where
  - $\mathcal{C}_{x_0,x_1,x_2} = \{\eta\{x_0^0, x_1^0, x_2^0\}, \eta\{x_0^1, x_1^0, x_2^1\}\}$
  - $\mathcal{C}_{x_0,x_2,x_3} = \{\eta\{x_0^1, x_2^1, x_3^1\}\}$
  - $\mathcal{C}_{x_0,x_1,x_3} = \{\eta\{x_0^0, x_1^1, x_3^0\}\}$
  - $\mathcal{C}_{x_1,x_2,x_3} = \{\eta\{x_1^0, x_2^1, x_3^0\}\}$

Figure 4.1 (from Section 3.3.3) represents the *micro-structure complement* of this CSP. We can see clearly that this is a DOMAIN mapping since the component sets (new variable domains) map to each variable in $\mathcal{V}$, the nodes to $\mathcal{D}$, and the 3-hypercliques (the new constraints) to the *nogoods* defined by $\mathcal{C}$. That is, the *SAT* variables map to CSP variables, and the clauses to the CSP *nogoods* (defined by the constraints).

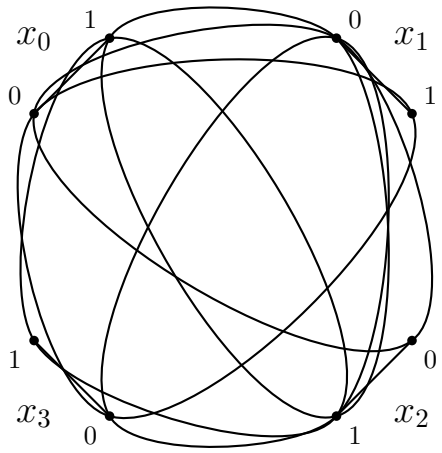

Figure 4.1: Formula 2.1 as a $\bar{G}_2^4$ 3-hypergraph using the NON-BINARY encoding. Variables are represented at the component sets, whereas the ternary clauses are represented as hyperedges that connect three nodes (i.e. the unsatisfiable assignments to each clause).

None of the other *SAT* to CSP encodings described in Chapter 3 fit into the DOMAIN category. This is in contrast to the CSP to *SAT* encodings. Recall the definitions of the CSP to *SAT* DIRECT, SUPPORT and LOG encodings described in Section 3.1. It is clear that the variable domains of the CSP map to new *SAT* variables (and *positive* clauses), with the CSP *nogoods* mapping to *SAT* clauses. Although this is less obvious in the case of the LOG encoding we can see that the *SAT* variables result from the logarithmic encoding of the CSP variable domains, with the *SAT* clauses constructed from the *nogoods* defined by the CSP constraints.

By observing how the CSP instances encoded to *SAT* we can see that *all* of the CSP to *SAT* encodings described in Chapter 3 encode the CSP variable domains to *SAT* variables, and CSP *nogoods* to *SAT*

clauses (i.e. they are all DOMAIN mappings). This is summarised in Table 4.1. Notice that there are no CONSTRAINT or COMBINED CSP to *SAT* encodings.

DOMAIN	CONSTRAINT	COMBINED
DIRECT		
SUPPORT		
LOG		
MULTIVALUED		

Table 4.1: Categorising the CSP to *SAT* encodings.

However, the most common *SAT* to CSP encodings (DUAL and LITERAL) map the *SAT* clauses to CSP variables, with the *SAT* variables encoded as CSP constraints. This type of transformation I call the CONSTRAINT mapping.

4.1.2 CONSTRAINT Mapping

The CONSTRAINT mapping is quite different from the DOMAIN mapping. Encodings that fall into the CONSTRAINT category are constructed in such a way that the resulting CSP variable domains map to some representation of the satisfying tuples of the original constraints, and the resulting constraints are enforced by the variable domains in the original CSP. Given a CSP P we construct a new CSP P' using some encoding, the distinction between the DOMAIN and CONSTRAINT mappings is as follows:

- DOMAIN Mapping:
  - P variables $\mapsto$ P' variables.
  - P constraints $\mapsto$ P' constraints.
- CONSTRAINT Mapping:
  - P constraints $\mapsto$ P' variables.
  - P variables $\mapsto$ P' constraints.

Figure 4.2 (from Section 3.3.1) below represents the CSP *micro-structure complement* of the LITERAL encoding of Formula 2.1. Notice that the nodes in the component sets (the new CSP variable domains) are constructed from the *SAT* clauses. The edges enforce that two variables cannot take different values. The LITERAL encoding can therefore be categorised as a CONSTRAINT mapping.

Similarly, recall the definition of the DUAL encoding in Section 3.3.2.

Definition 4.1.2 (DUAL encoding)**.** *With the* DUAL *encoding each clause C_i is associated with a variable $c_i \in \mathcal{V}$. The domain of c_i is the set of satisfying assignments to the clause C_i. The set of constraints $\mathcal{C}$ are binary and are posted between the CSP variables that have opposing proposition assignments.*

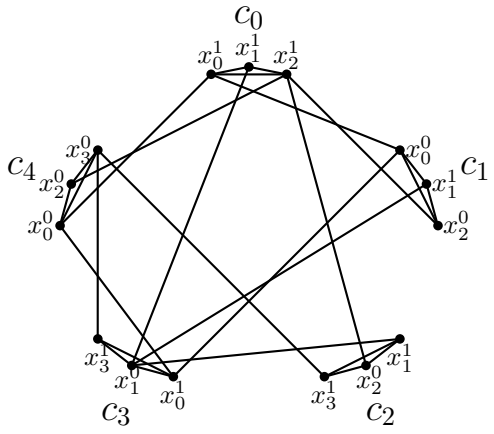

Figure 4.2: Formula 2.1 as a $\bar{G}_3^5$ graph using the LITERAL encoding.

The DUAL encoding takes the partial satisfying assignments to a clause and represents each clause as a new CSP variable with its domain as those partial assignments satisfying it. The CSP binary constraints enforce that no original propositional variable can take different domain values. Loosely, the DUAL encoding maps satisfying solutions of *SAT* clauses to CSP variables, with the *SAT* domains enforce the CSP constraints Therefore we can see that it fits into the CONSTRAINT mapping category.

4.1.3 COMBINED Mapping

Encodings that fall into the COMBINED mapping category simply take the variables resulting from the DOMAIN and CONSTRAINT mappings, and constructs the constraints in such a way that these variables cannot take conflicting domain values that would violate the variables in the original CSP.

Both the PLACE and HIDDEN VARIABLE encodings are instances of the COMBINED mapping. Take the PLACE encoding of Formula 2.1 (Section 3.3.4) as an example, Figure 4.3 illustrates how this encoding produces the two component set sets (i.e. the variables from the LITERAL and NON-BINARY encodings) and that the edges between the nodes ensure that the original variable domains are not violated. Notice that these edges infer the constraints (the dotted binary and ternary edges) that are explicitly stated in the two resulting CSPs that it is constructed from. In this case, the nodes and implied edges encode the combined *micro-structure complements* of the LITERAL and NON-BINARY encodings. Take the dotted edge between the nodes $\{c_0^{x_0^1}, c_4^{x_0^0}\}$ in Figure 4.3 for instance. Owing to some of the constraints this edge cannot form an independent set with the nodes in the component set x_0. Since the dotted edge cannot form a local independent set, it cannot be part of a global independent set, so an edge can be safely added to the graph. Similarly, the dotted ternary independent set between the nodes $\{x_0^0, x_1^0, x_2^0\}$ cannot be extended to an independent set (of size 4) with any node in the component set c_0, so a 3-clique can also be safely added to the graph. In the next chapter I demonstrate that this inference method is equivalent to *Consistency* and *Resolution* proof techniques in the *SAT* and CSP fields.

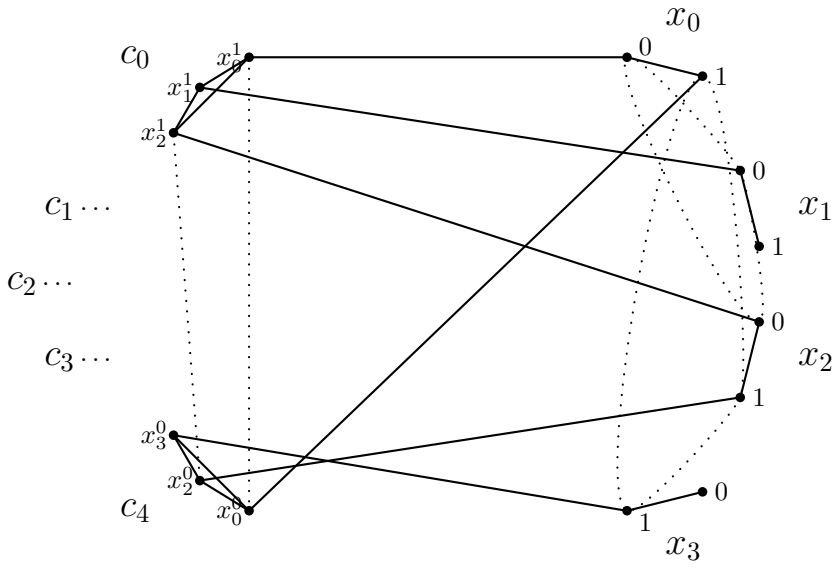

Figure 4.3: Illustrating that the PLACE encoding is a combined LITERAL and NON-BINARY encoding.

Similarly, it is easy to see that the HIDDEN VARIABLE encoding is a combination of both the DUAL and NON-BINARY encodings in exactly the same way that the PLACE encoding is a combination of the LITERAL and NON-BINARY encodings. Table 4.2 shows the *SAT* to CSP encodings described in this Chapter 3 categorised as DOMAIN, CONSTRAINT, COMBINED mappings.

DOMAIN	CONSTRAINT	COMBINED
NON-BINARY	LITERAL	PLACE
	DUAL	HIDDEN VARIABLE

Table 4.2: Categorising the *SAT* to CSP encodings.

4.2 INVERSE Encoding

In this section I introduce a new type of encoding that I call the INVERSE encoding inspired by the three categories of mappings identified in this chapter. The CSP to *SAT* encodings described thus far fall into the DOMAIN family, since with all these encodings the CSP variable domains map to *SAT* variables and the constraints map to *constraint* clauses. Encodings that fall into the CONSTRAINT category have the opposite mapping such that CSP constraints map to *SAT* variables and the CSP variable domains map to the *constraint* clauses. COMBINED mappings are a combination between DOMAIN and CONSTRAINT mappings.

Definition 4.2.1 (INVERSE encoding)**.** *The* INVERSE *encoding maps CSP satisfying assignments to* SAT *variables, and constrains these variables using the CSP domains:*

- positive: *a* positive *clause C_i is generated for each constraint, $\mathcal{C}_i$, in $\mathcal{C}$. The literals of C_i each map to a unique tuple satisfying $\mathcal{C}_i$ (i.e. there is a* bijection *between the satisfying-tuples and literals). For instance if c_j and c_k are adjacent nodes in a* GRAPH 3-COLOURABILITY PROBLEM *(i.e. a constraint $\mathcal{C}_{c_j c_k}$) then the set of satisfying tuples for (c_j, c_k) would be $\{\gamma\{c_j^R, c_k^G\}, \gamma\{c_j^R, c_k^B\}, \gamma\{c_j^G, c_k^R\}, \gamma\{c_j^G, c_k^B\}, \gamma\{c_j^B, c_k^R\}, \gamma\{c_j^B, c_k^G\}\}$, each mapping to a literal in C_i $(l_{c_j^R c_k^G} \vee l_{c_j^R c_k^B} \vee l_{c_j^G c_k^R} \vee l_{c_j^G c_k^B} \vee l_{c_j^B c_k^R} \vee l_{c_j^B c_k^G})$.*

- constraint: constraint *clauses are specified by the domains of the CSP variables, they contain two literals that have been mapped to the same variables but assigned different domain values. Using the* GRAPH 3-COLOURABILITY *for example, if* positive *clause C_a contains the literal $l_{c_j^R c_k^G}$ and C_b contains the literal $l_{c_k^R c_p^B}$ then the* constraint *clause $(\bar{l}_{c_j^R c_k^G} \vee \bar{l}_{c_k^R c_p^B})$ would be generated. Intuitively, the* constraint *clauses specify that two literals cannot assign two different domain values to the same variable.*

- negative: negative *clauses may be included to constrain the* positive *clauses to only have one satisfying assignment. That is, for each positive clause C_i*

$$\bigwedge_{\substack{S \subset C_i \\ |S|=2}} \left(\bigvee_{j \in S} \bar{l}_j \right)$$

4.2.0.1 INVERSE Encoding Complexity

On the assumption that a CSP has n variables (each with a fixed domain size m) and q constraints (with arity r), the INVERSE encoding will generate a *SAT* instance with $O(qm^r)$ variables and $q + O(q^2 m^r)$ clauses:

- q *positive* clauses of size $O(m^r)$
- $O(q^2 m^r)$ binary *constraint* clauses

Recall Example 2.2.3 which has 5 variables (each with ternary domains) and 6 binary constraints. There are 6 ($O(3^3)$) partial satisfying assignments to each of the 6 constraints. This generates 6 *positive* clauses of size 6. The maximum number of unique binary clauses over these 36 literals is $6^2 + 3^3$. Appendix B.2 shows the *SAT* encoding of Example 2.2.3 using the INVERSE encoding (excluding the *negative* clauses), and Figure 4.4 represents its (partial) G_6^6 graph (note that this is the *micro-structure*, not the *complement*).

Table 4.3 shows a summary of the complexity of the INVERSE encoding with respect to the others. These encodings are based on a CSP with n variables with domain size m, and q r-ary constraints. The notation $a : b$ denotes that a clauses are generated of arity b.

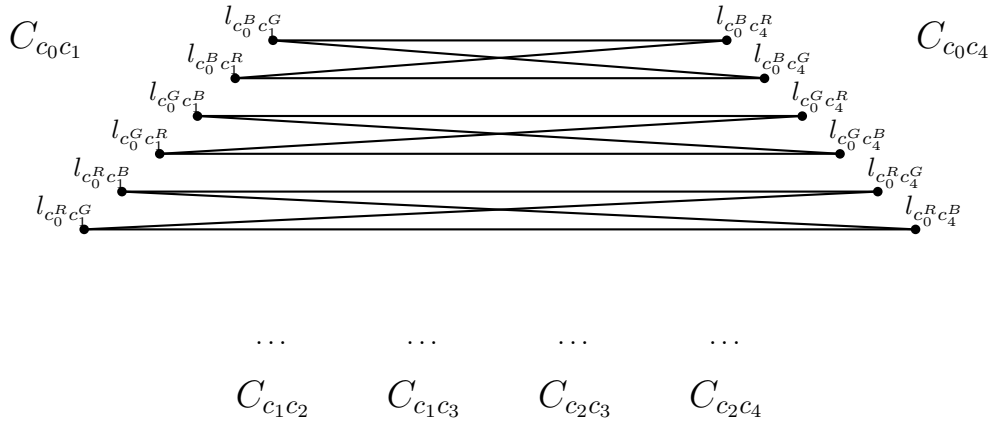

Figure 4.4: The (partial) G_6^6 graph of Example 2.2.3 mapped to *SAT* using the INVERSE encoding. Notice that the partial satisfying assignments to a constraint map to component sets, and binary edges connect nodes that represent variables that do not violate the variable domain assignments.

encoding	**propositions**	**positive**	**negative**	**constraint**
DIRECT	nm	$n : m$	$n\binom{m}{2} : 2$	$q : r$
SUPPORT	nm	$n : m$	$n\binom{m}{2} : 2$	$2q : m$
LOG	$n\lceil\log_2(m)\rceil$	0	$q(2^{\lceil\log_2(m)\rceil} - n) : \lceil\log_2(m)\rceil$	$q : r\lceil\log_2(m)\rceil$
MULTIVALUED	nm	$n : m$	0	$q : r$
INVERSE	$O(qm^r)$	$q : O(m^r)$	0	$O(q^2m^r) : 2$

Table 4.3: The CSP to *SAT* encoding complexity, including the INVERSE encoding.

As highlighted in Section 3.1.1, it is not always feasible to encode CSP problems into *SAT* using the DIRECT encoding. Take a simple example CSP.

Example 4.2.1. *Given a CSP* $= (\mathcal{V}, \mathcal{D}, \mathcal{C})$*, such that*

- $\mathcal{V} : \{a, b, c\}$
- $\mathcal{D} : \mathcal{D}_a = \mathcal{D}_b = \mathcal{D}_c = \{1, 2, \ldots, 1024\}$
- $\mathcal{C} : b = 2^a, c = (b-1)^2, a = \sqrt[3]{c}$

Translated to *SAT* using the DIRECT encoding, this example will generate:

- 3 *positive* clauses of arity 1024
- over 3 million binary *constraint* clauses
- over 1.5 million binary *negative* clauses

The INVERSE encoding is much more compact, generating only:

- 3 *positive* clauses:
  - 1 of arity 10 ($C_{a,b}$)
  - 1 of arity 6 ($C_{a,c}$)
  - 1 of arity 32 ($C_{b,c}$)
- less than 600 binary *constraint* clauses
- approximately 550 binary *negative* clauses

The INVERSE encoding is preferable when the number of partial satisfying assignments to the constraint is small, since each partial satisfying assignment gets mapped to a *SAT* variable. If, on the other hand, a problem has constraints that have many partial satisfying assignments, then careful consideration should be taken to decide which encoding is best to use.

4.3 Chapter Summary and Discussion

After defining three types of mapping categories, I demonstrate that the encodings described in the previous chapter can be categorised as either a DOMAIN, CONSTRAINT or COMBINED mapping. As a result of this categorisation a new encoding from CSP to *SAT*, the INVERSE encoding, is defined and it is shown to have some benefits over other encodings with respect to the compactness of the representation. I use a simple example to demonstrate that the INVERSE encoding is preferable when the number of partial satisfying assignments to the constraints are small. With a simple calculation researchers might find that they can now practically represent some CSP problems as *SAT* using the INVERSE encoding that were previously impractical.

The INVERSE encoding resulting from this categorisation demonstrates another benefit of examining the relationship between *SAT* and CSP research. With the introduction of this new encoding it is now possible to define COMBINED mappings for CSP to *SAT* encodings and explore the benefits that these new encodings may bring. For instance, the same way that the PLACE encoding is the combination of the LITERAL and NON-BINARY encodings a new encoding for CSP to *SAT* can be formed by combining the DIRECT and INVERSE encodings. I summarise these in Table 4.4, however, since the focus of this thesis is on *SAT* to CSP encodings, exploring the relative benefits of these new encodings is left as an open challenge.

DOMAIN	CONSTRAINT	COMBINED
DIRECT	INVERSE	INVERSE + DIRECT
SUPPORT		INVERSE + SUPPORT
LOG		INVERSE + LOG

Table 4.4: Categorising the CSP to *SAT* encodings.

Chapter 5

Characterising *SAT* to CSP Encodings

Though the fields of Constraint Satisfaction and Propositional Satisfiability have developed relatively independently, I have shown that working with the *micro-structure* can help us understand the similarities and differences between the two disciplines. In Chapter 4 the focus was on the expressive power of *SAT* and CSPs, in this chapter I provide a theoretical analysis comparing the work that polynomial-time techniques achieve on problems encoded as *SAT* and CSP, where 'work' is defined as the level of consistency that these polynomial-time techniques achieve on CSP and *SAT* instances.

As discussed in Chapter 3 the *solution-density* is one measure that can sometimes be used to determine performance of stochastic and branching algorithms on problem instances. In particular, it has been shown that stochastic algorithms can perform better on problems with a higher *solution-density*. In the first part of this chapter I separate *SAT* to CSP encodings according to their *solution-density* and prove that some encodings will result in CSPs with more solutions than others. This is important, as it may help guide which encoding should be used in a particular situation (i.e. problem type and available algorithm).

The second part of this chapter addresses the gap in the theoretical research that compares the performance of *Resolution* and *Consistency* based techniques on the various encodings. For instance, I show that although there is a space overhead associated with DUAL encoded problems in comparison to using the LITERAL encoding, enforcing local-consistency on the resulting CSP does much more work. I also prove that enforcing *path*-consistency on LITERAL encoded *3-SAT* instances does zero work if each clause has distinct literals, which explains the empirical results presented in the next chapter.

Finally, I investigate the proof-systems in both the *SAT* and CSP research areas, and show that the graph-theoretic approach is a useful framework to explore the similarities and differences between these techniques. In particular, I demonstrate that (strong-)*Consistency* and *Resolution* are exactly the same technique; in graph-theoretic terms both infer implicit *nogoods* by taking a clique and showing that it cannot extend to a larger clique and hence cannot be part of a *global* clique. Inspired by the work of Tseitin, Baker and Mitchell I define Extended-*Consistency* (the generalisation of Extended-*Resolution*)

which allows synergies between these two extended proof-systems to be explored. Using an example I demonstrate a simple automated method for introducing auxiliary variables with the aim of promoting the development of algorithms that might exploit the potential of these powerful proof-systems.

5.1 Solution Separation of *SAT* to CSP Encodings

In this section I provide a characterisation of *SAT* to CSP encodings according to their *solution-density*. By analysing the resulting *micro-structure* I show that CSP encoded *SAT* problems result in different *solution-densities*.

5.1.1 LITERAL Encoded Solutions

When encoding *SAT* problems to CSP using the LITERAL encoding, it is important to notice that there is a *surjective* and *non-injective* mapping from the n-cliques in the G_k^n graph to the *solution-tuples* satisfying the original *SAT* instance. To phrase it another way, there may be *more* n-cliques in the resulting CSP *micro-structure* than satisfying assignments to the original *SAT* instance. That is, the CSP resulting from the LITERAL encoding of a *SAT* instance can include more *solution-tuples* than the number of satisfying assignments.

Recall the CSP *micro-structure complement* of the LITERAL encoding of Formula 2.1 in Section 3.3.1 (shown in Figure 5.1). Notice for instance that a (dotted) 5-clique, $\gamma\{c_0^{x_0^1}, c_1^{x_2^0}, c_2^{x_2^0}, c_3^{x_0^1}, c_4^{x_2^0}\}$ can be formed in Figure 5.1 stating that $\{x_0^1, x_2^0\}$ satisfies Formula 2.1 regardless of the assignments to the remaining literals (this can be verified by looking at Table 3.3).

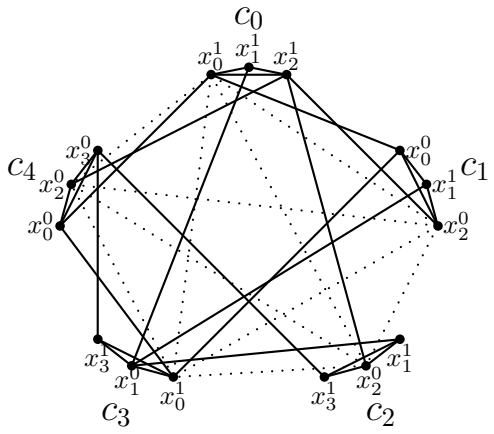

Figure 5.1: The $\bar{G}_3^5$ LITERAL encoded Formula 2.1, highlighting the $\gamma\{x_0^1, x_2^0\}$ solution.

Remark 5.1.1. *The CSP resulting from the* LITERAL *encoding of a satisfiable* SAT *instance* Ψ *may contain more (but no less)* solution-tuples *than the number of assignments satisfying* Ψ.

In terms of the *micro-structure*,

Remark 5.1.2. *the G_k^n graph resulting from the* LITERAL *encoding of a satisfiable* SAT *instance Ψ may contain more (but no less) n-cliques than the number of assignments satisfying Ψ.*

Remark 5.1.3. *The size of the search-space of a* LITERAL *encoded* k-SAT *instance Ψ is k^m, where m is the number of clauses in Ψ.*

5.1.2 DUAL Encoded Solutions

In contrast to the LITERAL encoding, given a *SAT* instance Ψ, the DUAL encoding produces a G_k^n graph where there is a *bijection* between the n-cliques and satisfying assignments to Ψ. This means that the G_k^n graph contains exactly the same number of n-cliques as there are satisfying solutions to Ψ. As far as I am aware this distinction has not been made previously.

Theorem 5.1.4. *Let Ψ be a* 3-SAT *formula and let $(G_7^n, (S_i : i < n))$ be the instance of $\mathcal{G}_7^n$, which is the* micro-structure *of the CSP obtained from Ψ using the* DUAL *encoding (where S_i is an independent set of nodes of G_7^n). There is a* bijection *from the set of valuations satisfying Ψ to the set of n-cliques of G_7^n.*

Proof. Let υ be any valuation to the propositions occurring in a *3-SAT* formula $\Psi = \bigwedge_{i<n} C_i$. For each $i < n$ let υ_i be the partial valuation obtained by restricting υ to the propositions in C_i.

The required *bijection* is

$$\theta : \upsilon \mapsto \{(i, \upsilon_i) : i < n\}.$$

Note that if υ satisfies Ψ (i.e. $\upsilon(\Psi) = \top$) then υ satisfies each of the n clauses of Ψ, hence the restriction υ_i satisfies the i'th clause, so $(i, \upsilon_i) \in S_i$. Also, for any $i, j < n$, the partial valuations υ_i, υ_j cannot contradict each other, since they are restrictions of the same valuation υ. Hence, if υ satisfies Ψ then $\{(i, \upsilon_i) : i < n\}$ is an n-clique of G_7^n, so θ is well-defined. To see that θ is *injective*, let $\upsilon \neq \omega$ be two valuations on the propositions occurring in Ψ. Since $\upsilon \neq \omega$ there must be $i < n$ such that $\upsilon_i \neq \omega_i$, hence $\theta(\upsilon) \neq \theta(\omega)$. To see that θ is *surjective*, let X be any n-clique of the graph G_7^n. X contains exactly one node from each independent set S_i, so let $(i, \omega_i) \in X$ (each $i < n$) where $\omega_i \in S_i$. Since X is a clique, none of the ω_i's contradict each other, hence $\omega = \bigcup\{\omega_i : i < n\}$ is a well-defined valuation satisfying Ψ, and ω maps to X, as required. □

Remark 5.1.5. *The size of the search-space of a* DUAL *encoded* k-SAT *instance Ψ is $(2^{k-1})^m$, where m is the number of clauses in Ψ.*

For DUAL encoded problems I show that there is a *bijection* between the CSP solutions and satisfying assignments to the original *SAT* instance. This is significant because it highlights a way to differentiate between the 'nature' of these encodings. Also, since the *solution-density* of these encodings

may differ previous literature suggests that satisfiable LITERAL encoded problems could be easier to solve by means of applying a stochastic algorithm than other encodings.

If we take Formula 2.1 as an example. Table 3.3 (Chapter 3) lists the eight satisfying assignments to Formula 2.1. The CSP resulting from the DUAL encoding contains exactly eight full satisfying assignments (eight n-cliques in the CSP *micro-structure*), whereas the CSP generated by the LITERAL encoding contains 48 full satisfying assessments. The difference between the two encodings is even more compelling when one compares the *solution-densities*. The original *SAT* instance has 2^4 (16) possible full assignments, of which 50% (8) are satisfiable. Notice that there are 7^5 possible full assignments in the DUAL encoding, whereas there are only 3^5 when LITERAL encoded. The DUAL encoded version thus has a *solution-density* of $\frac{8}{16,807}$ with the LITERAL encoding having a *solution-density* of $\frac{48}{243}$.

5.1.3 NON-BINARY Encoded Solutions

Similarly for the NON-BINARY encoding, the G_2^n also contains the same number of n-cliques[1] as there are satisfying solutions to a *SAT* instance translated to CSP using the NON-BINARY encoding.

Theorem 5.1.6. *Let Ψ be a* 3-SAT *formula and let $(G_2^n, (S_i : i < n))$ be the instance of $\mathcal{G}_2^n$, which is the* micro-structure *of the CSP obtained from Ψ by the* NON-BINARY *encoding (where S_i is an independent set of nodes of G_2^n). There is a* bijection *from the set of valuations satisfying Ψ to the set of n-cliques of G_2^n.*

Proof. Since each n-clique must include one node from each set S_i, and each S_i represents a proposition, then every n-clique must map to a valuation that includes all propositions (i.e. there are no possible partial valuations to the propositions as in the case of the LITERAL encoding). □

Remark 5.1.7. *The size of the search-space of a* NON-BINARY *encoded* k-SAT *instance Ψ is 2^n, where n is the number of propositions in Ψ.*

5.1.4 PLACE and HIDDEN VARIABLE Encoded Solutions

In Chapter 4 I showed that the PLACE and HIDDEN VARIABLE encodings are combinations of NON-BINARY + LITERAL, and NON-BINARY + DUAL respectively. Given this, determining the mapping between *SAT* satisfying assignments and CSP *solution-tuples* for these encodings is straightforward. For instance, the number of *solution-tuples* resulting from the translation of *SAT* to CSP using the PLACE encoding is the maximum number of *solution-tuples* in either the NON-BINARY or LITERAL encoding. This means that the number of *solution-tuples* in CSP resulting from the PLACE encoding is equal to the number of *solution-tuples* resulting from the LITERAL encoding, since the LITERAL encoding has at least as many *solution-tuples* as the CSP generated using the NON-BINARY encoding. As a result, we can make the following remark:

[1] Though in this case the n-cliques are made up of hyperedges rather than edges.

Remark 5.1.8. *The* PLACE *encoding of a* SAT *instance* Ψ *produces a CSP with the same number of* solution-tuples *as the CSP resulting from the* LITERAL *encoding of* Ψ. *This CSP encoding can produce more (but no less)* solution-tuples *than the number of satisfying solutions to the original* SAT *instance.*

Similarly the CSP resulting from the HIDDEN VARIABLE encoding has the same number of *solution-tuples* as the maximum number of *solution-tuples* resulting from using either the NON-BINARY or DUAL encoding. I have proven that there is a *bijection* between the CSP *solution-tuples* and *SAT* satisfying assignments for both the NON-BINARY and DUAL encodings. This means that the *SAT* and resulting CSP instance both have the same number of solutions.

Remark 5.1.9. *The* HIDDEN VARIABLE *encoding of a* SAT *instance* Ψ *produces a CSP with the same number of* solution-tuples *as the CSP resulting from the* NON-BINARY *and* DUAL *encodings of* Ψ. *This encoding produces a CSP with the same number of solutions as the original* SAT *instance.*

It is easy to see that enforcing *path*-consistency on PLACE and HIDDEN VARIABLE encoded problems generate the *nogoods* that are already explicit in their constituent parts. For example, a PLACE encoded instance contains the same nodes from both the NON-BINARY and LITERAL encodings. Recall Figure 5.2 below from the previous chapter. Notice the binary constraints are posted between the two groups of component set (c and x), but not within each of the two sets. The dotted edges between c nodes will only be found by enforcing *path*-consistency, whereas only 4-consistency will find the ternary (dotted) hyperedges between the x nodes.

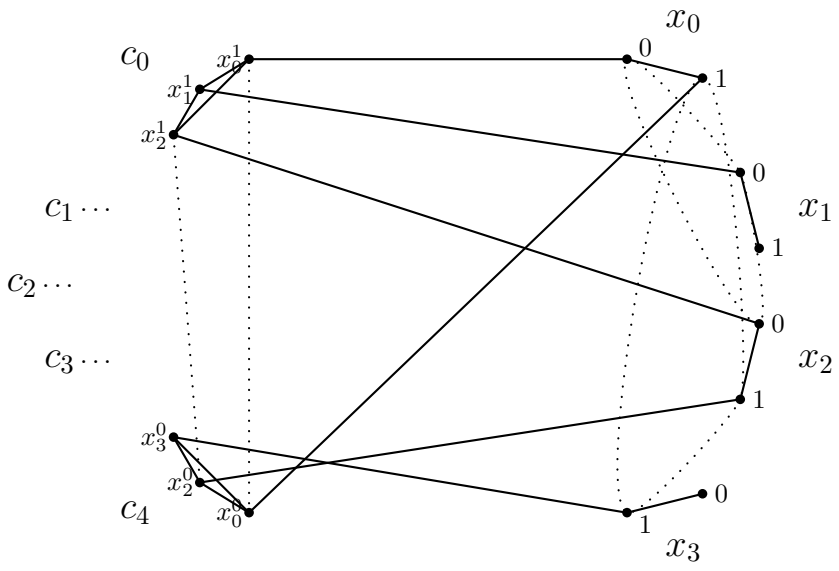

Figure 5.2: Illustrating that the PLACE encoding is a combined LITERAL and NON-BINARY encoding.

Given this, although Jarvisalo & Niemela (2004) points out that the PLACE encoding is the only one that generates a linear number of clauses, this does come at a cost. This encoding requires en-

forcing a level of local-consistency for it to explicitly represent the same constraints as the LITERAL and NON-BINARY encodings. The *nogoods* in Figure 5.2 (dotted lines) are not explicitly added until strong-4-consistency is applied to the CSP. In addition, although enforcing strong-k-consistency on PLACE encoded 'strict' *k-SAT* instances will find the same constraints explicitly represented in the corresponding LITERAL encoding, it does zero extra work than when simply encoding the instance into its two constituent parts (i.e. a LITERAL and NON-BINARY encoding).

5.2 Local-Consistency Analysis of *SAT* to CSP Encodings

In Chapter 3 I argued that although a large amount of theoretical analysis comparing stochastic and branching *SAT* and CSP algorithmic techniques has been carried out very little research has been done comparing local-consistency algorithms for each framework. As the performance of *SAT*-Solvers begins to plateau the *SAT* community are looking for alternative methods to improve the performance of their algorithms, and in recent years a flurry of work has looked at the use of 'practical' preprocessing. Enforcing local-levels of consistency is one form of preprocessing, and in this section I provide a comprehensive theoretical comparison of the work *Resolution* and *Consistency* do on the various *SAT* to CSP encodings.

5.2.1 *Resolution* $\equiv$ *Consistency*

In this section I show that it is possible to compare strong-consistency with *nogood-Resolution* for these encodings, but first it is necessary to define several extensions to the ***NG-RES*** proof-system.

Definition 5.2.1 (*k-**NG-RES***)**.** *Given that the domain of a variable x is $\{0, 1, \ldots, d-1\}$, the* k-nogood Resolution Rule *allows one to infer* nogoods *of arity less than k:*

$$\frac{\begin{array}{c}\eta\{\{x^0\} \cup X_0\} \\ \eta\{\{x^1\} \cup X_1\} \\ \vdots \\ \eta\{\{x^{d-1}\} \cup X_{d-1}\}\end{array}}{N = \eta\{X_0 \cup X_1 \cup \cdots \cup X_{d-1}\}, \textit{ where } |N| < k}$$

where X_i is a partial assignment (for $i < d$), and $\eta\{\{x^0\} \cup X_0\}, \eta\{\{x^1\} \cup X_1\}, \ldots, \eta\{\{x^{d-1}\} \cup X_{d-1}\}$ are nogoods.

It is well known that the various CSP techniques I have discussed are essentially constructing resolution proofs (Baker (1995)). This is easy to see using graphs as an intermediary framework and observing the behaviour of the respective algorithmic techniques. In graph-theoretic terms both the *Resolution* and *Consistency* algorithms state the same thing, namely, given a G_k^n graph *if a k-clique cannot extend to a $(k+1)$-clique, then this k-clique cannot be part of an n-clique*. These two concepts can be seen in Figures 5.3 and 5.4.

Figure 5.3 uses the GRAPH COLOURABILITY Example 2.2.3 and shows how a 3-ary *nogood*, $\{x_0^R, x_3^G, x_2^B\}$, can be added to the graph (using a strong-4-consistency algorithm) because this set cannot extend to any domain value (node) in x_1.

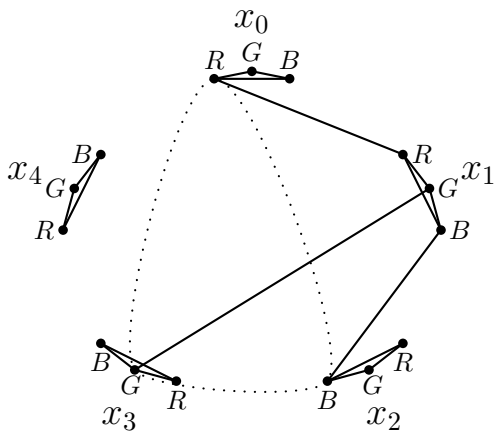

Figure 5.3: *Consistency* as a *clique* proof-system. Enforcing 4-consistency would uncover the *nogood* $\eta\{x_0^R, x_2^B, x_3^G\}$ since it cannot to extend to any domain value in variable x_1. Enforcing 4-***NG-RES*** achieves the same result.

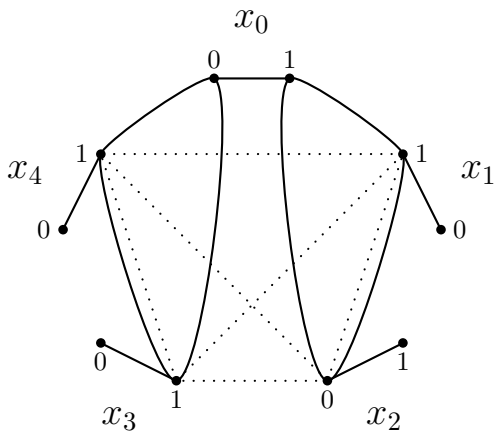

Figure 5.4: *Resolution* as a *clique* proof-system. Enforcing 5-consistency would uncover the *nogood* $\eta\{x_1^1, x_2^0, x_3^1, x_4^1\}$ since it cannot to extend to any domain value in variable x_0. Enforcing 5-***NG-RES*** achieves the same result.

Similarly, Figure 5.4 shows a hypothetical situation where two ternary constraints, $\langle\langle x_0, x_3, x_4\rangle, \setminus\{\langle 0,1,1\rangle\}\rangle$ and $\langle\langle x_0, x_1, x_2\rangle, \setminus\{\langle 1,1,0\rangle\}\rangle$, (derived from the clauses $(x_0 \vee \bar{x}_3 \vee \bar{x}_4)$ and $(\bar{x}_0 \vee \bar{x}_1 \vee x_2)$) imply that the set $\{x_1^1, x_2^0, x_3^1, x_4^1\}$ cannot extend to any domain node in x_0, so the corresponding 4-ary *nogood* can be safely added to the graph:

$$\frac{\eta\{x_0^0, x_3^1, x_4^1\} \quad \eta\{x_0^1, x_1^1, x_2^0\}}{\eta\{x_1^1, x_2^0, x_3^1, x_4^1\}}$$

This is exactly the same as resolving on x_0 over the two clauses $(x_0 \vee \bar{x}_3 \vee \bar{x}_4)$ and $(\bar{x}_0 \vee \bar{x}_1 \vee x_2)$ as follows:

$$\frac{x_0 \vee \bar{x}_3 \vee \bar{x}_4 \qquad \bar{x}_0 \vee \bar{x}_1 \vee x_2}{\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_1 \vee x_2}$$

5.2.2 Local-Consistency on the NON-BINARY Encoding

Hooker (2007) showed that k-resolution is equivalent to strong-k-consistency; where k-resolution is a version of the *Resolution* proof method restricting all resolvent clauses to less than size k.

Theorem 5.2.1 (Hooker (2007), Theorem 3.22)**.** *Enforcing k-Resolution on a* SAT *instance Ψ is equivalent to enforcing strong-k-consistency on the CSP resulting from the* NON-BINARY *encoding of Ψ. More precisely, the diagram shown in Table 5.1 is a commuting diagram.*

For instance, enforcing strong-3-consistency on NON-BINARY encoded *SAT* instances achieves 3-*Resolution* on the original instance.

k-SAT$_n^m$	$\longrightarrow$	NON-BINARY	$\longrightarrow$	CSP_2^n
Ψ				Φ
$\downarrow$				$\downarrow$
k-*Resolution*				strong-k-consistency
$\downarrow$				$\downarrow$
Ψ'	$\longrightarrow$	NON-BINARY	$\longrightarrow$	Φ'

Table 5.1: The algorithmic equivalence of *Resolution* and *Consistency* on NON-BINARY encodings.

5.2.3 Local-Consistency on the LITERAL Encoding

de Kleer (1989) originally defined the set of CNF-based inference rules (described in Section 3.2.2.1) that do the same work as strong-k-consistency on DIRECT encoded CSP instances:

$$\frac{\begin{array}{c}(x_0 \vee x_1 \vee \cdots \vee x_{i-1}) \\ (\bar{x}_0 \vee X_0) \\ (\bar{x}_1 \vee X_1) \\ \vdots \\ (\bar{x}_{i-1} \vee X_{i-1})\end{array}}{(X_0 \vee X_1 \vee \cdots \vee X_{i-1})}$$

where X_j are clauses and x_j are literals, for $j < i$.

de Kleer's inference rules describe a *Resolution* procedure - that is equivalent to enforcing *Consistency* - of a CSP that has been encoded to CNF using the DIRECT encoding. Without wishing to cause confusion I will refer to de Kleer's inference rules as ***NG-RES*** so as to be able to distinguish between the application of *Resolution* and *Consistency*-based techniques on their respective problem domains. In summary,

- ***NG-RES*** is a *Resolution*-based procedure applied to *SAT* instances.
- strong-consistency is a *Consistency*-based procedure applied to CSP instances.

Remark 5.2.2 (de Kleer (1989))**.** *Enforcing strong-k-consistency on a CSP Φ is equivalent to enforcing k-**NG-RES** on the* SAT *instance resulting from the* DIRECT *encoding of Φ. More precisely, the diagram shown in Table 5.2 is a commuting diagram.*

CSP	$\longrightarrow$	DIRECT	$\longrightarrow$	*SAT*
Φ				Ψ
$\downarrow$				$\downarrow$
strong-k-consistency				k-***NG-RES***
$\downarrow$				$\downarrow$
Φ'	$\longrightarrow$	DIRECT	$\longrightarrow$	Ψ'

Table 5.2: The algorithmic equivalence of *Resolution* and *Consistency* on DIRECT encodings.

Similarly for the LITERAL encoding of a *SAT* instance, enforcing strong-k-consistency on the resulting CSP performs exactly the same procedure as k-***NG-RES*** (see Table 5.3). When applying strong-k-consistency, a partial assignment (of arity $(k-1)$ or less) becomes a *nogood* if it cannot extend to at least one partial satisfying assignment that includes each other variable. Recall that the LITERAL encoding derives each CSP variable from each *SAT* clause. With k-***NG-RES***, clauses are resolved if a set of literals cannot extend to at least one other literal in each clause. Since both of these methods have the same variables and domains, and both 'exhaust the domain' to resolve conflicts, they both do exactly the same amount of work. This can be verified easily by observing the behaviour of two algorithmic procedures on the resulting *micro-structure*.

Theorem 5.2.3. *Enforcing k-**NG-RES** on a* SAT *instance Ψ is equivalent to enforcing strong-k-consistency on the CSP resulting from the* LITERAL *encoding of Ψ. More precisely, the diagram shown in Table 5.3 is a commuting diagram.*

j-SAT$_n^m$	$\longrightarrow$	LITERAL	$\longrightarrow$	CSP_j^m
Ψ				Φ
$\downarrow$				$\downarrow$
k-***NG-RES***				strong-k-consistency
$\downarrow$				$\downarrow$
Ψ'	$\longrightarrow$	LITERAL	$\longrightarrow$	Φ'

Table 5.3: The algorithmic equivalence of *Resolution* and *Consistency* on LITERAL encodings.

For instance, enforcing strong-3-consistency on LITERAL encoded *SAT* instances achieves 3-***NG-RES*** on the original instance.

A very important practical result of this theorem is that enforcing strong-k-consistency achieves *nothing* on *k-SAT* instances with clauses containing distinct literals.

Theorem 5.2.4. *Given any* k-SAT *instance encoded to a CSP using the* LITERAL *encoding, enforcing strong-k-consistency has no effect when the* k-SAT *instances have clauses containing distinct literals.*

Proof. Proof by contradiction. Let us assume that some *nogood*, N, of arity $k-1$ has been discovered by enforcing strong-k-consistency. This means that the partial assignment N cannot form a partial satisfying assignment with some variable, I, that has a domain of size k. I has been derived from some clause of size k, and by definition each literal was distinct. However, since the size of N is less than k, some element of N must be inconsistent with two values of I. Given that, in the LITERAL encoding, binary constraints are posted between variables that have complementary assignments, this means that I must contain literals that are not distinct. Hence, we have a contradiction. □

For example, enforcing strong-3-consistency on a LITERAL encoded *3-SAT* instance (with each clause having three distinct literals) will prune zero edges from the corresponding *micro-structure*.

Figure 5.5 is the CSP *micro-structure complement* of the LITERAL encoding of *3-SAT* Formula 2.1. Notice that if we enforce strong-3-consistency on this instance that every binary independent set can form an independent set with at least one node from each other component set. For a binary *nogood* (edge) to be derived it must not extend to a solution to any node in at least one component set, but this would only be possible if a component set did not have three distinct literals. For example, the dotted edge $\{C_0^{x_0^1}, C_3^{x_1^0}\}$ can extend (dashed lines) to only one node in the component set C_1 ($C_1^{x_2^0}$), since the arity of the component set is greater than the arity of the *nogood* we are trying to derive.

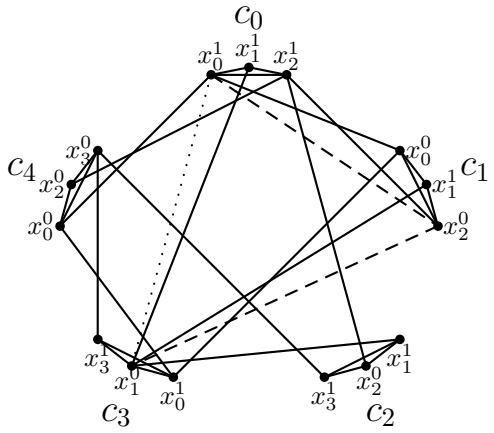

Figure 5.5: Illustrating that enforcing strong-3-consistency on LITERAL encoded *3-SAT* instances (that have distinct literals) does zero work.

Notice also that the relative work done by the *Resolution* and *Consistency* techniques on instances encoded using the NON-BINARY and LITERAL encodings appear to be identical. However, there are cases in which more *nogoods* are found in instances using the NON-BINARY encoding than using the LITERAL when applying strong-k-consistency techniques. For instance, given two clauses $(x_0 \vee x_1 \vee x_2)$ and $(\bar{x}_0 \vee x_1 \vee x_2)$, strong-3-consistency applied to the NON-BINARY encoding would resolve:

$$\frac{\eta\{x_0^0, x_1^0, x_2^0\} \quad \eta\{x_0^1, x_1^0, x_2^0\}}{\eta\{x_1^0, x_2^0\}}$$

Whereas in the LITERAL encoded instance the pair $\{x_1^0, x_2^0\}$ can form a partial satisfying solution with the x_0^0 and x_0^1 variables in the two component sets derived from the two clauses. Another practical consequence of this result is that (in this instance) enforcing a local-level of consistency on NON-BINARY encoded *SAT* instances does more work than when they are encoded using the LITERAL encoding.

5.2.4 Local-Consistency on the DUAL Encoding

Definition 5.2.2 ((i,j)-*NG-RES*). *Given a CSP = $(\mathcal{V}, \mathcal{D}, \mathcal{C})$. The* (i,j)-nogood Resolution Rule *allows one to infer* nogoods *of arity i or less. Given a partial assignment $A, |A| \leq i$, if for all partial satisfying assignments $B, |B| \leq j$ $A \cup B$ is a conflict, then we can deduce that A is a* nogood, $\eta(A)$.

That is, if some partial assignment I (of size i and less) cannot extend to a partial satisfying assignment J (of arity j and less), then I is a *nogood*.

Theorem 5.2.5. *Enforcing $(j(k-1), j)$-**NG-RES** on a* j-SAT *instance Ψ is equivalent to enforcing strong-k-consistency on the CSP resulting from the* DUAL *encoding of Ψ. More precisely, the diagram shown in Table 5.4 is a commuting diagram.*

Proof. For any *j-SAT* instance, CSP variable domains represent partial satisfying assignments to j literals when DUAL encoded. This means that any set of $(k-1)$ CSP variables encodes at most $j(k-1)$ *SAT* literals. When enforcing strong-k-consistency, if a partial satisfying variable assignment of size $(k-1)$ (a partial satisfying assignment to $j(k-1)$ *SAT* literals) cannot extend to a k^{th} partial satisfying assignment (a partial satisfying assignment to j *SAT* literals) in each domain then it becomes a *nogood*. By definition, this is exactly the same procedure as $(j(k-1), j)$-***NG-RES***. □

For instance, enforcing strong-3-consistency on DUAL encoded *3-SAT* instances achieves $(3(3-1), 3)$-***NG-RES*** on the original instance, i.e. $(6, 3)$-***NG-RES***. Notice that enforcing a local-level of consistency on DUAL encoded *k-SAT* instances does more work than it does on either the NON-BINARY and LITERAL encoded instances (3-*Resolution* and 3-***NG-RES*** respectively).

5.2.5 Local-Consistency on the PLACE and HIDDEN VARIABLE Encodings

Using the framework defined in Chapter 4 it is easy to extend these theoretical results to the PLACE and HIDDEN VARIABLE encodings. I showed in Chapter 4 that these two encodings were constructed by

j-SAT^m_n	$\longrightarrow$	DUAL	$\longrightarrow$	$CSP^m_{2^j-1}$
Ψ				Φ
$\downarrow$				$\downarrow$
$(j(k-1),j)$-***NG-RES***				strong-k-consistency
$\downarrow$				$\downarrow$
Ψ'	$\longrightarrow$	DUAL	$\longrightarrow$	Φ'

Table 5.4: The algorithmic equivalence of *Resolution* and *Consistency* on DUAL encodings.

combining NON-BINARY + LITERAL and NON-BINARY + DUAL, respectively. Let us label the CSP variables resulting from the PLACE and HIDDEN VARIABLE encodings in the following way:

- PLACE: PLACE$_{\text{NON-BINARY}}$ + PLACE$_{\text{LITERAL}}$
- HIDDEN VARIABLE: HIDDEN VARIABLE$_{\text{NON-BINARY}}$ + HIDDEN VARIABLE$_{\text{DUAL}}$

As a result we can prove the following theorems:

Theorem 5.2.6. *Enforcing strong-k-consistency on* PLACE *encoded* SAT *instances achieves k-**NG-RES** on the* SAT *literals represented by the* PLACE$_{\text{LITERAL}}$ *variables and k-**NG-RES** on the* PLACE$_{\text{NON-BINARY}}$ *variables.*

Proof. Theorem 5.2.1 and Theorem 5.2.3 states that enforcing strong-k-consistency on NON-BINARY and LITERAL encoded *SAT* instances (respectively) achieves k-***NG-RES*** on each instance. Since PLACE encoded *SAT* instances are constructed by adding constraints between NON-BINARY and LITERAL encoding variables (defined by PLACE$_{\text{NON-BINARY}}$ and PLACE$_{\text{LITERAL}}$) that have contradictory assignments, then enforcing strong-k-consistency on these instances achieves k-***NG-RES*** on PLACE$_{\text{NON-BINARY}}$ and PLACE$_{\text{LITERAL}}$. □

Theorem 5.2.7. *Enforcing strong-k-consistency on* HIDDEN VARIABLE *encoded* j-SAT *instances achieves $(j(k-1),j)$-**NG-RES** on the* SAT *literals represented by the* HIDDEN VARIABLE$_{\text{DUAL}}$ *variables and k-**NG-RES** on the* HIDDEN VARIABLE$_{\text{NON-BINARY}}$ *variables.*

Proof. Theorem 5.2.1 states that enforcing strong-k-consistency on NON-BINARY encoded *SAT* instances achieves k-***NG-RES***. Theorem 5.2.5 states that enforcing strong-k-consistency on DUAL encoded *SAT* instances achieves $(j(k-1),j)$-***NG-RES***. Since HIDDEN VARIABLE encoded *SAT* instances are constructed by adding constraints between NON-BINARY and DUAL encoding variables (defined by HIDDEN VARIABLE$_{\text{NON-BINARY}}$ and HIDDEN VARIABLE$_{\text{DUAL}}$) that have contradictory assignments, then enforcing strong-k-consistency on these instances achieves k-***NG-RES*** on HIDDEN VARIABLE$_{\text{NON-BINARY}}$ and $(j(k-1),j)$-***NG-RES*** on HIDDEN VARIABLE$_{\text{DUAL}}$. □

5.3 Extended-*Consistency*

In the same way that Baker (1995) generalises the *Resolution Rule* for CSP (which Mitchell (2003) calls ***NG-RES***, described in Section 5.2.1) here I generalise the Extended-*Resolution Extension Rule* for any CSP with a finite size domain. I call this Extended-*Consistency*.

As discussed in Chapter 3, the *Consistency* proof-system was only recently explicitly defined by Atserias et al. (2004) who argued importance of 'mapping' the proof-system space. In this section, I demonstrate the value of using a graph-theoretic approach and use it to extend the *Consistency* proof-system. Inspired by the relationship between the *Resolution* and *Consistency* algorithmic techniques described throughout this chapter, the aim of introducing Extended-*Consistency* is to provide a general platform to allow the CSP and *SAT* communities to share knowledge about these extended proof-systems and to begin to exploit synergies that might be mutually beneficial to each field.

Recall that given that the domain of a variable x is $\{0, 1, \ldots, (d-1)\}$, the *nogood Resolution Rule* allows one to infer *nogoods* by resolving on x:

$$\frac{\begin{array}{c}\eta\{\{x^0\} \cup X_0\} \\ \eta\{\{x^1\} \cup X_1\} \\ \vdots \\ \eta\{\{x^{d-1}\} \cup X_{d-1}\}\end{array}}{\eta\{X_0 \cup X_1 \cup \cdots \cup X_{d-1}\}}$$

where X_i is a partial assignment (for $i < d$), and $\eta\{\{x^0\} \cup X_0\}, \eta\{\{x^1\} \cup X_1\}, \ldots, \eta\{\{x^{d-1}\} \cup X_{d-1}\}$ are *nogoods*. This may result in the inference of a *nogood* with arity $\sum_{i=0}^{d-1} |X_i|$. Sometimes is it undesirable to generate constraints with large arity, so by introducing new variables one can ensure a problem's arity never exceeds either $|(d-1)|$ or $max(|X_i| + 1)$, whichever is larger.

Definition 5.3.1 (Extended-*Consistency*)**.** *Given a CSP* $= (\mathcal{V}, \mathcal{D}, \mathcal{C})$ *(for simplicity assume that each variable has the same domain cardinality* d*, apart from the new variables, which are Boolean), and a set of* nogoods $\eta\{\{x^0\} \cup X_0\}, \eta\{\{x^1\} \cup X_1\}, \ldots, \eta\{\{x^{d-1}\} \cup X_{d-1}\}$*, where a variable* x *has the domain* $\{0, 1, \ldots, (d-1)\}$*. We add a new Boolean variable* v_i *to* $\mathcal{V}$ *(not previously in* $\mathcal{V}$*) for each* X_i *such that*

$$v_i \leftrightarrow X_i.$$

That is, for each $y_i^b \in X_i$ *the following set of constraints are added to* $\mathcal{C}$*:*

$$\bigcup_{j=0, j\neq b}^{d-1} \langle\langle v_i, y_i\rangle, \setminus\{\langle 0, j\rangle\}\rangle$$

where b *is the domain assignment to variable* y_i*. Also, for each* X_i *we add* $\langle\langle v_i, V(X_i)\rangle \setminus \{\langle 1, D(X_i)\rangle\}\rangle$*, where* $V(X_i)$ *is the list of variables of* X_i *and* $D(X_i)$ *is the list of domain assignments that map to those*

variables. Once the set of new variables have been included the last constraint that remains to be added to $\mathcal{C}$ is $\langle\langle v_1, v_2, \ldots, v_d\rangle, \setminus\{\langle 1, 1, \ldots, 1\rangle\}\rangle$.

This is best illustrated by use of an example.

Example 5.3.1. *Suppose we have the CSP =* $(\mathcal{V}, \mathcal{D}, \mathcal{C})$*:*

- $\mathcal{V} : \{x_0, x_1, x_2, x_3, x_4, x_5, x_6\}$
- $\mathcal{D} : \mathcal{D}_{x_0} = \mathcal{D}_{x_1} = \mathcal{D}_{x_2} = \mathcal{D}_{x_3} = \mathcal{D}_{x_4} = \mathcal{D}_{x_5} = \mathcal{D}_{x_6} = \{0, 1, 2\}$
- $\mathcal{C}$:
  - $\langle\langle x_0, x_5, x_6\rangle, \setminus\{\langle 0, 2, 0\rangle\}\rangle$
  - $\langle\langle x_0, x_3, x_4\rangle, \setminus\{\langle 1, 1, 2\rangle\}\rangle$
  - $\langle\langle x_0, x_1, x_2\rangle, \setminus\{\langle 2, 2, 0\rangle\}\rangle$

The *micro-structure complement* for this CSP is shown in Figure 5.6, which has seven variables (with domains $\{0, 1, 2\}$) and three ternary constraints. In this example the three constraints $\langle\langle x_0, x_5, x_6\rangle, \setminus\{\langle 0, 2, 0\rangle\}\rangle$, $\langle\langle x_0, x_3, x_4\rangle, \setminus\{\langle 1, 1, 2\rangle\}\rangle$ and $\langle\langle x_0, x_1, x_2\rangle, \setminus\{\langle 2, 2, 0\rangle\}\rangle$ imply that the set of nodes $\{x_5^2, x_6^0, x_3^1, x_4^2, x_1^2, x_2^0\}$ cannot extend to any domain value in x_0, generating the *nogood* $\eta\{x_5^2, x_6^0, x_3^1, x_4^2, x_1^2, x_2^0\}$ (represented as new dotted 6-ary hyperedge):

$$\frac{\begin{array}{c}\eta\{x_0^0, x_5^2, x_6^0\}\\ \eta\{x_0^1, x_3^1, x_4^2\}\\ \eta\{x_0^2, x_1^2, x_2^0\}\end{array}}{\eta\{x_5^2, x_6^0, x_3^1, x_4^2, x_1^2, x_2^0\}}\ \mathcal{D}_{x_0} = 0, 1, 2$$

Let us assume that it is undesirable to generate constraints with arity larger than three. We can introduce three new Boolean variables $\{v_0, v_1, v_2\}$ such that

$$\begin{array}{c} v_0 \leftrightarrow \{x_5^2, x_6^0\}\\ v_1 \leftrightarrow \{x_3^1, x_4^2\}\\ v_2 \leftrightarrow \{x_1^2, x_2^0\}.\end{array}$$

Take $v_0 \leftrightarrow \{x_5^2, x_6^0\}$, for instance, the following constraints are required to represent this formula (this is illustrated in Figure 5.7, where the dashed tri-clique is a hyperedge).

$$\begin{array}{c}\langle\langle v_0, x_6\rangle, \setminus\{\langle 0, 1\rangle\}\rangle\\ \langle\langle v_0, x_6\rangle, \setminus\{\langle 0, 2\rangle\}\rangle\\ \langle\langle v_0, x_5\rangle, \setminus\{\langle 0, 0\rangle\}\rangle\\ \langle\langle v_0, x_5\rangle, \setminus\{\langle 0, 1\rangle\}\rangle\\ \langle\langle v_0, x_5, x_6\rangle, \setminus\{\langle 1, 2, 0\rangle\}\rangle.\end{array}$$

The desired outcome is that when $\{x_5^2, x_6^0\}$ is assigned, then we have v_0^0, otherwise v_0^1. Notice in Figure 5.7 that v_0^0 can only form a clique with $\{x_5^2, x_6^0\}$, whereas all other combinations of x_5 and x_6 can only extend to v_0^1.

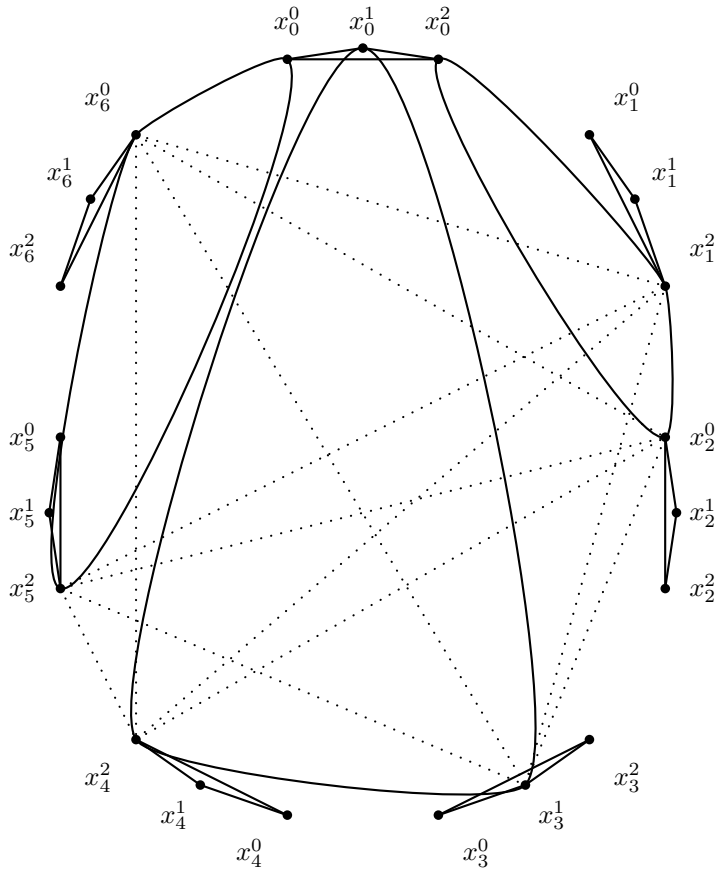

Figure 5.6: Example 5.3.1 CSP *micro-structure complement*.

We do the same for v_1 and v_2, finally adding the constraint $\langle\langle v_0, v_1, v_2\rangle, \setminus\{\langle 0,0,0\rangle\}\rangle$ (implying $\eta\{x_5^2, x_6^0, x_3^1, x_4^2, x_1^2, x_2^0\}$) as illustrated in Figure 5.8.

Figure 5.9 is the final result of encoding this 6-ary *nogood*, by including three auxiliary Boolean variables, twelve binary constraints and four ternary constraints, resulting in the new CSP:

- $\mathcal{V}$: $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6, v_0, v_1, v_2\}$
- $\mathcal{D}$:
  - $\mathcal{D}_{x_0} = \mathcal{D}_{x_1} = \mathcal{D}_{x_2} = \mathcal{D}_{x_3} = \mathcal{D}_{x_4} = \mathcal{D}_{x_5} = \mathcal{D}_{x_6} = \{0, 1, 2\}$
  - $\mathcal{D}_{v_0} = \mathcal{D}_{v_1} = \mathcal{D}_{v_2} = \{0, 1\}$
- $\mathcal{C}$:
  - $\langle\langle x_0, x_5, x_6\rangle, \setminus\{\langle 0,2,0\rangle\}\rangle$
  - $\langle\langle x_0, x_3, x_4\rangle, \setminus\{\langle 1,1,2\rangle\}\rangle$
  - $\langle\langle x_0, x_1, x_2\rangle, \setminus\{\langle 2,2,0\rangle\}\rangle$
  - $\langle\langle v_0, x_6\rangle, \setminus\{\langle 0,1\rangle\}\rangle$

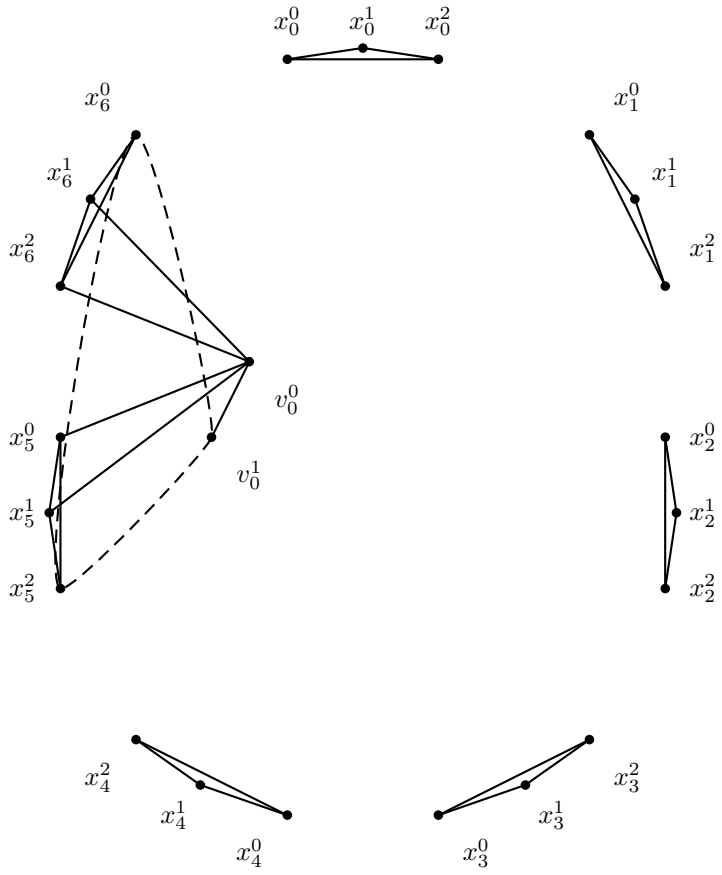

Figure 5.7: Example 5.3.1 CSP *micro-structure complement* with added variable v_0.

- $\langle\langle v_0, x_6\rangle, \setminus\{\langle 0, 2\rangle\}\rangle$
- $\langle\langle v_0, x_5\rangle, \setminus\{\langle 0, 0\rangle\}\rangle$
- $\langle\langle v_0, x_5\rangle, \setminus\{\langle 0, 1\rangle\}\rangle$
- $\langle\langle v_0, x_5, x_6\rangle, \setminus\{\langle 1, 2, 0\rangle\}\rangle$
- $\langle\langle v_1, x_4\rangle, \setminus\{\langle 0, 1\rangle\}\rangle$
- $\langle\langle v_1, x_4\rangle, \setminus\{\langle 0, 0\rangle\}\rangle$
- $\langle\langle v_1, x_3\rangle, \setminus\{\langle 0, 0\rangle\}\rangle$
- $\langle\langle v_1, x_3\rangle, \setminus\{\langle 0, 2\rangle\}\rangle$
- $\langle\langle v_1, x_3, x_4\rangle, \setminus\{\langle 1, 1, 2\rangle\}\rangle$
- $\langle\langle v_2, x_1\rangle, \setminus\{\langle 0, 0\rangle\}\rangle$
- $\langle\langle v_2, x_1\rangle, \setminus\{\langle 0, 1\rangle\}\rangle$
- $\langle\langle v_2, x_2\rangle, \setminus\{\langle 0, 1\rangle\}\rangle$
- $\langle\langle v_2, x_2\rangle, \setminus\{\langle 0, 2\rangle\}\rangle$

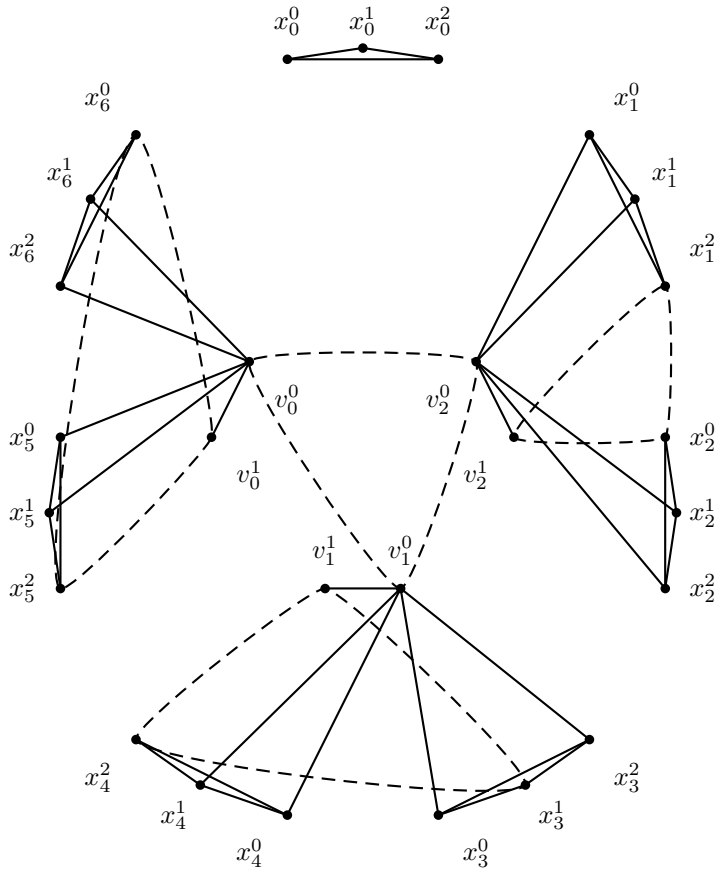

Figure 5.8: Example 5.3.1 CSP *micro-structure complement* with added variables v_0, v_1 and v_2.

- $\langle\langle v_2, x_1, x_2\rangle, \setminus\{\langle 1, 2, 0\rangle\}\rangle$
- $\langle\langle v_0, v_1, v_2\rangle, \setminus\{\langle 0, 0, 0\rangle\}\rangle$

It could be argued that the fact that no hard problems are known for extended proof-systems is not so astonishing, since very little is known about how to use auxiliary variables effectively. However, it is only in the past five years that researchers have begun to understand how to guide basic *Resolution* effectively, as evident by the remarkable performance of some of the most recent *SAT*-Solvers. By introducing Extended-*Consistency* it might be possible for the *SAT* community to benefit from the wealth of theoretical results from CSP research, including the idea of symmetry (Crawford et al. (1996)), m-tightness (van Beek & Dechter (1994)), etc. Similarly, the Constraints community may be able employ the efficient implementation of Extended-*Resolution* techniques of *SAT* research to improve the effectiveness of CSP algorithms.

MINION, developed by Gent et al. (2006), is an excellent example of the CSP community adopting techniques from *SAT* research to improved the practical performance of CSP solving. MINION is a fast and scalable CSP-Solver that has been optimised for solving large and hard problems. Several of MINIONS design decisions are modelled on those of ZCHAFF, including the use of very small data struc-

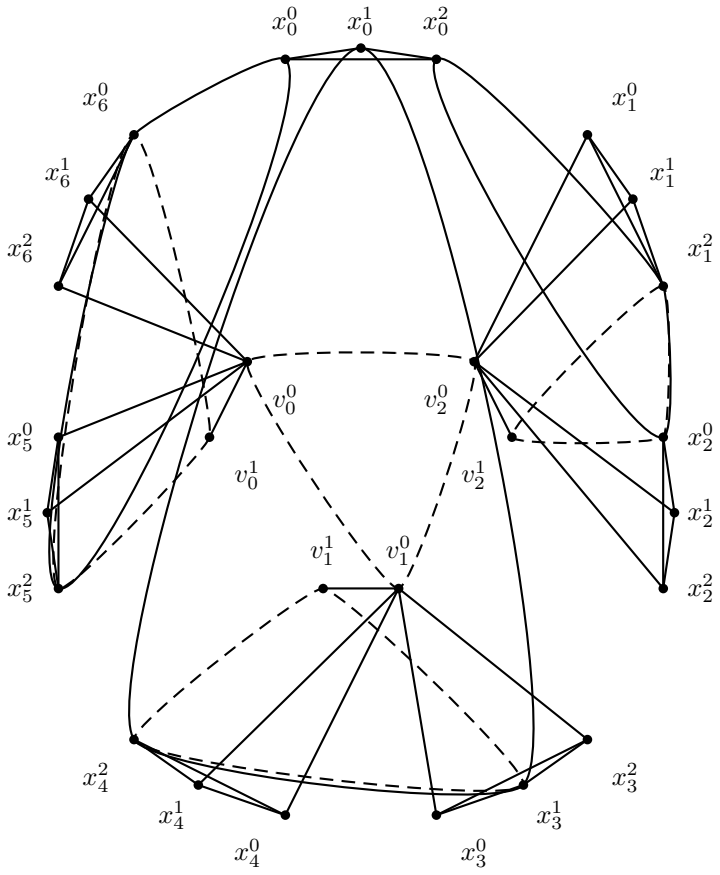

Figure 5.9: Combining Figures 5.6 and 5.8 together.

tures, making memory usage very small and greatly increasing speed on modern computer architectures. MINION has been shown to perform between one and two orders of magnitude faster than state-of-the-art CSP-Solvers.

5.4 Chapter Summary and Discussion

In this chapter I prove that transforming *SAT* instances into CSP using the LITERAL encoding produces problems that typically have more solutions than those produced by using the DUAL and NON-BINARY encodings. This is summarised in Table 5.5; for notation, we consider X vs. Y. $X = Y$ denotes that X and Y have equal number of solutions, and $X \geq Y$ denotes that X has at least the same number of solution tuples as Y. I mentioned in Chapter 3 that stochastic algorithms can perform better on problems with a higher *solution-density*, so these theoretical results might be practically useful in helping guide which encoding should be chosen for a particular problem instance, as this choice could mean that the problem could be solved more easily by using one encoding than another.

Having characterised the encodings according to their *solution-density*, the second part of this chapter addresses the gap in theoretical research identified in Chapter 3. I compared the performance of *Res-*

	NON-BINARY	LITERAL	DUAL	HIDDEN VARIABLE	PLACE
CSP vs. *SAT*	$=$	$\geq$	$=$	$=$	$\geq$

Table 5.5: Comparing the number of solution of *SAT* to CSP encodings.

olution and *Consistency* based techniques on the various encodings. This is summarised in Table 5.6, where the row values indicate the level of ***NG-RES*** when applied to a CSP encoded *SAT* instance. I show that although there is a space overhead associated with DUAL encoded problems in comparison to using the LITERAL encoding, enforcing local-consistency on the resulting CSP does much more work. I also prove that enforcing strong-k-consistency on LITERAL encoded *3-SAT* instances does zero work if each clause has distinct literals, which explains the empirical results presented in the next chapter.

	NON-BINARY	LITERAL	DUAL	HIDDEN VARIABLE	PLACE
NG-RES	k(*-Resolution*)	k	$(j(k-1), j)$	k (NON-BINARY) k (LITERAL)	k (NON-BINARY) $(j(k-1), j)$ (DUAL)

Table 5.6: The level of ***NG-RES*** when enforcing strong-k-consistency on *j-SAT* to CSP encodings. Notice that we have to make a distinction between the amount of local-consistency achieved on two constituent component sets in the COMBINED encodings.

This analysis highlights several important questions. Since HIDDEN VARIABLE and PLACE encodings are combinations of DUAL, LITERAL and NON-BINARY encodings, what advantages do they bring? This I leave as an open question, but my theoretical analysis predicts that enforcing local-consistency on HIDDEN VARIABLE and PLACE encodings should never outperform at least one of the 'constituent' encodings from which they are constructed. Indeed, the extra space cost associated with the HIDDEN VARIABLE and PLACE encodings may even hinder algorithmic performance.

Since the DUAL encoding produces relatively large data-structures compared to the LITERAL encoding, it might be the case that enforcing *path*-consistency on DUAL encoded instances is currently too expensive. In the next chapter I provide empirical results of applying *path*-consistency to unsatisfiable *SAT* instances with up to 10,000 clauses. However Kask & Dechter (1995) showed that enforcing partial-*path*-consistency[2] can perform almost as well as full *path*-consistency, especially for problems with tight constraints. Hence, enforcing partial-*path*-consistency on DUAL encoded problems may address this complexity issue by significantly pruning the search-space for complete algorithms without the space/time overhead of full *path*-consistency.

I propose a simple heuristic for choosing encodings and algorithms for instances that have been predicted[3] and be satisfiable or not:

[2] A slight variant of *path*-consistency where *path*-consistency is performed on the sub-problem induced by a chosen subset of the variables that have the highest degree and are tightly grouped together.

[3] In Section 7.2.1 I discuss the idea of using feature analysis to correlate between instances and the performance of *SAT*-Solvers.

1. Predicted satisfiable instances: choose the LITERAL encoding and use a stochastic algorithm.

2. Predicted unsatisfiable instances: choose the DUAL encoding and run a branching algorithm.

In Case 1. it is ill-advised to enforce local-consistency, since my theoretical (and empirical) results suggest that it might not have any tangible pruning effect. However, it is strongly advised to enforce local-consistency in Case 2, since this might significantly (or entirely) prune the search-space in polynomial-time. In the next chapter I demonstrate the practical benefit of cross-fertilising techniques between the *SAT* and CSP research. I show that representing *SAT* instances as a particular type of CSP and enforcing a low-level of consistency can solve many *SAT* benchmarks.

Finally, I introduce the concept of Extended-*Consistency*. This extends the work of Tseitin, Baker and Mitchell, and provides a platform to communicate the wealth of theoretical results that have been discovered about the power of extended proof-systems in each of these two domains. Whilst the notion of Extended-*Consistency* is not a major contribution to the field per-se, this does highlight another benefit of bridging the fields. By introducing the concept of Extended-*Consistency* I provide a more complete picture of the *SAT* and CSP proof-systems. Relatively little practical work has been done on extended proof-systems. *SAT* and CSP algorithms that employ additional variables tend to apply them in an 'ad-hoc' manner. Extended-*Consistency* describes a technique to automate this process, and may allow synergies between these two extended proof-systems to be cultivated and explored.

Chapter 6

Empirical Analysis of *SAT* to CSP Encodings

Strictly speaking, the title of this thesis should be "The *Path* to Unsatisfaction", since the focus of this research is on how effective polynomial algorithms (encoding plus local-consistency enforcement) are at determining unsatisfiability. Moe specifically, this thesis is not concerned with the effectiveness of *SAT*-Sovlers prior to enforcing local-consistency. One of the main results of the previous chapter is that enforcing local-consistency on DUAL encoded problems does more work than on LITERAL encoded instances. Although this might not be totally surprising (since DUAL encoded problems are typically larger than LITERAL encoded) the empirical results on each of these encodings in this chapter are surprising. Not only do the empirical results strongly support the theoretical results in the previous chapter, but we find that enforcing a low-level of local consistency can solve what are regarded as 'hard' unsatisfiable *SAT* instance and can even compete with state-of-the-art *SAT*-Solvers.

Stochastic and branching algorithms are two families of algorithm adopted by the *SAT* community; branching algorithms can prove both satisfiability and unsatisfiability[1], whereas stochastic algorithms can only prove satisfiability. In Chapter 3 I suggested that local-consistency algorithms might be a viable approach to redress this balance. In this chapter I address the two experimental issues raised in Chapter 3 and provide empirical evidence to support the theoretical results presented in the previous chapter.

More specifically, I apply a well-known *Consistency*-based procedure (Algorithm 4) proposed by Freuder (1982) to many 'hard' unsatisfiable *SAT* instances encoded using the DUAL and LITERAL encodings. This algorithm incrementally enforces strong-1-, 2-, 3- $\ldots$ n-consistency until the problem is solved.

Ten years ago, enforcing *path*-consistency (i.e. strong-3-consistency in binary constraint networks) was extremely difficult owing to the availability of computer resources. Nowadays it is possible to enforce this level of consistency on problem instances that have over 15,000 clauses. The number of bits used to represent the DUAL encoding data-structure of a *k-SAT* is $2^{k^2} \times \binom{c}{2}$, which is $O(c^2)$, where c is

[1] Though it is well known that branching algorithms perform better on satisfiable instances than on unsatisfiable ones.

Algorithm 4 Incremental *k-CON* algorithm.

```
Given a CSP F
for k = 1 to n do
  if k-CON(F) ⊢ ⊥ then
    return ⊥
  end if
end for
return ⊤
```

the number of clauses.

The procedure used to enforce *path*-consistency is described by Algorithm 5 and has a worst time complexity of $O(c^4)$, where c is the number of clauses. Notice that although this procedure is more expensive than some of the *path*-consistency procedures it is extremely simple and does not use any of the heuristics or queuing mechanisms described in the more recent *Consistency* algorithms (see Tsang (1993)). One reason for this is that since the *SAT* instances are in 3CNF I can utilise bitwise calculations of blocks of bytes; each representing the seven edges that can connect each node to each other variable. Recursing over each variable and determining whether a bit (an edge $\{i, j\}$) should be 0 or 1 is calculated by simply intersecting three bytes and checking whether the result is non-zero:

$$if((i_v \cap j_v) \cap v) = 0 : \{i, j\} \rightarrow 0; \{i, j\} \rightarrow 1$$

where i_v and j_v represent the edges in node-set v to which the nodes i and j are connected. For *3-SAT* instances i_v, j_v and v are all bytes.[2] This process is repeated until no more changes are made to the data-structure, i.e. that no more edges are pruned.

Table 6.1 represents the memory (in GigaBytes) required to run these experiments to enforce strong-k-consistency on DUAL encoded *3-SAT* instances with a varying number of clauses. The values for the number of clauses in the table have not been chosen arbitrarily, they correspond to the level of local-consistency requiring 1GB of memory (along the diagonal from right to left). For instance, row 3: given 36 clauses, enforcing strong-6-consistency requires approximately 1GB of memory. All experiments presenting in the chapter were performed on a DELL PRECISION 470 DUAL CORE ZEON 5.6GHZ with 3GB RAM and the units of time are seconds.

These experiments originally focused on applying the incremental local-consistency algorithm on randomly generated *SAT* instances and SATLIB benchmarks (see Section 3.2.1.1). The aim was to compare the effect of enforcing local-consistency on problems encoded using both the LITERAL and DUAL encodings. Recall that both of these encodings are classed as CONSTRAINT mappings. Although the LITERAL encoding creates graph instances with $k^2 \times \binom{c}{2}$ nodes - compared to $2^{k^2} \times \binom{c}{2}$ for the DUAL

[2]The simplicity of this algorithm is one of its main strengths. Indeed it may be possible to implement in hardware and utilise parallel bit processing, which may also improve its performance.

	enforcing the level of strong-k-consistency					
clauses	**3**	**4**	**5**	**6**	**7**	**8**
15	7.4E-07	2.2E-05	4.7E-04	7.2E-03	8.4E-02	**7.6E-01**
21	1.5E-06	6.5E-05	2.1E-03	4.9E-02	**9.1E-01**	1.4E+01
36	4.4E-06	3.5E-04	2.0E-02	**9.1E-01**	3.3E+01	9.8E+02
92	2.9E-05	6.2E-03	**9.6E-01**	1.2E+02	1.2E+04	1.0E+06
497	8.6E-04	**1.0E+00**	8.6E+02	5.9E+05	3.4E+08	1.7E+11
16903	**1.0E+00**	3.9E+04	3.9E+04	2.8E+13	5.4E+17	9.2E+21

Table 6.1: Memory (GB) required to enforce strong-k-consistency on DUAL encoded *SAT* problems.

Algorithm 5 *strong-3-CON* algorithm.

```
Given a DUAL encoded CSP constraint graph F
while changed(F) do
  for each edge {i, j} do
    for each set of nodes represented by variable v do
      if v is empty then
        return UNSAT
      end if
      if (i,j) cannot form a triangle with a domain value in v then
        remove (i,j)
      end if
    end for
  end for
end while
return UNKNOWN
```

graph - the *edge-density*[3] of the DUAL graph can be much less than that of the LITERAL graph (for satisfiable instances; unsatisfiable instance have no solutions in either graph). In Chapter 4 I proved that enforcing strong-k-consistency does at least as much work on DUAL encoded instances as it does when using the LITERAL encoding. In addition, I proved that enforcing strong-k-consistency on *k-SAT* instances (encoded using the LITERAL encoding) does zero work when the literals in each clause are distinct. It should be noted that several of the SATLIB benchmark families are in 'strict' 3CNF format (UUF, AIM and DUBOIS), to it is expected that enforcing *path*-consistency on these LITERAL encoded instances will have no effect. However, several benchmark families are constructed from industrial problems, and have binary and even unary clauses. In these cases enforcing *path*-consistency on LITERAL encoded instances will have some effect.

6.0.1 DIMACS CNF Format

There exists a widely used format for *SAT* instances: the `cnf` format from the Second DIMACS Benchmark Challenge. The DIMACS (1993) `cnf` format (described in detail in Appendix A.1) is currently accepted by almost all of the best-performing *SAT*-Solvers and tools. It is simple to parse and generate,

[3] the number of binary edges over the number of possible binary edges.

reasonably concise and flexible, portable across different platforms, and human-readable. Therefore, all benchmark instances in this thesis are in this format.

To utilise the bitwise calculations of my strong-3-consistency algorithm it is necessary only to work with *3-SAT* instances. Since the DIMACS CNF format does not strictly enforce 3CNF, I converted the `cnf` files into 3CNF using De Morgan's Laws. These instances have been converted to *3-SAT*, so in many cases the numbers of *propositions* and *clauses* are larger than they were in the original. The conversion rules for clauses with n literals are shown in Table 6.2.

# lits	**original**	**new**
one	(x_1)	$(x_1 \vee x_1 \vee x_1)$
two	$(x_1 \vee x_2)$	$(x_1 \vee x_2 \vee x_2)$
four	$(x_1 \vee x_2 \vee x_3 \vee x_4)$	$(x_1 \vee x_2 \vee y_1) \wedge (-y_1 \vee x_3 \vee x_4)$
five	$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5)$	$(x_1 \vee x_2 \vee y_1) \wedge (-y_1 \vee x_3 \vee y_2) \wedge (-y_2 \vee x_4 \vee x_5)$
≥ 6	$(x_1 \vee x_2 \vee \cdots \vee x_{n-1} \vee x_n)$	$(x_1 \vee x_2 \vee y_1) \wedge (-y_1 \vee \cdots \vee x_{n-1} \vee x_n)$

Table 6.2: Rules to convert a general CNF formula into 3CNF.

6.0.2 UUF

Uniform Random-*3-SAT* is a family of *SAT* problems obtained by randomly generating 3CNF formulae then determining their satisfiability by using a complete *SAT*-Solver. These were generated in the following way:

- For an instance with n variables and c clauses, each of the c clauses is constructed from 3 literals, which are randomly drawn from the $2n$ possible literals such that each possible literal is selected with the same probability.

- Clauses are not accepted for the construction of the problem instance if they contain multiple copies of the same literal or if they are tautological (i.e. they contain a variable and its negation).

Uniform Random-*3-SAT* is the union of these distributions over all n and c.

These instances are sampled from the 'hard' region of the phase transition, the complete set is shown in Table 6.3. Empirical analysis, by Yokoo (1997) and Crawford & Auton (1996), shows that problem instances from the phase transition region of uniform Random-*3-SAT* tend to be particularly hard for both tree-based and stochastic *SAT*-Solvers.

Table 6.4 shows the results of applying *path*-consistency to many DUAL encoded UUF instances. Note that instances were solved from the uuf50-218, uuf75-325 and uuf100-430 groups. All 1000 uuf50-218 instances were solved by enforcing *path*-consistency, as well as 55 from the 100 uuf75-325 instances. The remaining 45 uuf75-325 instances were solved using strong-4-consistency (shown in Table 6.5), but

test-set	instances	clause-len	vars	clauses
uuf50-218	1000	3	50	218
uuf75-325	100	3	75	325
uuf100-430	1000	3	100	430
uuf125-538	100	3	125	538
uuf150-645	100	3	150	645
uuf175-753	100	3	175	753
uuf200-860	100	3	200	860
uuf225-960	100	3	225	960
uuf250-1065	100	3	250	1065

Table 6.3: Uniform Random-*3-SAT* unsatisfiable instances.

unfortunately uuf100-430 were too large to enforce this level of consistency. When the instances are encoded using the LITERAL encoding the *path*-consistency algorithm failed to have any effect, which is not surprising given Theorem 5.2.4, and that these are 'strict' 3CNF instances.

benchmark set	inst's	ran	solves	props	clauses	avg. time (s)
uuf50-218	1000	1000	1000	50	218	1.16
uuf75-325	100	100	55	75	325	25.05
uuf100-430	1000	1000	4	100	430	120.45
uuf125-538	100	100	0	125	538	378.34
uuf150-645	100	100	0	150	645	783.98
uuf175-753	100	100	0	175	753	1034.45

Table 6.4: Enforcing *path*-consistency on DUAL encoded UUF benchmarks. Time is in seconds.

benchmark set	inst's	ran	solves	props	clauses	avg. time
uuf75-325 (unsolved)	45	45	45	75	325	4766.51

Table 6.5: Enforcing 4-consistency on unsolved DUAL encoded UUF benchmarks.

6.0.3 DUBOIS

The DUBOIS instances are considered the easiest unsatisfiable SATLIB family. However, LITERAL encoded instances from this family cannot be solved using *path*-consistency, though they can all easily be solved when represented using the DUAL encoding (as shown in Table 6.6).

Out of the 23 *SAT*-Solvers submitted to *SAT*-Competition 2003 this algorithm would have ranked in the top 10 on this benchmark set, with twelve *SAT*-Solvers failing to solve every instance.

benchmark set	inst's	ran	solves	props	clauses	avg. time
dubois20	1	1	1	60	160	0.24
dubois21	1	1	1	63	168	0.27
dubois22	1	1	1	66	176	0.31
dubois23	1	1	1	69	184	0.34
dubois24	1	1	1	72	192	0.39
dubois25	1	1	1	75	200	0.44
dubois26	1	1	1	78	208	0.49
dubois27	1	1	1	81	216	0.56
dubois28	1	1	1	84	224	0.61
dubois29	1	1	1	87	232	0.67
dubois30	1	1	1	90	240	0.75
dubois50	1	1	1	150	400	3.75
dubois100	1	1	1	300	800	30.2

Table 6.6: Enforcing *path*-consistency on DUAL encoded DUBOIS benchmarks.

6.0.4 AIM

The AIM instances are all generated with a particular Random-*3-SAT* instance generator described by Asahiro et al. (1993). It should be noted that all of these instances can be solved with polynomial pre-processing, therefore, they cannot be considered as intrinsically hard. However, some stochastic search algorithms, such as WALKSAT and GSAT variants, show a very poor performance on the instances with low clause/variable ratio; in particular for ratios of 1.6 and 2.0.

All instances are easily solvable by *path*-consistency when transformed to a CSP using the DUAL encoding (see Table 6.7), yet when they are LITERAL encoded *path*-consistency fails to solve a single instance.

benchmark set	inst's	ran	solves	props	clauses	avg. time
aim-50-1_6-no	4	4	4	50	80	0.032
aim-50-2_0-no	4	4	4	50	100	0.062
aim-100-1_6-no	4	4	4	100	160	0.27
aim-100-2_0-no	4	4	4	100	200	0.40
aim-200-1_6-no	4	4	4	200	320	1.85
aim-200-2_0-no	4	4	4	200	400	3.00

Table 6.7: Enforcing *path*-consistency on DUAL encoded AIM benchmarks.

6.0.5 JNH

JNH instance are generated randomly in the following way:

benchmark set	inst's	ran	solves	props	clauses	avg. time
jnh10	1	1	1	1796	2546	108.02
jnh11	1	1	1	1749	2499	305.20
jnh13	1	1	1	1763	2513	126.41
jnh14	1	1	1	1780	2530	142.94
jnh15	1	1	1	1759	2509	168.14
jnh16	1	1	0	1777	2527	10,436.73
jnh18	1	1	1	1794	2544	662.94
jnh19	1	1	1	1759	2509	239.71
jnh2	1	1	1	1819	2569	83.19
jnh20	1	1	1	1786	2536	159.87
jnh202	1	1	1	1733	2433	115.57
jnh203	1	1	1	1683	2383	314.92
jnh206	1	1	1	1678	2378	446.45
jnh208	1	1	1	1686	2386	308.70
jnh211	1	1	1	1672	2372	102.17
jnh214	1	1	1	1667	2367	271.44
jnh215	1	1	1	1665	2365	187.97
jnh216	1	1	1	1669	2369	307.02
jnh219	1	1	1	1665	2365	480.90
jnh3	1	1	1	1797	2547	436.97
jnh302	1	1	1	1924	2724	82.09
jnh303	1	1	1	1859	2659	322.20
jnh304	1	1	1	1887	2687	118.10
jnh305	1	1	1	1895	2695	130.85
jnh306	1	1	1	1887	2687	1,702.63
jnh307	1	1	1	1854	2654	72.80
jnh308	1	1	1	1877	2677	273.41
jnh309	1	1	1	1892	2692	124.30
jnh310	1	1	1	1854	2654	57.36
jnh4	1	1	1	1770	2520	178.89
jnh5	1	1	1	1797	2547	166.14
jnh6	1	1	1	1770	2520	462.83
jnh8	1	1	1	1780	2530	113.17
jnh9	1	1	1	1766	2516	117.78

Table 6.8: Enforcing *path*-consistency on DUAL encoded JNH benchmarks.

- For an instance with n variables and c clauses, each of the c clauses is constructed by including a variable with a fixed probability p, and then negating the variable with probability 0.5.
- Empty clauses and unit clauses are rejected in the generation process.

Empirical evidence provided by Mitchell & Levesque (1996) indicate that instances generated by this model tend to be much easier to solve than the Uniform Random-*3-SAT* (UUF) instances described earlier.

There are 34 unsatisfiable JNH benchmarks, 33 of which can be solved using *path*-consistency on the DUAL encoding, and none using *path*-consistency on the LITERAL encoding (as shown in Table 6.8). The 34th unsolvable instance (jnh16) is too large to enforce 4-consistency, and it is unknown as to why this particular instance was not solved. It is also unknown as to why instance jnh306 took significantly longer to solve than the other instances, though, compared to the others, it took several more iterations through the data-structure before the algorithm was able to determine unsatisfiable

6.0.6 BF and SSA

benchmark set	inst's	ran	solves	props	clauses	avg. time
ssa0432-003	1	1	1	504	1096	21.32
ssa2670-130	1	1	1	1583	3545	1309.45
ssa2670-141	1	1	1	1129	2458	503.51
ssa6288-047	1	0	0	10410	34238	-

Table 6.9: Enforcing *path*-consistency on DUAL encoded SSA benchmarks.

The BF and SSA instances are selected formulae from those generated by a circuit-fault-analysis test-pattern generation program called *Nemesis*. These formulae are in CNF and contain clauses of lengths 1-6, with more than half the clauses of size 2. The instances are of rather large size (most of them have more than 1000 variables), though some of the instances contain unit clauses and hence can be simplified by *unit propagation*. According to the benchmark description, of the unsatisfiable instances only instances ssa2670-130 and ssa2670-141 may be considered hard. However, when encoded using the DUAL encoding all instances can be solved by enforcing *path*-consistency. The results are shown in Tables 6.9 and 6.10. Unfortunately, ssa6288-047 was too large to enforce *path*-consistency, since the DUAL encoding would require approximately 600MB of RAM to represent the data-structure.

6.0.7 PRET

The PRET instances are an encoding of GRAPH-2-COLOURABILITY, along with a parity constraint to force unsatisfiability. According to SATLIB, these instances are considered amongst the hardest unsatisfiable benchmarks. In fact, as shown in Table 6.11, solving these instances required enforcing strong-5-consistency using the DUAL encoding. Thirteen of the 23 *SAT*-Solvers submitted to *SAT*-Competition

benchmark set	inst's	ran	solves	props	clauses	avg. time
bf0432-007	1	1	1	1417	4045	2675.78
bf1355-075	1	1	1	2706	7304	4949.39
bf1355-638	1	1	1	2701	7292	3908.81
bf2670-001	1	1	1	1625	3666	550.36

Table 6.10: Enforcing *path*-consistency on DUAL encoded BF benchmarks.

benchmark set	inst's	ran	solves	props	clauses
pret60_25	1	1	1	60	160
pret60_40	1	1	1	60	160
pret60_60	1	1	1	60	160
pret60_75	1	1	1	60	160
pret150_25	1	1	1	150	400
pret150_40	1	1	1	150	400
pret150_60	1	1	1	150	400
pret150_75	1	1	1	150	400

Table 6.11: Enforcing strong-5-consistency on DUAL encoded PRET benchmarks.

2003 failed to solve every PRET benchmark, meaning that they reached the time-out limit enforced by the competition.

6.0.8 Pigeon Hole

As discussed in Chapter 3, the PIGEON-HOLE PROBLEM is probably one of the most important and well-studied class of problems. It asks whether it is possible to place $n + 1$ pigeons in n holes without two pigeons being in the same hole.

The *SAT* encoding of this problem (also referred to as PHOLE) is very straightforward and although there are various different mappings, these instance were constructed from the original CSP definition to *SAT* using the DIRECT encoding in the following way:

- For each pigeon p we have a variable x_{ph} meaning that pigeon p is placed in hole h.
- $n + 1$ clauses say that a pigeon has to be placed in some hole.
- For each hole a set of clauses ensure that only one single pigeon is placed into that hole.

There are five instances available, which encode the PIGEON-HOLE PROBLEM for six to ten holes (and therefore seven to eleven pigeons); details on the instances are given in Table 6.12.

Haken (1985) proved the first exponential bounds for *Resolution*, showing that a *Resolution refutation* would require an exponential number of clauses to be generated to solve this problem. To solve these

instance	holes	vars	clauses	satisfiable?
hole6	6	42	133	no
hole7	7	56	204	no
hole8	8	72	297	no
hole9	9	90	415	no
hole10	10	110	561	no

Table 6.12: The Pigeon-Hole instances.

instance would require strong-p-consistency, where p is the number of pigeons, and it is not surprising that *path*-consistency failed to solve a single instance.

6.1 *SAT* Competition Benchmarks

Given the unexpected success of enforcing such a low-level of consistency on DUAL encoded SATLIB problems I sought harder benchmarks to which to apply this algorithm. The obvious place to find difficult *SAT* instances is from the *SAT* Competition. Although the majority of these benchmarks are from Formal Verification applications and tend to be very large, some were small enough to represent using the my DUAL encoding data-structure. Since the space complexity is $O(n^2)$, instances with above $n = 10,000$ clauses require over 700MBytes of RAM. Many of these benchmarks are therefore untestable on a standard computer. For larger benchmarks I recommend that sparse-matrix techniques are used to represent the data-structure, which may help address this space-complexity issue. Since these are competition benchmarks, relatively little information is given about their construction. Tables 6.13-6.18 show how enforcing *path*-consistency performs on the following *SAT*-Competition benchmark families encoded using the DUAL encoding.

- BMC1, Table 6.13
- GRAPHCOLORS3, Table 6.14
- BARREL, Table 6.15
- CMPADD, Table 6.16
- DINPHIL, Table 6.17
- LONGMULT, Table 6.18

The success of enforcing a local-level of consistency on DUAL encoded unsatisfiable *SAT* instances is very interesting. By comparing these results with the *SAT*-Solvers submitted to the *SAT* Competition 2003, we see that this basic constraints-inspired polynomial-time algorithm can compete favourably with

benchmark set	inst's	ran	solves	props	clauses	avg. time	k-con
goldberg/bmc1/23.shuffled	1	1	1	217	493	1.50	3
goldberg/bmc1/4.shuffled	1	1	1	217	493	1.53	3
goldberg/bmc1/42.shuffled	1	1	1	417	943	9.69	3
goldberg/bmc1/61.shuffled	1	1	1	417	943	9.81	3

Table 6.13: Enforcing *path*-consistency on BMC1 *SAT* Competition benchmarks.

benchmark set	inst's	ran	solves	props	clauses	avg. time	k-con
graphcolors3/3col20	20	20	20	190	326	0.38	3
graphcolors3/3col40	20	20	20	380	646	4.87	3
graphcolors3/3col60	20	20	20	570	966	35.6	3

Table 6.14: Enforcing *path*-consistency on GRAPHCOLORS3 *SAT* Competition benchmarks.

benchmark set	inst's	ran	solves	props	clauses	avg. time	k-con
barrel2	1	1	1	58	167	0.11	3
barrel3	1	1	1	307	974	20.49	3
barrel4	1	1	1	654	2111	172.313	3
barrel5	1	1	1	1703	5679	3364.344	3
barrel6	1	1	1	2806	9431	14977.031	3
barrel7	1	0	0	4303	14545	-	-
barrel8	1	0	0	6254	21231	-	-

Table 6.15: Enforcing *path*-consistency on BARREL *SAT* Competition benchmarks.

benchmark set	inst's	ran	solves	props	clauses	avg. time	k-con
biere/cmpadd/ca002.shuffled	1	1	1	26	70	0.00	3
biere/cmpadd/ca004.shuffled	1	1	1	60	168	0.06	3
biere/cmpadd/ca008.shuffled	1	1	1	130	370	1.23	3
biere/cmpadd/ca016.shuffled	1	1	1	272	780	15.77	3
biere/cmpadd/ca032.shuffled	1	1	1	558	1606	195.83	3
biere/cmpadd/ca064.shuffled	1	1	1	1132	3264	1936.453	3

Table 6.16: Enforcing *path*-consistency on CMPADD *SAT* Competition benchmarks.

highly optimised and efficient **DLL**-based *SAT*-Solvers. Moreover, these empirical results suggest that many of the SATLIB and Competition benchmark families are tractable, that is, they can be solved by enforcing a constant level of local-consistency.

A list of unsatisfiable benchmark families that enforcing *path*-consistency fails to solve are shown in Table 6.19.

benchmark set	inst's	ran	solves	props	clauses	avg. time	k-con
biere/dinphil/dp02u01.shuffled	1	1	1	213	376	0.28	3
biere/dinphil/dp03u02.shuffled	1	1	1	478	1007	10.16	3
biere/dinphil/dp04u03.shuffled	1	1	1	1018	2412	345.076	3
biere/dinphil/dp05u04.shuffled	1	1	1	1573	3904	1485.036	3

Table 6.17: Enforcing *path*-consistency on DINPHIL *SAT* Competition benchmarks.

benchmark set	inst's	ran	solves	props	clauses	avg. time	k-con
longmult0	1	1	1	558	1327	8.94	3
longmult1	1	1	1	1035	2579	249.938	3
longmult2	1	1	1	1535	3896	1082.625	3
longmult3	1	1	0	2051	5263	7122.125	3
longmult4	1	0	0	2587	6690	-	-
longmult5	1	0	0	3143	8177	-	-
longmult6	1	0	0	3719	9724	-	-
longmult7	1	0	0	4315	11331	-	-
longmult8	1	0	0	4931	12998	-	-
longmult9	1	0	0	5567	14725	-	-
longmult10	1	0	0	6223	16512	-	-
longmult11	1	0	0	6899	18359	-	-
longmult12	1	0	0	7595	20266	-	-
longmult13	1	0	0	8311	22233	-	-
longmult14	1	0	0	9047	24260	-	-

Table 6.18: Enforcing *path*-consistency on LONGMULT *SAT* Competition benchmarks.

benchmark set	inst's	ran	solves
crafted/jarvisalo05/mod2-rand3bip-unsat/	15	15	0
crafted/jarvisalo05/mod2c-rand3bip-unsat/	15	15	0
crafted/jarvisalo05/mod2c-3cage-unsat/	6	6	0
crafted/jarvisalo05/mod2-3cage-unsat/	24	24	0
crafted/sabharwal05/counting/fphp/unsat/	28	28	0
crafted/sabharwal05/counting/php/unsat/	28	28	0

Table 6.19: SATLIB Benchmark families *path*-consistency fails to solve.

The MOD2 benchmark families, created by Haanpaa et al. (2005), are very small but hard instances based on linear equations modulo 2, where the underlying structure is derived from 3-regular graphs:

- mod2-rand3bip-unsat: Unsatisfiable instances with the number of variables v = 90, 105, 120, 135,

150. For each v there are 15 instances. The underlying graphs are random 3-regular bipartite graphs.

- mod2c-rand3bip-unsat: As mod2-rand3bip-unsat but with "simple camouflage" applied on the instances.

- mod2-3cage-unsat: The underlying graphs are (3, g)-cages with girth $g \in \{9, 10, 11, 12\}$. 18 (3, 9)-cages, 3 (3, 10)-cages, 1 (3, 11)-cages, and 1 (3, 12)-cage exist.

- mod2c-3cage-unsat: As mod2-3cage-unsat but with "simple camouflage" applied on the instances.

6.1.1 Competition Comparison

I have shown that enforcing such a low-level of consistency can determine the unsatisfiability of many DUAL encoded *SAT* instances. It appears from a general comparison with the 23 *SAT*-Solvers submitted to *SAT* Competition 2003 that this polynomial-time algorithm is competitive on many of the unsatisfiable DUAL encoded benchmarks. Table 6.20 provides an overview of how enforcing *path*-consistency in this way compares to *SAT*-Solvers on many SATLIB and *SAT* Competition 2003 instances. The left column lists the *SAT*-Solver, with each row describing how many instances of the benchmark families were solved which are listed at the top. The **tested** row shows the number of instance that were tested in my experiments and the ***path*-consistency** row shows the number of those instances that *path*-consistency solves[4].

Comparing the execution time of this polynomial-time algorithm with the competition *SAT*-Solvers is not interesting since they have ran in different architectures and my code was not designed to test such large instances. However, out of the 276 benchmark-algorithm combinations this simple polynomial-time algorithm outperformed 70 cases and was as successful as 96 cases; a total of 166 out of 276. Notice that ZCHAFF solved every instance correctly.

6.2 Chapter Summary and Discussion

This investigation began by comparing the performance of enforcing a local-level of consistency on LITERAL and DUAL encoded *SAT* instances. I found that enforcing *path*-consistency on the LITERAL encoding does not solve any of the *SAT* benchmarks. This is in stark contrast to problems encoded using the DUAL encoding, which took these experiments into a new direction by showing that enforcing such a low-level of consistency can not only effectively prune the problem, but also solve a surprising number of what are considered 'hard' *SAT* benchmarks.

We know from the theoretical results presented in Chapter 5 that:

[4]Note that all of the PRET instances required strong-5-consistency to determine unsatisfiability, and that they were small enough to enforce this level of consistency.

family	**aim-100**	**aim-200**	**aim-50**	**barrel**	**bf**	**dubois**	**phole**	**jnh**	**longmult**	**pret-150**	**pret-60**	**ssa**
# instances	8	8	8	8	4	13	5	34	16	4	4	4
tested	8	8	8	6	4	13	5	34	4	4	4	3
***path*-consistency**	8	8	8	6	4	13	0	33	3	4	4	3
asat	8	0	8	7	1	9	5	34	15	0	4	1
calcres	4	0	8	3	2	13	1	0	4	0	4	4
csat	8	8	8	5	2	0	5	34	16	0	0	4
dr	0	0	5	1	0	13	1	0	2	4	4	2
eqsatz	8	8	8	8	4	13	5	34	16	4	4	4
heerhugo	8	8	8	7	4	13	3	34	7	4	4	4
modoc	8	8	8	7	4	7	5	34	4	0	4	4
modoc-2.0	8	8	8	7	4	5	4	34	5	0	4	4
nsat	8	8	8	0	3	13	4	34	16	4	4	2
ntab	5	0	5	5	1	8	0	34	6	0	4	2
ntab back	5	0	5	5	3	7	0	34	7	0	4	3
ntab back2	5	3	5	5	3	7	0	34	7	0	4	3
posit	8	0	8	3	2	8	5	34	15	0	4	3
relsat	8	8	8	5	4	13	5	34	9	4	4	4
relsat-200	8	8	8	7	4	13	5	34	16	4	4	4
sat-grasp	8	8	8	4	4	13	4	34	9	4	4	4
sato	8	8	8	7	4	13	5	34	15	4	4	4
sato-3.2.1	8	8	8	7	4	13	5	34	16	4	4	4
satz	8	8	8	6	4	11	5	34	12	0	4	4
satz-213	8	8	8	7	4	9	5	34	16	0	4	4
satz-215	8	8	8	8	4	9	5	34	16	0	4	4
zchaff	8	8	8	8	4	13	5	34	16	4	4	4
zres	4	0	8	3	3	13	3	0	4	4	4	3

Table 6.20: The number of DUAL encoded SATLIB benchmarks solved by enforcing a local-level of consistency compared to the number solved by competition entrants.

1. Enforcing strong-k-consistency on LITERAL encoded 'strict' *k-SAT* instances has no pruning effect.

2. Enforcing strong-k-consistency on DUAL encoded *j-SAT* instances is equivalent to performing $(j(k-1), j)$-***NG-RES***.

Given this, it is not surprising that the empirical tests demonstrated that enforcing *path*-consistency

on LITERAL encoded *3-SAT* instances had very little impact on pruning the search-space and failed to solve a single instance. In fact, it was observed that in some cases a little pruning did take place on LITERAL encoded instances, but (as predicted) only when the level of consistency being enforced was greater than the number of literals in some of the clauses.

Recall from Chapters 2 and 3 that there are two families of algorithm used, stochastic and branching. Branching algorithms can prove both satisfiability and unsatisfiability (though proving unsatisfiability is typically more difficult[5]), whereas stochastic algorithms can only prove satisfiability. Although local-consistency algorithms are unable to determine whether an instance is satisfiable, they are able to prove unsatisfiability. The development of sophisticated local-consistency algorithms might thus be a viable approach to redress the 'algorithmic' bias.

The empirical results presented in this chapter strengthen the theoretical results that enforcing a local-level of consistency on unsatisfiable DUAL encoded problems does more work when LITERAL encoded. In Chapter 3 I mentioned that Prestwich (2003) had established a correlation between *solution-density* and problem solubility. Since these experiments are primarily concerned with the effect of local-consistency algorithms on CSP encoded unsatisfiable *SAT* instances, the *solution-density* is the same (zero) in every case. Therefore, *solution-density* cannot be a factor in understanding what conditions are necessary for local-consistency to determine unsatisfiability. As with previous research I could find no definable characteristic that distinguished unsatisfiable instances that were solvable versus ones that were not. The question of whether we can characterise instances (and families) that are 'polynomially solvable' is open. Such analysis is left as future work, but a specific pointer to future work would be to look for the characteristics of the solved benchmark families that made them 'polynomially solvable' and extrapolate (hopefully new) generic features that we can use to test for tractability.

As described in Section 3.5, in recent years there has been a large amount of research devoted to analysing the effect of preprocessing *SAT* instances prior to applying a branching or stochastic algorithms. Whilst these resolution-based preprocessors have been proven (on average) to improve problem solubility, it is still unclear as to what type of preprocessing should be applied, when it should be applied and to what part of the problem. Preprocessors such as HYPRE and SATELITE are now routinely employed by modern *SAT*-Solvers, however most research is still unconcerned with how the problem is encoded. The main result of this thesis is the remarkable success of enforcing a local-level of consistency on DUAL encoded unsatisfiable *SAT* instances. What is also important to highlight is that although this technique managed totally to prune the search-space of instances from a variety of problem domains, it is not yet a viable practical technique for two reasons:

1. DUAL encoded instances typically require much more memory than LITERAL or NON-BINARY encoded instances.

[5] Verbal communication with members of Princeton's *SAT* Research Group.

2. It is still computationally expensive and time consuming to enforce larger levels of local-consistency.

In Chapter 7 I discuss how we might overcome some of these issues. The purpose of this chapter, however, is not to present a comprehensive solution, but to show the potential that lies in encoding instances differently. One stand-out result of preprocessing research is that despite the preprocessing overhead and number of additional constraints and variables produced, in many cases the *SAT*-Solver could still find solutions in less time. Such results are encouraging, I propose that developing refined versions of this method is a sensible approach for researchers wishing to explore practical preprocessing techniques. A natural extension of this research is to explore practical implementations of these techniques as well as to explore effect that the resulting encoded/preprocessed instances have on the effectiveness of *SAT*-Solvers on both unsatisfiable and satisfiable instances.

Chapter 7

Discussion, Exploitation and Future Work

In this thesis I show how we might use Graph Theory as a framework to bridge *SAT* and CSP research. I hinted at the benefits of this approach by demonstrating that even the most basic of polynomial-time *Consistency* algorithms on DUAL encoded instances can be used to solve a variety of problems that might be encountered in practice. I suggested that using more sophisticated constraint-based algorithms and techniques than the ones used in my experiments could highlight tractable *SAT* benchmark families and problems, yielding many of the practical benefits described throughout this thesis. Whilst research into the *structure* of these problems has started to established itself in recent years relatively little research has begun to exploit these findings either to gain a deeper understanding of the complexity of problems, or to develop better performing algorithms and tools to solve them. Future research will aim to utilise these findings in order to develop new techniques to solve a larger set of problems.

This work relates to a much wider programme which aims to contribute to bridging the fields of Constraint Satisfaction and Propositional Satisfiability. More specifically, the work in this thesis explores the *structural* relationship between CSP and *SAT* problems and their algorithmic methods by viewing them in a graph-theoretic framework. Using this framework we are able to identify synergies between these two research domains. Ten years ago Selman et al. (1997) proposed "Ten Challenges in Propositional Reasoning and Search". Although many of these challenges have been met, several still remain open. This research addresses three in particular:

1. **CHALLENGE 1**: Prove that a hard 700 variable random *3-SAT* instance is unsatisfiable.
2. **CHALLENGE 3**: Demonstrate that a propositional proof-system more powerful than Resolution can be made practical for satisfiability testing.
3. **CHALLENGE 8**: Characterise the computational properties of different encodings of a real-world problem domain, and/or give general principles that hold over a range of domains.

In each section below I summarise this research and propose how it relates to Selman's 'challenges'. I discuss how the findings of this research could be used to address several prominent issues with modern

SAT-Solvers and have wider implications in the field.

7.1 Theoretical Studies

7.1.1 Categorising Encodings

After introducing the necessary background in Chapter 2, in Chapter 3 I provided a detailed survey of CSP and *SAT* encodings, algorithmic techniques and proof-systems. After providing a detailed example-led survey of *SAT* and CSP encodings, in Chapter 4 I show how all of the encodings can be categorised as one of three types of mapping. I introduce the new INVERSE encoding (Section 4.2), the only member of the CONSTRAINT mapping group, and in this chapter I also defined the DOMAIN and COMBINED mappings. The CSP to *SAT* encodings described in Chapter 3 are categorised and shown in Table 7.1. By using an example, I demonstrate that the INVERSE encoding is preferable when the number of partial satisfying solutions to the constraints is small, since each partial satisfying solution gets mapped to a *SAT* variable. In fact, given certain kinds of constraints the INVERSE encodings can be several orders of magnitude smaller than DIRECT encodings. Thus, careful consideration should be taken when deciding which encoding is best to use, and using the result of simple calculation (the encoding space complexity) researchers might find that they can now practically represent problems in CNF that were previously incredibly difficult.

The new INVERSE encoding (defined in Chapter 4) is the only member of the CONSTRAINT category. As mentioned, with the introduction of the INVERSE encoding it is now possible to define COMBINED mappings for CSP to *SAT* encodings (listed in Table 7.1) and explore the potential benefits that these new encodings may bring. Analysis of these new encodings I leave as open questions. I aim to cultivate this important link between *SAT* and CSP further and perform a comprehensive study on how expressing a wide variety of real problems - using various encodings - affects the performance of polynomial-time algorithms as well as state-of-the-art *SAT*-Solvers.

DOMAIN	CONSTRAINT	COMBINED
DIRECT	INVERSE	INVERSE + DIRECT
SUPPORT		INVERSE + SUPPORT
LOG		INVERSE + LOG
MULTIVALUED		INVERSE + MULTIVALUED

Table 7.1: Categorising the CSP to *SAT* encodings.

Similarly, Table 7.2 shows the *SAT* to CSP encodings described in Chapter 3 categorised as DOMAIN, CONSTRAINT, and COMBINED mappings.

Recently I have been developing the notion of HYBRID mappings, which are constructed using a

DOMAIN	CONSTRAINT	COMBINED
NON-BINARY	LITERAL	PLACE
	DUAL	HIDDEN VARIABLE

Table 7.2: Categorising the *SAT* to CSP encodings.

mixture of DOMAIN and CONSTRAINT mappings. In this case one can choose how individual constraints are encoded. For example, it might be more compact to encode *Alldiff* constraints using the DOMAIN mapping, yet encode *not-equals* constraints using CONSTRAINT mappings. A *hybrid* approach might yield encodings that are more compact and might potentially improve the effectiveness of preprocessing and search-based algorithms. However, this is still in the concept stage and a great deal of work is required to determine whether there are any tangible benefits to using this approach.

7.1.2 Characterising Encodings

In Chapter 5 I prove that these encodings can be characterised in accordance with the number of solutions they encapsulate, that the different *SAT* to CSP encodings can produce *micro-structures* that have different levels of *solution-density*. I prove that transforming *SAT* instances into CSP using the LITERAL encoding produces problems that can have a higher proportion of solutions than the DUAL and NON-BINARY encodings, for instance. These results are summarised in Table 7.3. This theoretical insight might be practically useful in helping guide which encoding should be chosen for a particular problem instance, as the literature shows that *solution-density* can affect the solubility of a problem, particularly for stochastic algorithms.

	NON-BINARY	**LITERAL**	**DUAL**	**HIDDEN VARIABLE**	**PLACE**
CSP vs. *SAT*	$=$	$\geq$	$=$	$=$	$\geq$

Table 7.3: Comparing the number of solution of *SAT* to CSP encodings. If we consider X vs. Y. $X = Y$ denotes that X and Y have an equal amount of solutions, and $X \geq Y$ denotes that X has at least the same number of solution tuples as Y.

> **CHALLENGE 8:** Characterise the computational properties of different encodings of a real-world problem domain, and/or give general principles that hold over a range of domains.

The work presented in Chapter 5 directly addresses this challenge. Having characterised the encodings according to their *solution-density*, the second part of this chapter addresses the gap in theoretical research identified in Chapter 3. I compare the performance of *Resolution* and *Consistency* based techniques on the various encodings (summarised in Table 7.4) and show that enforcing local-consistency on

the resulting CSP does varying amounts of work on different encodings [1]. Notice that enforcing the same level of local-consistency on DUAL encoded problems does more work than on LITERAL and NON-BINARY encoded instances. Since DUAL encoded problems can be significantly larger than LITERAL or NON-BINARY instances it is left as an open question as to whether the extra 'work' achieved is worth the space overhead. However, the empirical results in Chapter 6 suggest that this is certainly worth it, at least with respect to the types of problems in used in these experiments.

I also prove that enforcing strong-k-consistency on LITERAL encoded *k-SAT* instances does zero work if each clause has distinct literals and I suggest that these theoretical results might be practically useful in helping guide which encoding should be chosen for a particular problem instance. The empirical results in Chapter 6 demonstrate that this is most likely true.

	NON-BINARY	LITERAL	DUAL	HIDDEN VARIABLE	PLACE
NG-RES	k(*-Resolution*)	k	$(j(k-1), j)$	k (NON-BINARY) k (LITERAL)	k (NON-BINARY) $(j(k-1), j)$ (DUAL)

Table 7.4: The level of ***NG-RES*** when enforcing strong-k-consistency on *SAT* to CSP encodings. The row values indicate the level of ***NG-RES*** when applied to a CSP encoded *SAT* instance.

7.2 Empirical Studies

I attempt to address two issues raised in the literature review that may bias empirical studies of encodings, and identify a set of tests that have not been investigated previously. More specifically, I apply a relatively basic polynomial-time constraint-based technique to an extensive suite of *SAT* benchmarks represented as a CSP using the DUAL and LITERAL encodings. Table 7.5 provides a summary of the results of applying *path*-consistency to DUAL encoded SATLIB benchmarks discussed in Chapter 6.

Given the theoretical results in Chapter 5, it is not surprising that the empirical results demonstrate that enforcing *path*-consistency on LITERAL encoded *3-SAT* instances had very little impact; failing to solve a single instance. However, the success of enforcing a local-level of consistency on DUAL encoded unsatisfiable *SAT* instances is very interesting. By comparing these results with the *SAT*-Solvers submitted to the *SAT* Competition 2003, we see that this basic constraints-inspired polynomial-time algorithm can compete favourably with highly optimised and efficient **DLL**-based *SAT*-Solvers on unsatisfiable instances. As discussed, branching algorithms can prove both satisfiability and unsatisfiability, whereas stochastic algorithms can only prove satisfiability. Proving unsatisfiability has been shown to be more difficult for branching algorithms, so the results in this thesis encourage the development of sophisticated local-consistency algorithms which might be a viable approach to redress this balance. I conjecture that

[1] Remember that the notion of 'work' refers to the relative level of consistency achieved on the encoded problem instance (compared to the original) when enforcing a certain level of local-consistency.

benchmark set	inst's	ran	solves	props	clauses	avg. time (s)
dubois(20-30)	11	11	11	60-90	160-240	0.24-0.75
dubois50	1	1	1	150	400	3.75
dubois100	1	1	1	300	800	30.2
aim-50-1_6-no	4	4	4	50	80	0.032
aim-50-2_0-no	4	4	4	50	100	0.062
aim-100-1_6-no	4	4	4	100	160	0.27
aim-100-2_0-no	4	4	4	100	200	0.40
aim-200-1_6-no	4	4	4	200	320	1.85
aim-200-2_0-no	4	4	4	200	400	3.00
uuf50-218	1000	1000	1000	50	218	1.16
uuf75-325	100	100	55	75	325	25.05
uuf100-430	1000	1000	4	100	430	120.45
bf0432-007	1	1	1	1417	4045	2675.78
bf1355-075	1	1	1	2706	7304	4949.39
bf1355-638	1	1	1	2701	7292	3908.81
bf2670-001	1	1	1	1625	3666	550.36
ssa0432-003	1	1	1	504	1096	21.32
ssa2670-130	1	1	1	1583	3545	1309.45
ssa2670-141	1	1	1	1129	2458	503.51
goldberg/bmc1 4 & 23	2	2	2	217	493	1.5
goldberg/bmc1 42 & 61	2	2	2	417	943	9.7
graphcolors3/3col20	20	20	20	190	326	0.38
graphcolors3/3col40	20	20	20	380	646	4.87
graphcolors3/3col60	20	20	20	570	966	35.6
barrel-(2-9)	8	6	6	58-2806	167-9431	0.11-14977.031
biere/cmpadd	6	6	6	26-1132	70-3264	0.01-1936.453
biere/dinphil	4	4	4	213-1573	376-3904	0.28-1485.036
jnh	38	38	34	1665-2119	2365-2919	254.70
longmult(0-15)	16	4	3	558-2051	1327-5263	8.94-7122.125

Table 7.5: Summary of the unsatisfiable benchmark instances solved by enforcing *path*-consistency on DUAL encoded *SAT* instances. The notations $X - Y$ represents the spread of times that correspond to the spread of benchmarks in each family. A more detailed set of results are shown in Chapter 6.

enforcing higher levels of consistency may even outperform state-of-the-art *SAT*-Solvers in determining the unsatisfiability of many instances.

CHALLENGE 1: Prove that a hard 700 variable random *3-SAT* instance is unsatisfiable.

Although modern *SAT*-Solvers have solved 700 variable instances [2], random problems are still notoriously difficult for search-based and stochastic algorithms. One aim of this (and related) research is to design hybrid algorithms - exploiting the strengths of each field - yielding benefits that are mutually advantageous not only to the CSP and *SAT* communities, but also to the wider scientific community and to industry.

An obvious line of research is to compare the performance of stochastic and branching algorithms on DUAL encoded problems that have been made locally consistent (or partially consistent). The empirical results in Chapter 6 and the results described by Prestwich (2003) and Een & Biere (2005) suggest that practical techniques to prune search-spaces will most likely result in problems that are easier for search-based algorithms to solve.

Another future direction of this research is to utilise the wealth of theoretical work in the CSP community, and use it to test *SAT* instances for tractability. Tractability testing can help researchers decide when to use polynomial-time or search-based algorithms (Jeavons et al. (1996)).

Inspired by Prestwich's research, I currently have several ideas to explore in the future that could help alleviate in particular some bottlenecks in the **DLL** algorithm (the core of most modern *SAT*-Solvers), two of which are:

1. The time spent performing *unit propagation*.
2. The time spent exploring dead-end branches.

I intend to tackle the former by introducing an *exactly-1-SAT* pruning concept inspired by the difference between the DIRECT and MULTIVALUED CSP to *SAT* encodings. I also intend to improve the branching rule in **DLL** to explore only local areas of the search-tree with the aim to prevent the algorithm from exploring dead-end branches thereby potentially reducing unnecessary backtracking and greatly increasing performance.

7.2.1 Feature Analysis

Recently, the idea of dynamically applying heuristics according to some 'fitness measure' has been proven to be successful by Shacham & Yorav (2006) who propose the notion of *Adaptive Solving*, in which a *SAT*-Solver monitors the effectiveness of the search 'on-the-fly'. Using a *Performance Metric* to score the search progress one or more heuristics are turned on or off dynamically. The goal is to use a specific heuristic or strategy when it is advantageous, and to turn it off when it is not. Shacham & Yorav (2006) suggest several possible metrics and compare their effectiveness, showing that their adaptive solver achieves significant speedups on a large set of examples.

[2] Verbal communication with Ian Gent, St Andrews Constraints Research Group.

The notion of using 'problem features' to guide algorithm search (or choice) has been shown to be very successful by SATZILLA2007. SATZILLA incorporates so-called *empirical hardness models* that calculate features of the problem instance and use these to choose amongst a portfolio of *SAT*-Solvers. Nudelman et al. (2004) describe 91 *SAT* instance features, of which SATZILLA2007 uses around 70. It is widely known that there is no dominant *SAT*-Solver and that different solvers perform better or worse according to the 'type' of the problem instance. SATZILLA2007, described by Xu et al. (2007), is a 'meta-algorithm' that decides to apply one of a portfolio of state-of-the-art *SAT*-Solvers on a per-instance basis. Using *empirical hardness models* to choose among its constituent solvers, SATZILLA2007 significantly outperforms its constituent algorithms on every data set; winning three gold medals, one silver, and one bronze in 2007's *SAT* Competition.

We can extend this paradigm to the choice of encodings and (multiple) preprocessors. For example, in Chapter 5 I propose a simple heuristic for choosing encodings and algorithms for instances that have been predicted to be satisfiable or not. If an instance is likely to be satisfiable, then it might be appropriate to choose the LITERAL encoding and use a stochastic algorithm. Whereas if an instance is likely to be unsatisfiable, then choosing the DUAL encoding and running a branching algorithm might be more successful. In the former example it is ill-advised to enforce local-consistency, since it might not have any tangible pruning effect. However, it might be strongly advised to enforce local-consistency in the latter example, since this might entirely prune the search-space.

Given the success of these types of *dynamic* algorithms described above, I suspect that the research in these areas will become extremely important over the next few years. At present these features are *SAT* orientated, utilising none of the wealth of theoretical knowledge from the Constraints Satisfaction field devoted to understanding the tractability of these problems. If the performance of *SAT*-Solvers continues to plateau, the *SAT* community may turn to Constraint Satisfaction and Graph Theory research to provide a richer set of problem-features that these *empirical hardness models* can exploit. These methods may alleviate the plateau in the short to medium term, but it will take new paradigm shifts to make larger leaps in algorithmic performance. This supports the importance of providing a common framework to bridge the research between these fields. Moreover, I intend to investigate the 'meta-algorithm' paradigm further and explore other fields in search of 'better' features.

7.3 Extended Proof-Systems and Symmetry

In Chapter 3 I survey the pertinent theoretical work that is related to both *Resolution* and *Consistency* research. I argue that although extended proof-systems are the most powerful at our disposal (and that no known hard problems exist for them) little is known about how to use auxiliary variables effectively. Indeed, this area of research has not progressed greatly over the past ten years. Kullmann (1999) made some advances in proving lower-bounds for restricted versions of Extended-*Resolution*. Kullmann's

research may have a huge impact in the future, since the proof of exponential bounds would likely result in a paradigm shift in the way that complexity bounds are proved, as Haken (1985) results did when proving the first exponential bounds for standard *Resolution*.

7.3.1 Symmetry

Symmetries in the search-space can be broken by adding appropriate symmetry-breaking predicates to a *SAT* instance. These predicates prune the search-space by acting as a filter that confines the search to non-symmetric regions of the space without affecting the satisfiability of the instance. Crawford et al. (1996) attempted to utilise these techniques to *break* symmetry in some CSPs. This work showed how symmetries can be utilised by adding additional constraints to search problems, thereby ensuring that the *SAT*-Solver never visits two points in the search-space that are equivalent under some symmetry of the problem. The complexity results in this study suggested that generating symmetry-breaking predicates is exponential in the general case. Crawford's empirical results were not compelling, but suggested that partial symmetry breaking - that can be done in polynomial-time - might address this issue. As with any form of preprocessing, for symmetry-breaking to be effective in practice, the computational overhead of generating and manipulating the predicates must be significantly less than the run-time savings they yield due to search-space pruning. Recently Aloul et al. (2006) showed that formulae can be simplified using symmetry-breaking techniques that lead to run-time reductions on many benchmarks.

7.3.2 Extended Proof-Systems

Notable research has been developed around symmetry during the past 20 years, and despite extended proof-systems appearing in the literature 40 years ago, very little practical and theoretical advances have been made. There is a close relationship between extended proof-systems and symmetry; both add predicates with the aim to improve the solubility (or tractability) of a problem.

> **CHALLENGE 3:** Demonstrate that a propositional proof-system more powerful than Resolution can be made practical for satisfiability testing.

Although I have provided no major contribution to the area of extended proof-systems in this thesis, I imagine that this topic will become more important over the next decade. I argue that although almost nothing is known about how to 'intelligently' use auxiliary variables, in Section 5.3 I have described a simple way to automate this process. As with recent advances in branching heuristics and use of symmetry-breaking clauses, I predict that the use of auxiliary variables will provide the next great leaps in algorithmic performance.

The aim of introducing Extended-*Consistency* was to provide a general platform to allow the Constraint Satisfaction and *SAT* communities to share knowledge about these extended proof-systems and to begin to exploit synergies that are mutually beneficial to each field. This extends the work of Tseitin,

Baker and Mitchell, and provides a platform to communicate the wealth of theoretical results that have been previously discovered about the power of extended proof-systems.

By demonstrating how this graph-theoretic framework can help unify the findings in these various lines of work, I hope to utilise techniques from Graph Theory (and the wider Theoretical Computer Science community) that can be exploited and that are mutually beneficial. For instance, Yang (2005) published a Graph Theory paper describing an algorithm for 'Finding k-cliques on a k-partite Graphs'. These techniques (as with constraint-based techniques) are largely ignored by the *SAT* community. I intend to further the research into extended proof-systems in two ways:

1. To utilise the wealth of theoretical knowledge from the CSP community to develop Extended *SAT* algorithms that more effectively employ auxiliary variables, yet attempting to maintain the performance benefits of modern *SAT*-Solvers.

2. To look for (in)tractable cases for Extended-*Consistency* and Extended-*Resolution* and identify worse-case complexity bounds for these types of proof-systems.

Since there are no known hard problems for extended proof-systems, this research may prove useful in answering a question raised by Rossi et al. (1990), which is "Can you reduce in polynomial-time a *SAT* problem to a polynomial bounded graph that contains the same amount of information?". In fact, I envisage that some of the most important theoretical discoveries yet to be made in Complexity Theory will centre around extended proof-systems.

From a practical perspective, it is well-known that many modern *SAT*-Solvers suffer from wide clauses (Van Gelder (2006)), so by introducing auxiliary variables we can implement techniques to maintain a constant arity (width), employ symmetry-breaking, and develop this research to exploit the potential of these powerful proof-systems. We are still quite far from understanding the reasoning power of even the most simple *Resolution* algorithms with many of the basic questions still open (Ben-Sasson & Wigderson (1999)). The main aim of this thesis (and the subsequent research I propose) is to attempt to cultivate cross-fertilisation between the CSP and *SAT* research, providing the opportunity to gain deeper insights into Theoretical Computer Science by bridging the fields of Constraint Satisfaction and Propositional Satisfiability.

7.4 Exploitation

In this section I provide an overview of the use of *SAT* algorithms in EDA and highlight many of the prominent issues facing the industry. I discuss the direct benefits that improvements is *SAT* technology may have on the Formal Verification aspect of the EDA industry and beyond.

7.4.1 Electronic Design Automation

Electronic Design Automation (EDA) is the umbrella term for the category of tools used for designing and producing electronic systems ranging from Printed Circuit Boards (PCBs) to Integrated Circuits (also referred to as IC, microchips, silicon chips, or chips). Formal Verification is the process of testing chip designs for correctness using exhaustive and complex mathematical algorithms. It is estimated that around two thirds of IC projects fail to complete on schedule. As the design complexity of chips rapidly increases, it is becoming more and more difficult to test for errors with 'functional failure' cited as the number one cause by a wide margin[3].

7.4.1.1 Chip Verification

There are two mechanisms to test a chip design for correctness, 1. Simulation-based Verification 2. Formal-based Verification. The consensus is that Simulation and Formal Verification are complementary and that together they have the potential of giving the highest verification coverage in the fastest time[4].

In 2003, a typical 90nm design project cost about $25million to undertake and approximately half of the costs of any design project are on verification[5]. About 70% of new designs come back with errors when fabricated and tested. This is not because of problems with yield or timing, but because of functional problems that were missed during design verification[6]. On average 50% of the entire design effort is currently focused on verification and as the design of chips become more complex this increasing verification bottleneck is lengthening the design cycle, delaying time-to-market and reducing profits. Consequently, chip manufacturers - faced with increasingly costly designs and delays, potential recall of faulty chips and huge reputational risks - are applying pressure to EDA tool vendors to deliver more effective and robust verification solutions. This is just one issue adding the "design catastrophe" facing chip designers and threatening the accepted prediction of Moore's Law (Mann (2000)).

As design complexity has increased the tools that design engineers have at their disposal have become insufficient. Desperate to provide sufficient verification services, but limited by the capabilities of the verification algorithms, EDA suppliers are struggling to meet current needs and expected demands[7]. Traditional simulation-based verification techniques are being pushed past their limits in an effort to produce functionally correct new designs. It is not uncommon for IC design teams to write hundreds or thousands of tests in addition to many months of pseudo-random simulations in an effort to test all chip functionality and hit all the 'corner cases' (difficult cases) of the design.

Ultimately, trying to verify large designs by simulation alone is a losing battle. The size and complexity of chips are growing much faster than development teams can generate tests and servers can run

[3]Mentor Graphics: European ESL Survey 2005.
[4]Formal Verification Usage with FPGAs - `http://latticeblogs.typepad.com/frontier/2006/08/formal_verifica.html`
[5]Nowadays a typical 65nm design project costs $60, with around 70% of the cost on verification.
[6]Fragmentation of the IC Verification Process - `http://www.edat.com/NEA21.htm`
[7]FPGA Explosion Will Test EDA - `http://www.elecdesign.com/Articles/Index.cfm?ArticleID=15910&bypass=1`

them. A fundamentally different approach is needed in order to address this dilemma. The only viable alternative to simulation is Formal Verification - the use of mathematical analysis to prove properties about a design or, when proofs cannot be found, to generate diagnostic data to show how the design is inconsistent with the properties. Formal Verification has been around for decades in academia, used in a few large companies for nearly as long, and been available in commercial EDA products for about ten years[8]. While large companies like IBM, Intel and Motorola have routinely hosted Formal Verification experts since the early '90s (Gupta et al. (2003)) the algorithms to support this type of analysis were not sufficient to use as a routine part of the design process.

7.4.1.2 Adopting *SAT*-Solvers

Verification methods based on *SAT*-Solvers have recently emerged as a promising solution (Prasad et al. (2005)). For this reason *SAT* has been extensively studied in theory for the past 50 years and in practice for the past 15 years. Hundreds of academic and industrial researchers are actively working on *SAT* algorithms, and thousands of articles have been published to date. Dramatic improvements in *SAT*-solver technology over the past decade have led to the development of several powerful *SAT*-Solvers.

As a result the use of Formal Verification has increased significantly over the last few years and it is now estimated that about 15% of the bugs in chip designs can only be found using Formal Verification and up to 40% of the logic in a typical design project can be tested using Formal Verification analysis subject to careful planning (Ludden et al. (2002)). This is in marked contrast with the situation a decade and a half ago, when the fact that "every problem in **NP** can be reduced to *SAT*in polynomial-time" was widely regarded as being of only theoretical interest. Conventional wisdom was that a general purpose algorithm could not be expected to perform well on real-world search or decision problems.

So, once a bit of a novelty, *SAT*-Solving is now a commercially valuable and competitive activity, with widespread interest in academia and increasingly attracting the attention of a growing base in industry. *SAT*-based verification is now been applied to software as well as hardware (Jackson et al. (2000)), with the *SAT*-Solver's major advantage over simulation-based techniques attributed to their capacity either to assert that a property holds, or to compute a sequence that violates the property. Indeed, Formal Verification often uncovers bugs and corner cases that are very unlikely to be discovered by simulation alone. Formal Verification can also be initiated earlier in the design cycle and therefore improves time-to-market over the conventional verification by simulation. This means that Formal Verification can improve the design process in two ways, reduce the time to tape out (time-to-market), and reducing risk through better verification coverage.

However, the transfer of Formal Verification technology is a time consuming and costly business. This means that although the returns in investing in learning about and using formal methods in the

[8]FPGA Explosion Will Test EDA-`http://www.elecdesign.com/Articles/Index.cfm?ArticleID=15910&bypass=1`

long term may be large, the effects and benefits are less obvious than some of the other more popular techniques that come and go with fashion (Bowen & Hinchey (1997)).

Despite this, even marginal improvements in *SAT* algorithms can help IC design engineers to test their designs more quickly, allowing IC manufacturers to significantly reduce time-to-market, decrease the risk of faulty chips and thereby increase profits. As I have discussed throughout this thesis, although it is well-know that the encoding can have a significant impact on the solubility of a problem the *SAT* community typically adopt only one. Moreover, I have shown that coupling the encoding with an efficient preprocessor, it is possible to solve many of the problems thought to be 'hard' for traditional *SAT*-Solvers.

Many 'natural' CSPs can be encoded into *SAT*. This property makes the value of improving *SAT*-Solvers extend beyond the area of EDA and Verification, offering the possibility to benefit other industries by incorporating a whole range of 'hard' natural problems, such as timetabling or scheduling. Developing theoretical knowledge should be complemented by practical application. With some research, too much emphasis is paid to either one or the other, which is why I attempt to find a balance between both in this thesis.

Appendix A

Benchmark File Format

A.1 DIMACS CNF Format

To represent an instance of CNF problems, DIMACS (1993) has suggested a file format that contains all of the information needed to define a satisfiability problem. This file will be an ASCII file consisting of a two major sections: the preamble and the clauses.

The Preamble. The preamble contains information about the instance. This information is contained in lines. Each line begins with a single character (followed by a space) that determines the type of line. The type of lines are as follows:

- **Comments.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines appear at the beginning of the preamble. Each comment line begins with a lower-case character `c`.

  ```
  c This is an example of a comment line.
  ```

- **Problem line.** There is one problem line per input file. The problem line must appear before any node or arc descriptor lines. For cnf instances, the problem line has the following format.

  ```
  p FORMAT VARIABLES CLAUSES
  ```

  The lower-case character `p` signifies that this is the problem line. The `FORMAT` field allows programs to determine the format that will be expected, and should contain the word 'cnf'. The `VARIABLES` field contains an integer value specifying n, the number of variables in the instance. The `CLAUSES` field contains an integer value specifying m, the number of clauses in the instance. This line must occur as the last line of the preamble.

The Clauses. The clauses appear immediately after the problem line. The variables are assumed to be numbered from 1 up to n. It is not necessary that every variable appear in an instance. Each clause will be represented by a sequence of numbers, each separated by either a space, a tab, or a new line

character. The non-negated version of a variable i is represented by i; the negated version is represented by $-i$.

Each clause is terminated by the value 0. Unlike many formats that represent the end of a clause by a new-line character, this format allows clauses to be on multiple lines.

Example. Using the example

$$(x_1 \vee x_3 \vee \bar{x}_4) \wedge (x_4) \wedge (x_2 \vee \bar{x}_3) \tag{A.1}$$

a possible input file would be

```
c Example CNF format file
c
p cnf 4 3
1 3 -4 0
4 0 2
-3
```

and the input file for formula 2.1 could be

```
c 3SAT Equation example
c Daniel J Hulme
c
p cnf 4 4
1 2 3 0
-1 2 -3 0
1 -2 4 0
-2 -3 -4 0
```

CNF format files will generally have a `.cnf` extension.

Appendix B

SAT Encodings

B.1 LOG Encoding of Example 2.2.3

If we use the LOG encoding, the *SAT* instance of the GRAPH 3-COLOURABILITY Example 2.2.3 (on the assumption that the binary values map to $R = (0,0), G = (0,1)$ and $B = (1,0)$) is:

- *negative*: $(\bar{v}_{x_0}[1] \vee \bar{v}_{x_0}[1]) \wedge (\bar{v}_{x_1}[1] \vee \bar{v}_{x_1}[1]) \wedge (\bar{v}_{x_2}[1] \vee \bar{v}_{x_2}[1]) \wedge (\bar{v}_{x_3}[1] \vee \bar{v}_{x_3}[1]) \wedge (\bar{v}_{x_4}[1] \vee \bar{v}_{x_4}[1])$
- *constraint*: $(v_{x_0}[0] \vee v_{x_0}[1] \vee v_{x_1}[0] \vee v_{x_1}[1]) \wedge (v_{x_0}[0] \vee v_{x_0}[1] \vee v_{x_2}[0] \vee v_{x_2}[1]) \wedge (v_{x_0}[0] \vee v_{x_0}[1] \vee v_{x_3}[0] \vee v_{x_3}[1]) \wedge (v_{x_0}[0] \vee v_{x_0}[1] \vee v_{x_4}[0] \vee v_{x_4}[1]) \wedge (v_{x_1}[0] \vee v_{x_1}[1] \vee v_{x_2}[0] \vee v_{x_2}[1]) \wedge (v_{x_1}[0] \vee v_{x_1}[1] \vee v_{x_3}[0] \vee v_{x_3}[1]) \wedge (v_{x_1}[0] \vee v_{x_1}[1] \vee v_{x_4}[0] \vee v_{x_4}[1]) \wedge (v_{x_2}[0] \vee v_{x_2}[1] \vee v_{x_3}[0] \vee v_{x_3}[1]) \wedge (v_{x_2}[0] \vee v_{x_2}[1] \vee v_{x_4}[0] \vee v_{x_4}[1]) \wedge (v_{x_3}[0] \vee v_{x_3}[1] \vee v_{x_4}[0] \vee v_{x_4}[1]) \wedge (v_{x_0}[0] \vee \bar{v}_{x_0}[1] \vee v_{x_1}[0] \vee \bar{v}_{x_1}[1]) \wedge (v_{x_0}[0] \vee \bar{v}_{x_0}[1] \vee v_{x_2}[0] \vee \bar{v}_{x_2}[1]) \wedge (v_{x_0}[0] \vee \bar{v}_{x_0}[1] \vee v_{x_3}[0] \vee \bar{v}_{x_3}[1]) \wedge (v_{x_0}[0] \vee \bar{v}_{x_0}[1] \vee v_{x_4}[0] \vee \bar{v}_{x_4}[1]) \wedge (v_{x_1}[0] \vee \bar{v}_{x_1}[1] \vee v_{x_2}[0] \vee \bar{v}_{x_2}[1]) \wedge (v_{x_1}[0] \vee \bar{v}_{x_1}[1] \vee v_{x_3}[0] \vee \bar{v}_{x_3}[1]) \wedge (v_{x_1}[0] \vee \bar{v}_{x_1}[1] \vee v_{x_4}[0] \vee \bar{v}_{x_4}[1]) \wedge (v_{x_2}[0] \vee \bar{v}_{x_2}[1] \vee v_{x_3}[0] \vee \bar{v}_{x_3}[1]) \wedge (v_{x_2}[0] \vee \bar{v}_{x_2}[1] \vee v_{x_4}[0] \vee \bar{v}_{x_4}[1]) \wedge (v_{x_3}[0] \vee \bar{v}_{x_3}[1] \vee v_{x_4}[0] \vee \bar{v}_{x_4}[1]) \wedge (\bar{v}_{x_0}[0] \vee v_{x_0}[1] \vee \bar{v}_{x_1}[0] \vee v_{x_1}[1]) \wedge (\bar{v}_{x_0}[0] \vee v_{x_0}[1] \vee \bar{v}_{x_2}[0] \vee v_{x_2}[1]) \wedge (\bar{v}_{x_0}[0] \vee v_{x_0}[1] \vee \bar{v}_{x_3}[0] \vee v_{x_3}[1]) \wedge (\bar{v}_{x_0}[0] \vee v_{x_0}[1] \vee \bar{v}_{x_4}[0] \vee v_{x_4}[1]) \wedge (\bar{v}_{x_1}[0] \vee v_{x_1}[1] \vee \bar{v}_{x_2}[0] \vee v_{x_2}[1]) \wedge (\bar{v}_{x_1}[0] \vee v_{x_1}[1] \vee \bar{v}_{x_3}[0] \vee v_{x_3}[1]) \wedge (\bar{v}_{x_1}[0] \vee v_{x_1}[1] \vee \bar{v}_{x_4}[0] \vee v_{x_4}[1]) \wedge (\bar{v}_{x_2}[0] \vee v_{x_2}[1] \vee \bar{v}_{x_3}[0] \vee v_{x_3}[1]) \wedge (\bar{v}_{x_2}[0] \vee v_{x_2}[1] \vee \bar{v}_{x_4}[0] \vee v_{x_4}[1]) \wedge (\bar{v}_{x_3}[0] \vee v_{x_3}[1] \vee \bar{v}_{x_4}[0] \vee v_{x_4}[1])$

B.2 INVERSE Encoding of Example 2.2.3

The CNF formula (excluding *negative* clauses) resulting from the INVERSE encoding of Example 2.2.3, is the conjunction of:

- *positive*: $(l_{c_0^R c_1^G} \vee l_{c_0^R c_1^B} \vee l_{c_0^G c_1^R} \vee l_{c_0^G c_1^B} \vee l_{c_0^B c_1^R} \vee l_{c_0^B c_1^G}) \wedge (l_{c_0^R c_4^G} \vee l_{c_0^R c_4^B} \vee l_{c_0^G c_4^R} \vee l_{c_0^G c_4^B} \vee l_{c_0^B c_4^R} \vee l_{c_0^B c_4^G}) \wedge (l_{c_1^R c_2^G} \vee l_{c_1^R c_2^B} \vee l_{c_1^G c_2^R} \vee l_{c_1^G c_2^B} \vee l_{c_1^B c_2^R} \vee l_{c_1^B c_2^G}) \wedge (l_{c_1^R c_3^G} \vee l_{c_1^R c_3^B} \vee l_{c_1^G c_3^R} \vee l_{c_1^G c_3^B} \vee l_{c_1^B c_3^R} \vee l_{c_1^B c_3^G}) \wedge (l_{c_2^R c_3^G} \vee l_{c_2^R c_3^B} \vee l_{c_2^G c_3^R} \vee l_{c_2^G c_3^B} \vee l_{c_2^B c_3^R} \vee l_{c_2^B c_3^G}) \wedge (l_{c_2^R c_4^G} \vee l_{c_2^R c_4^B} \vee l_{c_2^G c_4^R} \vee l_{c_2^G c_4^B} \vee l_{c_2^B c_4^R} \vee l_{c_2^B c_4^G})$
- *constraint*: $(\bar{l}_{c_0^R c_1^G} \vee \bar{l}_{c_0^G c_4^R}) \wedge (\bar{l}_{c_0^R c_1^G} \vee \bar{l}_{c_0^B c_4^R}) \wedge (\bar{l}_{c_0^R c_1^B} \vee \bar{l}_{c_0^G c_4^R}) \wedge (\bar{l}_{c_0^R c_1^B} \vee \bar{l}_{c_0^B c_4^R}) \wedge (\bar{l}_{c_0^G c_1^R} \vee \bar{l}_{c_0^R c_4^G}) \wedge (\bar{l}_{c_0^G c_1^R} \vee \bar{l}_{c_0^B c_4^G}) \wedge (\bar{l}_{c_0^G c_1^B} \vee \bar{l}_{c_0^R c_4^G}) \wedge (\bar{l}_{c_0^G c_1^B} \vee \bar{l}_{c_0^B c_4^G}) \wedge (\bar{l}_{c_0^B c_1^R} \vee \bar{l}_{c_0^R c_4^B}) \wedge (\bar{l}_{c_0^B c_1^R} \vee \bar{l}_{c_0^G c_4^B}) \wedge$

$$(\bar{l}_{c_0^B c_1^G} \vee \bar{l}_{c_0^R c_4^B}) \wedge (\bar{l}_{c_0^B c_1^G} \vee \bar{l}_{c_0^G c_4^B}) \wedge (\bar{l}_{c_1^R c_0^G} \vee \bar{l}_{c_1^G c_2^R}) \wedge (\bar{l}_{c_1^R c_0^G} \vee \bar{l}_{c_1^B c_2^R}) \wedge (\bar{l}_{c_1^R c_0^B} \vee \bar{l}_{c_1^G c_2^R}) \wedge (\bar{l}_{c_1^R c_0^B} \vee \bar{l}_{c_1^B c_2^R}) \wedge (\bar{l}_{c_1^G c_0^R} \vee \bar{l}_{c_1^R c_2^G}) \wedge (\bar{l}_{c_1^G c_0^R} \vee \bar{l}_{c_1^B c_2^G}) \wedge (\bar{l}_{c_1^G c_0^B} \vee \bar{l}_{c_1^R c_2^G}) \wedge (\bar{l}_{c_1^G c_0^B} \vee \bar{l}_{c_1^B c_2^G}) \wedge (\bar{l}_{c_1^B c_0^R} \vee \bar{l}_{c_1^R c_2^B}) \wedge (\bar{l}_{c_1^B c_0^R} \vee \bar{l}_{c_1^G c_2^B}) \wedge (\bar{l}_{c_1^B c_0^G} \vee \bar{l}_{c_1^R c_2^B}) \wedge (\bar{l}_{c_1^B c_0^G} \vee \bar{l}_{c_1^G c_2^B}) \wedge (\bar{l}_{c_1^R c_0^G} \vee \bar{l}_{c_1^G c_3^R}) \wedge (\bar{l}_{c_1^R c_0^G} \vee \bar{l}_{c_1^B c_3^R}) \wedge (\bar{l}_{c_1^R c_0^B} \vee \bar{l}_{c_1^G c_3^R}) \wedge (\bar{l}_{c_1^R c_0^B} \vee \bar{l}_{c_1^B c_3^R}) \wedge (\bar{l}_{c_1^G c_0^R} \vee \bar{l}_{c_1^R c_3^G}) \wedge (\bar{l}_{c_1^G c_0^R} \vee \bar{l}_{c_1^B c_3^G}) \wedge (\bar{l}_{c_1^G c_0^B} \vee \bar{l}_{c_1^R c_3^G}) \wedge (\bar{l}_{c_1^G c_0^B} \vee \bar{l}_{c_1^B c_3^G}) \wedge (\bar{l}_{c_1^B c_0^R} \vee \bar{l}_{c_1^R c_3^B}) \wedge (\bar{l}_{c_1^B c_0^R} \vee \bar{l}_{c_1^G c_3^B}) \wedge (\bar{l}_{c_1^B c_0^G} \vee \bar{l}_{c_1^R c_3^B}) \wedge (\bar{l}_{c_1^B c_0^G} \vee \bar{l}_{c_1^G c_3^B}) \wedge (\bar{l}_{c_1^R c_2^G} \vee \bar{l}_{c_1^G c_3^R}) \wedge (\bar{l}_{c_1^R c_2^G} \vee \bar{l}_{c_1^B c_3^R}) \wedge (\bar{l}_{c_1^B c_2^R} \vee \bar{l}_{c_1^G c_3^B}) \wedge (\bar{l}_{c_1^G c_2^R} \vee \bar{l}_{c_1^B c_3^G}) \wedge (\bar{l}_{c_1^G c_2^B} \vee \bar{l}_{c_1^R c_3^G}) \wedge (\bar{l}_{c_1^G c_2^B} \vee \bar{l}_{c_1^B c_3^G}) \wedge (\bar{l}_{c_1^B c_2^R} \vee \bar{l}_{c_1^G c_3^B}) \wedge (\bar{l}_{c_1^B c_2^G} \vee \bar{l}_{c_1^R c_3^B}) \wedge (\bar{l}_{c_1^B c_2^G} \vee \bar{l}_{c_1^G c_3^B}) \wedge (\bar{l}_{c_2^R c_1^G} \vee \bar{l}_{c_2^G c_3^R}) \wedge (\bar{l}_{c_2^R c_1^G} \vee \bar{l}_{c_2^B c_3^R}) \wedge (\bar{l}_{c_2^R c_1^B} \vee \bar{l}_{c_2^G c_3^R}) \wedge (\bar{l}_{c_2^R c_1^B} \vee \bar{l}_{c_2^B c_3^R}) \wedge (\bar{l}_{c_2^G c_1^R} \vee \bar{l}_{c_2^B c_3^G}) \wedge (\bar{l}_{c_2^G c_1^B} \vee \bar{l}_{c_2^R c_3^G}) \wedge (\bar{l}_{c_2^G c_1^B} \vee \bar{l}_{c_2^B c_3^G}) \wedge (\bar{l}_{c_2^B c_1^R} \vee \bar{l}_{c_2^R c_3^B}) \wedge (\bar{l}_{c_2^B c_1^R} \vee \bar{l}_{c_2^G c_3^B}) \wedge (\bar{l}_{c_2^B c_1^G} \vee \bar{l}_{c_2^R c_3^B}) \wedge (\bar{l}_{c_2^B c_1^G} \vee \bar{l}_{c_2^B c_3^G}) \wedge (\bar{l}_{c_2^R c_1^G} \vee \bar{l}_{c_2^G c_4^R}) \wedge (\bar{l}_{c_2^R c_1^G} \vee \bar{l}_{c_2^B c_4^R}) \wedge (\bar{l}_{c_2^R c_1^B} \vee \bar{l}_{c_2^G c_4^R}) \wedge (\bar{l}_{c_2^R c_1^B} \vee \bar{l}_{c_2^B c_4^R}) \wedge (\bar{l}_{c_2^G c_1^R} \vee \bar{l}_{c_2^B c_4^G}) \wedge (\bar{l}_{c_2^G c_1^R} \vee \bar{l}_{c_2^R c_4^G}) \wedge (\bar{l}_{c_2^G c_1^B} \vee \bar{l}_{c_2^B c_4^G}) \wedge (\bar{l}_{c_2^B c_1^R} \vee \bar{l}_{c_2^R c_4^B}) \wedge (\bar{l}_{c_2^B c_1^R} \vee \bar{l}_{c_2^G c_4^B}) \wedge (\bar{l}_{c_2^B c_1^G} \vee \bar{l}_{c_2^R c_4^B}) \wedge (\bar{l}_{c_2^B c_1^G} \vee \bar{l}_{c_2^B c_4^G}) \wedge (\bar{l}_{c_2^R c_3^G} \vee \bar{l}_{c_2^G c_4^R}) \wedge (\bar{l}_{c_2^R c_3^G} \vee \bar{l}_{c_2^B c_4^R}) \wedge (\bar{l}_{c_2^R c_3^B} \vee \bar{l}_{c_2^G c_4^R}) \wedge (\bar{l}_{c_2^R c_3^B} \vee \bar{l}_{c_2^B c_4^R}) \wedge (\bar{l}_{c_2^G c_3^R} \vee \bar{l}_{c_2^R c_4^G}) \wedge (\bar{l}_{c_2^G c_3^R} \vee \bar{l}_{c_2^B c_4^G}) \wedge (\bar{l}_{c_2^G c_3^B} \vee \bar{l}_{c_2^R c_4^G}) \wedge (\bar{l}_{c_2^G c_3^B} \vee \bar{l}_{c_2^B c_4^G}) \wedge (\bar{l}_{c_2^B c_3^R} \vee \bar{l}_{c_2^R c_4^B}) \wedge (\bar{l}_{c_2^B c_3^R} \vee \bar{l}_{c_2^G c_4^B}) \wedge (\bar{l}_{c_2^B c_3^G} \vee \bar{l}_{c_2^R c_4^B}) \wedge (\bar{l}_{c_2^B c_3^G} \vee \bar{l}_{c_2^G c_4^B}) \wedge (\bar{l}_{c_4^R c_0^G} \vee \bar{l}_{c_4^G c_2^R}) \wedge (\bar{l}_{c_4^R c_0^G} \vee \bar{l}_{c_4^B c_2^R}) \wedge (\bar{l}_{c_4^R c_0^B} \vee \bar{l}_{c_4^G c_2^R}) \wedge (\bar{l}_{c_4^R c_0^B} \vee \bar{l}_{c_4^B c_2^R}) \wedge (\bar{l}_{c_4^G c_0^R} \vee \bar{l}_{c_4^R c_2^G}) \wedge (\bar{l}_{c_4^G c_0^R} \vee \bar{l}_{c_4^B c_2^G}) \wedge (\bar{l}_{c_4^G c_0^B} \vee \bar{l}_{c_4^R c_2^G}) \wedge (\bar{l}_{c_4^G c_0^B} \vee \bar{l}_{c_4^B c_2^G}) \wedge (\bar{l}_{c_4^B c_0^R} \vee \bar{l}_{c_4^R c_2^B}) \wedge (\bar{l}_{c_4^B c_0^R} \vee \bar{l}_{c_4^G c_2^B}) \wedge (\bar{l}_{c_4^B c_0^G} \vee \bar{l}_{c_4^R c_2^B}) \wedge (\bar{l}_{c_4^B c_0^G} \vee \bar{l}_{c_4^G c_2^B})$$

Bibliography

Achlioptas, D., Chtcherba, A., Istrate, G., & Moore, C. (2001). The phase transition in 1-in-k SAT and NAE 3-SAT. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, (pp. 721–722). Society for Industrial and Applied Mathematics Philadelphia, PA, USA.

Achlioptas, D., Naor, A., & Peres, Y. (2005). Rigorous location of phase transitions in hard optimization problems. *Nature*, *435*, 759–764.

Adleman, L., & Manders, K. (1977). Reducibility, randomness, and intractibility (Abstract). In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, (pp. 151–163). New York, NY, USA: ACM Press.

Ajtai, M. (1994). The complexity of the Pigeonhole Principle. *Combinatorica*, *14*(4), 417–433.

Aloul, F., Sakallah, K., & Markov, I. (2006). Efficient Symmetry Breaking for Boolean Satisfiability. *IEEE TRANSACTIONS ON COMPUTERS*, *55*, 549–558.

Anbulagan, & Slaney., J. (2006). Multiple preprocessing for systematic sat solvers. In *IWIL-6, 2006. Workshop in conjunction with LPAR-2006*.

Ansotegui, C., & Manya, F. (2004). Mapping problems with finite-domain variables into problems with boolean variables. *SAT 2004*.

Asahiro, Y., Iwama, K., & Miyano, E. (1993). Random generation of test instances with controlled attributes. *Second DIMACS Challenge Workshop*.

Atserias, A., & Dalmau, V. (2003). A combinatorial characterization of resolution width. *Computational Complexity, 2003. Proceedings. 18th IEEE Annual Conference on*, (pp. 239–247).

Atserias, A., Kolaitis, P., & Vardi, M. (2004). Constraint Propagation as a Proof System. In *10th Int.Conf. on Principles and Practice of Constraint Programing, LN in Computer Science*, vol. 3258, (pp. 77–91). Springer.

Bacchus, F., & Grove, A. (1995). On the Forward Checking Algorithm. *Principles and Practice of Constraint Programming*, (pp. 292–309).

Bacchus, F., & van Beek, P. (1998). On the conversion between non-binary and binary constraint satisfaction problems. *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, (pp. 311–318).

Bacchus, F., & Winter, J. (2003). Effective preprocessing with hyper-resolution and equality reduction. In *In SAT*, (pp. 341–355).

Bachmair, L., & Ganzinger, H. (2001). Resolution theorem proving. *Handbook of Automated Reasoning*, *1*, 19–99.

Bailleux, O., & Boufkhad, Y. (2003). Efficient CNF encoding of boolean cardinality constraints. *Principles and Practice of Constraint Programming - 9th International Conference, CP*.

Baker, A. (1995). *Intelligent Backtracking on Constraint Satisfaction Problems: Experimental and Theoretical Results*. Ph.D. thesis, University of Oregon.

Beame, P., & Pitassi, T. (1996). Simplified and improved resolution lower bounds. *37th Annual Symposium on Foundations of Computer Science, 274282*.

Beldiceanu, N., Carlsson, M., & Rampon, J. (2005). Global Constraint Catalog. *Technical Report T2005-06, Swedish Institute of Computer Science, Kista.*.

Ben-Sasson, E. (2001). *Expansion in Proof Complexity*. Ph.D. thesis, Hebrew University.

Ben-Sasson, E., & Wigderson, A. (1999). Short proofs are narrowresolution made simple. *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, (pp. 517–526).

Bennaceur, H. (1996). The satisfiability problem regarded as a constraint satisfaction problem. *Proc. ECAI*, *96*, 155–159.

Bennaceur, H. (2004). A Comparison between SAT and CSP Techniques. *Constraints*, *9*(2), 123–138.

Bessiere, C., & Cordier, M.-O. (1994). Arc-Consistency and Arc-Consitency Again. In M. Meyer (Ed.) *Proceedings ECAI'94 Workshop on Constraint Processing*. Amsterdam.
URL `citeseer.ist.psu.edu/bessiere94arcconsistency.html`

Bessiére, C., Freuder, E., & Regin, J. (1999). Using constraint metaknowledge to reduce arc consistency computation. *Artificial Intelligence*, *107*(1), 125–148.

Bessiere, C., Hebrard, E., & Walsh, T. (2003). Local Consistencies in SAT. *Proc. SAT-2003*.

Bessiere, C., Meseguer, P., Freuder, E., & Larrosa, J. (2002). On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, *141*, 205224.

Bonet, M., Buss, S., & Pitassi, T. (1994). Are there hard examples for Frege systems. *Feasible Mathematics II*, (pp. 30–56).

Bordeaux, L., Hamadi, Y., & Zhang, L. (2006). Propositional Satisfiability and Constraint Programming: A comparative survey. *ACM Computing Surveys (CSUR)*, *38*(4).

Bowen, J., & Hinchey, M. (1997). The Use of Industrial-Strength Formal Methods. *Proc. COMPSAC97: 21st IEEE Annal International Computer Software and Application Conference, Washington DC, USA*, (pp. 332–337).

Brafman, R. (2004). A Simplifier for Propositional Formulas With Many Binary Clauses. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, *34*(1), 52–59.

Brailsford, S., Potts, C., & Smith, B. (1999). Constraint satisfaction problems: Algorithms and applications. *European Journal of Operational Research*, *119*(3), 557–581.

Broadfoot, G., & Broadfoot, P. (2003). Academia and industry meet: some experiences of formal methods in practice. *Software Engineering Conference, 2003. Tenth Asia-Pacific*, (pp. 49–58).

Buchberger, B., & Winkler, F. (1998). *Gröbner Bases and Applications*. Cambridge University Press.

Buss, S. (1987). Polynomial Size Proofs of the Propositional Pigeonhole Principle. *The Journal of Symbolic Logic*, *52*(4), 916–927.

Buss, S., & Turan, G. (1988). Resolution Proofs of Generalized Pigeonhole Principles. *TCS*, *62*(3), 311–317.

Cheeseman, P., Kanefsky, B., & Taylor, W. (1991). Where the really hard problems are. *Proceedings of the 12th IJCAI*, (pp. 331–337).

Clark, D., Frank, J., Gent, I., MacIntyre, E., Tomov, N., & Walsh, T. (1996). Local search and the number of solutions. *Proc. 2nd Int. Conf. on the Principles and Practices of Constraint Programming*, (pp. 119–133).

Clegg, M., Edmonds, J., & Impagliazzo, R. (1996). Using the Groebner basis algorithm to find proofs of unsatisfiability. *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, (pp. 174–183).

Cohen, D., Jeavons, P., Jonsson, P., & Koubarakis, M. (2000). Building tractable disjunctive constraints. *Journal of the ACM (JACM)*, *47*(5), 826–853.

Condrat, C., & Kalla, P. (2007). A Grobner Basis Approach to CNF-Formulae Preprocessing. In *Tools and Algorithms for the Construction and Analysis of Systems 13th International Conference, TACAS*

2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007: Proceedings. Springer.

Cook, S. (1976). A short proof of the pigeon hole principle using extended resolution. *ACM SIGACT News*, *8*(4), 28–32.

Cook, S. (2003). The importance of the P versus NP question. *J. ACM*, *50*(1), 27–29.
URL citeseer.ist.psu.edu/634970.html

Cook, S., & Mitchell, D. (1997). Finding hard instances of the satisfiability problem: A survey. *Satisfiability Problem: Theory and Applications*, *35*, 1–18.

Cook, S., & Reckhow, R. (1979). The Relative Efficiency of Propositional Proof Systems. *The Journal of Symbolic Logic*, *44*(1), 36–50.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *STOC '71: Proceedings of the third annual ACM symposium on Theory of computing*, (pp. 151–158). New York, NY, USA: ACM Press.

Cook, S. A. (2000). The P Versus NP Problem. Computer Science Department, University of Toronto. Available at http://www.cs.toronto.edu/sacook/homepage/PvsNP.ps.
URL citeseer.ist.psu.edu/302888.html

Cooper, M. C., Cohen, D. A., & Jeavons, P. (1994). Characterising Tractable Constraints. *Artificial Intelligence*, *65*(2), 347–361.
URL citeseer.ist.psu.edu/cooper94characterising.html

Crawford, J., & Auton, L. (1996). Experimental results on the crossover point in random 3-SAT. *Artificial Intelligence*, *81*(1-2), 31–57.

Crawford, J., Ginsberg, M., Luks, E., & Roy, A. (1996). Symmetry-breaking predicates for search problems. *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*.

Davis, M., Logemann, G., & Loveland, D. (1962). A machine program for theorem-proving. *Communications of the ACM*, *5*(7), 394–397.

Davis, M., & Putnam, H. (1960). A Computing Procedure for Quantification Theory. *Journal of the ACM (JACM)*, *7*(3), 201–215.

de Kleer, J. (1989). A comparison of ATMS and CSP techniques. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, (pp. 290–296).

Dechter, R. (1990). On the Expressiveness of Networks with Hidden Variables. In *Eighth national conference on Artificial intelligence*, (pp. 556–562).

Dechter, R. (1992a). *Constraint Networks*. Information and Computer Science, University of California, Irvine.

Dechter, R. (1992b). From Local to Global Consistency. *Artificial Intelligence*, *55*(1), 87–108.
URL citeseer.ist.psu.edu/dechter92from.html

DIMACS (1993). Satisfiability Suggested Format.

Do, M., & Kambhampati, S. (2001). Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP. *Artificial Intelligence*, *132*(2), 151–182.

Een, N., & Biere, A. (2005). Effective preprocessing in SAT through variable and clause elimination. *SAT 2005: international conference on theory and applications of satisfiability testing*, *3569*, 61–75.

Een, N., & Sorensson, N. (2003). MiniSat: A SAT solver with conflict clause minimization. *Proc. SAT*, *5*.

Esteban, J., & Torán, J. (2001). Space Bounds for Resolution. *Information and Computation*, *171*(1), 84–97.

Fortnow, L., & Homer, S. (2002). A Short History of Computational Complexity. *Bull. Eur. Assoc. Theor. Comput. Sci*, *80*, 95–133.
URL citeseer.ist.psu.edu/fortnow02short.html

Freuder, E. (1978). Synthesizing constraint expressions. *Communications of the ACM*, *21*(11), 958–966.

Freuder, E. (1982). A Sufficient Condition for Backtrack-Free Search. *Journal of the ACM (JACM)*, *29*(1), 24–32.

Frisch, A., & Peugniez, T. (2001). Solving non-boolean satisfiability problems with stochastic local search. *Proc. IJCAI*, *1*, 282–288.

Galil, Z. (1977). On Resolution with Clauses of Bounded Size. *SIAM Journal on Computing*, *6*, 444.

Garey, M. R., & Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co.

Gasarch, W. (2002). Guest column: The P=? NP poll. *SIGACT NEWS*, *33*(2), 34–47.

Gaschnig, J. (1974). A constraint satisfaction method for inference making. In *Proceedings of the Twelfth Annual Allerton Conference on Circuit Systems Theory*, (pp. 866–874).

Gent, I. (2002). Arc consistency in SAT. *Fifteenth European Conference on Artificial Intelligence*, (pp. 121–125).

Gent, I., Jefferson, C., & Miguel, I. (2006). Minion: A Fast Scalable Constraint Solver. *FRONTIERS IN ARTIFICIAL INTELLIGENCE AND APPLICATIONS*, *141*, 98.

Gent, I., MacIntyre, E., Prosser, P., & Walsh, T. (1996). The constrainedness of search. *Proceedings of AAAI-96*, *1*, 246–252.

Gent, I., Prosser, P., & Walsh, T. (2003). The extended literal encoding of SAT into CSP. Tech. rep., Technical Report APES-73-2003, APES Research Group, November 2003. Available from http://www. dcs. stand. ac. uk/apes/apesreports. html.

Gent, I., & Walsh, T. (1996). The Satisfiability Constraint Gap. *Artificial Intelligence*, *81*(1), 59–80.

Gent, I. P., & Walsh, T. (1994). The sat phase transition. In *Proceedings of 11th ECAI*, (pp. 105–109). John Wiley & Sons.

Gent, I. P., & Walsh, T. (1995). Phase transitions from real computational problems. In *In Proceedings of the 8th International Symposium on Artificial Intelligence*, (pp. 356–364).

Goldberg, E., & Novikov, Y. (2002). BerkMin: A fast and robust SAT-solver. *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, (pp. 142–149).

Grant, S., & Smith, B. (1995). *The Phase Transition Behaviour of Maintaining Arc Consistency*. University of Leeds, School of Computer Studies.

Green, M. (2005). *New Methods for the Tractability of Constraint Satisfaction Problems*. Ph.D. thesis, University of London, Department of Computer Science, Royal Holloway, Egham, Surrey, UK.
URL `http://www.cs.rhul.ac.uk/home/green/publications/thesis/MJGreen_PhDThesis.ps`

Gu, J., Purdom, P., Franco, J., & Wah, B. (1997). Algorithms for the Satisfiability (SAT) Problem: a Survey. *Discrete Mathematics and Theoretical Computer Science: Satisfiability*, (pp. 378–383).
URL `citeseer.ist.psu.edu/56722.html`

Gupta, R., et al. (2003). Panel: Formal Verification: Prove It or Pitch It. *Design Automation Conference, June*.

Haanpaa, H., Jarvisalo, M., Kaski, P., & Niemela, I. (2005). SAT Benchmarks based on 3-Regular Graphs. In *SAT Competition 2005*.

Haken, A. (1985). The Intractability of Resolution. *TCS*, *39*, 297–308.

Haralick, R., & Elliott, G. (1980). Increasing Tree Search Efficiency for Constraint Satisfaction Problems. *Artificial Intelligence*, *14*(3), 263–313.

Hartmanis, J., & Stearns, R. (1965). On the Computational Complexity of Algorithms. *Transactions of the American Mathematical Society*, *117*, 285–306.

Hooker, J. (2007). *Integrated Methods for Optimization*. Springer.

Hoos, H. (1999). *Stochastic Local Search-Methods, Models, Applications*. IOS Press.

Hoos, H. H., & Stützle, T. (2000). SATLIB: An Online Resource for Research on SAT. In *SAT'2000*, (pp. 283–292). IOS Press.
URL citeseer.ist.psu.edu/hoos00satlib.html

Hulme, D. J., Hirsch, R., Buxton, B., & Lotto, R. (2007). A new reduction from 3sat to n-partite graphs. In *FOCI'07: IEEE Symposium on Foundations of Computational Intelligence*, IEEE Symposium on Computational Intelligence. Honolulu, Hawaii, USA: IEEE Computer Society.

Hwang, C. (2004). *A Theoretical Comparison of Resolution Proof Systems for CSP Algorithms*. Ph.D. thesis, SIMON FRASER UNIVERSITY.

Istrate, G. (2002). Phase Transitions and all that. *Arxiv preprint cs.CC/0211012*.

Jackson, D., Schechter, I., & Shlyakhter, I. (2000). Alcoa: the Alloy constraint analyzer. *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, (pp. 730–733).

Jarvisalo, M., & Niemela, I. (2004). A Compact Reformulation of Propositional Satisfiability as Binary Constraint Satisfaction. *Modelling and Reformulating Constraint Satisfaction Problems*, (pp. 111–124).

Jeavons, P., Cohen, D., & Cooper, M. C. (1998). Constraints, consistency and closure. *Artif. Intell.*, *101*(1-2), 251–265.

Jeavons, P., Cohen, D., & Gyssens, M. (1997). Closure properties of constraints. *Journal of the ACM*, *44*(4), 527–548.
URL citeseer.ist.psu.edu/jeavons97closure.html

Jeavons, P., & Cooper, M. C. (1995). Tractable Constraints on Ordered Domains. *Artificial Intelligence*, *79*(2), 327–339.
URL citeseer.ist.psu.edu/jeavons95tractable.html

Jeavons, P. G., Cohen, D. A., & Gyssens, M. (1996). A Test for Tractability. In *Proceedings 2nd International Conference on Constraint Programming—CP'96 (Boston, August 1996)*, vol. 1118, (pp.

267–281). Springer-Verlag.
URL citeseer.ist.psu.edu/jeavons96test.html

Jegou, P. (1993). Decomposition of domains based on the micro-structure of finite constraint satisfaction problems. *Proceedings of the 11th (US) National Conference on Artificial Intelligence (AAAI-93)*, (pp. 731–736).

Jussila, T., Sinz, C., & Biere, A. (2006). Extended resolution proofs for symbolic SAT solving with quantification. *9th Intl. Conf. on Theory and Applications of Satisfiability Testing*, *4121*, 54–60.

Karp, R. (1972). Reducibility among combinatorial problems. *Complexity of Computer Computations*, *43*, 85–103.

Kasif, S. (1990). On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, *45*(3), 275–286.

Kask, K., & Dechter, R. (1995). GSAT and local consistency. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, (pp. 616–622).

Krajicek, J., & Pudlak, P. (1989). Propositional Proof Systems, the Consistency of First Order Theories and the Complexity of Computations. *The Journal of Symbolic Logic*, *54*(3), 1063–1079.

Kulikov, A. (2005). An upper bound O (2 0.16254 n) for exact 3-satisfiability: a simpler proof. *Journal of Mathematical Sciences*, *126*(3), 1195–1199.

Kullmann, O. (1999). On a generalization of extended resolution. *Discrete Applied Mathematics*, *96*, 149–176.

Kullmann, O. (2004). Upper and Lower Bounds on the Complexity of Generalised Resolution and Generalised Constraint Satisfaction Problems. *Annals of Mathematics and Artificial Intelligence*, *40*(3), 303–352.

Lecoutre, C., Boussemart, F., & Hemery, F. (2003). Exploiting multidirectionality in coarse-grained arc consistency algorithms. *Proceedings of CP03*, (pp. 480–494).

Ludden, J., Roesner, W., Heiling, G., Reysa, J., Jackson, J., Chu, B., Behm, M., Baumgartner, J., Peterson, R., Abdulhafiz, J., et al. (2002). Functional verification of the POWER 4 microprocessor and POWER 4 multiprocessor systems. *IBM Journal of Research and Development*, *46*(1), 53–76.

Lynce, I., & Marques-Silva, J. (2001). The interaction between simplification and search in propositional satisfiability. In *CP2001 Workshop on Modelling and Problem Formulation (Formul01)*.

MacIntyre, E., Prosser, P., Smith, B., & Walsh, T. (1998). Random constraint satisfaction: theory meets practice. *Proceedings of CP-98, Pisa, Italy*, *19*, 325–339.

Mackworth, A. (1975). Consistency in Networks of Relations. Tech. rep., University of British Columbia Vancouver, BC, Canada, Canada.

Mackworth, A., & Freuder, E. (1985). The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, *25*(1), 65–74.

Madsen, B., & Rossmanith, P. (2004). Maximum exact satisfiability: NP-completeness proofs and exact algorithms. Tech. rep., BRICS.

Mann, C. (2000). The End of Moores Law. *Technology Review–MITs Magazine of Innovation, May/June*.

Marques-Silva, J., & Lynce, I. (2007). Towards Robust CNF Encodings of Cardinality Constraints. In *International Conference on Principles and Practice of Constraint Programming*, vol. 4741, (p. 483). Springer.

McAllester, D., Selman, B., & Kautz, H. (1997). Evidence for invariants in local search. *Proceedings of AAAI*, *97*, 321–326.

Mitchell, D. (1998). Hard problems for CSP algorithms. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, (pp. 398–405).

Mitchell, D. (2002). *The Resolution Complexity of Constraint Satisfaction*. Ph.D. thesis, University of Toronto.

Mitchell, D. (2003). Resolution and constraint satisfaction. *CP*, *2833/2003*, 555–569.

Mitchell, D., & Levesque, H. (1996). Some pitfalls for experimenters with random SAT. *Artificial Intelligence*, *81*(1-2), 111–125.

Mitchell, D. G., Selman, B., & Levesque, H. J. (1992). Hard and Easy Distributions for SAT Problems. In P. Rosenbloom, & P. Szolovits (Eds.) *Proceedings of the Tenth National Conference on Artificial Intelligence*, (pp. 459–465). Menlo Park, California: AAAI Press.
URL `citeseer.ist.psu.edu/mitchell92hard.html`

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., & Malik, S. (2001). Chaff: engineering an efficient SAT solver. *Proceedings of the 38th conference on Design automation*, (pp. 530–535).

Nudelman, E., Leyton-Brown, K., Hoos, H., Devkar, A., & Shoham, Y. (2004). Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. *Principles and Practice of Constraint Programming–CP 2004: 10th International Conference, CP 2004, Toronto, Canada, September 27-October 1, 2004: Proceedings*.

Paris, L., Benhamou, B., & Siegel, P. (2006). A Boolean Encoding Including SAT and n-ary CSPs. *Artificial Intelligence: Methodology, Systems, and Applications, 4183*, 33–44.

Parkes, A. (1999). *Lifted Search Engines for Satisfiability*. Ph.D. thesis, University of Oregon Eugene, OR, USA.

Piette, C., Hamadi, Y., & Sais, L. (2008). Vivifying propositional clausal formulae. Tech. rep., MSR-TR-(to appear) Microsoft Research (april 2008).

Porschen, S., Randerath, B., & Speckenmeyer, E. (2002). X3SAT is Decidable in Time O (2 n= 5). In *Proceedings of the Fifth International Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*, (pp. 231–235).

Prasad, M., Biere, A., & Gupta, A. (2005). A survey of recent advances in SAT-based formal verification. *International Journal on Software Tools for Technology Transfer (STTT), 7*(2), 156–173.

Prestwich, S. (2003). Local search on sat-encoded csps. *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT-03)*.

Prosser, P. (1996). Empirical study of phase transitions in binary constraint satisfaction problems. *Artificial Intelligence, 81*(1), 81–109.

Raz, R. (2001). Resolution lower bounds for the weak pigeonhole principle. *Journal of the ACM (JACM), 51*(2), 115–138.

Razborov, A. A. (2001). Improved Resolution Lower Bounds for the Weak Pigeonhole Principle. *Electronic Colloquium on Computational Complexity (ECCC), 8*(55).
URL `citeseer.ist.psu.edu/article/razborov01improved.html`

Régin, J. (1994). A filtering algorithm for constraints of difference in CSPs. *Proceedings of the twelfth national conference on Artificial intelligence (vol. 1) table of contents*, (pp. 362–367).

Rish, I., & Dechter, R. (2000). Resolution versus Search: Two Strategies for SAT. *Journal of Automated Reasoning, 24*(1/2), 225–275.
URL `citeseer.ist.psu.edu/article/rish00resolution.html`

Robinson, J. (1965). A Machine-Oriented Logic Based on the Resolution Principle. *Journal of the ACM (JACM), 12*(1), 23–41.

Rossi, F., Petrie, C., & Dhar, V. (1990). On the Equivalence of Constraint Satisfaction Problems. In L. C. Aiello (Ed.) *ECAI'90: Proceedings of the 9th European Conference on Artificial Intelligence*, (pp. 550–556). Stockholm: Pitman.
URL citeseer.ist.psu.edu/rossi90equivalence.html

Rossi, F., Van Beek, P., & Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier Science.

Roussel, O. (2004). Another SAT to CSP Conversion. *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04)-Volume 00*, (pp. 558–565).

Schaefer, T. J. (1978). The complexity of satisfiability problems. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, (pp. 216–226). New York, NY, USA: ACM Press.

Schaerf, A. (1999). A Survey of Automated Timetabling. *Artificial Intelligence Review*, *13*(2), 87–127.

Selman, B., Kautz, H., & McAllester, D. (1997). Ten challenges in propositional reasoning and search. *Proc. IJCAI*, *97*, 5054.

Selman, B., Levesque, H. J., & Mitchell, D. (1992). A New Method for Solving Hard Satisfiability Problems. In P. Rosenbloom, & P. Szolovits (Eds.) *Proceedings of the Tenth National Conference on Artificial Intelligence*, (pp. 440–446). Menlo Park, California: AAAI Press.
URL citeseer.ist.psu.edu/selman92new.html

Shacham, O., & Yorav, K. (2006). Adaptive Application of SAT Solving Techniques. *Electronic Notes in Theoretical Computer Science*, *144*(1), 35–50.

Sinz, C. (2005). Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP, 2005*.

Sinz, C., & Biere, A. (2006). Extended resolution proofs for conjoining BDDs. *First International Computer Science Symposium in Russia*, (pp. 600–611).

Sipser, M. (1992). The history and status of the P versus NP question. In *Proceedings of the 24th ACM Symposium on Theory of Computing*, (pp. 603–618).
URL citeseer.ist.psu.edu/sipser92history.html

Smith, B. (1993). *The Phase Transition in Constraint Satisfaction Problems: A Closer Look at the Mushy Region*. University of Leeds, School of Computer Studies.

Smith, B., & Dyer, M. (1996). Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, *81*(1), 155–181.

Smith, B., Grant, S., of Leeds, U., & of Computer Studies, S. (1995). *Where the Exceptionally Hard Problems are*. University of Leeds, School of Computer Studies.

Smith, B., Stergiou, K., & Walsh, T. (2000). Using auxiliary variables and implied constraints to model non-binary problems. *Proceedings of the 16th National Conference on AI*, (pp. 182–187).

Stergiou, K., & Walsh, T. (1999). Encodings of non-binary constraint satisfaction problems. *Proceedings of AAAI-99*, (pp. 163–168).

Subbarayan, S., & Pradhan, D. (2004). NiVER: Non increasing variable elimination resolution for preprocessing SAT instances. *Proc. 7th International Conference on Theory and Applications of Satisfiability Testing (SAT)*.

Toran, J. (2004). Space and width in propositional resolution. *Bulletin of the European Association for Theoretical Computer Science*, *83*, 86–104.

Trakhtenbrot, B. A. (1984). A survey of Russian approaches to perebor (brute-force search) algorithms. *Annals of the History of Computing*, *6*(4), 384–400. Partial English translation of L. Levin, *Universal Search Problems*, 9(3), pp. 265–266, (1973).
URL `http://dlib.computer.org/an/books/an1984/pdf/a4384.pdf`

Tsang, E. (1993). *Foundations of Constraint Satisfaction*. London: Academic Press.
URL `http://cswww.essex.ac.uk/CSP/papers/Tsang-Fcs1993.pdf/`

Tseitin, G. (1968). On the complexity of derivation in propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, *2*, 115–125.

Urquhart, A. (1987). Hard examples for resolution. *Journal of the ACM (JACM)*, *34*(1), 209–219.

Urquhart, A. (1995). The Complexity of Propositional Proofs. *The Bulletin of Symbolic Logic*, *1*(4), 425–467.

van Beek, P., & Dechter, R. (1994). *Constraint Tightness Versus Global Consistency*. Information and Computer Science, University of California, Irvine.

van Beek, P., & Dechter, R. (1997). Constraint tightness and looseness versus local and global consistency. *Journal of the ACM (JACM)*, *44*(4), 549–566.

Van Gelder, A. (2006). Preliminary Report on Input Cover Number as a Metric for Propositional Resolution Proofs. *Theory and applications of satisfiability testing*, *4121*, 48–53.

Van Gelder, A. (2008). Another look at graph coloring via propositional satisfiability. *Discrete Applied Mathematics*, *156*(2), 230–243.

Wallace, M. (1996). Practical applications of constraint programming. *Constraints*, *1*(1), 139–168.

Walsh, T. (2000a). Reformulating propositional satisfiability as constraint satisfaction. *Symposium on Abstraction, Reformulation and Approximation (SARA), 1864/2000*, 233–246.

Walsh, T. (2000b). SAT v CSP. *Proceedings CP*, *2000*, 441–456.

Waltz, D. (1975). Understanding line drawings of scenes with shadows. In P. H. Winston (Ed.) *The Psychology of Computer Vision*. New York: McGraw-Hill.
URL `citeseer.ist.psu.edu/waltz75understanding.html`

Xu, L., Hutter, F., Hoos, H., & Leyton-Brown, K. (2007). SATzilla-07: The Design and Analysis of an Algorithm Portfolio for SAT. *Solver description, SAT competition*.

Yang, J. (2005). *Finding k-cliques on a k-partite Graph*. Ph.D. thesis, National Dong Hwa University.

Yokoo, M. (1997). Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. *Lecture notes in computer science*, (pp. 356–370).

Zhang, L. (2003). *Searching for Truth: Techniques for Satisfiability of Boolean Formulas*. Ph.D. thesis, Princeton University.