Aspects of a Constraint Optimisation Problem

by

Johan Thapper
Abstract

In this thesis we study a constraint optimisation problem called the maximum solution problem, henceforth referred to as MAX SOL. It is defined as the problem of optimising a linear objective function over a constraint satisfaction problem (CSP) instance on a finite domain. Each variable in the instance is given a non-negative rational weight, and each domain element is also assigned a numerical value, for example taken from the natural numbers. From this point of view, the problem is seen to be a natural extension of integer linear programming over a bounded domain. We study both the time complexity of approximating MAX SOL, and the time complexity of obtaining an optimal solution. In the latter case, we also construct some exponential-time algorithms.

The algebraic method is a powerful tool for studying CSP-related problems. It was introduced for the decision version of CSP, and has been extended to a number of other problems, including MAX SOL. With this technique we establish approximability classifications for certain families of constraint languages, based on algebraic characterisations. We also show how the concept of a core for relational structures can be extended in order to determine when constant unary relations can be added to a constraint language, without changing the computational complexity of finding an optimal solution to MAX SOL. Using this result we show that, in a specific sense, when studying the computational complexity of MAX SOL, we only need to consider constraint languages with all constant unary relations included.

Some optimisation problems are known to be approximable within some constant ratio, but are not believed to be approximable within an arbitrarily small constant ratio. For such problems, it is of interest to find the best ratio within which the problem can be approximated, or at least give some bounds on this constant. We study this aspect of the (weighted) MAX CSP problem for graphs. In this optimisation problem the number of satisfied constraints is supposed to be maximised. We introduce a method for studying approximation ratios which is based on a new parameter on the space of all graphs. Informally, we think of this parameter as an approximation distance; knowing the distance between two graphs, we can bound the approximation ratio of one of them, given a bound for the other. We further show how the basic idea can be implemented also for the MAX SOL problem.
Acknowledgements

I want to thank my supervisor Peter Jonsson for the opportunity to do my thesis work in his group, and for countless inspiring discussions. My studies were also in no small part influenced by Svante Linusson, who advised my licentiate thesis and gave me a certain combinatorial view on things.

For their contribution of ideas, encouragement, and perspective, I would further like to thank the following people:

- my co-authors: Ola, Peter, Gustav, Tommy, and Robert;
- Ola, Rustem, and Tommy, for reading and commenting on parts of the manuscript;
- Tommy for many interesting discussions and encouragement;
- everyone at TCSLAB for the open and creative work atmosphere.

Much gratitude also goes to my friends and family, and to Aurélie for her endless love and support, and for her patience in me during these last years.

Linköping, November 2009
List of papers

Parts of this thesis have previously been published in the following refereed papers:


- Peter Jonsson and Johan Thapper. Approximability of the maximum solution problem for certain families of algebras. In Anna Frid, Andrey Morozov, Audrey Rybalchenko, and Klaus W. Wagner, editors,

Additionally, the results from the following paper are also included:  

Peter Jonsson and Johan Thapper. Approximating integer programs with positive right-hand sides. Information Processing Letters, submitted. Accepted subject to minor revisions. [87]
Contents

I Introduction 1

1 Introduction 3
  1.1 Constraint satisfaction .......................... 10
  1.2 Optimisation .................................. 14
    1.2.1 Example: Maximum Independent Set ....... 14
    1.2.2 Example: Max cut .......................... 15
  1.3 Approximability ................................. 16
  1.4 The maximum solution problem ................. 17
  1.5 Methods and results ............................ 18
  1.6 Outline of the thesis ......................... 22

2 Preliminaries 25
  2.1 Basic definitions and notation ................ 25
    2.1.1 Graphs .................................. 25
  2.2 Constraint satisfaction ........................ 28
    2.2.1 Constraint languages and satisfying assignments .... 28
    2.2.2 Relational structures and homomorphisms ........ 29
  2.3 The maximum solution problem ................. 29
  2.4 Approximability and reductions ............... 30
    2.4.1 Some complete problems ................... 33
    2.4.2 The unique games conjecture ............... 33
  2.5 The algebraic method .......................... 34
  2.6 Post's lattice ................................ 36

II Approximability 39

3 Affine Algebras and Approximability 41
  3.1 Universal algebra ............................... 44
    3.1.1 Endomorphism rings ........................ 45
  3.2 Affine algebras ................................ 45
  3.3 Linear equations over abelian groups ......... 48
    3.3.1 The case $M = 0 \times R\overline{A}$ ........ 51
    3.3.2 The case $M = R \times 0$ .................. 57
## Contents

3.4 Supplementary toolbox ........................................ 59  
3.4.1 Two-element domain ..................................... 61  
3.5 Strictly simple surjective algebras ...................... 62  
3.6 Symmetric algebras ................................. 65  
3.7 Discussion ............................................. 68  

4 Minimisation of Integer Linear Programs ............... 71  
4.1 Unbounded domain ..................................... 73  
4.1.1 Lower bounds .................................... 75  
4.2 Bounded domain ...................................... 76  
4.3 Discussion and future work ............................ 78  

5 Approximation Distance .................................. 79  
5.1 Approximation distance for \textsc{Max} $H$-Col .......... 83  
5.2 Properties of the parameter $s$ ....................... 84  
5.2.1 Exploiting symmetries ............................. 85  
5.2.2 A sandwiching lemma ............................... 88  
5.2.3 The space $(G_{\mathbb{Z}}, d)$ ....................... 90  
5.2.4 A linear program formulation .................... 91  
5.3 Circular complete graphs .............................. 94  
5.3.1 Maps to $K_2$ .................................... 97  
5.3.2 Constant regions .................................. 98  
5.3.3 Maps to odd cycles ............................... 99  
5.4 Approximation bounds for \textsc{Max} $H$-Col .......... 100  
5.4.1 Sparse graphs .................................... 102  
5.4.2 Dense and random graphs ......................... 105  
5.5 Fractional covering by $H$-cuts ....................... 108  
5.5.1 An upper bound ................................. 111  
5.5.2 Confirmation of a scale ........................... 112  
5.6 Approximation distance for \textsc{Max} $\text{Sol}$ ........ 114  
5.7 Discussion and open problems ....................... 118  

III Computational Complexity .......................... 123  

6 Constant Production .................................... 125  
6.1 Preliminaries ......................................... 127  
6.1.1 Weighted pp-definability ......................... 128  
6.1.2 The measure $m(f)$ as a bilinear form .......... 131  
6.1.3 Connected substructures ......................... 134  
6.2 Constant production .................................. 138  
6.3 Application: graphs .................................. 142  
6.3.1 Paths and cycles .................................. 145  
6.3.2 Graphs on at most three vertices ............... 149  
6.3.3 Proof of Lemma 6.40 ............................. 151  
6.3.4 Graphs on the vertex set $\{0, 1, 2, 3\}$ ....... 153
### Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
<td>Discussion</td>
<td>158</td>
</tr>
<tr>
<td>7</td>
<td>Conservative Constraint Languages and Arbitrary Weights</td>
<td>161</td>
</tr>
<tr>
<td>7.1</td>
<td>Basic reductions</td>
<td>163</td>
</tr>
<tr>
<td>7.2</td>
<td>Undirected graphs with loops</td>
<td>165</td>
</tr>
<tr>
<td>7.3</td>
<td>A complete classification</td>
<td>171</td>
</tr>
<tr>
<td>7.4</td>
<td>Conclusions</td>
<td>175</td>
</tr>
<tr>
<td>8</td>
<td>Exponential-Time Algorithms</td>
<td>177</td>
</tr>
<tr>
<td>8.1</td>
<td>The microstructure</td>
<td>180</td>
</tr>
<tr>
<td>8.2</td>
<td>The covering method</td>
<td>181</td>
</tr>
<tr>
<td>8.3</td>
<td>Algorithms for Max Sol</td>
<td>184</td>
</tr>
<tr>
<td>8.4</td>
<td>The Max Hamming Distance problem</td>
<td>189</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Algorithm for Max Hamming Distance(2, 2)-CSP</td>
<td>190</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Algorithm for Max Hamming Distance($d$, $l$)-CSP</td>
<td>200</td>
</tr>
<tr>
<td>8.5</td>
<td>Discussion and open problems</td>
<td>204</td>
</tr>
</tbody>
</table>

### Bibliography

| Page | 207 |
Part I

Introduction
Chapter 1

Introduction

In the research field of computational complexity theory one studies problems, and the efficiency with which these problems can be solved. Both of these terms come with non-standard interpretations which really have quite little to do with what one would normally consider “a problem” or “efficient”. Let us therefore begin with some examples in order to highlight this discrepancy, and to introduce some related vocabulary.

Problems, instances, and algorithms

Every day we encounter problems of different kinds. This may be anything from appropriately dividing our time and energy between a number of different tasks, or figuring out our current location on a map of a lesser known city, to determining the precise set of documents required in the application for a Russian visa. Some problems are easy, others are hard. Most of these problems do not lend themselves very well to formalisation into the sort of problems we will be dealing with; for us, it will be important that both the problem and a solution can be precisely described, and that the method with which we want to obtain the solution has a predictable outcome. To simplify things, we therefore narrow our discussion to problems which are in some sense “mathematical”, or “combinatorial”. As in all sciences, such idealised models and approximations of real-world problems always come equipped with limitations of applicability, but they have the advantage of being much more easily analysed in a formal setting. Let us have a look at the following familiar problem:

\[
\text{Solve for } x: \quad x^2 - x - 1 = 0
\]

From the viewpoint of complexity theory, this is in fact not a problem at all, but an instance of a problem. A problem consists of a collection of such instances. We solve a problem by means of an algorithm: a sequence of precise instructions which produces a solution to any instance of the problem. The quadratic equation above can be considered an instance of
the problem of solving arbitrary quadratic equations. This problem can be solved by means of the “completing the square” algorithm. To specify a problem we must specify the set of instances of the problem, but we also need to specify what constitutes a solution. In the aforementioned example, the algorithm will deliver two solutions:

\[ x = \frac{1 \pm \sqrt{5}}{2} \approx 1.6180, -0.6180 \]

If we are happy with solutions in radicals and elementary operations, then the first answer is acceptable, but since both solutions are irrational, we cannot produce a correct, finite, decimal expansion, for either of them. If we decide to settle for four correct decimal places, then the second answer is acceptable. In either case, we need to make sure that the form of the desired output is precisely specified, and we have to choose an appropriate computer representation for it. Similar considerations must be given to the representation of the instances.

We can of course define other problems which include the given equation \( x^2 - x - 1 = 0 \) as an instance. Consider the problem of solving an arbitrary polynomial equation, regardless of degree. We know that if we ask for the solutions in terms of radicals and elementary operations, then for polynomials of degree greater than four, we are at a loss. Here, we will have to settle for approximate solutions in the general case, even if we can obtain exact solutions for many such problems.

**Efficiency and resources**

The idea of considering the general, worst case appears again when we study the efficiency of our chosen algorithm. There are many ways of defining the efficiency of an algorithm, but they all involve restricting the resources which the algorithm needs to perform its task. Two of the main resources in modern day computers are time and memory. Either choice leads to a rich theory and poses many fascinating open problems, but we will restrict ourselves to the study of time. In fact, “time” is a slightly misleading term.

One generally starts out observing that the same algorithm will complete in different amounts of time on different computer hardware, or even when run twice on the same hardware, given today’s elaborate computer architectures. We remedy this by counting not the number of clock cycles it takes to perform a task, but rather the number of instructions, or steps, which are carried out by the algorithm. One therefore settles for some ‘reasonable’ formal model of computation, and regards differences in multiplicative and additive constants between such models as uninteresting.

In principle, we could now study a function \( t \) which assumes the number of steps a specific algorithm will need, given an instance of the problem. This would probably become a very complicated function to determine, even for simple problems, and in addition, it might not tell us much of what we want to know. Often we are interested in solving “larger” instances as we
1. Introduction

gather more information, or as to better approximate some real-world problem. Thus, it would make more sense to study the number of steps needed for the algorithm to complete its computation as a function of the size of the instance. The size of an instance is a function which is directly related to the concrete representation that has been chosen for the problem. We can formally define it as the number of bits needed to encode the instance, but we will usually settle for a function which grows asymptotically as the actual size. The function \( t \) is then a function from natural numbers, representing the size of the instance, to natural numbers, representing the \textit{maximal} number of steps required to solve an instance of a specific size. This is called a \textbf{worst case} analysis; there may be instances of a given size \( n \) which the algorithm can solve very quickly, but \textit{in general} it can do no better than \( t(n) \). Also note that \( t \) may be a partial function since there may be no instances at all for some particular values of \( n \). Now, we are not actually interested in the exact values which \( t \) assumes, but rather in how it behaves as \( n \) grows large. This is formalised in the following asymptotic notation. For partial functions \( t \) and \( g \), we write

\[ t(n) \in \mathcal{O}(g(n)), \]

if there are constants \( C \) and \( n_0 \) such that

\[ t(n) \leq C \cdot g(n) \text{ for all } n > n_0. \]

This notation, at the same time, takes care of reasonable approximations in the choice of instance size, and the uninteresting additive and multiplicative constants due to the computational model. For notational convenience, we write \( 't \in \mathcal{O}(n^2)' \) instead of \( 't(n) \in \mathcal{O}(g(n))' \) for a function \( g(n) = n^2 \). In this case, we also say that the \textbf{time complexity} of the algorithm associated to \( t \) is \( \mathcal{O}(n^2) \).

\textbf{Example 1.1} Consider again the problem of solving general quadratic equations. A quadratic equation can be given on a standard form as:

\[ ax^2 + bx + c = 0. \]

An instance can thus be encoded by three numbers: \( a, b, \) and \( c \). If we assume that \( a, b, \) and \( c \) are rational coefficients, each given by the ratio of two relatively prime integers, then the number of bits to represent the instance is approximately the sum of the 2-logarithms of the absolute values of the six integers. Since we are disregarding constants, we can actually let the size be the 2-logarithm of the largest among the absolute values of the six integers. If we choose to produce an expression in radicals and elementary operations, then essentially all operations of the algorithm are a fixed number of additions, subtractions, and multiplications. It is reasonable to assume that additions and subtractions can be carried out in a number of steps which grows linearly as the size of the operands involved, and that multiplications can be performed in a quadratic number of steps. It follows
that for this algorithm, \( t(n) \in \mathcal{O}(n^2) \); we say that the algorithm runs in quadratic time. It is also common to count the number of arithmetical operations, as opposed to the number of bit operations. In this case we would say that the previous algorithm runs in constant time. Normally, we are much less specific in how we represent instances of our problems, relying heavily on accepted conventions, and the absorption of constants into the \( \mathcal{O} \)-notation.

‘How about a nice game of chess?’

The game of chess takes place on an idealised battlefield where the opponents practise their strategic and tactical skills. It is the archetypal “war game”. A chess problem, or puzzle, is a prespecified position given on a chess board, often accompanied by a short instruction on the form ‘white to play and win’. The objective of the problem is to come up the the best move in the given situation. For the ‘...to play and win’-instances, we can define a best move as one which always leads to a win for the player. An example puzzle is given in Figure 1.1. This is clearly in line with what we mean by a problem; who is

![Chessboard](image)

Figure 1.1: White to play and win in 2.

to play, the legal positions, and the moves between them are all well-defined and easy to encode in a computer readable format. However from the point of view of computational complexity theory, the problem as stated is trivial: it can be solved by an algorithm which runs in a constant number of steps. This was observed already in 1950 by Claude Shannon [125], where he gives a rough estimate on the number of possible positions in chess to about \( 10^{43} \).

\(^3\)This particular example was composed by Susan Polgár, one of the famous Polgár sisters, at the age of 4.
1. Introduction

We may want to consider also some positions which can never occur from normal play but which still pose interesting chess problems. In any case, since this number is bounded by some (admittedly very large) constant, the number of possible instances to a chess puzzle-solving algorithm is constant. But then, we can simply hard-code each position and its solution into one gigantic table. This table may take a long time to produce, and it would hardly fit into any existing computer memory, but the point is that it is finite in size and could, in principle, be constructed. The asymptotic growth of the number of steps required for an algorithm, which given a puzzle looks up the answer in this table is thus also constant: there is some puzzle which takes the most number of computational steps to “solve”. The chess puzzle problem can be turned into a (complexity theoretically) non-trivial problem, for example by having it played on a generalised $n$ times $n$ chess board.

Linear programming

We now take a look at perhaps the most well-known problem in combinatorial optimisation: linear programming. An instance of linear programming is specified by a vector $c$ of size $n$, a vector $b$ of size $m$, and an $m \times n$ matrix $A$. All entries in the vectors and in the matrix are supposed to be rational numbers. The linear program corresponding to this instance can be written as follows:

\[
\begin{align*}
\text{Maximise} \quad & c^T x \\
\text{subject to} \quad & Ax \leq b, \\
& x \geq 0.
\end{align*}
\]

The entries in the vector $x = (x_1, \ldots, x_n)^T$ are called variables, and the aim is to assign rational values to the variables so that $x$ resides in the polytope specified by the inequalities, and so that the objective function $f(x) = c^T x$ is maximised.

The classic algorithm for solving linear programs is the simplex algorithm, described by George Dantzig in 1947. (First mention of the term ‘linear programming’ seems to be Dantzig [36]. See also [37].) The simplex algorithm performs very well in practice, and simplex-based algorithms are still at the heart of many linear program solvers today. The algorithm can roughly be described as follows: a first observation is that either the linear objective function can grow arbitrarily, in which case a maximal solution does not exist, or it must attain its maximum in a vertex of the polytope. Furthermore, from a non-optimal vertex, we can always reach a better vertex along some edge of the polytope. The idea is therefore to pick a starting vertex, and to walk along the edges of the polytope, from vertex to vertex, always in a direction which increases the objective function. Unfortunately, this strategy has an, at least theoretical, draw-back; it is possible to construct a polytope, a slightly “twisted” version of the hypercube [97], and pick a starting vertex such that the simplex algorithm will walk through
every vertex of the polytope before reaching the maximum. This construction works in any dimension, and in this fashion we can produce an infinite sequence of polytopes whose vertex count grow exponentially with the size of the instances. The worst case time complexity of the simplex algorithm is therefore exponential, and the algorithm, from our point of view, cannot be considered efficient at all.

It took over three decades, and a completely different approach, to come up with a complexity theoretically satisfying improvement (during which time, of course, the simplex algorithm continued to solve an ever increasing number of instances in practical applications all over the world.) In 1979, Leonid Khachiyan presented the first polynomial-time algorithm for linear programming, based on the ellipsoid method introduced by Nemirovsky and Yudin, and independently by Shor, in the 1970s. Paradoxically, this method proved to perform worse in practice than the simplex algorithm. Advances in science, however, must be judged both on their immediate practical impact as well as on the ideas and possibilities they bring to light, and the importance of this result is hard to overstate. Today, interior point methods, introduced by Karmarkar [90], provide linear program solvers which both perform well in practice, and run in worst case polynomial time.

Now that we have a better understanding of what we mean by a problem, and how we measure efficiency, we will move on to the most famous question in computer science, compactly stated \( P \neq NP \).

**Complexity classes, reductions and completeness**

One of the interests of the computational complexity theorist is to decide upon some relevant measure of efficiency, and then to try and express in some more or less explicit way which problems can (and often more importantly, which problems \textit{cannot}) be solved with this efficiency. The collection of all such problems constitutes what we call a \textit{complexity class}. The inclusion of a problem in a complexity class can be proved by giving an algorithm for the problem, and to prove that the algorithm is efficient in the right sense.

We begin by considering only \textbf{decision problems}, which are problems where the set of solutions is taken to be \{YES, NO\}. The most basic class of problems is called \textbf{P}, which stands for \textit{polynomial time}. This is the class of decision problems for which we can write an algorithm that, in the worst case, runs in a number of steps that is bounded by a polynomial in the size of the input. In this class we find problems such as \textit{Is x a prime number?}, and deciding whether there is a connected path in a network from one point to another. Many problems are more naturally stated as computing some function value in a larger domain than \{YES, NO\}. For example, the greatest common divisor of two integers can be computed in polynomial time using Euclid’s algorithm. Such problems formally belong to a class called \textbf{FP}, but can usually be turned into decision problems by a reformulation, e.g.
1. Introduction

"Is the greatest common divisor of \(a\) and \(b\) greater than or equal to \(c\)?" It is generally considered that if a problem cannot be solved in a polynomial number of steps, then as the size of the input grows, it will quickly become infeasible to obtain a solution, even with tomorrow's computers\(^2\). We have seen that linear programming falls into this class, although it was not proved before 1979. We have also seen that even though the simplex algorithm in general cannot be guaranteed to run within a polynomial number of steps, it is nonetheless the algorithm of choice over the ellipsoid method for this problem. This issue seems to arise partly from the fact that we study worst case complexity. The simplex algorithm may indeed run in an exponential number of steps for a cleverly contrived input, but such inputs do not occur frequently in practice. It is worth mentioning that the worst case complexity approach is not the only one taken, and that, for example, the study of average case complexity is a growing and interesting field in its own right. Nevertheless, the distinction between problems inside and outside of \(P\), and the continued effort to try and understand this boundary, has proved extraordinarily fruitful, both from a theoretical and practical point of view.

The second class which we will look at is called \(NP\), which stands for non-deterministic polynomial time. It can be roughly defined as the class of (decision) problems for which a correct solution can be checked in a number of steps which is bounded by a polynomial in the size of the solution\(^3\).

The class \(NP\) is known to contain, in addition to all of \(P\), also quite a few problems which are believed not to be in \(P\). To return to our chess example, although it did not constitute a very interesting problem from our point of view, we intuitively find that it should be "easier", in some sense, to verify a correct solution than to find one. Yet, almost 40 years after the conception of the class \(NP\) \cite{29,104}, we still have no clue how to decide whether or not \(P \neq NP\), i.e. whether it is really harder to solve a problem in \(NP\) from scratch, than it is to verify a solution which is given to us. Of course, we strongly believe that it is harder, and it is not a controversial hypothesis to build one's work upon. Still, since most of this thesis is about tracing this boundary between easy and hard, if it were to turn out that \(P = NP\), i.e. that there is no boundary at all, then many of our results would be rendered null and void.

One of the reasons why the question \(P \neq NP\)? seems so hard to answer is that a proof seems to require an explicit problem in \(NP\) which is not a member of \(P\), and a proof of such a property must necessarily consider all possible known and unknown algorithms for solving this particular problem. There is however a technique for singling out problems which are "harder" than others (or at least "not easier").) The basic components in this technique has become known as reductions, and the study of complexity theory is so much built around them that it could, with some accuracy, be called the

\(^2\)However, quantum computing may put a dent in this long held "truth".

\(^3\)The size of the solution is itself supposed to be polynomially bounded by the size of the input.
1.1. Constraint satisfaction

In this thesis we consider a particular class of problems called constraint satisfaction problems, or Csp. The formal definition of a Csp is given in the preliminaries, Section 2.2. For the purposes of this introduction, it will suffice with an informal description, which is given after the following example.

Example 1.3 The structure in Figure 1.2 is known as a (combinatorial) graph. Such a graph consists of a set of points, called vertices, and a set of lines connecting pairs of points, called edges. They can be used to model relationships between various entities. As an example, we can think of the graph in Figure 1.2 as a model of live variables in a computer language.

The Complexity Zoo is found at http://complexityzoo.uw.edu.
compiler. The register allocation step in a compiler assigns physical CPU registers to the variables. In the graph, we let the vertices represent the variables, and an edge between two vertices represents the fact that their live ranges overlap, i.e. there is some point during the execution of the program when both variables may carry important values. An edge between two vertices thus implies that these two vertices may not be allocated to the same physical register.

A proper vertex colouring of a graph is an assignment of “colours” to the vertices of the graph so that no two vertices which share an edge are assigned the same colour. In our example, the colours represent the physical registers, and a proper colouring corresponds to a realisable variable-register allocation. We will use the set \{\text{REG1, REG2, REG3}\} as colours, and show that three physical registers suffice to allocate all ten variables of the Petersen graph. The following assignment \( f \) is a proper colouring:

\[
\begin{align*}
    f(u_1) &= f(u_2) = f(v_3) = f(v_5) = \text{REG1} \\
    f(u_3) &= f(u_4) = f(v_2) = \text{REG2} \\
    f(u_5) &= f(v_1) = f(v_4) = \text{REG3}
\end{align*}
\]

The problem 3-COLOURING is that of deciding, for an arbitrary input graph, whether there exists a proper colouring using no more than three distinct colours. The 3-COLOURING problem is thus a decision problem, and it is a standard example of an NP-complete graph problem.
1.1. Constraint satisfaction

Informal definition

A CSP instance is determined by three components, a set of variables, a set of values, and a set of constraints. The set of variables, which we will denote by \( V \), is in the case of Example 1.3 given by the set of vertices of the graph: \( \{ u_1, \ldots, u_5, v_1, \ldots, v_5 \} \). We also need a set of values, or domain elements. This set, which we will denote by \( D \), is called the domain. In the example, it is given by \( \{ \text{reg1}, \text{reg2}, \text{reg3} \} \). Finally, we need a set of constraints, which we will denote by \( C \). Constraints are local conditions on the values allowed for each variable. The constraints in Example 1.3 are given by the rule that no two vertices sharing an edge may have the same colour. For a specific graph, this rule can be modelled in many different ways. It turns out that the most natural way to have each constraint apply only to a single pair of vertices, e.g., \( u_1 \) and \( u_3 \) must be assigned different colours. The example then contains 15 different constraints, one for each edge in the graph.

A function \( f : V \to D \) is called an assignment. The assignment \( f \) satisfies a constraint \( c \in C \) if it gives values to the variables in such a way that the given constraint \( c \) holds. In the example, each constraint \( c \) applies to two vertices, say \( u_1 \) and \( u_2 \), so we can verify that \( f \) satisfies \( c \) by checking that \( f(u_1) \neq f(u_2) \). A satisfying assignment, or solution, to a CSP instance is an assignment which satisfies all of the constraints in \( C \) simultaneously. In the 3-Colouring example, the satisfying assignments are in one-to-one correspondence with proper colourings.

For a given CSP instance, and a suggested assignment \( f \), it is easy to check, in polynomial time, whether or not \( f \) is satisfying. Hence CSP is in the class \( \text{NP} \), and from Example 1.3 we conclude that the problem of deciding whether or not a given CSP instance has a satisfying assignment is \( \text{NP} \)-complete.

Parametrisation and the dichotomy conjecture

Since the CSP framework can be used to model so many interesting and important problems, but is \( \text{NP} \)-complete in general, one can try to restrict its power in different ways. One possibility is to restrict the size of the domain \( D \) to obtain a parametrised problem in which any instance must satisfy \( |D| \leq k \). Unfortunately even the problem CSP restricted to a two-element, or Boolean, domain is \( \text{NP} \)-complete. It contains the problem of deciding the satisfiability of a propositional logic formula, which was among the first problems shown to be \( \text{NP} \)-complete. A different restriction on CSP is to allow only certain types of constraints. For example, we have seen that 3-Colouring can be described with constraints of the form \( f(u) \neq f(v) \). The binary relation \( \neq \) on the domain \( D \) is thus the only type of constraint we need, and the restriction of CSP to only such constraints will be denoted CSP(\( \{ \neq \} \)). In general, we can take any set \( \Gamma \) of constraints and let CSP(\( \Gamma \)) denote the restriction of CSP where only constraints from the set \( \Gamma \) are
1. Introduction

allowed. Schaefer [119] managed to classify the restriction of Csp over a Boolean domain \( \text{any} \) fixed set of constraints \( \Gamma \) as being either in \( P \), or being \( \text{NP-complete} \). This dichotomy theorem is quite remarkable. By Ladner’s theorem [101], we know that, unless \( P = \text{NP} \), there are problems which are neither in \( P \), nor in \( \text{NP} \). Yet, the Csp problem over a Boolean domain contains only two types of problems: easy and hard.

A similar phenomenon occurs for another class of Csp, this time related to graphs. A graph homomorphism from a graph \( G \) to a graph \( H \) is a function from the vertices of \( G \) to the vertices of \( H \) such that the image of each edge in \( G \) is an edge in \( H \). The graph homomorphism problem parametrised by a graph \( H \), and denoted \( \text{Hom}(H) \), is the problem of deciding, for a given graph \( G \), whether there exists a graph homomorphism from \( G \) to \( H \). In particular, the problem \( \text{Hom}(K_3) \), where \( K_3 \) denotes the complete graph on three vertices, is nothing more than the 3-COLOURING problem in disguise. Hell and Nešetřil [71] showed that for undirected graphs, \( \text{Hom}(H) \) is in \( P \) if \( H \) is either bipartite, or if it has a loop. In all other cases, \( \text{Hom}(H) \) is \( \text{NP-complete} \). Note that the case \( H = K_3 \) is not bipartite, and has no loop, hence \( \text{Hom}(K_3) \) is \( \text{NP-complete} \). This is in agreement with our claim in Example 1.3 that 3-COLOURING is \( \text{NP-complete} \).

Guided by the occurrence of such a dichotomy in two important subclasses of Csp, Feder and Vardi [53] conjectured that Csp(\( \Gamma \)) in general must exhibit a dichotomy. Their argument was based on a study of a class named MMSNP (monotone monadic SNP without inequalities). They showed that if either of the three restrictions monotonicity, monadicity, or no inequalities is removed from MMSNP, the resulting class is polynomial-time equivalent to all of \( \text{NP} \). In this sense they identified MMSNP as the largest subclass of \( \text{NP} \) which can potentially exhibit a dichotomy. They further showed that Csp and MMSNP are polynomial-time equivalent. This means that while in a strict sense Csp is a subclass of MMSNP, there exists a polynomial-time reduction from every problem in MMSNP to a problem in Csp. Thus they asked the question: Is every problem in Csp either in \( P \) or \( \text{NP-complete} \)? The conjecture that this is indeed the case has become known as ‘the dichotomy conjecture’:

**Conjecture 1.4 (The Dichotomy Conjecture)** Given a family of constraints \( \Gamma \), the problem Csp(\( \Gamma \)) is either in \( P \), or it is \( \text{NP-complete} \).

The dichotomy conjecture has inspired much of the theoretical research on the constraint satisfaction problem. Progress towards settling Conjecture 1.4 has largely been driven by a method which has become known as the algebraic approach, or the algebraic method. An early and influential paper in this direction is Jeavons, Cohen, and Gyssens [79]. Bulatov, Jeavons and Krokhin [22] use characterisations from universal algebra to study a special case of the full conjecture. Other important cases which have been solved include maximal constraint languages [23], conservative constraint
languages [18], constraint languages over a three-element domain [20], and a polynomial-time algorithm for so-called Mal'tsev constraints [21].

1.2 Optimisation

To merely know the existence of a solution to a given problem may be insufficient. One may wish to obtain the “best” solution, in some specific sense. This is the case in particular for linear programming, where the existence of a solution only asserts that the polytope specified by the inequalities is non-empty, while the goal generally is to optimise an objective function over the polytope.

Similar need for optimisation occurs for the solution spaces of general constraint satisfaction problems. There are several approaches to introducing an optimisation criterion on CSP instances. We will primarily be concerned with the following two types of criteria:

1. One can assign weights to the variables and domain elements and introduce an objective function based on these weights.

2. One can allow constraints to be unsatisfied in a solution, and base the objective function on the number of satisfied constraints.

For both of these approaches, the resulting problem is in general not easier than its non-optimising counterpart. In the first case, if we can find an optimal solution to a particular problem using some restricted resources, then we can clearly find one. In the second case, there are solutions to the decision problem if and only if a maximal solution satisfies all constraints. We will now illustrate these two approaches with two examples, the problems **Maximum Independent Set** and **Max cut**. Both examples also illustrate that an optimisation problem can be strictly harder (in this case provided $P \neq NP$) than the original problem.

1.2.1 Example: **Maximum Independent Set**

A subset $S$ of the vertices of a graph is called an **independent set** if there is no edge between any pair of vertices $u, v \in S$. For the Petersen graph in Figure 1.2 it is easy to find an independent set of size four, e.g. the set \( \{u_1, u_2, v_3, v_5\} \), which is the set of vertices of “colour” \textcolor{red}{\text{red}} in Example 1.3. We also see that it is not possible to choose more than two vertices from the outer pentagon, nor more than two vertices from the inner pentagram, if we wish to obtain an independent set. Hence the size of a largest independent set in the Petersen graph is exactly four.

The problem **Maximum Independent Set** is to find an independent set of maximum size in a given input graph $G$. This optimisation problem will appear many times throughout this thesis as a typical “hard” problem; it can be shown to be hard even to approximate within certain bounds. In
1. Introduction

particular one can find a polynomial-time reduction from an \textbf{NP}-complete

problem to \textbf{MAXIMUM INDEPENDENT SET}. We often express this by saying

that \textbf{MAXIMUM INDEPENDENT SET} is \textbf{NP}-hard. However, it is clearly easy

to find one solution: the empty set is a subset of the vertices of any graph,

and it is always an independent set.

To obtain a Csp-formulation of the problem \textbf{MAXIMUM INDEPENDENT SET},

we again take the vertices as variables \( V \), but this time we let the domain

\( D \) be \( \{0, 1\} \). Our intended interpretation of an assignment \( f \) from \( V \) to \( D \)

is that if a vertex \( v \) is assigned the value 1 by \( f \), then \( v \in S \), and if \( f(v) = 0 \),

then \( v \notin S \). For each edge between \( u \) and \( v \), we have the constraint:

\[
(f(u), f(v)) \in \{(0, 0), (0, 1), (1, 0)\}.
\]

In order to solve \textbf{MAXIMUM INDEPENDENT SET}, we thus need to maximise

the number of ones assigned by a solution \( f \) subject to constraints of the form

\( R = \{(0, 0), (0, 1), (1, 0)\} \). This problem of maximising the number of

ones in a solution to a Csp instance, expressed using only the constraint \( R \),

is called \textbf{MAX ONES}(\( R \)).

The problem \textbf{MAX ONES} is an optimisation problem for Csp of the first

type with the linear objective function

\[
\sum_{v \in V} f(v).
\]

1.2.2 Example: \textbf{MAX CUT}

The most studied problem in the second category is called \textbf{MAX CUT}. Let

\( G \) be a graph with vertex set \( V \) and edge set \( E \), and let \( S \) be a subset of

\( V \). We say that \( S \) induces a \textbf{cut} in \( G \), which is the subset of edges that are

attached to one vertex in \( S \) and one vertex in the complement, \( \overline{S} = V \setminus S \).

The problem \textbf{MAX CUT} is then to find a cut of largest size in a given input

graph \( G \). This problem is also \textbf{NP}-hard, and has been extensively studied.

Similar to our previous Csp-formulations, we identify the set of vertices \( V \)

as variables and we let the domain \( D \) be \( \{a, b\} \). For each edge between two

vertices \( u \) and \( v \), we have a constraint

\[
(f(u), f(v)) \in \{(a, b), (b, a)\}.
\]

It is easy to see that the resulting Csp instance is satisfiable if and only if

\( G \) is bipartite. This is precisely when the maximum cut in \( G \) is given by

the entire set \( E \). The problem of determining whether or not a graph \( G \)

is bipartite is in \textbf{P}.

We obtain the problem \textbf{MAX CUT} by asking not if the instance is satisfiable,

but instead for an assignment \( f \) which satisfies as many of the constraints as possible.

Each constraint satisfied by \( f \) corresponds to an edge in the cut induced by \( f \).

The set \( S \) can be recovered from \( f \), for example by letting \( S = f^{-1}(a) \).
1.3 Approximability

In the previous section, we encountered two optimisation problems: Maximum Independent Set and Max Cut. Both of these problems are NP-hard to solve to optimality. There is however a way to differentiate the hardness of the two, if we consider approximation algorithms in place of exact algorithms.

**Example 1.5** Let $G = (V, E)$ be a graph. Place each vertex $v \in V$ into $S$ or $\overline{S}$ with equal probability. For each edge $e \in E$, let $X_e = 1$ if the edge $e$ is in the cut induced by $S$, and let $X_e = 0$ otherwise. By linearity of expectation, we have

$$E \left( \sum_{e \in E} X_e \right) = \sum_{e \in E} E(X_e) = \frac{1}{2} \cdot |E|.$$ 

Since the expected measure of a solution, picked at random, is half of the total number of edges, it follows that there exists some solution with a measure at least this big. We can obtain such a solution by the following recursive procedure. If $G$ consists of only two vertices, then we put one of them in $S$, and the other in $\overline{S}$. This is a solution with more than half of the edges in the cut. Otherwise, arbitrarily pick a vertex $v$, and remove $v$, as well as the edges connected to $v$, from the graph $G$. We thus obtain a new graph $G' = (V', E')$, with fewer vertices. Recursively apply this procedure to obtain a solution for the graph $G'$ with, by assumption, a measure of at least $\frac{1}{2} \cdot |E'|$. In this solution, some of the neighbours of $v$ are in $S$, and the edges from $v$ to these vertices constitute a set $E_1$. The remaining neighbours of $v$ are in $\overline{S}$, and the edges from $v$ to these vertices constitute another set $E_2$. The final solution to $G$ is obtained by assigning $v$ to either $S$ or $\overline{S}$, depending on the size of the sets $E_1$ and $E_2$. The measure of this solution is bounded from below by

$$\frac{1}{2} \cdot |E'| + \max \{|E_1|, |E_2|\} \geq \frac{1}{2} \cdot |E|,$$

since $E = E' \cup E_1 \cup E_2$.

We say that an algorithm approximates a maximisation problem within a constant $r$, if it always returns a solution $f$, such that the measure of $f$ divided by the measure of an optimal solution is bounded from below by $r$. We call $r$ the approximation ratio of the algorithm. We similarly say that an algorithm approximates a maximisation problem within some function $r(n)$, if the ratio between the measures of the solution provided by the algorithm, and that of the optimal solution is bounded from below by $r(n)$. Here, the bound $r(n)$ varies with the instance size $n$.

---

8The approximation ratio for an algorithm of a maximisation problem is sometimes defined as the ratio between the measure of an optimal solution and that of the algorithm, i.e., as $1/r$. 

---
1. Introduction

There are two natural complexity classes for optimisation problems, corresponding to the classes \( P \) and \( NP \) for decision problems. These are called \( PO \) and \( NPO \). The complexity class \( APX \) is the class of optimisation problems in \( NPO \) which can be approximated within some constant \( r > 0 \). In Example 1.5, we gave an example of an algorithm which approximates \( \text{Max cut} \) within \( \frac{1}{2} \). Hence we have shown that the problem \( \text{Max cut} \) is in \( APX \). It can further be shown that \( \text{Maximum Independent Set} \) is not in \( APX \), and so, we have differentiated the hardness of these two problems. The problem \( \text{Maximum Independent Set} \) resides instead in a complexity class called \( \text{poly}-APX \), which strictly contains \( APX \) (if \( P \neq NP \)). Both of these classes have complete problems under the appropriate reductions (see Section 2.4.) In fact, \( \text{Max cut} \) is \( APX \)-complete, and \( \text{Maximum Independent Set} \) is \( \text{poly}-APX \)-complete.

The two problems \( \text{Max cut} \) and \( \text{Maximum Independent Set} \) were chosen as representatives for two distinct approaches to optimisation on instances of \( \text{CSP} \). Khanna et al. [93] studies both of these approaches in the case of Boolean \( \text{CSP} \), from a perspective of approximation. They consider both maximisation and minimisation problems, resulting in the four problems \( \text{Max Ones} \), \( \text{Min Ones} \), \( \text{Max CSP} \), and \( \text{Min CSP} \). In all cases they find that the problems, parametrised by a constraint language, fall into a small number of approximation classes. In the case of \( \text{Max Ones} \), which is of primary concern to us, they find that every problem \( \text{Max Ones}(\Gamma) \) is either in \( PO \), it is \( APX \)-complete, it is \( \text{poly}-APX \)-complete, it is an \( NP \)-complete problem to find a solution of non-zero measure, or the problem of finding any solution is \( NP \)-complete. This situation resembles the dichotomy phenomenon, on a more refined scale.

1.4 The maximum solution problem

In this section, we will finally meet the main problem of the thesis. The Maximum Solution Problem, or \( \text{Max Sol} \), for short, can be seen either as a natural generalisation of \( \text{Max Ones} \) to larger domains, or as a generalisation of integer linear programming. It is most easily defined as the maximising optimisation version of \( \text{CSP} \), with values associated to the domain elements, and with a linear objective function. It is often natural to take \( D = \{ 0, 1, \ldots, m-1 \} \), so that the numerical values are implicitly given by the domain elements themselves. The objective function, or measure, of a satisfying assignment \( f : V \rightarrow D \), is in this case given by

\[
\sum_{v \in V} \omega(v) \cdot f(v),
\]

where the coefficients \( \omega(v) \) are non-negative rationals, and are considered to be a part of the instance.

**Example 1.6** Assume that \( V = \{ v_1, \ldots, v_n \} \), that \( a = (a_1, \ldots, a_n) \) is a row vector of rational numbers, and that \( b \) is a rational number. For each such
1.5 Methods and results

choice of \(a, b\), we can define an \(n\)-ary constraint (a relation) \(\text{Ineq}_{a,b}\) as follows:

\[
\text{Ineq}_{a,b} = \{(d_1, \ldots, d_n) \in D^n \mid \sum_{i=1}^n a_i \cdot d_i \leq b\},
\]

If we let \(\Gamma = \{\text{Ineq}_{a,b} \mid a \in \mathbb{Q}^n, b \in \mathbb{Q}\}\), then \(\text{Max Sol}(\Gamma)\) is the problem integer linear programming over a bounded domain \(D\). Note that \(\Gamma\) contains a finite number of constraints, even though it is not immediately apparent from its definition; there are only \(2^{|D^n|}\) distinct relations of arity \(n\).

Exact algorithms for \(\text{Max Sol}\) (under the name of \textsc{Max Value}) were given by Angelikskam and Thapper [8]. For the constraint language parameterisation, the problem has been studied for languages describing solutions to equations over abelian groups [99], for problems in multi-valued logic [83], and for certain undirected graphs [83]. The algebraic approach for studying the approximability of the maximum solution problem was introduced in Jonsson \textit{et al.} [82]. It was used there for studying \textit{maximal} as well as \textit{homogeneous constraint languages}. Building on this work, Jonsson and Thapper [86] used the algebraic approach to classify the approximability of \(\text{Max Sol}\) for constraint languages invariant under two large families of algebras. From the approximation side, all obtained results for the general maximum solution problem align nicely with those of Klaman \textit{et al.} [83] on the \text{Max Ones} problem. In particular, they suggest the following conjecture:

**Conjecture 1.7** Let \(\Gamma\) be a constraint language over a finite domain \(D \subseteq \mathbb{N}\). The problem \(\text{Max Sol}(\Gamma)\) is either in \textsc{Po}, it is \textsc{ApX}-complete, it is \textsc{poly-ApX}-complete, it is \textsc{Np}-hard to obtain a solution of non-zero measure, or it is \text{Np}-hard to obtain any solution.

A relatively up-to-date survey of the currently known results relating to the problem can be found in Jonsson and Nordh [84].

1.5 Methods and results

In this thesis, we study the maximum solution problem from a number of different points of view. We employ both well-established and novel methods of analysis, as well as a few different different notions of efficiency. A rough division of the thesis distinguishes the part that focuses on approximability and the part that deals with computational complexity. More specifically, in the first part, we look at a more fine grained scale of efficiency, where we aim to determine how hard it is to \textit{approximate} an optimal solution, while in the second, we aim to decide how hard it is to obtain an optimal solution. However, there are variations even within this framework. As an
1. Introduction

example, two of the chapters on approximability deal with specific approximation constants for problems that are known to be hard to approximate "arbitrarily well". On the other hand, in the third chapter on approximability, Chapter 3, we treat a more general class of problems, and consequently we focus on classifying these in the appropriate approximation classes: PO, APX, poly-APX, etc. Similarly, while the main subject of the thesis is the maximum solution problem, we make contact with, and sometimes give more attention to other problems, such as MAX $H$-Col. (a restriction of MAX CSP) in Chapter 5, and MAX HAMMING DISTANCE in Chapter 8. We now briefly present the different techniques used, and describe our main results and contributions.

The algebraic approach

This is one of the main techniques used, and it appears several times throughout the thesis. It exploits a connection between constraint languages represented as sets of relations, and universal algebra, which deals with sets of polymorphisms: functions operating on the domain. The technique has been used primarily for the CSP decision problem, with impressive success. It was extended to the maximum solution problem by Jonsson et al. [82] and used, among other things, to classify homogeneous constraint languages. These are constraint languages which include a constraint of the form

$$\{(d, \pi(d)) \mid d \in D\},$$

for every permutation $\pi : D \rightarrow D$. We continue this effort by classifying the approximability of MAX $\text{Sol}(\Gamma)$ for constraint languages $\Gamma$ which are preserved by families of affine algebras. Such constraint languages are described by certain systems of linear equations over abelian groups. This result allows us to give a simplified proof for the classification of MAX $\text{Sol}$ for homogeneous constraint languages (as a special case of languages preserved by symmetric algebras.) We also classify languages preserved by strictly simple surjective algebras, which can be seen as building blocks for more complex algebras [22].

There exists another, more general, optimisation framework for CSP, called the valued constraint satisfaction problem, or VCSP. It was introduced by Schiex et al. [120], and includes MAX $\text{Sol}$ as a special case. An algebraic approach based on two generalised polymorphism concepts, namely multimorphisms and fractional polymorphisms, has been developed for the study of VCSP. While these show great promise, we will not deal with them in this thesis. We refer to the work of Cohen et al. [25, 26, 27] for more on this topic.
1.5. Methods and results

Approximation by relaxation and rounding

This is a commonly applied technique for constructing approximation algorithms for various restrictions of integer linear programming. It is based on the idea of relaxing the integrality constraints of the program, after which a solution to the relaxed program can be found in polynomial time. The obtained solution is in general fractional, and some kind of “rounding procedure” is needed to generate an integer solution. We use this technique for various restrictions of the following integer linear program:

Minimise \( c^T x \)

subject to \( Ax \geq b, \)
\( x \in X, \)

where \( c \) is non-negative, \( A \) is an integer matrix, and \( b \geq 1 \) (componentwise.)

We show that such programs, for which the maximum absolute row sums are bounded by \( k \), can be approximated within \( k \) when \( X = \mathbb{N}^n \), but cannot be approximated within \( k - \epsilon, \epsilon > 0 \), if Khot’s Unique Games Conjecture holds (cf. Section 2.4.2.) We also show that finding a feasible solution to the same problem is \( \text{NP} \)-hard in almost all cases when \( X = \{0, \ldots, a - 1\}^n \). Previous studies have focused mainly on the special case when \( A \) is non-negative, so-called covering integer programs.

This problem is an example of a CSP optimisation problem over an infinite domain. Strictly speaking, it is a “MIN SOL”, rather than a MAX SOL-problem.

Approximation distance

We introduce a novel method designed to extend known approximation ratios for one problem to bounds on the ratio for other problems. The method works equally well for positive approximation results as for negative, inapproximability results. The basis for the method is a binary parameter on graphs which, in a sense, measures a distance between the graphs. This distance can be used to relate the approximation ratios of the problems parameterised by the corresponding graphs. Our main application of this method is to the problem (Weighted) Maximum H-Colourable Subgraph (Max H-Col.) which is a restriction of the general Max CSP problem to a single, binary, and symmetric relation. We also show how the method can be applied to the maximum solution problem, although in this setting much less is known about the approximation ratios, which limits the applicability.

Example 1.8 Goemans and Williamson [58] have designed an algorithm for Max cut using semidefinite programming, which approximates the optimum within 0.87856. Our technique allows us to “measure the distance” between \( K_2 \), which is the graph corresponding to Max cut, and \( C_{11} \), the cycle on 11 vertices. This “distance” turns out to be \( s(K_2, C_{11}) = 10/11 \), and
1. Introduction

we can apply Lemma 5.3 to find that $\text{MAX } C_{11} \text{-Col}$ can be approximated within $0.87856 \cdot s(K_2, C_{11}) \approx 0.79869$.

Conversely, Khot et al. [95] have shown that, if the Unique Games Conjecture holds (cf. Section 2.4.2), then it is $\text{NP}$-hard to approximate $\text{MAX cut}$ within any constant greater than 0.87856. We can now apply our technique to this negative result, and find that, conditioned on the same conjecture, it is $\text{NP}$-hard to approximate $\text{MAX } C_{11} \text{-Col}$ within $0.87856/s(K_2, C_{11}) \approx 0.96642$.

In addition to the application of the technique to $\text{MAX } H \text{-Col}$ for a wide range of graphs $H$, we also establish a close connection to work by Sámal on cubical colourings [122, 123]. This connection shows that our parameter $s(H, G)$ has the same numerical value as $1/\chi_H(G)$, where $\chi_H$ is a special type of “chromatic number”. We believe that this insight may turn out to be crucial for understanding the behaviour of $s$, and in the extension, for understanding the approximability of $\text{MAX } H \text{-Col}$.

Constant production

Given a domain $D$, define

$$C_D = \{(d) \mid d \in D\}.$$  

The set $C_D$ contains all constant unary relations. For the decision problem $\text{Csp}(\Gamma)$, it is known that when $\Gamma$ is a core (i.e. when all endomorphisms are surjective,) then $\text{Csp}(\Gamma)$ is in $\text{P} \ (\text{NP}$-hard) if and only if $\text{Csp}(\Gamma \cup C_D)$ is in $\text{P} \ (\text{NP}$-hard.) Furthermore, the restriction to a core can be done without loss of generality. Previously, no such result was known for $\text{MAX Sol}$. The concept of a core has previously been generalised to that of a max-core (cf. Section 2.5,) but the possibility of adding constants to the language does not carry over. We present a new property which also aims to mimic the core concept, but is more restrictive than the max-core (in the sense that being a max-core is a necessary, but not sufficient condition for satisfying the property.) We show that this property identifies cases when adding the constant relations can be done without changing the computational complexity of the problem. We also show that for a language $\Gamma$ without this property, we can deduce the computational complexity of $\text{MAX Sol}(\Gamma)$ from that of its smaller endomorphic images.

By considering what happens when constants cannot be added to the constraint language, we show that the general $\text{MAX Sol}(\Gamma)$ exhibits a dichotomy if and only if $\text{MAX Sol}(\Gamma \cup C_D)$ does. For the purpose of answering the dichotomy question for $\text{MAX Sol}$, we can therefore assume that all constants are in the language. The technique is used to classify $\text{MAX Sol}(\{H\})$ for certain undirected graphs $H$ with a simple structure.

Example 1.9 Let $H$ be a cyclic graph on an even number of vertices, at least six. If we can add the constant relations to $H$, then it is easy to
see that we can reduce to this problem from the retraction problem on $H$, and this problem is $\text{NP}$-complete [130]. If we cannot add the constants, then our results show that the computational complexity of $\text{Max Sol} \{\{H\}\}$ is determined by that of the set of problems $\text{Max Sol} \{\{g(H)\}\}$ for non-surjective endomorphisms $g$ on $H$. But for all such endomorphisms, $g(H)$ is a path, and for paths, the maximum solution problem is in $\text{PO}$. We can conclude that $\text{Max Sol}(H)$ is in $\text{PO}$ in this case, hence we have established a dichotomy for cycles.

The covering method

In addition to the previously mentioned complexity analyses, we will also address the issue of constructing exact, but exponential-time algorithms for $\text{CSP}$ related problems. In this context, the common and preferred parameterisation is to restrict the problem to $(d, l)$-$\text{CSP}$ instances, where $d$ indicates the maximal allowed domain size and $l$ the maximal arity of any constraint.

The covering method is a technique for extending an exponential-time algorithm which solves some specific problem for $(d', l)$-$\text{CSP}$ instances to an algorithm which solves the same problem for $(d, l)$-$\text{CSP}$ instances, where $d$ is strictly greater than $d'$. The covering method can be seen as a way of derandomising a standard probabilistic technique for doing the same thing. The basic idea is as follows: assume that we restrict the domain of each variable of the $(d, l)$-$\text{CSP}$ instance to a random $d'$-subset of the domain, chosen with uniform distribution. Under some mild conditions on the specific problem, we can then use the existing algorithm for solving this resulting instance. The probability of success, i.e. the probability that the intended solution still remains in the restricted instance, can be bounded by repeating this process a prespecified number of times. Typically, one will need to do about $(d/d')^n$ repetitions to obtain a constant bound on this probability, where $n$ is the number of variables of the instance. What the covering method does, is to replace this probabilistic algorithm with a deterministic one, with a small penalty on the running time. We use this technique to construct algorithms for $\text{Max Sol}(d, 2)$, the version of $\text{Max Sol}$ restricted to $(d, 2)$-$\text{CSP}$ instances. It has also been successfully applied to a large number of decision, counting, and optimisation problems, cf. Angelsmark [5].

1.6 Outline of the thesis

We conclude this introduction with a “road map” of the thesis. The main body of thesis is, as has already been mentioned, divided into two main parts, roughly separating considerations of approximability from those of computational complexity. These two parts are preceded by a smaller part containing this introduction, and a chapter with preliminaries.
Part I

This part contains the introduction, as well as some preliminaries. In Chapter 2, we begin by (re-)introducing constraint satisfaction and the maximum solution problem, and provide a proper formal framework for these problems. We then proceed with formal treatments of approximability, and the related reductions. Finally, we introduce the basic concepts of the algebraic approach. This chapter also serves to establish conventions and notation which will be assumed throughout the thesis.

Part II

In the first main part we consider approximability properties of MAX SOL, integer linear programming, and of MAX CSP for undirected graphs. In Chapter 3, we apply the algebraic approach to two large families of algebras: strictly simple surjective algebras, and symmetric algebras. The main contribution of this chapter is the results for affine algebras.

Chapter 4 studies some restricted integer linear programs over an unbounded domain. Modulo the truth of Khot's unique games conjecture, this chapter provides tight approximation ratios for these integer linear programs using a deterministic rounding technique.

The final chapter on approximability, Chapter 5, introduces the concept of approximation distance, a novel technique for studying approximability. We apply this technique to MAX H-Col (a restriction of MAX CSP) as well as to MAX SOL. The main concrete results are obtained for MAX H-Col, but there is also a substantial part which deals with the technique itself. A section is also devoted to the MAX SOL problem. The fact that the approximability of MAX SOL is much less studied, however, restricts the applicability of the technique.

Part III

In this part, we take a coarser view on MAX SOL, and try to determine simply when it is polynomial-time solvable to optimality, and when it is not. Jonsson et al. [85] conjectured that MAX SOL with arbitrary weights and parameterised by a conservative constraint language is polynomial-time equivalent to MIN HOM for the same constraint language. Thanks to a recent result by Takhanov [126], we prove this conjecture, in Chapter 7.

Chapter 6 deals with the technique of constant production. It extends the ideas from the paper 'The maximum solution problem on graphs' [85] to arbitrary finite constraint languages. Applications are given for certain undirected graphs with relatively simple structure.

Finally, in Chapter 8, we give some exact, exponential-time algorithms for two CSP problem: MAX SOL and the MAX HAMMING DISTANCE problem. In particular, for the first of these, we discuss the covering technique for CSP-related problems.
Chapter 2

Preliminaries

In this chapter we formally define constraint satisfaction and the maximum solution problem. We also give the relevant theoretical background to approximability, and its related reductions. Finally, we introduce the basic concepts of the algebraic approach. This chapter also serves to establish conventions and notation which will be assumed throughout the thesis.

2.1 Basic definitions and notation

Integers, natural numbers, rational numbers, and real numbers are denoted by \( \mathbb{Z}, \mathbb{N}, \mathbb{Q}, \) and \( \mathbb{R} \), respectively. Positive integers, rational numbers and real numbers are denoted by \( \mathbb{Z}^+, \mathbb{Q}^+, \) and \( \mathbb{R}^+ \), respectively. Non-negative rational numbers, \( \mathbb{Q}^+ \cup \{0\} \), are denoted by \( \mathbb{Q}_{\geq 0} \).

Let \( S \) be any set, and \( k \) a positive integer. We denote by \( \binom{S}{k} \), the set of all subsets \( S' \subseteq S \) such that \( |S'| = k \). We denote by \([N] \) the set \( \{1, \ldots, N\} \).

2.1.1 Graphs

Definition 2.1 An (undirected) graph \( G \) is a tuple \((V, E)\), where \( V \) is a finite set of vertices, and \( E \subseteq \binom{V}{2} \) is a set of edges.

For a graph \( G = (V, E) \), we let \( n(G) = |V| \) denote the number of vertices of \( G \), and \( e(G) = |E| \) denote the number of edges. When \( e(G) > 0 \), we say that \( G \) is non-empty. The neighbourhood \( N_G(v) = N(v) \), of the vertex \( v \in V \) in the graph \( G \), is defined as the set \( \{u \in V \mid \{v, u\} \in E\} \). The degree, \( \text{deg}_G(v) = \text{deg}(v) \), of a vertex \( v \in V \), can be defined as \( |N_G(v)| \). A graph \( G = (V, E) \) is called \( d \)-regular if \( \text{deg}(v) = d \) for all \( v \in V \).

Let \( G = (V, E) \) be a graph. If \( E' \subseteq E \), then we say that the graph \( G' = (V, E') \) is a subgraph of \( G \), and we denote this by \( G' \subseteq G \). If \( V' \subseteq V \) is a subset of the vertices of \( G \), then the graph \( G|_{V'} = (V', E \cap V' \times V') \) is called the subgraph induced by \( V' \).
2.1. Basic definitions and notation

At times, we will allow edges, called loops, which go from a vertex \( v \) to itself. Sometimes one also allows for more than one edge to go from \( u \) to \( v \) for a given pair of vertices \( u, v \in V \). Such edges are called parallel edges, and it is clear from our definition that we do not allow them. A graph \((V, E)\) is called simple if it does not contain any loops or parallel edges.

**Example 2.2** The complete graph, \( K_n \), on vertex set \([n]\), is the graph with edge set \( E(K_n) = \binom{n}{2} \). It is \( n-1 \)-regular, and contains \( \binom{n}{2} = n(n-1)/2 \) edges. The cycle graph, \( C_n \), on vertex set \([n]\), is the graph with edge set \( E(K_n) = \{(a, b) \mid |a - b| \equiv 1 \mod n\} \). It is \( 2 \)-regular, and contains \( n \) edges. The graphs \( K_n \) and \( C_n \) are both simple. \( C_n \) is a subgraph of \( K_n \), but not an induced subgraph. The graphs \( K_5 \) and \( C_7 \) are depicted in Figure 2.1.

![Graphs K5 and C7](image)

**Figure 2.1:** The graphs \( K_5 \) (left), and \( C_7 \) (right).

**Definition 2.3** Let \( G \) and \( H \) be graphs. A **graph homomorphism** from \( G \) to \( H \) is a function \( f : V(G) \rightarrow V(H) \) such that

\[
\{u, v\} \in E(G) \implies \{f(u), f(v)\} \in E(H).
\]

The existence of a graph homomorphism from \( G \) to \( H \) is denoted by \( G \rightarrow H \). In this case, we say that \( G \) is **homomorphic to** \( H \).

A **vertex colouring** of a graph \( G \) is a function \( f : V(G) \rightarrow C \), from the vertices of \( G \) to a set of colours, \( C \). If \( f \) is such that for any edge \((u, v) \in E(G)\), we have \( f(u) \neq f(v) \), then \( f \) is called a **proper vertex colouring**, or simply a **proper colouring**. Clearly, whether or not \( G \) has a proper colouring with colours \( C \) depends only on the cardinality of \( C \). This observation suggests the definition of the following important graph parameter:
Definition 2.4 The chromatic number, \( \chi(G) \), of a graph \( G \) is the size of a smallest set \( C \) with which \( G \) can be properly coloured.

A graph parameter related to the chromatic number concerns the notion of cliques: a clique in \( G \) is a complete subgraph of \( G \).

Definition 2.5 The clique number, \( \omega(G) \), of a graph \( G \) is the size of a largest clique in \( G \).

Clearly, one needs \( k \) colours to properly colour any clique of size \( k \) in \( G \), hence \( \omega(G) \leq \chi(G) \), for all graphs \( G \). With the definition of a graph homomorphism in mind, we could alternatively define \( \chi(G) \) as the least positive integer \( k \) such that \( G \rightarrow K_k \), and \( \omega(G) \) as the largest positive integer \( l \) such that \( K_l \rightarrow G \).

In connection to graph homomorphisms, we will state and prove a lemma, which will be needed in Chapters 3 and 5.

**Lemma 2.6** For two non-empty, undirected, and simple graphs \( G \) and \( H \), let \( mc_H(G) \)\(^1\) denote the maximal number of edges in a subgraph \( G' \) of \( G \) such that \( G' \rightarrow H \). Then,

\[
\frac{mc_H(G)}{e(G)} > \sum_{\{u,v\} \in E(H)} \frac{\deg(u) \deg(v)}{2e(H)^2}.
\]

**Proof:** The proof is a straightforward application of the probabilistic method (cf. [4].) In fact, it is only a slight generalisation of the calculation from Example 1.5 in the introduction. That particular result is recovered when \( H = K_2 \).

Let \( f : V(G) \rightarrow V(H) \) be a function, chosen randomly as follows: for every \( v \in V(G) \), and \( v \in V(H) \), the probability that \( f(v) = v \) is equal to \( \deg(v)/2e(H) \). Note that the subset of edges in \( E(G) \) that are mapped to an edge in \( E(H) \) by \( f \) defines a subgraph of \( G' \) such that \( G' \rightarrow H \). Every possible function \( f \) appears with non-zero probability, and we will show that there is at least one such \( f \) which defines a subgraph with the right number of edges.

For \( e \in E(G) \), and \( e' = \{u,v\} \in E(H) \), let \( Y_{e,e'} = 1 \) if \( f \) maps \( e \) to \( e' \), and \( Y_{e,e'} = 0 \) otherwise. Then, \( E(Y_{e,e'}) = 2 \cdot \deg(u) \deg(v)/(2e(H))^2 \). Let \( X_e = 1 \) if \( f \) maps \( e \) to some edge in \( E(H) \), and \( X_e = 0 \) otherwise, so that \( X_e = \sum_{e' \in E(H)} Y_{e,e'} \). Then, the total number of edges in \( G' \) is equal to \( \sum_{e \in E(G)} X_e \), and by linearity of expectation,

\[
E(e(G')) = \sum_{e \in E(G)} E(X_e) = e(G) \sum_{\{u,v\} \in E(H)} \frac{\deg(u) \deg(v)}{2e(H)^2}.
\]

Finally, we note that for an arbitrary fixed vertex \( v_k \in V(H) \), the function defined by \( f(v) = v_k \) for all \( v \in V(G) \) defines the empty subgraph, and has

\(^1\)The notation \( mc_H(G) \) will be explained in Chapter 5.
a non-zero probability. Since $G$ and $H$ are non-empty we have $E(e(G')) > 0$, so there must exist at least one $f$ which defines a $G'$ with strictly more than the expected total number of edges.

An interesting special case of Lemma 2.6 is when $H$ is a $d$-regular graph. Then, $d = 2e(H)/n(H)$, and we have $mc_H(G)/e(G) > d/n(H)$. In particular, for $H = K_n$, $mc_H(G)/e(G) = 1 - 1/n$.

### 2.2 Constraint satisfaction

There are two different, but equivalent, approaches to defining constraint satisfaction problems. In the first, which corresponds to the informal definition given in the introduction, we work with constraints, constraint applications, and satisfying assignments. In the second, as suggested by Feder and Vardi [33], we instead consider homomorphism problems between relational structures. We introduce both here, and use whichever is more convenient in a specific situation.

#### 2.2.1 Constraint languages and satisfying assignments

Let $V$ be a finite set of variables, and let $D$, the domain, be a finite set of values, or domain elements. For a non-negative integer $n$, the set of all $n$-tuples of elements from $D$ is denoted by $D^n$. Any subset of $D^n$ is an $n$-ary relation on $D$. The set of all finitary relations over $D$ is denoted by $R_D$.

A constraint $\varrho$ is a relation in $R_D$, and a constraint application $c$ is a pair $(s; \varrho)$, where $s$ is a list of variables from $V$ of length $m$, called the constraint scope, and $\varrho$ is an $m$-ary relation (constraint) in $R_D$. A constraint language over $D$ is set of constraints, i.e. a subset of $R_D$.

**Definition 2.7** A constraint satisfaction instance over a constraint language $\Gamma$, is a 3-tuple $(V, D, C)$, where

- $V$ is a finite set of variables,
- $D$ is a finite set of values, and
- $C$ is a finite set of constraint applications $\{(s_1; \varrho_1), \ldots, (s_q; \varrho_q)\}$, such that $\{\varrho_i\}_{1 \leq i \leq q} \subseteq \Gamma$.

**Remark 2.8** We did not make the distinction between constraint and constraint application in the introduction. In fact, we will generally not make this distinction, but let constraint denote both concepts, and rely on the context to resolve the ambiguity.

**Definition 2.9** The Constraint Satisfaction Problem over a constraint language $\Gamma$, denoted $\text{CSP}(\Gamma)$, is defined to be the following decision problem:
2. Preliminaries

**Instance:** A CSP-instance \((V, D, C)\) over \(\Gamma\).

**Question:** Is there an assignment \(f : V \to D\) such that for each constraint application \((s; q) \in C\), the image \(f(s)\) of the constraint scope is in \(q\).

**Example 2.10** Let \(NAE\) be the following ternary relation on \(\{0, 1\}\):
\[
NAE = \{(0, 1, 1)\}.
\]

It is easy to see that the well-known \(NP\)-complete problem \textsc{Not-All-Equal Sat} [119] can be expressed as \(Csp(\{NAE\})\).

### 2.2.2 Relational structures and homomorphisms

A (relational) signature \(\tau\), is a set of tuples \((R_i, k_i)\), where \(R_i\) is a relation symbol and \(k_i \in \mathbb{N}\) is called the *arity* of \(R_i\).

**Definition 2.11** A (relational) structure \(\Gamma\) over the signature \(\tau\), sometimes called a \(\tau\)-structure, is a finite set \(D_\Gamma\), the *domain*, and for each relation symbol \(R_i\), a \(k_i\)-ary relation \(g_i^\Gamma \in R_{D_\Gamma^k}\).

**Definition 2.12** Let \(\Delta\) and \(\Gamma\) be relational \(\tau\)-structures. A *homomorphism* from \(\Delta\) to \(\Gamma\) is a function \(h : D_\Delta \to D_\Gamma\) such that for every relation symbol \(R_i\), if \((d_1, \ldots, d_{k_i}) \in g_i^\Delta\), then \((h(d_1), \ldots, h(d_{k_i})) \in g_i^\Gamma\).

**Definition 2.13** The Constraint Satisfaction Problem parameterised by the \(\tau\)-structure \(\Gamma\) is defined to be the following decision problem:

**Instance:** A finite \(\tau\)-structure \(I\).

**Question:** Is there a homomorphism from \(I\) to \(\Gamma\)?

The correspondence to the definition given in Section 2.2.1 is as follows: the domain of the problem is \(D_\Gamma\), and the relations \(g_i^\Gamma \in \Gamma\) are the *constraints*. The domain of the structure \(I\) is the set of variables of the instance, and each tuple in \(g_i^\Gamma \in \Gamma\) corresponds to a constraint application of \(g_i^\Gamma\).

### 2.3 The maximum solution problem

We will use the setting of constraints and satisfying assignments to give the formal description of \textsc{Max Sol}. In Chapter 6 we will redefine it in terms of valued relational structures.

**Definition 2.14** The Maximum Solution Problem over a constraint language \(\Gamma\), or \textsc{Max Sol}(\(\Gamma\)) for short, is the maximisation problem with

**Instance:** A 4-tuple \((V, D, C, \omega)\), where \((V, D, C)\) is a CSP-instance over \(\Gamma\), \(D \subseteq \mathbb{N}\), and \(\omega : V \to \mathbb{Q}_{\geq 0}\) is a weight function.
2.4 Approximability and reductions

Solution: A satisfying assignment \( f : V \rightarrow D \).

Measure: \( m(f) = \sum_{v \in V} \omega(v) \cdot f(v) \).

At times, it will be more natural to let \( D \) be some arbitrary finite set and introduce an explicit function \( \nu : D \rightarrow \mathbb{Q}_{\geq 0} \) to assign numerical values to the domain elements. We call \( \nu \) a valuation on \( D \), or even on \( \Gamma \), and we call the pair \((\Gamma, \nu)\) a valued constraint language. For any subset \( D' \subseteq D \), we will let \( \nu(D') \) denote \( \sum_{d \in D'} \nu(d) \).

Definition 2.15 The Maximum Solution Problem over a valued constraint language \((\Gamma, \nu)\), or MAX SOL(\(\Gamma, \nu\)) for short, is the maximisation problem with

Instance: A 4-tuple \((V, D, C, \omega)\), where \((V, D, C)\) is a Cst-instance over \(\Gamma\), and \(\omega : V \rightarrow \mathbb{Q}_{\geq 0}\) is a weight function.

Solution: A satisfying assignment \( f : V \rightarrow D \).

Measure: \( m(f) = \sum_{v \in V} \omega(v) \cdot \nu(f(v)) \).

The choice between using \( \mathbb{N} \) or \( \mathbb{Q}_{\geq 0} \) as numerical values for \( D \) is not important: if we multiply the rational values by their lowest common denominator, we obtain an equivalent problem using natural numbers. However, the formulation using an explicit valuation function \( \nu \) is more general, compared to implicit numerical values given by \( D \) as a subset of \( \mathbb{N}_0 \), in that distinct domain elements can be assigned equal numerical values. At times, there is also a clear notational advantage of having an explicit valuation.

2.4 Approximability and reductions

In this section, we review the basic definitions of an optimisation problem, the basic classes of such problems (\(\text{NPO, APX, etc.}\)) and the related reductions. A complete introduction to the subject can be found in Ausiello et al. [9].

Definition 2.16 A combinatorial optimisation problem \( \Pi \) is defined over a set of instances (admissible input data); each instance \( I \) has an associated finite set \( \text{Sol}(I) \), of feasible solutions. The objective is, given an instance \( I \), to find a feasible solution of optimum value, with respect to some measure (objective function) \( m : \text{Sol}(I) \rightarrow \mathbb{Q} \). The optimum value is the largest one for maximisation problems and the smallest one for minimisation problems, and is denoted by \( \text{Opt}(I) \).

Definition 2.17 A combinatorial optimisation problem belongs to the class \(\text{NPO}\) if its instances and solutions can be recognised in polynomial time, the solutions are polynomially bounded in the input size, and the objective function can be computed in polynomial time. An \(\text{NPO}\) problem belongs to the class \(\text{PO}\) if it can be solved to optimality in polynomial time.
2. Preliminaries

We will sometimes say that an NPO problem is NP-hard. By this we mean that we can reduce an NP-complete problem to it in polynomial time:

**Definition 2.18** An NPO problem \( \Pi_1 \) is said to be \( \text{NP-hard} \) if there exists an NP-complete problem \( \Pi_2 \) and a polynomial-time algorithm \( A \) which solves \( \Pi_2 \) using queries to an oracle for \( \Pi_1 \).

**Definition 2.19** A solution \( \sigma \in \text{Sol}(I) \) to an instance \( I \) of an NPO problem \( \Pi \) is called \( r \)-approximate if it satisfies

\[
\max \left\{ \frac{m(\sigma)}{\text{Opt}(I)}, \frac{\text{Opt}(I)}{m(\sigma)} \right\} \leq r.
\]

An approximation algorithm for an NPO problem \( \Pi \) has approximation ratio \( r(n) \) if, given any instance \( I \) of \( \Pi \), it outputs an \( r(|I|) \)-approximate solution.

**Definition 2.20** An NPO problem \( \Pi \) is in the class \( \text{APX} \) (poly-APX) if there is a polynomial-time approximation algorithm for \( \Pi \) whose approximation ratio is bounded by a constant (a polynomial in the input size).

Completeness in \( \text{APX} \) and poly-APX is defined using appropriate reductions, called \( \text{AP} \)-reductions and \( \alpha \)-reductions, respectively [30, 93].

**Definition 2.21** (\( \alpha \)-reducibility) An NPO problem \( \Pi_1 \) is said to be \( \alpha \)-reducible to an NPO problem \( \Pi_2 \) if two polynomial-time computable functions \( F \) and \( G \), and a constant \( \alpha \) exist such that

(a) for any instance \( I \) of \( \Pi_1 \), \( F(I) \) is an instance of \( \Pi_2 \);

(b) for any instance \( I \) of \( \Pi_1 \), and any feasible solution \( \sigma' \) of \( F(I) \), \( G(I, \sigma') \) is a feasible solution of \( I \);

(c) for any instance \( I \) of \( \Pi_1 \), and any \( r \geq 1 \), if \( \sigma' \) is an \( r \)-approximate solution of \( F(I) \), then \( G(I, \sigma') \) is an \( \alpha \cdot r \)-approximate solution of \( I \).

**Definition 2.22** (\( \text{AP} \)-reducibility) An NPO problem \( \Pi_1 \) is said to be \( \text{AP} \)-reducible to an NPO problem \( \Pi_2 \) if two polynomial-time computable functions \( F \) and \( G \), and a constant \( \alpha \) exist such that

(a) for any instance \( I \) of \( \Pi_1 \), \( F(I) \) is an instance of \( \Pi_2 \);

(b) for any instance \( I \) of \( \Pi_1 \), and any feasible solution \( \sigma' \) of \( F(I) \), \( G(I, \sigma') \) is a feasible solution of \( I \);

(c) for any instance \( I \) of \( \Pi_1 \), and any \( r \geq 1 \), if \( \sigma' \) is an \( r \)-approximate solution of \( F(I) \), then \( G(I, \sigma') \) is an \( (1 + (r-1)\alpha + o(1)) \)-approximate solution of \( I \), where the \( o \)-notation is with respect to \( |I| \).
2.4. Approximability and reductions

Definition 2.23 An NPO problem \( \Pi \) is called \textit{APX-hard} (\textit{poly-APX-hard}) if every problem in \textit{APX} (\textit{poly-APX}) is \textit{AP-reducible} (\textit{A-reducible}) to it. If in addition, \( \Pi \) is in \textit{APX} (\textit{poly-APX}), then \( \Pi \) is called \textit{APX-complete} (\textit{poly-APX-complete}).

Every \textit{AP}-reduction is also an \textit{A}-reduction [33]. It is a well-known fact (see, e.g., Section 8.2.1 in [9]) that \textit{AP-reductions} compose. It is also well-known (and not difficult to prove) that every \textit{APX}-hard problem is \textit{NP-hard}. We say that a reduction \textit{preserves} a class \( C \), if the existence of a reduction from a problem \( \Pi_1 \) to a problem \( \Pi_2 \in C \) implies \( \Pi_1 \in C \). \textit{AP-reductions} do not generally preserve membership in \textit{PO} [33]. For this reason, we will often use a stronger type of reductions, called \textit{S-reductions}. Every \textit{S-reduction} is also an \textit{AP}-, and an \textit{A}-reduction. \textit{S-reductions} preserve all classes that are of interest to us. In particular, they preserve \textit{PO}, \textit{APX}, \textit{poly-APX}, the property that finding a solution of non-zero measure is \textit{NP-hard}, and the property that finding any solution is \textit{NP-hard}.

Definition 2.24 (\textit{S-reducibility} [32]) An NPO problem \( \Pi_1 \) is said to be \textit{S-reducible} to an NPO problem \( \Pi_2 \) if two polynomial-time computable functions \( F \) and \( G \) exist such that

\[(a) \text{ for any instance } I \text{ of } \Pi_1, I' = F(I) \text{ is an instance of } \Pi_2 \text{ such that } \text{Opt}(I') = \text{Opt}(I); \]

\[(b) \text{ for any instance } I \text{ of } \Pi_1, \text{ and any feasible solution } \sigma' \text{ to } I' = F(I), \sigma = G(I, \sigma') \text{ is a feasible solution to } I \text{ such that }\]

\[ m_1(\sigma) = m_2(\sigma'), \]

where \( m_1 \) is the measure for \( \Pi_1 \) and \( m_2 \) is the measure for \( \Pi_2 \).

In some of our hardness proofs, it will be convenient to use a different type of approximation-preserving reduction called \textit{L-reductions} [9].

Definition 2.25 (\textit{L-reducibility}) An NPO problem \( \Pi_1 \) is said to be \textit{L-reducible} to an NPO problem \( \Pi_2 \) if two polynomial-time computable functions \( F \) and \( G \), and two positive constants \( \beta \) and \( \gamma \) exist such that

\[(a) \text{ for any instance } I \text{ of } \Pi_1, I' = F(I) \text{ is an instance of } \Pi_2 \text{ such that } \text{Opt}(I') \leq \beta \cdot \text{Opt}(I); \]

\[(b) \text{ for any instance } I \text{ of } \Pi_1, \text{ and any feasible solution } \sigma' \text{ to } I' = F(I), \sigma = G(I, \sigma') \text{ is a feasible solution to } I \text{ such that }\]

\[ |m_1(\sigma) - \text{Opt}(I)| \leq \gamma \cdot |m_2(\sigma') - \text{Opt}(I')|, \]

where \( m_1 \) is the measure for \( \Pi_1 \) and \( m_2 \) is the measure for \( \Pi_2 \).
2. Preliminaries

We use $L$-reductions as follows: assume that $\Pi$ is a problem in $\text{APX}$, and that $\Pi$ is $L$-reducible to an $\text{APX}$-hard problem. $\text{APX}$-completeness of $\Pi$ is then deduced from the following lemma:

**Lemma 2.26** (Lemma 8.2 in [9]) Let $\Pi_1, \Pi_2 \in \text{NPO}$ such that $\Pi_1$ is $L$-reducible to $\Pi_2$. If $\Pi_1 \in \text{APX}$, then $\Pi_1$ is $\text{AP}$-reducible to $\Pi_2$.

### 2.4.1 Some complete problems

The following is a generalisation of the problem $\text{Max Cut}$ from the introduction.

**Definition 2.27** $\text{Max } k\text{-cut}$ is the maximisation problem with

**Instance:** A graph $G = (V, E)$.

**Solution:** A partition of $V$ into $k$ disjoint sets $C_1, C_2, \ldots, C_k$.

**Measure:** The number of edges between the disjoint sets, i.e.,

$$\sum_{i \neq j} |E \cap (C_i \times C_j)|.$$

Using the idea of Lemma 2.6, we can construct a simple algorithm which approximates $\text{Max } k\text{-cut}$ within $1 - 1/k$, but better algorithms exist (see [55, 58]). Papadimitriou and Yannakakis [110] showed that $\text{Max } k\text{-cut}$ is complete for the syntactically defined class $\text{Max SNP}$. It was then shown by Khanna et al. [92] how to essentially identify a closure of $\text{Max SNP}$ under approximation-preserving reductions with the class $\text{APX}$. As a consequence, results for $\text{Max SNP}$-completeness translates into results for $\text{APX}$-completeness.

**Proposition 2.28** The problem $\text{Max } k\text{-cut}$ is $\text{APX}$-complete, for $k \geq 2$.

**Proposition 2.29** (Lemma 4.3 and 4.4 in [82]) Let $R$ be the relation $\{(a, a), (a, b), (b, a)\}$. Then,

- $\text{Max Sol}(\{R\})$ is $\text{APX}$-complete, if $0 < a < b$; and
- $\text{Max Sol}(\{R\})$ is $\text{poly-APX}$-complete, if $0 = a < b$.

### 2.4.2 The unique games conjecture

**Definition 2.30** An algorithm $A$ for an $\text{NPO}$ problem $\Pi$ is called a polynomial-time approximation scheme if, for any instance $I$ of $\Pi$, and any rational $r > 1$, $A(I, r)$ returns an $r$-approximate solution to $I$ in time polynomial in $|I|$.
2.5. The algebraic method

The class **PTAS** consist of those NPO problems for which a polynomial-time approximation scheme exists. It is a subclass of **APX**, and if P ≠ NP, it is a proper subclass. In particular, under this assumption, there is a problem II in **APX**, and a rational number c > 1 such that II cannot be approximated within any r ≤ c. In the absence of a **PTAS**, it is interesting to determine the best possible approximation ratio c within which a problem can be approximated or, alternatively the smallest c for which it can be proved that no polynomial-time approximation algorithm exists (typically under some complexity-theoretic assumption such as P ≠ NP).

Khot [94] suggested the Unique Games Conjecture (UGC) as a possible direction for proving inapproximability properties of some important constraint satisfaction problems. Its subsequent applicability to a wide range of problems has been remarkable. In order to state the conjecture, we need to introduce the following problem:

**Definition 2.31** The Unique Label Cover Problem is the maximisation problem with

**Instance:** A 5-tuple (V, W, E, [M], {π^v,w}_(v,w)∈E), where (V ∪ W, E) is a bipartite graph, and π^v,w : [M] → [M] is bijective for each (v, w) ∈ E.

**Solution:** An assignment, label : V ∪ W → [M].

**Measure:** The fraction of edges (v, w) ∈ E such that

π^v,w(label(w)) = label(v).

Now, UGC is the following:

**Conjecture 2.32 (Unique Games Conjecture)** For any η, γ > 0, there exists a constant M = M(η, γ) such that it is NP-hard to distinguish whether the Unique Label Cover problem with label set of size M has optimum at least 1 − η or at most γ.

2.5 The algebraic method

Let A be a finite domain. An **operation** on A is an arbitrary function f : A^k → A. The set of all finitary operations on A is denoted by OA. Any operation on A can be extended in a standard way to an operation on tuples over A, as follows: Let f be a k-ary operation on A and let a_1, a_2, . . . , a_k be n-tuples in A^n. The n-tuple f(a_1, a_2, . . . , a_k) is defined as follows:

\[ f(a_1, a_2, . . . , a_k) = \begin{cases} f(a_1[1], a_2[1], . . . , a_k[1]), \\ f(a_1[2], a_2[2], . . . , a_k[2]), \\ \vdots \\ f(a_1[n], a_2[n], . . . , a_k[n]) \end{cases} \]
2. Preliminaries

where $a_j[i]$ is the $i$-th component in the tuple $a_j$. This construction is extended to relations $\varrho \in R_A$ by $f(\varrho) = \{ f(t) \mid t \in \varrho \}$, and to sets of relations $\Gamma \subseteq R_A$, by $f(\Gamma) = \{ f(\varrho) \mid \varrho \in \Gamma \}$.

**Definition 2.33** Let $\varrho \in R_A$. If $f$ is an operation such that for all choices of tuples $a_1, a_2, \ldots, a_k \in \varrho$, we have $f(a_1, a_2, \ldots, a_k) \in \varrho$, then $\varrho$ is said to be \textbf{invariant under} $f$, and $f$ is said to \textbf{preserve} $\varrho$. We say that $\Gamma \subseteq R_A$ is invariant under $f$, or that $f$ preserves $\Gamma$, if $\varrho$ is invariant under $f$ for all $\varrho \in \Gamma$.

An operation $f$ which preserves $\Gamma$ is called a \textbf{polymorphism} of $\Gamma$. The set of all polymorphisms of $\Gamma$ is denoted by $\text{Pol}(\Gamma)$. Given a set of operations $F$, the set of all relations that are invariant under all the operations in $F$ is denoted by $\text{Inv}(F)$.

Sets of operations of the form $\text{Pol}(\Gamma)$ are known as \textbf{clones}, and they are well-studied objects in algebra (cf. [113, 135]). We remark that the operators $\text{Inv}$ and $\text{Pol}$ form a Galois correspondence between the set of relations over $A$ and the set of operations on $A$. In particular, for $\Gamma, \Gamma' \subseteq R_A$, and $F, F' \subseteq O_A$, we have

$$\Gamma \subseteq \Gamma' \implies \text{Pol}(\Gamma') \subseteq \text{Pol}(\Gamma),$$

and

$$F \subseteq F' \implies \text{Inv}(F') \subseteq \text{Inv}(F).$$

A basic introduction to this correspondence can be found in [112], and a comprehensive study in [113].

A first-order formula $\varphi$ over a constraint language $\Gamma$ is said to be \textbf{primitive positive} (we say $\varphi$ is a pp-formula for short) if it is of the form

$$\varphi \equiv \exists x : (g_1(x_1) \land \cdots \land g_k(x_k))$$

where $g_1, \ldots, g_k \in \Gamma$ and $x_1, \ldots, x_k$ are lists of variables of length equal to the arity of the corresponding relation. Note that a pp-formula $\varphi$ with $m$ free variables defines an $m$-ary relation $\varrho \subseteq A^m$; the relation $\varrho$ is the set of assignments to the $m$ free variables which satisfy $\varphi$:

$$\varrho = \{(a_1, \ldots, a_m) \in A^m \mid \varphi(a_1, \ldots, a_m)\}.$$ 

In this case, we say that $\varrho$ is \textbf{pp-definable} from $\Gamma$. Define a closure operation $\langle \cdot \rangle$ on subsets of $R_A$ such that $\varrho \in \langle \Gamma \rangle$ if and only if the relation $\varrho$ is pp-definable from $\Gamma$. Sets of relations of the form $\langle \Gamma \rangle$ are called \textbf{relational clones}, or cooclones. We have

$$\langle \Gamma \rangle = \text{Inv}(\text{Pol}(\Gamma)).$$

The following theorem states that we have access to a handy $S$-reduction from $\text{Max Sol}$ over finite subsets of $\langle \Gamma \rangle$ to $\text{Max Sol}$ over $\Gamma$ itself.

**Theorem 2.34** ([82]) Let $\Gamma$ be a constraint language and $\Gamma' \subseteq \langle \Gamma \rangle$ a finite subset. Then, $\text{Max Sol}(\Gamma')$ is $S$-reducible to $\text{Max Sol}(\Gamma)$. 
Max-cores

The concept of a core of a constraint language \( \Gamma \) has previously shown its value when classifying the complexity of \( \text{Csp}(\Gamma) \). Jonson \textit{et al.} \cite{Jonson} defined the analogous concept of a max-core for the optimisation problem \( \text{MAX SOL}(\Gamma) \). When \( A \subseteq \mathbb{N} \), we say that a unary operation \( f \) on \( A \) is increasing if \( f(a) \geq a \) for all \( a \in A \).

**Definition 2.35** ([\cite{Jonson}]) A constraint language \( \Gamma \) is a max-core if and only if there is no non-injective, increasing, unary operation \( f \in \text{Pol}(\Gamma) \). A constraint language \( \Gamma' \) is a max-core of \( \Gamma \) if and only if it is a max-core, and \( \Gamma' = f(\Gamma) \) for some increasing, unary operation \( f \in \text{Pol}(\Gamma) \).

The following lemma shows that when studying the approximability of \( \text{MAX SOL}(\Gamma) \), it is sufficient to consider constraint languages \( \Gamma \) that are max-cores.

**Lemma 2.36** ([\cite{Jonson}]) If \( \Gamma' \) is a max-core of \( \Gamma \), then \( \text{MAX SOL}(\Gamma) \) and \( \text{MAX SOL}(\Gamma') \) are equivalent under \( S \)-reductions.

### 2.6 Post’s lattice

The lattice of all Boolean clones, i.e. clones on the domain \( \{0, 1\} \), was described by Emil Post in 1941 [114], and is referred to as Post’s lattice. We will however be more interested in the lattice of all Boolean relational clones (or co-clones.) Fortunately, the Hasse diagram of this lattice is obtained from the Hasse diagram of Post’s lattice by simply turning it upside down; the two lattices are anti-isomorphic (cf. [113].) The result is shown in Figure 2.2. The bold circles indicate the maximal Boolean relational clones.

It remains to assign meaning to the content of the nodes. We will do this in terms of plain bases, as described by Creignou \textit{et al.} [31]. A \textit{basis} for a relational clone \( IC \) is a set of relations \( R \subseteq IC \) such that every relation \( g(x_1, \ldots, x_k) \) in \( IC \) is pp-definable from \( R \). A \textit{plain basis} is a stronger notion, and is defined similarly, but without existential quantification, i.e. every relation \( g(x_1, \ldots, x_k) \) in \( IC \) must be definable as a formula using only conjunction and relations from \( R \). In particular, only the variables \( x_1, \ldots, x_k \) may appear in the definition. For our purposes, the choice between using bases and plain bases does not make any difference, and we make the choice of the latter based solely on the, in our opinion, slightly more appealing appearance of the plain bases.

The plain bases for a selection of the relational clones in Figure 2.2 are given in Table 2.1. This particular subset is chosen for the reason that it contains all the relational clones which will appear in Chapters 3 and 7. The notation is given as follows: the constant unary relations \( (0) \) and \( (1) \) are denoted by \( c_0 \) and \( c_1 \), respectively. A clause on the form \( (\neg x \vee y) \) denotes the relation consisting of the assignments which satisfies the clause, in this case \( \{(0, 0), (0, 1), (1, 1)\} \). The positive clause \( (x_1 \vee
\[ \cdots \lor x_k \] of width \( k \geq 1 \) is denoted by \( P_k \), and similarly, \( N_k \) denotes the negative disjunctive clause of width \( k \). \( Eq \) denotes the equality relation \( \{ (0, 0), (1, 1) \} \). \( \text{Compl}_{k,l} \) denotes the complementation relation \( \{ (0, 1)^{k+l}\} \backslash \{ (0, \ldots, 0, 1, \ldots, 1), (1, \ldots, 1, 0, \ldots, 0) \} \), where in the first tuple there are \( k \) 0s, and \( l \) 1s, and in the second there are \( k \) 1s, and \( l \) 0s.

![Figure 2.2: The lattice of Boolean relational clones. (The graph is from [13].)](image)

The nature of some of the clones may be a bit hard to digest from these descriptions alone. In particular \( \Pi_0 \) (\( \Pi_1 \)) is the relational clone of all relations which contain an all-0 (all-1) tuple, so-called \textbf{0-valid} (\textbf{1-valid}) relations. The clone \( \Pi_2 \) contains all relations. The clauses in the plain bases
Table 2.1: Plain bases for a selection of the Boolean relational clones. This partial list is taken from Créignou et al. [31]

for \( \text{IE}_2 \) and \( \text{IV}_2 \) are usually called **Horn** and **dual Horn**, respectively. The clones \( \text{II}_1 \) and \( \text{II}_i \) are described by appropriately restricted systems of linear equations over \( \mathbb{Z}_2 \). For a more comprehensive introduction to this lattice and its clones, we refer to [12, 13], and for an exhaustive description of the plain bases see [31].
Part II

Approximability
Chapter 3

Affine Algebras and Approximability

In this chapter we study the problem $\text{Max Sol}(\Gamma)$ from the point of view of approximation. The ultimate goal of such a study is to fully classify the approximability of all constraint languages $\Gamma$ according to some scale. One can imagine several candidates for such a scale, as well as several different choices of granularity for some particular scale. We make our choice of scale based on the observation that our complete knowledge to this day suggests the following pentachotomy:

**Conjecture 3.1** For any constraint language $\Gamma$ over a finite domain $A$, and valuation $\nu : A \rightarrow \mathbb{Q}_{\geq 0}$, the problem $\text{Max Sol}(\Gamma, \nu)$ is either polynomial-time solvable, APX-complete, poly-APX-complete, it is NP-hard to obtain a solution of non-zero measure, or it is NP-hard to obtain any solution.

The main method will be the algebraic approach, or algebraic method, described in Section 2.5. The operators Pol and Inv introduced there come from universal algebra. In this section we will need additional concepts from this field. We start with the definition of an algebra.

**Definition 3.2** Let $A$ be a domain. An algebra $A$ over $A$ is a tuple $(A; F)$, where $F \subseteq O_A$ is a family of finitary operations on $A$.

For all our purposes, the algebras will be finite, i.e. the domain $A$ will be finite. The domains of our algebras will also act as domains of constraint languages, which explains why we will see more of the letter $A$ than $D$ in this chapter.

Our interest in the algebra $A = (A; F)$ lies in the fact that the relations invariant under its operations define a relational clone (cf. Section 2.5). Therefore, when we study relational clones, we can make use of known characterisations of the dual concept of clones, and the algebras that they define. Assuming $(\Gamma) = \text{Inv}(F)$, we can therefore, by referring to Theorem 2.34, use
\textsc{Max Sol}(\Gamma), \textsc{Max Sol}(\langle \Gamma \rangle), \textsc{Max Sol}(\lnw(F)), \text{or Max} \textsc{Sol}(\lnw(A)) \text{ as equivalent problems (under S-reductions.) Occasionally, we will abuse notation slightly further, and simply write Max \textsc{Sol}(A) for the same thing.}

This chapter naturally separates into two parts. In the first part, which is introduced in Section 3.2, we study \textit{affine algebras} and the relations which are invariant under the \textit{term operations} of such algebras. These relations turn out to be the solutions to certain sets of linear equations over abelian groups. The general problem of finding a solution to a system of linear equations over an abelian group was shown to be in \textsf{P} by Goldmann and Russell [60]. Kuivinen [99] has given tight inapproximability bounds (provided that \textsf{P} \neq \textsf{NP}) for the problem of maximising a solution to a system of linear equations over a cyclic group of prime order. As a part of a (conjectured complete) classification of the approximability of \textsc{Max Sol}(\Gamma) for \textit{maximal constraint languages} \footnote{A constraint language \Gamma over \mathcal{A} is called \textit{maximal} if \langle \Gamma \cup \{a\} \rangle = \mathcal{R}_A, for any \ a \not\in \Gamma.}, Jonsson \textit{et al.} [82] showed APX-hardness for the set of relations invariant under \textit{minimal} affine algebras. Here, we investigate subsets of such relations, most notably relations obtained by restricting the sum of the coefficients in the linear equations to 0, and relations obtained from homogeneous equations.

The second part of this chapter classifies completely, using some of the results of the first part, and previously developed containment and hardness results, mainly from [82], the approximability of \textsc{Max Sol}(\Gamma) for two large families of constraint languages:

\textbf{Strictly simple surjective algebras}

The first classification determines the approximability of \textsc{Max Sol}(\mathcal{A}) when \mathcal{A} is a strictly simple surjective algebra. Such an algebra has a very ‘simple’ structure: all its smaller homomorphic images and all its proper subalgebras are one-element. These algebras can be viewed as building blocks for more complex algebras and they are well-studied in the literature; an understanding of such algebras is probably needed in order to make further progress using the algebraic approach. We note, for example, that the proof of our second classification result is partly based on the results for strictly simple surjective algebras. Concrete examples include when \mathcal{A} is a finite field of prime order or a Post algebra (cf. Szendrei [135]). Furthermore, these algebras generalise the two-element case nicely since every surjective two-element algebra is strictly simple. Our proof is based on Szendrei’s characterisation of strictly simple surjective algebras [137]. In each case of the characterisation, we either use results from Jonsson \textit{et al.} [82], or the results obtained in Section 3.3 and Section 3.4 for affine, and two-element algebras. The corresponding classification of the \textsc{Csp}-problem was carried out in [22].
Symmetric algebras

The second classification considers algebras that are symmetric in the sense of Szendrei [136]. Examples include algebras whose automorphism group contains the alternating group (i.e. the permutation group containing only even permutations) and certain three-element algebras with cyclic automorphism groups [136]. Well-known examples are the homogeneous algebras: an algebra \( A \) is homogeneous if its automorphism group Aut(\( A \)) is the full symmetric group. The approximability of MAX SOL(\( A \)) is known for all homogeneous algebras [82] and our result generalises this result.

Example 3.3 Consider the equation \( x = y + 1 \pmod{3} \) over the domain \( A = \{0, 1, 2\} \). For brevity, we define \( R = \{(x, y) \mid x = y + 1 \pmod{3}\} \) and note that \( R = \{(x, \sigma(x)) \mid x \in A\} \) for the permutation \( \sigma(0) = 2, \sigma(1) = 0, \) and \( \sigma(2) = 1 \). Let \( \Gamma \) be any relational clone containing \( R \). It is known that for every permutation \( \pi : A \rightarrow A, \{(x, \pi(x)) \mid x \in A\} \in \Gamma \) if and only if \( \pi(x) \in \text{Aut}(\text{Pol}(\Gamma)) \) where \( \text{Pol}(\Gamma) \) denotes the algebra with universe \( A \) and the functions that preserve \( \Gamma \). It is now easy to see that \( \text{Aut}(\text{Pol}(\Gamma)) \) contains every even permutation on \( A \): \( \text{id}, \sigma, \) and \( \sigma^2 = \sigma^{-1}. \) Thus, \( \text{Pol}(\Gamma) \) is symmetric and the approximability of MAX SOL(\( \Gamma \)) can be determined using Theorem 3.36.

The proof of our result is based on Szendrei’s characterisation of symmetric algebras [136], and follows the same lines as that for strictly simple surjective algebras. In fact, the non-idempotent symmetric algebras are strictly simple and surjective (cf. the proof of Theorem 3.36.) It should be noted that the proof of this result is considerably simpler than the original proof for homogeneous algebras, which is a fairly tedious case-by-case analysis. As a by-product of the proof, we also get a classification of \( Csr(\text{Inv}(\mathcal{A})) \) for symmetric algebras (Theorem 3.37.) We note that this result agrees with Conjecture 7.5 in Bulatov et al. [22] on the source of intractability in finite, idempotent algebras.

Chapter outline

The rest of the chapter is organised as follows. In Section 3.1, we provide the definitions needed from universal algebra. We treat affine algebras in Section 3.2 and Section 3.3. We prove \( \text{APX} \)-hardness for maximising the solution of systems of equations over finite vector spaces using a restricted set of equations. This provides \( \text{APX} \)-hardness of MAX SOL(\( A \)) for a family of affine algebras. In Section 3.4, review the main algebraic tools for MAX SOL, from Jonsson et al. [82]. The section is concluded with a complete classification of MAX SOL(\( \Gamma, \nu \)), for arbitrary two-element constraint language \( \Gamma \), and arbitrary \( \nu \). This is a slight, but useful, extension of the result by Khaava et al. [93]. While not a lengthy exercise in itself, this result immediately finds its use in one of the main classifications. Strictly simple surjective algebras are treated in Section 3.5 and symmetric algebras...
in Section 3.6. The chapter is concluded in Section 3.7 with a discussion of the obtained results and possible future extensions.

3.1 Universal algebra

An operation $f$ on $A$ is called a term operation of $(A; F)$ if $f \in \text{Pol}([\text{Inv}(F)])$. The set of all term operations of $A$ are denoted by $\text{Term}(A)$. Two algebras over the same universe are called term equivalent if they have the same set of term operations. An operation $f$ is called idempotent if $f(a, \ldots, a) = a$ for all $a \in A$. The set of all idempotent term operations of $A$ will be denoted by $\text{Term}_{id}(A)$.

Example 3.4 An abelian group $\overline{A} = (A; +)$ is an algebra with term operations

$$\text{Term}(\overline{A}) = \{ \sum_{i=1}^{n} c_i x_i \mid c_i \in \mathbb{Z}, n \geq 1 \}.$$ 

We will let $0_\overline{A}$ denote the neutral element in $\overline{A}$.

Let $A = (A; F)$ be an algebra and let $B \subseteq A$. For an operation $f \in F$, let $f|_B$ denote the restriction of $f$ to $B$. The restriction of $F$ to $B$ is defined by $F|_B = \{ f|_B \mid f \in F \}$. If $f|_B$ preserves $B$, i.e. if $f|_B(B) \subseteq B$, for every $f \in F$, then the algebra $B = (B; F|_B)$ is called a subalgebra of $A$. In this case, we also say that $B$ supports the subalgebra $B$. $B$ is called a proper subalgebra of $A$, if $|B| < |A|$.

We will now define algebra homomorphisms. To do this, it is convenient to index the operations of the algebras involved by an index set $I$.

Definition 3.5 Let $A = (A; F)$ and $B = (B; F')$ be two algebras such that $F = \{ f_i \}_{i \in I}$ and $F' = \{ f'_i \}_{i \in I}$ for an index set $I$. Furthermore, assume that $F$ and $F'$ are indexed so that $f_i$ and $f'_i$ have the same arity, $k_i$, for all $i \in I$. Then, an algebra homomorphism from $A$ to $B$ is a map $h : A \to B$ such that for all $i \in I$,

$$h(f_i(a_1, \ldots, a_{k_i})) = f'_i(h(a_1), \ldots, h(a_{k_i})).$$

If $h$ is surjective, then $B$ is called a homomorphic image of $A$.

A bijective algebra homomorphism is called an algebra isomorphism. An algebra homomorphism from an algebra $A$ to itself is called an algebra endomorphism, and an algebra isomorphism from $A$ to itself is called an algebra automorphism. For brevity, we will often drop the ‘algebra’ prefix, and simply refer to homomorphisms, isomorphisms, endomorphisms, and automorphisms. Note that the set of endomorphisms and the set of unary polymorphisms of an algebra $A$ coincide.
3. Affine Algebras and Approximability

3.1.1 Endomorphism rings

Let \( \mathfrak{A} = (A; +) \) be an abelian group, and let \( g \) and \( h \) be endomorphisms of \( \mathfrak{A} \). If we define \((g + h)(a) = g(a) + h(a)\) and \((g \cdot h)(a) = (g \circ h)(a)\), then the endomorphisms of \( \mathfrak{A} \) form a unitary ring under + and \( \cdot \). This ring will be denoted by \( \text{End} \mathfrak{A} \).

**Definition 3.6** Let \( \mathfrak{A} \) be an abelian group, and \( R \) be a subring of \( \text{End} \mathfrak{A} \). The module \( R \mathfrak{A} \) over \( R \) is an algebra with term operations

\[
\text{Term}(R \mathfrak{A}) = \{ \sum_{i=1}^{n} r_i x_i \mid r_i \in R, n \geq 1 \}.
\]

The abelian group \( \mathfrak{A} \) can be seen as a module over the integers.

We denote by \( _K \mathfrak{A} \) the finite-dimensional vector space on \( \mathfrak{A} \) over the finite field \( K \). It is a module over \( K \) (seen as a subring of \( \text{End} \mathfrak{A} \)). Its term operations are given by

\[
\text{Term}(K \mathfrak{A}) = \{ \sum_{i=1}^{n} k_i x_i \mid k_i \in K, n \geq 1 \}.
\]

The endomorphism ring \( \text{End} _K \mathfrak{A} \) consists of all linear transformations on \( K \mathfrak{A} \). We can consider \( \mathfrak{A} \) as a module over \( \text{End} _K \mathfrak{A} \) and this module will be denoted \( (\text{End} _K \mathfrak{A}) \mathfrak{A} \). Although the notation becomes slightly cumbersome, the following inclusions should be clear:

\[
\text{Term}(\mathfrak{A}) \subseteq \text{Term}(K \mathfrak{A}) \subseteq \text{Term}(\text{End} _K \mathfrak{A}) \subseteq \text{Term}(\text{End} _L \mathfrak{A}).
\]

When \( |A| \) is a prime, all sets are identical, but all inclusions can be strict.

We make a final remark regarding the finite vector space \( K \mathfrak{A} \). If \( K \) is of cardinality \( q = p^m \) for a positive integer \( m \), then inside \( K \) there is a prime subfield \( L \) of cardinality \( p \). The abelian group \( \mathfrak{A} \) can then be seen as a finite vector space \( L \mathfrak{A} \) over \( L \) instead. Since we have the inclusion \( \text{Term}(L \mathfrak{A}) \subseteq \text{Term}(K \mathfrak{A}) \), the set of endomorphisms of \( L \mathfrak{A} \) is larger than that of the original vector space. Hence,

\[
\text{Term}(\text{End} _L \mathfrak{A}) \subseteq \text{Term}(\text{End} _L \mathfrak{A}). \tag{3.1}
\]

3.2 Affine algebras

We begin with an example of a system of linear equations over \( \mathbb{Z}_2 \). We then take a look at such equations “from the other side”, using the language of universal algebra and polymorphisms.
3.2. Affine algebras

Let \( G \) be an undirected graph on \( n \) vertices. Consider the following system of linear equations in variables \( \{x_i \mid 1 \leq i \leq n\} \cup \{y_e \mid e \in E(G)\} \) over \( \mathbb{Z}_2 \):

\[
\begin{align*}
\text{maximise} & \quad \sum_{e \in E(G)} y_e \\
\text{subject to} & \quad y_e = x_i - x_j, \text{ for each } e = \{i, j\}, \\
\text{where} & \quad x_i, x_j, y_e \in \mathbb{Z}_2.
\end{align*}
\]

(3.2)

where the sum in the objective function is taken over the integers, rather than \( \mathbb{Z}_2 \). The program (3.2) describes the problem \( \text{MAX} \ \text{cut} \) (cf. Section 2.4.1.) From a solution to (3.2), we obtain a cut by taking the edge set \( \{e \mid y_e \neq 0\} \). In fact, the reduction from \( \text{MAX} \ \text{cut} \) to the program (3.2) can be shown to be an \( \text{AP} \)-reduction. In general, the set of solutions to an equation of the form \( \sum_{i=1}^{n} x_i = d \), where \( d \in \mathbb{Z}_2 \) is a constant, describes a relation over \( \mathbb{Z}_2 \) as follows:

\[
\{(x_1, \ldots, x_n) \in \mathbb{Z}_2^n \mid \sum_{i=1}^{n} x_i = d\}.
\]

(3.3)

Let \( \Lambda \) be the set of all relations of the form (3.3). The preceding discussion shows that \( \text{MAX} \ \text{Sol}(\Lambda, \nu) \) is \( \text{APX} \)-hard, for \( \nu(0) < \nu(1) \), and with minor adjustments \( \text{APX} \)-hardness can be shown for \( \nu(1) < \nu(0) \) as well. The set of polymorphisms is given by \( \text{Pol}(\Lambda) = \text{Pol}(\text{Inv}(F)) \), where \( F \) is the following set of operations:

\[
F = \{ f(y_1, \ldots, y_k) = \sum_{i=1}^{k} y_i \mid k \geq 1 \text{ and odd.} \}
\]

(3.4)

The algebra \( (A; F) \) is an example of an affine algebra.

**Definition 3.7** Let \( \overline{\Lambda} = (\Lambda; +) \) be a finite abelian group. The operation \( \omega_{\overline{\Lambda}}(x, y, z) = x - y + z \) is called the affine operation of \( \overline{\Lambda} \). An algebra \( A \) is said to be **affine with respect to** \( \overline{\Lambda} \) if

1. \( A \) and \( \overline{\Lambda} \) have the same universe \( A \);
2. the 4-ary relation \( Q_{\overline{\Lambda}} = \{(a, b, c, d) \in A^4 \mid a - b + c = d\} \) is in \( \text{Inv}(A) \);
3. \( \omega_{\overline{\Lambda}} \) is a term operation of \( A \).

The following proposition describes affine algebras up to term equivalence:

**Proposition 3.8** (Proposition 2.6 [135]) Given an algebra \( A = (A; F) \) which is affine with respect to an abelian group \( \overline{\Lambda} = (\Lambda; +) \), there exists a unique unitary subring \( R \) of \( \text{End} \ \overline{\Lambda} \) and a unique submodule \( M \) of the \( R \)-module \( R \times \overline{\Lambda} \) such that \( \text{Term}(A) \) is equal to the following set:

\[
\left\{ \sum_{j=1}^{k} r_j x_i + a \mid k \geq 1, r_1, \ldots, r_k \in R, (1 - \sum_{j=1}^{k} r_j, a) \in M \right\}
\]

(3.5)
3. Affine Algebras and Approximability

The systems of linear equations with integer coefficients over prime order groups studied by Kuivinen [99] correspond to the case when $\mathbb{A} = \mathbb{Z}_p$ ($p$ prime), $R \cong \mathbb{Z}_p$, and $M = (0, 0)$. The maximal affine algebras for which $\text{MAX SOL}$ was proved $\text{APX}$-hard by Jonsson et al. [82] correspond to the case when $\mathbb{A}$ is arbitrary, $R = \mathbb{Z}$, and $M = (0, 0)$. In the special case when $\mathbb{A} = \mathbb{Z}_2$, with $\nu(0) = 0$ and $\nu(1) = 1$, constraint languages invariant under the term operations of the affine algebras are included in the classification of $\text{MAX ONES}$ (Khanna et al. [93].) The five relational clones of such algebras are described in the following example.

Example 3.9 To illustrate the description of affine algebras given in Proposition 3.8, we work out the affine clones (and corresponding relational clones) of Post’s lattice. We thus let $\mathbb{A} = \mathbb{Z}_2$ and $R = \text{End } \mathbb{Z}_2 \cong \mathbb{Z}_2$, where the endomorphisms on $\mathbb{Z}_2$ are given by $0(a) = 0$ and $1(a) = a$. Assume that we have $k$ solutions $x_1, \ldots, x_k$ to an arbitrary relation of the form (3.3) and apply an operation $f(y_1, \ldots, y_k) = \sum_{j=1}^k y_j + a$. We find that

$$\sum_{i=1}^n \left( \sum_{j=1}^k x_j[i] + a \right) = n \cdot a + \sum_{j=1}^n \left( \sum_{i=1}^k x_j[i] \right) = n \cdot a + \sum_{j=1}^k d. \quad (36)$$

In order for the original relation to be invariant under $f$, it must thus satisfy the relation

$$n \cdot a + k \cdot d \equiv d \pmod{2}. \quad (37)$$

(L) When $M = R \times R \mathbb{A}$, the set (3.5) consists of all linear operations over $\mathbb{Z}_2$. In this case, the relation (3.7) splits into four cases, $0 \equiv d$, $1 \cdot d \equiv d$, $n \cdot 1 \equiv d$, and $n \cdot 1 + 1 \cdot d \equiv d \pmod{2}$. We conclude that the equations invariant under the operations of the clone L are those for which $d = 0$ and $n$ is even. These generate the relational clone $\text{L}_e$, of even, homogeneous, linear equations.

(L2) We see that (3.4) is given by using the submodule $\mathbb{M} = (0, 0)$ in (3.5). This is the clone $\text{L}_3$ in Post’s lattice, the clone of all linear, $0$, and $1$-reproducing operations. The corresponding relational clone, $\text{L}_2$, is generated by relations of the form (3.3), i.e. relations described by arbitrary linear equations over $\mathbb{Z}_2$.

(L0) Next, we consider the submodule $M = R \times 0$. In addition to (3.4), we now have operations $\sum_{j=1}^k y_j$ for even $k$ as well. Such operations are no longer 1-reproducing, so the resulting subclone of $\text{L}$ is $\text{L}_0$. Since $a = 0$, the relation (3.7) is satisfied for both even and odd $k$ if and only if $d = 0$, i.e. if the equation is homogeneous. The relational clone generated by relations described by homogeneous linear equations is $\text{L}_0$. 

3.3 Linear equations over abelian groups

(L1) Similarly, we can consider the submodule \( M = \{(0, 0), (1, 1)\} \). The operations are \( \sum_{j=1}^{2k+1} y_j \) and \( \sum_{j=1}^{2k} y_j + 1 \), hence 1-, but not 0-reproducing and make up the clone \( L_1 \). The relation (3.7) is satisfied if and only if \( (2k + 1) \cdot d \equiv d \) and \( n + 2k \cdot d \equiv d \) (mod 2), i.e. if and only if \( n \equiv d \) (mod 2). These generate the relational clone \( \Pi L_1 \) of all 1-valid linear equations.

(L3) Among the proper submodules, we are left with \( M = 0 \times R \), where we have operations \( \sum_{j=1}^{k} y_j + a \) for odd \( k \), and arbitrary \( a \in \mathbb{Z}_2 \). Such operations are self-dual, i.e. \( f(y_1, \ldots, y_k) = 1 + f(1 + y_1, \ldots, 1 + y_k) \), and \( L_3 \) is the clone of all linear, self-dual operations. The relation (3.7) is satisfied if and only if \( (2k + 1) \cdot d \equiv d \) and \( n + 2k \cdot d \equiv d \) (mod 2), i.e. if and only if \( n \equiv d \) (mod 2). This is the relational clone \( \Pi L_3 \) of even linear equations.

3.3 Linear equations over abelian groups

Throughout this section, \( \overline{A} = (A; +) \) will be a finite abelian group. Recall Definition 3.7 and Proposition 3.8 from the previous section, which give the complete description of the term operations of the affine algebras.

Lemma 3.10 ([82]) Let \( \overline{A} \) be a finite abelian group. Then, the problem \( \text{MAX SOL}(\text{Inv}(a \overline{A})) \) is in \( \text{APX} \).

Hence, \( \text{MAX SOL}(\mathcal{A}) \) is in \( \text{APX} \) for all affine algebras \( \mathcal{A} \). Jonsson et al. [82] also showed that \( \text{MAX SOL}(\mathcal{A}) \) is \( \text{APX} \)-hard for the (minimal) affine algebra \( \mathcal{A} = (A; a \overline{A}) \). We extend this result to cover some affine algebras with a larger set of term operations.

We consider linear equations over \( \overline{A} \) of the form

\[
g(x) = \sum_{i=1}^{n} c_i x_i = d,
\]

(3.8)

where \( c_i \in \mathbb{Z} \) and \( d \in \overline{A} \). Each such linear equation defines an \( n \)-ary relation \( \{(x_1, \ldots, x_n) \in \mathbb{A}^n \mid g(x) = d\} \) on \( \mathcal{A} \). Let \( \Lambda(\overline{A}) \) denote the set of all such relations. In Example 3.9, we determined a condition (3.7) on a relation in \( \Lambda(\mathbb{Z}_2) \) to be invariant under the term operations of an affine algebra \( \mathcal{A} \) over the group \( \mathbb{Z}_2 \). Here, we determine a similar condition, for a general abelian group \( \overline{A} \).

Let \( R \) be a unitary subring of \( \text{End} \overline{A} \), in particular, \( R \) always contains the endomorphisms \( n(a) = n \cdot a \) for \( n \in \mathbb{Z} \), \( a \in \overline{A} \). Let \( M \) be an \( R \)-submodule of \( R \times R \overline{A} \), and let \( \mathcal{A} \) be the affine algebra determined by \( R \) and \( M \), according to Proposition 3.8. Let \( f(y_1, \ldots, y_k) = a + \sum_{i=1}^{k} r_i y_j \) be a \( k \)-ary term operation.
of $\mathcal{A}$, and $x_1, \ldots, x_k$ be $k$ solutions to the equation $g(x) = d$ in (3.8). Note that $c_i \in \mathbb{Z}$ commutes with $r_j \in R$ for all $i$ and $j$. We find that
\[
g(f(x_1, \ldots, x_k)) = \sum_{i=1}^{n} c_i \left(a + \sum_{j=1}^{k} r_j x_j[i]\right) = c \cdot a + \sum_{j=1}^{k} r_j \sum_{i=1}^{n} c_i x_j[i] = c \cdot a + r \cdot d,
\]
where $c = \sum_{i=1}^{n} c_i$ and $r = \sum_{i=1}^{k} r_i$. Therefore, we must have
\[
g(f(x_1, \ldots, x_k)) = d \Leftrightarrow c \cdot a + r \cdot d = d \Leftrightarrow c \cdot a = (1 - r) \cdot d
\]
for all $(1 - r, a) \in M$.

**Remark 3.11** The condition in (3.9) tells us exactly when a relation defined by a linear equation of the form (3.8) is in $\text{Inv}(\mathcal{A})$. However, $\text{Inv}(\mathcal{A})$ may contain additional types of equations: let $K\overline{\mathcal{A}}$ be a finite vector space over the finite field $K$. This is an affine algebra over $\overline{\mathcal{A}}$ with $R = K$, and $M = (R \times 0)$. In $\text{Inv}(K\overline{\mathcal{A}})$ we find all relations defined by homogeneous equations
\[
\sum_{i=1}^{n} l_i x_i = 0,
\]
where the $l_i$ are linear transformations on $K\overline{\mathcal{A}}$. With a larger $R$, this set of equations shrinks: for $R = \text{End}_{K}\overline{\mathcal{A}}$, the coefficients must be in $K$.

We have chosen to use only equations (3.8) with integer coefficients since they commute with any endomorphism of $\overline{\mathcal{A}}$. Thus, we will assume that $R = \text{End}_{\overline{\mathcal{A}}}$, unless otherwise specified.

Some particular examples which can be derived from (3.9) are the following:

- If $M = (0, 0)$, then (3.9) holds unconditionally, which means that $\Lambda(\overline{\mathcal{A}}) \subseteq \text{Inv}(\mathcal{A})$.
- If $M = 0 \times R\overline{\mathcal{A}}$, then we must have $c \cdot a = 0$ for all $a \in \overline{\mathcal{A}}$, hence $\sum_{i=1}^{n} c_i = 0$ in $R$. Let $\Lambda_0(\overline{\mathcal{A}})$ be the set of all such relations.
- If $M = R \times 0$, then we must have $r \cdot d = d$ for all $r \in R$, hence $d = 0$. Let $\Lambda^0(\overline{\mathcal{A}})$ be the set of all such relations.
- If $M = R \times R\overline{\mathcal{A}}$, then we must have $\sum_{i=1}^{n} c_i = 0$ in $R$ and $d = 0$. For a fixed $R$, this set of relations, which we will denote by $\Lambda_{\text{min}}(\overline{\mathcal{A}})$, is contained in $\text{Inv}(\mathcal{A})$, for every affine algebra $\mathcal{A}$ over $\overline{\mathcal{A}}$.

Before we move on to the classification of these special cases, we derive a general technique for proving $\text{APX}$-hardness of some problems $\text{Max S/01}(\Lambda, \nu)$, where $\Lambda_{\text{min}}(\overline{\mathcal{A}}) \subseteq \Lambda \subseteq \Lambda(\overline{\mathcal{A}})$, and $\nu$ is a valuation of $\Lambda$. We begin with an easy lemma, which allows us to combine two equations into one.
Lemma 3.12 If \( g(x) = d \) and \( h(z) = d' \) determine two relations in \( \text{Inv}(A) \) for an affine algebra \( A \), then the equation \( g(x) + h(z) = d + d' \) also determines a relation in \( \text{Inv}(A) \).

**Proof:** Let \( g(x) = \sum a_i x_i \) and \( h(y) = \sum b_i y_i \), and let \( f(y_1, \ldots, y_k) = a + \sum r_i y_i \) be a \( k \)-ary term operation of \( A \). Let \( c = \sum a_i c_i \), \( c' = \sum b_i c_i' \), and \( r = \sum r_i c_i \). From condition (3.9), we know that \( c \cdot a = (1-r) \cdot d \) and \( c' \cdot a = (1-r) \cdot d' \). Thus, \( (c + c') \cdot a = (1-r) \cdot (d + d') \), which proves the lemma.

Let \( I \) be an instance of \( \text{MaxSol}(A, \nu) \) described as a system of linear equations and fix one equation \( Eq \) in \( I \). Let \( \{I_a\}_{a \in A} \) be the instances obtained by replacing \( Eq : g(v) = d \) in \( I \) by \( Eq_a : g(v) = d + a \). We will write \( (A, \nu) \Rightarrow \nu' \) to indicate the existence of an instance \( I \) of \( \text{MaxSol}(A, \nu) \), and a valuation \( \nu' \) on \( A \), such that

\[ \nu'(a) = \text{Opt}(I_a). \]

Note that \( I_a \) is not necessarily an instance of \( A \).

Lemma 3.13 Let \( \overline{A} = (A; +) \) be a finite abelian group with a valuation \( \nu \), and let \( \Lambda \) be such that \( \Lambda_{\min}(\overline{A}) \subseteq \Lambda \subseteq \Lambda(\overline{A}) \). Assume that \( (\Lambda, \nu) \Rightarrow \nu' \) so that \( \nu' \) satisfies \( \nu'(0) = \nu'(A)/|A| \). Then, \( \text{MaxSol}(\Lambda, \nu) \) is \text{APX-hard} if \( \nu'(a + \overline{H}) = \nu'(A) \cdot |H|/|A| \) for the cosets \( a + \overline{H} \) of every prime order subgroup \( \overline{H} \) of \( \overline{A} \).

**Proof:** First, note that the assumption that \( \nu' \) is constant on the cosets \( a + \overline{H} \) for every prime order subgroup \( \overline{H} \) implies that \( \nu' \) is constant on the cosets of every (non-trivial) cyclic subgroup of \( \overline{A} \). We proceed by \( L \)-reducing from the \text{APX-hard} problem \( \text{Max k-cut} \), where \( k = |A| \).

Let \( I = (V, E) \) be an instance of \( \text{Max k-cut} \), and let \( \overline{I} \) be the instance of \( \text{Max Sol}(\Lambda, \overline{A}) \) which derived \( \nu' \). We create an instance \( F(I) \) of \( \text{Max Sol}(\Lambda, \overline{A}) \) as follows. For each vertex \( v_i \in V \) we add a variable \( x_i \) to \( F(I) \). For each edge \( e = \{v_i, v_j\} \in E \), and each \( e \in [k] \), we add a copy of \( I \), each with variables \( y_c^e \), where \( y \) are the variables in the instance \( I \). The fixed equation in \( I \) is replaced by

\[ c(x_i - x_j) = g(y_c^e) - d. \quad (3.10) \]

The equation (3.10) is a sum of one equation from \( \Lambda_{\min}(\overline{A}) \subseteq \Lambda \), and one equation from \( A \). By Lemma 3.12, the equation (3.10) is therefore also in \( \Lambda \). Finally, we let the weights of the \( x_i \) variables be \( \omega(x_i) = 0 \). Let \( \sigma' \) be a solution to the resulting instance \( F(I) \). Note that if \( c(\sigma'(x_i) - \sigma'(x_j)) = a \), then the contribution to the measure \( m(\sigma') \) from the variables \( y_c^e \) is less than, or equal to \( \nu'(a) \).
3. Affine Algebras and Approximability

Now, fix an edge \( e = \{v_i, v_j\} \in E \) and note that when \( \sigma'(x_i) \neq \sigma'(x_j) \), we have that \( \{e(\sigma'(x_i) - \sigma'(x_j)) \mid e \in [k]\} \) is equal to a coset \( a + \overline{\mathcal{H}} \) of a (non-trivial) cyclic subgroup \( \overline{\mathcal{H}} \) of \( \mathcal{H} \). Hence,

\[
\sum_{c=1}^{k} \sum_{y \in Y^c} \omega(y) \cdot \nu(\sigma'(y)) \leq \frac{|A|}{|\mathcal{H}|} \sum_{\sigma' \in a + \overline{\mathcal{H}}} \text{Opt} (I_{\sigma'}) = \nu'(A) = D,
\]

with equality if \( \sigma' \) is an optimal solution.

On the other hand, if \( \sigma'(x_i) = \sigma'(x_j) \), then

\[
\sum_{c=1}^{k} \sum_{y \in Y^c} \omega(y) \cdot \nu(\sigma'(y)) \leq \nu'(0_{\overline{\mathcal{A}}}) \cdot |A| = C,
\]

once again with equality if \( \sigma' \) is optimal.

Let \( \pi : A \to [k] \) be an arbitrary bijection between the elements of \( \overline{\mathcal{A}} \) and the index set of the partition in \( \text{Max } k\text{-cut} \). From a solution \( \sigma' \to F(I) \), we derive a solution \( \sigma \to I \) by letting \( \sigma(v_j) = C_j \), where \( j = \pi(\sigma'(x_i)) \). Assume that \( \sigma \) determines a \( k \)-cut in \( I \) with \( m(\sigma) \) edges. The measure \( m' \) of the solution \( \sigma' \) to \( F(I) \) can then be bounded by

\[
m'(\sigma') \leq |E| \cdot C + m(\sigma) \cdot (D - C). \tag{3.11}
\]

When \( \sigma' \) is an optimal solution, the inequality in (3.11) can be replaced by an equality and it follows that

\[
\text{Opt} (F(I)) = |E| \cdot C + \text{Opt}(I) \cdot (D - C). \tag{3.12}
\]

By Lemma 2.6, \( \text{Opt}(I) \geq |E| \cdot (1 - 1/k) \), which in turn implies that

\[
\text{Opt} (F(I)) = \text{Opt}(I) \left( \frac{|E| \cdot C}{\text{Opt}(I)} + (D - C) \right) \leq \text{Opt}(I) \cdot \left( \frac{C}{k - 1} + D \right).
\]

Note that both \( C \) and \( D \) are independent of the instance \( I \). We may therefore take \( \beta = C/(k - 1) + D \) as the first parameter in the \( L \)-reduction. By subtracting (3.11) from (3.12) we get

\[
\text{Opt} (F(I)) - m'(\sigma') \geq (\text{Opt}(I) - m(\sigma)) \cdot (D - C).
\]

Since \( C = \nu'(0_{\overline{\mathcal{A}}}) \cdot |A| < \nu'(A) \cdot |A|/|A| = D \), we can take \( \gamma = 1/(D - C) > 0 \) as the second parameter in the \( L \)-reduction, and the theorem follows. \( \blacksquare \)

3.3.1 The case \( M = 0 \times \mathbb{R} \\overline{\mathcal{A}} \)

Let \( \overline{\mathcal{A}} = (A; +) \) be a finite abelian group with a valuation \( \nu \). Let \( \Lambda_0(\overline{\mathcal{A}}) \) denote the constraint language consisting of all relations of the following form:

\[
\{(x_1, \ldots, x_n) \in A^n \mid \sum_{i=1}^{n} c_i \cdot x_i = a, \text{where } c_i \in \mathbb{Z}, \sum_{i=1}^{n} c_i = 0, a \in A \}.
\]
3.3. Linear equations over abelian groups

The aim of this section is to prove that $\text{Max Sol}(\Lambda_0/\Lambda)$ is $\text{APX}$-hard (Theorem 3.18) and to derive some consequences from this result. In particular, we show that the affine algebras which appear in Section 3.5 and Section 3.6 are $\text{APX}$-hard. First we derive a valuation $\nu^*$ from $\Lambda_0(\Lambda)$ and a valuation $\nu$ with some specific properties. We will then apply Lemma 3.13 to this $\nu^*$.

**Lemma 3.14** Let $\nu$ be non-constant, and assume that $\nu(a + \overline{H}) = \nu(A) \cdot |H|/|A|$ for the cosets $a + \overline{H}$ of every prime order subgroup $\overline{H}$ of $\Lambda$. Then there is a $\nu^*$ such that $(\Lambda_0(\Lambda), \nu) \Rightarrow \nu^*$, with

$$\nu^*(a) = \nu(c + a) + C, \quad (3.13)$$

for some fixed $c \in \arg\min_{b \in A} \nu(b)$, and a positive constant $C$. In particular, $\nu^*(0|_{\overline{H}}) < \nu^*(A)/|A|$, and $\nu^*(a + \overline{H}) = \nu^*(A) \cdot |H|/|A|$ for the cosets $a + \overline{H}$ of every prime order subgroup $\overline{H}$ of $\Lambda$.

**Proof:** We begin by showing the second part of the lemma, assuming that $\nu^*$ is given by the expression (3.13). Under this assumption, $\nu^*$ is just a ‘translation’ of $\nu$ with an added constant term. First we note that $\nu^*(A) = \sum_{b \in A} \nu(c + b) + C = \nu(A) + |A| \cdot C$. Since $\nu$ is non-constant, the inequality $\nu(c) + C \leq \nu(b) + C$ is strict for some $b \in A$. By summing $b$ over $A$, we get $|A| \cdot \nu^*(0|_{\overline{H}}) < \nu(A) + |A| \cdot C = \nu^*(A)$. This proves the first claim.

Similarly, for any $a \in A$, and any prime order subgroup $\overline{H}$:

$$\nu^*(a + \overline{H}) = \nu((c + a) + \overline{H}) + |H| \cdot C = \nu(A) \cdot |H|/|A| + |H| \cdot C = (\nu(A) + |A| \cdot C) \cdot |H|/|A|,$$

and the second claim follows since $\nu(A) + |A| \cdot C = \nu^*(A)$.

We now show how to construct an instance $\mathcal{I}$ of $\text{Max Sol}(\Lambda_0(\Lambda))$, from which we derive the valuation $\nu^*$ in (3.13). The instance has $|A| + 1$ variables: $y$, and $z_b$ for $b \in A$. It is defined by the equation

$$y - z_c = 0, \quad (3.14)$$

together with the equations

$$z_b - z_0 = b, \quad (3.15)$$

for $b \neq 0$. The coefficients in these equations do indeed sum to 0, so the instance can be expressed in $\Lambda_0(\Lambda)$.

To define the weights of $\mathcal{I}$, it will be convenient to introduce a few parameters. Let $\Delta = \max_{b, b' \in A} \nu(b) - \nu(b')$, $\nu_{\text{max}} = \max_{b \in A} \nu(b)$, $A_{\text{max}} = \{b \in A \mid \nu(b) = \nu_{\text{max}}\}$, and $\delta = \min_{b \in A \setminus A_{\text{max}}} \nu_{\text{max}} - \nu(b)$. That is, $\Delta$ is the ‘span’ of values, $\nu_{\text{max}}$ is the largest value, $A_{\text{max}}$ is the set of elements with maximal value, and $\delta$ is the difference between the largest and second largest
value in $\nu$. Due to $\nu$ being non-constant, $\Delta$, $\nu_{\text{max}}$, and $\delta$ are all greater than 0. Let $\gamma > \Delta/\delta$, and define the weights by

$$\omega(v) = \begin{cases} 1 & \text{if } v = y, \\ \gamma & \text{if } v = z_b \text{ with } b \in A_{\text{max}}, \text{ and} \\ 0 & \text{otherwise.} \end{cases}$$

In the instance $I_a$, we replace (3.14) by $'y - z_a = a'.$ The equations (3.15) force a solution to map the $z_b$ variables to a translation of $\bar{A}$. More precisely, $\text{Sol}(I_a)$ is given by the set $\{\sigma_t\}_{t \in \mathbb{T}}$ such that

$$\sigma_t(v) = \begin{cases} t + c + a & \text{if } v = y, \text{ and} \\ t + b & \text{if } v = z_b. \end{cases}$$

The valuation $\nu^*(a) = \text{Opt}(I_a)$ can therefore be expressed as

$$\nu^*(a) = \max_{t \in \mathbb{T}} \{\nu(t + c + a) + \gamma \sum_{b \in A_{\text{max}}} \nu(t + b)\}.$$

We now claim that $\sigma^*$ is the unique maximum solution to $I_a$. Assume to the contrary, that the maximum is attained for some $t' \neq 0_{\mathbb{T}}$, i.e.

$$\nu(t' + c + a) + \gamma \sum_{b \in A_{\text{max}}} \nu(t' + b) \geq \nu(c + a) + \gamma |A_{\text{max}}|\nu_{\text{max}}.$$

After a rearrangement of terms, we then obtain

$$\Delta \geq \nu(t' + c + a) - \nu(c + a) \geq \gamma \cdot (|A_{\text{max}}|\nu_{\text{max}} - \sum_{b \in A_{\text{max}}} \nu(t' + b)).$$

If $\nu(t' + b) < \nu_{\text{max}}$ for some $b \in A_{\text{max}}$, then the right-hand side is greater than or equal to $\gamma \cdot \delta > \Delta$, which is a contradiction. Otherwise $A_{\text{max}}$ is invariant under translations by $t'$, and trivially non-empty. Let $K = \{k \cdot t' \mid k \in \mathbb{Z}\}$ be the (non-trivial) cyclic subgroup generated by $t'$. For any $b' \in A_{\text{max}}$, the coset $b' + K$ must be a subset of $A_{\text{max}}$, so $\nu(b' + K) = |K|\nu_{\text{max}}$. Let $\bar{H}$ be a prime order subgroup of $\bar{K}$. In particular, $\bar{H}$ is a prime order subgroup of $\bar{A}$. By the assumption on $\nu$ we have $\nu(b' + \bar{H}) = \nu(A) \cdot |H|/|A|$. But we also have $b' + \bar{H} \subseteq b' + K$ which implies $\nu(b' + \bar{H}) = |H|\nu_{\text{max}}$. In this case $\nu(A) = |A|\nu_{\text{max}}$, i.e. $\nu$ is equal to $\nu_{\text{max}}$ on all of $A$. This in turn contradicts the choice of a non-constant $\nu$.

We conclude that $\sigma^*$ is indeed the unique solution to $I_a$, and hence $\nu^*$ is given by (3.13), with $C = \gamma |A_{\text{max}}|\nu_{\text{max}}$.}

We now combine Lemma 3.14 with Lemma 3.13 to obtain the following result, and its immediate corollary for prime order groups $\bar{A}$.

**Lemma 3.15** Let $\bar{A} = (A; +)$ be a finite abelian group with a non-constant valuation $\nu$ and assume that $\nu(a + \bar{H}) = \nu(A) \cdot |H|/|A|$ for the cosets $a + \bar{H}$ of every prime order subgroup $\bar{H}$ of $\bar{A}$. Then, $\text{MAX} \ \text{Sol}(\lambda_0(\bar{A}), \nu)$ is APX-hard.
3.3. Linear equations over abelian groups

Corollary 3.16 Let $\overline{A} = (A; +)$ be an abelian group of prime order, and let $\nu$ be a non-constant valuation on $A$. Then, $\text{MAX \ SOL}(\Lambda_0(\overline{A}), \nu)$ is APX-hard.

The question is now what to do when $\nu(a + H) \neq \nu(A) \cdot |H|/|A|$ for some coset $a + H$ of some prime order subgroup $H$. It turns out to be a good idea to pass to the quotient $\overline{A}/\overline{H}$ with the valuation $\nu_{\overline{\pi}}(a) = \sum_{a \in \pi} \nu(a)$. The reason for this is that the property $\nu(a + \overline{H}) \neq \nu(A) \cdot |H|/|A|$ implies that the valuation $\nu_{\overline{\pi}}$ is non-constant on $\overline{A}/\overline{H}$.

Lemma 3.17 Let $\overline{A} = (A; +)$ be a finite abelian group with valuation $\nu$, and let $\overline{H}$ be any subgroup of $\overline{A}$. Then, $\text{MAX \ SOL}(\Lambda_0(\overline{A}/\overline{H}), \nu_{\overline{\pi}})$ is $S$-valuable to $\text{MAX \ SOL}(\Lambda_0(\overline{A}), \nu)$, where $\nu_{\overline{\pi}}(a) = \sum_{a \in \pi} \nu(a)$ for $\pi \in \overline{A}/\overline{H}$.

Proof: Let $\overline{K} = (K; +) \cong \overline{A}/\overline{H}$ and let $\mathcal{I} = \{ \{x_i\}_{1 \leq i \leq n}, K, \text{Eqs}, \omega \}$ be an instance of $\text{MAX \ SOL}(\Lambda_0(\overline{K}), \nu)$. Here, $\text{Eqs}$ is a set of equations over $\overline{K}$, each of the form

$$\sum_{i=1}^{n} c_i x_i = d,$$

where $c_i \in \mathbb{Z}$ with $\sum_i c_i = 0$ and $d \in \overline{K}$. Variables of the instance $\mathcal{J}$ of $\text{MAX \ SOL}(\Lambda_0(\overline{A}))$ are given by the set $\{y^h_i \mid 1 \leq i \leq n, h \in \overline{H}\}$. The first set of equations of $\mathcal{J}$ is given by

$$y^h_i - y^0_i = h,$$

for $i = 1, \ldots, n$ and $h \in \overline{H}$. The equations (3.17) ensure that, in a solution $g$ to $\mathcal{J}$, we have $\{g(y^h_i) \mid h \in H\} = a + \overline{H}$ for some $a \in \overline{A}$. The second set of equations corresponds to the equations (3.16) of the original problem $\mathcal{I}$. For each such equation, and each $h \in \overline{H}$, we add the following equation to $\mathcal{J}$:

$$\sum_{i=1}^{n} c_i y^h_i = d.$$

Finally, the weight functions $\omega'$ of $\mathcal{J}$ take the values $\omega'(y^h_i) = \omega(x_i)$ for $1 \leq i \leq n$ and $h \in \overline{H}$.

Let $g$ be a solution to $\mathcal{J}$ and for each $i$, let $f(x_i)$ be the element $k_i \in \overline{K}$ that corresponds to the coset $\{g(y^h_i) \mid h \in H\} = a_i + \overline{H}$ under the isomorphism $\overline{K} \cong \overline{A}/\overline{H}$. The measure of this solution is given by

$$m(f) = \sum_{i=1}^{n} \omega(x_i) \nu_{\overline{\pi}}(k_i) = \sum_{i=1}^{n} \omega(x_i) \sum_{h \in H} \nu(a_i + H) = \sum_{1 \leq i \leq n} \sum_{h \in H} \omega'(y^h_i) \nu(g(y^h_i)) = m(g).$$
3. Affine Algebras and Approximability

This proves property (b) for the $S$-reduction, and furthermore shows that $\text{Opt}(I) \geq \text{Opt}(J)$.

Conversely, given a solution $f$ to $I$, we define the solution $g$ to $J$ by $g(y_i^h) = f(x_i) + h$. Indeed, $g$ is a solution: for every $i$ such that $1 \leq i \leq n$, and for every $h \in H$, we have

$$g(y_i^h) - g(y_i^0) = (f(x_i) + h) - (f(x_i) + 0) = h.$$ 

For the other set of equations, we have

$$\sum_{i=1}^{n} c_i g(y_i^h) = \sum_{i=1}^{n} c_i (f(x_i) + h) = d + \sum_{i=1}^{n} c_i h = d,$n

since $\sum_i c_i = 0$. It is easy to verify that $m(g) = m(f)$, so $\text{Opt}(I) \leq \text{Opt}(J)$, and equality, i.e. property (a) in the $S$-reduction, follows.

Theorem 3.18 Let $\overline{A} = (A; +)$ be a finite abelian group with a non-constant valuation $\nu$. Then, $\text{Max Sol}(\Lambda_0(\overline{A}), \nu)$ is $\text{APX}$-hard.

Proof: The proof is by induction over the size of the group $\overline{A}$. The base case is when $\overline{A}$ is a prime order group, for which the statement follows from Corollary 3.16. The induction hypothesis is the following: if for every abelian group $\overline{G}$ with non-constant valuation $\mu$, and $|G| < |A|$, the problem $\text{Max Sol}(\Lambda_0(\overline{G}), \mu)$ is $\text{APX}$-hard, then $\text{Max Sol}(\Lambda_0(\overline{A}), \nu)$ is $\text{APX}$-hard.

If $\overline{A}$ is not a prime order group, then it has some proper prime order subgroup. If for every such subgroup $\overline{H}$, and every coset $a + H$, it holds that $\nu(a + H) = \nu(A)/|H|$, then Lemma 3.15 is applicable and proves the statement.

We are left with the case that for some proper prime order subgroup $\overline{H}$, there is an $a \in A$ such that $\nu(a + H) \neq \nu(A)/|H|$. This in particular implies that $\nu$ is non-constant on the cosets of $\overline{H}$, i.e. $\omega_{\overline{H}}$ is non-constant. Lemma 3.17 now shows that we can reduce from $\text{Max Sol}(\Lambda_0(\overline{A}/\overline{H}), \nu_{\overline{H}})$. Since the group $\overline{A}/\overline{H}$ is abelian and strictly smaller than $\overline{A}$, and since $\nu_{\overline{H}}$ is non-constant, we can use the induction hypothesis, from which theorem follows.

The following corollary of Theorem 3.18 provides the special cases which are needed in Section 3.5 and Section 3.6. $T(\overline{A})$ denotes the group of translations $\{f(y) = y + a \mid a \in \overline{A}\}$.

Corollary 3.19 Let $\overline{K} \overline{A}$ be a finite dimensional vector space over a finite field $K$, and let $\nu$ be a non-constant valuation on $A$. Then, the following two problems are $\text{APX}$-complete:

- $\text{Max Sol}(A; \text{Term}_{id}(\overline{K} \overline{A}))$; and
- $\text{Max Sol}(A; \text{Term}_{id}(\text{End}_{\overline{K} \overline{A}}) \cup T(\overline{A}))$. 
3.3. Linear equations over abelian groups

Proof: Both algebras are affine, and the corresponding problems are therefore in APX by Lemma 3.10. We will now verify that both problems are also covered by Theorem 3.18. The theorem applies to the case when \( R = \text{End} \mathcal{A} \). The set \( \Lambda_0(\mathcal{A}) \) contains relations invariant under operations on the following form:

\[
 f(y_1, \ldots, y_k) = \sum_{j=1}^{k} r_j y_j + a,
\]

where \( r_j \in R, \sum_{j=1}^{k} r_j = 1 \) in \( R \), and \( a \in A \). The ring \( R \) includes \( \text{End} \mathcal{K} \mathcal{A} \), the linear transformations on \( \mathcal{K} \mathcal{A} \), which in turn includes \( K \), seen as a ring of endomorphisms on \( \mathcal{K} \mathcal{A} \). In particular then, all operations in \( \text{Term}_d(\mathcal{K} \mathcal{A}) \) and \( \text{Term}_d(\text{End}_n \mathcal{K} \mathcal{A}) \) are on the form (3.18), with \( a = 0 \mathcal{A} \). In addition, \( T(\mathcal{A}) = \{ f(y) = y + a | a \in A \} \) is also on this form. Hence \( \Lambda_0(\mathcal{A}) \) is invariant under all term operations of the respective algebras, so APX-hardness follows from Theorem 3.18.

Another proof of Corollary 3.19

We end this section by obtaining another proof of Corollary 3.19 in which we replace the induction needed in Theorem 3.18 by an analysis of the subspace structure in finite vector spaces.

Let \( \mathcal{K} \mathcal{A} \) be an \( n \)-dimensional vector space over a finite field \( K \) of size \( q = p^m \). From the final remark of Section 3.1, and in particular the relation in (3.1), we know that it suffices to treat the case when \( m = 1 \), i.e. when \( K \) is a prime field. In this case, we will show that there is always a subspace (i.e. subgroup) \( \mathcal{H} \) of \( \mathcal{K} \mathcal{A} \), of size \( q^{n-1} = p^{n-1} \), such that \( \mathcal{H}(\mathcal{A}) = \sum_{v \in \mathcal{H}} \nu(a) \) is non-constant on \( \mathcal{A} \). With this \( \mathcal{H} \), Corollary 3.16 shows that \( \text{Max Sol}(\mathcal{A}(\mathcal{K} \mathcal{A})/\mathcal{H}), \nu_{\mathcal{H}} \) is APX-hard, and by Lemma 3.17, so is \( \text{Max Sol}(\Lambda_0(\mathcal{K} \mathcal{A}), \nu) \). It remains to show that such a subspace always exists in \( \mathcal{K} \mathcal{A} \). This can be deduced from the contrapositive of Lemma 3.20, with \( g = \nu \); if \( \nu \) is non-constant on \( A \), then for some codimension 1 subspace \( \mathcal{H} \), a coset \( a + \mathcal{H} \) exists such that \( \nu(a + \mathcal{H}) \neq \nu(A)/q \).

We will need some definitions and notation. An affine hyperplane in \( \mathcal{K} \mathcal{A} \) is a coset \( a + \mathcal{H} \), where \( a \in A \) and \( \mathcal{H} \) is a codimension 1 subspace of \( \mathcal{K} \mathcal{A} \). Let \( \mathcal{H} \) be the set of affine hyperplanes in \( \mathcal{K} \mathcal{A} \). Denote by \( V \) the \( q^n \)-dimensional vector space over \( \mathbb{Q} \) with basis \( A \). For any subset \( B \subseteq A \), let \( \chi(B) \) denote the characteristic vector of \( B \), i.e. \( \chi(B) = \sum_{a \in B} a \). Let \( g : A \rightarrow \mathbb{Q} \) be any function from \( A \) to the rational numbers. We can then extend \( g \) to a linear transformation \( g : V \rightarrow \mathbb{Q} \) by letting \( g(v) = \sum_i v_i g(a_i) \), when \( v = \sum_i v_i a_i \). In particular, \( g(\chi(B)) = \sum_{a \in B} g(a) \). Note that the proof holds for any \( g \), prime or prime power.

Lemma 3.20 If \( g(\chi(H)) = C \) for all \( H \in \mathcal{H} \) and some constant \( C \), then \( g(a) = C/q^{n-1} \) for all \( a \in A \).
3. Affine Algebras and Approximability

**Proof:** We will show that the set \( X = \{ \chi(H) \mid H \in \mathcal{H} \} \) spans \( V \). From this it follows that \( g \) is uniquely determined by its values on \( X \).

The \textit{\( q \)-binomial coefficients} are defined by

\[
\binom{n}{k}_q = \frac{(q^n - 1)(q^{n-1} - 1) \cdots (q^{n-k+1} - 1)}{(q^k - 1)(q^{k-1} - 1) \cdots (q - 1)}.
\]

They count, among other things, the number of \( k \)-dimensional subspaces in an \( n \)-dimensional vector space over a finite field of size \( q \). The number of codimension 1 subspaces (hyperplanes) containing a fixed 1-dimensional subspace (line) is counted by \( \binom{n-1}{1}_q \) (cf. [59]). We let \( a \in A \) be fixed and for each \( v \in A \), \( v \neq a \), we count the number of (affine) hyperplanes through \( a \) that also contain \( v \). They are exactly the hyperplanes containing the unique line through \( a \) and \( v \). This number is \( \binom{n-1}{1}_q \). Thus,

\[
\sum_{a \in H \in \mathcal{H}} \chi(H) = \binom{n}{n - 1}_q a + \binom{n-1}{1}_q \chi(A - a) = q^{n-1} a + \binom{n-1}{1}_q \chi(A).
\]

Now, choose an arbitrary affine hyperplane \( H^1 \in \mathcal{H} \) and let \( H^2, \ldots, H^q \) denote its translations. Obviously, we have \( \sum_{i=1}^q \chi(H^i) = \chi(A) \). This implies that \( a \) can be written as the following linear combination of vectors in \( X \):

\[
a = q^{1-n} \left( \sum_{a \in H \in \mathcal{H}} \chi(H) \right) - \binom{n-1}{1}_q \cdot \sum_{i=1}^q \chi(H^i).
\]  \hspace{1cm} (3.19)

We now apply \( g \) to both sides of (3.19) and use linearity to obtain

\[
g(a) = q^{1-n} \left( \binom{n}{n-1}_q C - \binom{n-1}{1}_q qC \right) = q^{1-n} \left( \frac{q^n - 1}{q - 1} - \frac{q^n - q}{q - 1} \right) C = q^{1-n} C,
\]

from which the lemma follows.

\[\square\]

### 3.3.2 The case \( M = R \times 0 \)

We will let \( A^n(\overline{A}) \) denote the constraint language consisting of relations of the following form.

\[
\{(x_1, \ldots, x_n) \in A^n \mid \sum_{i=1}^n c_i \cdot x_i = 0, c_i \in \mathbb{Z}\}.
\]
3.3. Linear equations over abelian groups

In this case, we obtain a weaker result than in the last section. In particular, we only treat the case when \( \mathbb{A} \cong \mathbb{Z}_p \) for some prime \( p \).

Let \( \nu \) be a non-constant valuation, \( \nu_{\min} = \min_{b \in A} \nu(b) \), \( A_{\min} = \{ b \in A \mid \nu(b) = \nu_{\min} \} \), \( \nu_{\max} = \max_{b \in A} \nu(b) \), and \( A_{\max} = \{ b \in A \mid \nu(b) = \nu_{\max} \} \). First, note that since \( \Lambda^{0}(\mathbb{A}) \) is \( 0_{\mathbb{T}} \)-valid, it is clear that if \( 0_{\mathbb{T}} \in A_{\max} \), then \( \text{Max SOL}(\Lambda^{0}(\mathbb{A}), \nu) \) is trivially in \( \text{PO} \). We will therefore assume that \( 0_{\mathbb{T}} \not\in A_{\max} \). Hence, every \( d \in A_{\max} \) generates \( \mathbb{A} \). For \( r \in \mathbb{Z}^{+} \), define sets \( A_{\max}^{r} \) recursively as follows: let \( A_{\max}^{1} = A_{\max} \), and for \( r > 1 \), let \( A_{\max}^{r} \) be the set of those \( d \in A_{\max}^{r-1} \) which maximise \( \nu(r \cdot d) \). Then, for every \( r \geq 1 \), the set \( A_{\max}^{r} \) is non-empty, and the sequence \( (\nu(d), \nu(2 \cdot d), \ldots, \nu(r \cdot d)) \) is independent of the choice of \( d \in A_{\max}^{r} \). Let \( l \) be the least positive integer for which \( \nu(l \cdot d) = \nu_{\min} \) for all \( d \in A_{\max}^{r} \), and let \( I \) be the instance of \( \text{Max SOL}(\Lambda^{0}(\mathbb{A})) \), on variables \( \{ x_{1}, \ldots, x_{l-1}, y \} \), defined by the following set of equations:

\[
x_{r} - r \cdot x_{1} = 0_{\mathbb{T}},
\]

for \( 2 \leq r \leq l - 1 \), and

\[
y - l \cdot x_{1} = 0_{\mathbb{T}}.
\]

Next, we determine a weight function \( \omega \) for \( I \) so that in an optimal solution \( \sigma \) to \( I \), we must have \( \sigma(x_{1}) \in A_{\max}^{l} \). Let \( \omega(y) = 1 \), and for the variables \( x_{r} \), \( 1 \leq r \leq l - 1 \), we recursively define

\[
\omega(x_{r}) = \frac{\left( \omega(y) + \sum_{i=r+1}^{l-1} \omega(x_{i}) \right) \nu_{\max} + 1}{\delta},
\]

where \( \delta \) is the minimal absolute difference between two non-identical values of \( \nu \). The instances \( \{ I_{a} \} \) are now obtained by replacing the last equation, \( y - l \cdot x_{1} = 0_{\mathbb{T}} \), by \( y - l \cdot x_{1} = a' \). Then, for any solution \( \sigma \), and any two elements \( d \) and \( c \) such that \( d \in A_{\max} \) and \( c \not\in A_{\max} \),

\[
\omega(x_{1}) \cdot \nu(c) + \left( \omega(y) + \sum_{i=r+1}^{l-1} \omega(x_{i}) \right) \nu_{\max} < \omega(x_{1}) \cdot \nu(d).
\]

If we assume that \( \sigma \) is optimal, then we must have \( \sigma(x_{1}) \in A_{\max}^{l} \). Furthermore, \( \sigma(x_{2}) = 2 \cdot \sigma(x_{1}) \), and the weight \( \omega(x_{2}) \) guarantees that \( \nu(\sigma(x_{2})) \) is maximal among all choices for \( \sigma(x_{1}) \in A_{\max}^{l} \), hence \( \sigma(x_{1}) \in A_{\max}^{l} \). Repeating this argument, we arrive at proving \( \sigma(x_{1}) \in A_{\max}^{l} \) in an optimal solution \( \sigma \). Due to the construction of the sequence \( \{ A_{\max}^{r} \}_{r \geq 1} \), this implies \( \sigma(x_{1}) \in A_{\max}^{l} \).

To apply Lemma 3.13, it now remains to prove that the derived valuation \( \nu'(a) = \text{Opt}(I_{a}) \) has the desired properties. Specifically, since \( \mathbb{A} \) does not have any proper, non-trivial subgroups, we only need to verify that \( \nu'(0_{\mathbb{T}}) \) is strictly less than \( \nu'(A)/|A| \). First, note that \( \nu'(a) = C + \nu(\sigma(y)) \) for some constant \( C \). For \( a = 0_{\mathbb{T}} \), we have shown that \( y \) must be assigned an element in \( A_{\min} \), hence it follows that \( \nu'(0_{\mathbb{T}}) \leq \nu'(a) \) for all \( a \in A \). To see that \( \nu' \) is
3. Affine Algebras and Approximability

non-constant, so that this inequality must be strict, note that for \( a = d - l \cdot d \), with \( d \in A_{\max}^t \), the assignment \( \sigma \) with \( \sigma(x_1) = \sigma(y) = d \) is satisfying, hence \( \nu'(d - l \cdot d) = C + \nu_{\max} \). In conclusion, we have the following result:

**Lemma 3.21** Let \( \mathcal{A} \) be an abelian group of prime order \( p \) and let \( \nu \) be a non-constant valuation. If \( \nu(0_A) = \nu_{\max} \), then \( \text{MAX SOL}(A^0(\mathcal{A}), \nu) \) is in \( \text{PO} \). Otherwise, \( \text{MAX SOL}(A^0(\mathcal{A}), \nu) \) is \( \text{APX-hard} \).

### 3.4 Supplementary toolbox

The basis for the algebraic approach for \( \text{MAX SOL} \) is Theorem 2.34. In this section, we review some containment results proved by Jonsson et al. [82] for this framework. In Section 3.4.1, we extend the classification of Khanna et al. [93] for \( \text{MAX ONEs} \) to \( \text{MAX SOL}(\Gamma, \nu) \) for a two-element constraint language \( \Gamma \) with an arbitrary valuation \( \nu \). We begin by defining some basic operations.

**Definition 3.22** The **discriminator operation** \( t : A^3 \rightarrow A \) is the operation defined by

\[
t(a, b, c) = \begin{cases} 
  c & \text{if } a = b; \text{and} \\
  a & \text{otherwise.}
\end{cases}
\]

**Definition 3.23** The **dual discriminator operation** \( d : A^3 \rightarrow A \) is the operation defined by

\[
d(a, b, c) = \begin{cases} 
  a & \text{if } a = b; \text{and} \\
  c & \text{otherwise.}
\end{cases}
\]

**Definition 3.24** The **switching operation** \( s : A^3 \rightarrow A \) is the operation defined by

\[
s(a, b, c) = \begin{cases} 
  c & \text{if } a = b; \\
  b & \text{if } a = c; \text{and} \\
  a & \text{otherwise.}
\end{cases}
\]

**Definition 3.25** A binary function \( f : A^2 \rightarrow A \) is called a **generalised max-function** if, for all \( a, b \in A \),

- if \( a \neq b \) and \( f(a, b) \leq \min\{a, b\} \), then \( f(b, a) > \max\{a, b\} \); and
- \( f(a, a) \geq a \), for all \( a \in A \).

In particular, \( f(a, b) = \max\{a, b\} \) is a generalised max-function.

**Definition 3.26** An operation \( f \) on \( A \) is called a **2-semilattice operation** if, for all \( a, b \in A \), it holds that
\begin{itemize}
  \item $f(a, a) = a$;
  \item $f(a, b) = f(b, a)$; and
  \item $f(a, f(a, b)) = f(a, b)$.
\end{itemize}

The following proposition summarises the results from [82] that we will require. All proofs are found there. Remember that $\mathcal{C}_A$ denotes the set of unary constant relations on $A$.

**Proposition 3.27** The following holds:

1. Max Sol $\{\text{inv}(t)\}$ is in \textbf{PO}. (Lemma 7.8)
2. Max Sol $\{\text{inv}(s)\}$ is in \textbf{APX}. (Lemma 7.9)
3. For every generalised max-function $f$, Max Sol $\{\text{inv}(f)\}$ is in \textbf{PO}. (Theorem 5.10)
4. Let $f$ be a 2-semilattice operation on $A$. Assume there exist $a, b \in A$ such that $a < b$, $f(a, b) = a$, and $a^* > 0$ where $a^*$ is the minimal element such that there is a $b^*$ satisfying $a^* < b^*$ and $f(a^*, b^*) = a^*$. If this is the case, then Max Sol $\{\text{inv}(f)\}$ is in \textbf{APX}. (Lemma 6.16)
5. If $\text{CSP}(\Gamma)$ is in \textbf{P} and $0 \notin A$, then Max Sol $\{\text{inv}(\pi)\}$ is in \textbf{APX}. (Proposition 4.1)
6. If $\text{CSP}(\Gamma \cup \mathcal{C}_A)$ is in \textbf{P}, then Max Sol $\{\text{inv}(\pi)\}$ is in \textbf{poly-APX}. (Lemma 4.2)

Let $\mathcal{I}$ be a family of bijections between subsets of a set $A$. By $\mathcal{R}(\mathcal{I})$ we denote the set of operations on $A$ which preserve each relation of the form $\{(a, \pi(a)) \mid a \in A\}$ for $\pi \in \mathcal{I}$. By $\mathcal{R}_{id}(\mathcal{I})$ we denote the set of idempotent operations in $\mathcal{R}(\mathcal{I})$.

**Lemma 3.28** Let $\mathcal{I}$ be a family of bijections between subsets of $A$. Then,

$$\{t, d, s\} \subseteq \mathcal{R}_{id}(\mathcal{I}) \subseteq \mathcal{R}(\mathcal{I}).$$

**Proof:** We give the proof for $t$, the remaining two cases follow similarly. An operation $f \in \mathcal{R}(\mathcal{I})$ if and only if for each $\pi \in \mathcal{I}$ defined on a subset $B \subseteq A$ it holds that $B$ is closed under $f$ and $f|_B$ commutes with $\pi$ (cf. Proposition 1.1(c) in [135]). Let $a, b, c$ be three elements on which $\pi$ is defined. If $a = b$, then $\pi(a) = \pi(b)$ so $t(\pi(a), \pi(b), \pi(c)) = \pi(c) = \pi(t(a, b, c))$. If instead $a \neq b$, then $\pi(a) \neq \pi(b)$ since $\pi$ is a bijection between two subsets of $A$ which include $a, b$ and $\pi(a), \pi(b)$, respectively. Therefore, $t(\pi(a), \pi(b), \pi(c)) = \pi(a) = \pi(t(a, b, c))$ and we conclude that $t \in \mathcal{R}(\mathcal{I})$. Inclusion in $\mathcal{R}_{id}(\mathcal{I})$ now follows from the fact that $t$ is idempotent. \hfill \blacksquare
3.4.1 Two-element domain

We will now prove an extension of the classification of $\text{Max Ones}(\Gamma)$ by Khardon et al. [93]. Theorem 3.18 from the previous section plays a role in the latter part of the proof.

**Proposition 3.29** Let $\Gamma$ be a constraint language over a two-element domain. Then, $\text{Max Sol}(\Gamma)$ is either in $\text{PO}$, $\text{APX}$-complete, $\text{poly-APX}$-complete, it is $\text{NP}$-hard to find a solution of non-zero measure or it is $\text{NP}$-hard to find a feasible solution.

**Proof:** Let $D = \{a, b\}$, $0 \leq a < b$. For $a = 0$ the statement follows from the $\text{Max Ones}$-classification by Khardon et al. [93]. For $a > 0$ we show that $\text{Max Sol}(\Gamma)$ is either in $\text{PO}$, $\text{APX}$-complete, or $\text{Csp}(\Gamma)$ is $\text{NP}$-complete. Nevertheless, we carry out both cases, as it is a good illustration of the algebraic approach to $\text{Max Sol}$. In one case we refer to Lemma 6.23 from [93] for the proof that it is $\text{NP}$-hard to find a solution to $\text{Max Sol}(\Pi_0)$ of non-zero measure.

We use Post's lattice to guide the proof. (See Section 2.6 for the relevant notation and definitions.) Post's lattice has a symmetry under the action $f(x_1, \ldots, x_n) \mapsto \neg f(\neg x_1, \ldots, \neg x_n)$. Because of this, we will identify $a$ with 0 and $b$ with 1—the other case follows analogously.

By referring to Theorem 2.34 we may assume, without loss of generality, that $\Gamma = \{\Gamma\}$. From Schaefer's dichotomy theorem [119], $\text{Csp}(\Gamma)$ is $\text{NP}$-complete if and only if $\Gamma = \text{BR}$ or $\Gamma = \text{IN}_2$, so we assume henceforth that $\Gamma \not\in \{\text{BR}, \text{IN}_2\}$. Both for $a = 0$ and $a \neq 0$, the polynomial cases are given by the relational clones $\Pi_1$, $\Pi_2$ and $\Pi_3$. This follows from Proposition 3.27(3) and 3.27(1) respectively, since $\Pi_1$ is $b$-valid, $\Pi_2$ is invariant under $\text{max}(x, y)$, and $\Pi_3$ is invariant under the discriminator operation $t$.

Let $\mathbb{A} = (\mathbb{Z}, +)$ denote the cyclic group on $\{a, b\}$ with $a$ as the zero element and $b$ as unity. From Lemma 3.10 we have $\Pi_2 \subseteq \text{APX}$. The following relational clones were reviewed in the introduction to this chapter:

\[
\begin{align*}
\Pi_0 &= \{x_1 + \cdots + x_k = 0 \mid k \in \mathbb{N}\} \\
\Pi_2 &= \{x_1 + \cdots + x_k = c \mid k \in \mathbb{N}, c \in \{a, b\}\} \\
\Pi_3 &= \{x_1 + \cdots + x_k = c \mid k \text{ even}, c \in \{a, b\}\}
\end{align*}
\]

(3.20)

From Post's lattice, we have the inclusions $\Pi_0 \subseteq \Pi_2$ and $\Pi_3 \subseteq \Pi_2$. $\text{APX}$-hardness for $\text{Max Sol}(\Pi_0)$ now follows from Lemma 3.21, and for $\text{Max Sol}(\Pi_3)$ from Corollary 3.16. The relational clones $\Pi_1$ and $\Pi_0$ are contained in $\Pi_1$, hence $\text{Max Sol}(\Pi_2)$ and $\text{Max Sol}(\Pi_3)$ are in $\text{PO}$.

For $a \neq 0$ we have $\text{Max Sol}(\Gamma)$ in $\text{APX}$ for all $\Gamma \not\in \{\text{BR}, \text{IN}_2\}$ by Proposition 3.27(5) and for $a = 0$, we have $\text{Max Sol}(\Gamma)$ in $\text{poly-APX}$ for all relational clones contained in, and including, $\Pi_2$ and $\Pi_3$, by Proposition 3.27(6). The corresponding hardness results are obtained by observing that the relation $\{(a, a), (a, b), (b, a)\}$ is in $\text{IS}_2$, which implies $\text{APX}$-hardness ($a \neq 0$), and $\text{poly-APX}$-hardness ($a = 0$) by Proposition 2.29.
3.5. Strictly simple surjective algebras

Now everything has been classified for $a \neq 0$. For $a = 0$, the case $\langle \Gamma \rangle = I_0$ remains. In this case we will use Lemma 6.23 from [93]. This lemma states that if a binary constraint language is neither 1-valid (b-valid in our case), 2-CNF, affine, weakly positive (dual Horn), weakly negative (Horn) nor strongly 0-valid, then the problem of finding solutions of non-zero measure is NP-hard. A relation is strongly 0-valid if it contains all tuples with at most one non-zero element. A constraint language is strongly 0-valid if all its relations are strongly 0-valid. We now produce such a language $\Delta$:

$$\Delta = \{x \vee y \vee \neg z, \neg x \vee \neg y\} = \{(0, 1)^3 \setminus \{(0, 0, 1)\}, (0, 1)^2 \setminus \{(1, 1)\}\}.$$ 

The relation $x \vee y \vee \neg z$ is not in $IE_2$, $IL_2$, $ID_2$, nor is it strongly 0-valid, and the relation $x \vee y$ is neither in $IV_2$, nor is it 1-valid. Hence, it is NP-hard to find a solution to $\text{MAX SOL}(\Delta)$ of non-zero measure. Now, $\Delta \subseteq I_0 = (\Gamma)$, so $\text{MAX SOL}(\Delta)$ S-reduces to $\text{MAX SOL}(\Gamma)$ by Theorem 2.34. We can therefore conclude that it is NP-hard to find a solution to $\text{MAX SOL}(\Gamma)$ of non-zero measure.

3.5 Strictly simple surjective algebras

The strictly simple surjective algebras were classified by Szendrei in [137]. The complexity of the constraint satisfaction problem over such algebras was studied in [22]. Here, we do the corresponding classification of the approximability of $\text{MAX SOL}$. First, we will need a few definitions to be able to state Szendrei’s theorem. We adopt the following notion of a surjective algebra used in [22]:

**Definition 3.30** An algebra is called **surjective** if all of its term operations are surjective. As a consequence, a finite algebra is surjective if and only if its unary term operations are all surjective.

**Definition 3.31** An algebra is **simple** if all its smaller homomorphic images are trivial (one-element.) An algebra is **strictly simple** if, in addition, all its proper subalgebras are one-element.

Let $G$ be a permutation group on $A$. Then, $G$ is called **transitive** if, for any $a, b \in A$, there exists $g \in G$ such that $g(a) = b$. $G$ is called **regular** if it is transitive and each nonidentity member has no fixed point. $G$ is called **primitive**, or is said to act **primitively on** $A$, if it is transitive and the algebra $(A; G)$ is simple.

Let $F_k^\alpha$ denote the set of all operations preserving the relation

$$X_k^\alpha = \{(a_1, \ldots, a_k) \in A^k \mid a_i = a \text{ for at least one } i, 1 \leq i \leq k\}$$

where $a$ is some fixed element in $A$, and let $F_k = \bigcap_{\alpha=2}^{\infty} F_k^\alpha$.

With these definitions, we are ready to state the following characterisation of strictly simple surjective algebras:
3. Affine Algebras and Approximability

**Theorem 3.32 ([137])** Let $A$ be a finite strictly simple surjective algebra.

- If $A$ has no one-element subalgebras, then $A$ is term equivalent to one of the following algebras:
  
  $(a)$ $(A; R(G))$ for a regular permutation group $G$ acting on $A$;
  
  $(b)$ $(A; \text{Term}_{id}(\text{End}_{\kappa A}(A) \cup T(A)))$ for some vector space $\kappa A$ over a finite field $K$; or
  
  $(c)$ $(A, G)$ for a primitive permutation group $G$ on $A$.

- If $A$ has one-element subalgebras, then $A$ is idempotent and term equivalent to one of the following algebras:
  
  $(a^0)$ $(A; R_{id}(G))$ for a permutation group $G$ on $A$ such that every non-identity member of $G$ has at most one fixed point;
  
  $(b^0)$ $(A; \text{Term}_{id}(\text{End}_{\kappa A}(A)))$ for some vector space $\kappa A$ over a finite field $K$;
  
  $(d)$ $(A; R_{id}(G) \cap F_k)$ for some $k$ ($2 \leq k \leq \omega$), some element $a \in A$, and some permutation group $G$ acting on $A$ such that $a$ is the unique fixed point of every non-identity member of $G$;
  
  $(c^0)$ $(A; F)$ where $|A| = 2$ and $F$ contains a semilattice operation; or
  
  $(f)$ a two-element algebra with an empty set of basic operations.

It turns out that the type $(d)$ algebras require a slightly more careful analysis. We treat this case separately in the following lemma.

**Lemma 3.33** Let $A = (A; R_{id}(G) \cap F_k)$ be an algebra of type $(d)$ and let $a$ denote the unique fixed point of $G$.

1. If $a = \max A$, then $\text{MAX Sol}(A)$ is in $\text{PO}$.

2. If $0 \notin A$ and $a < \max A$, then $\text{MAX Sol}(A)$ is $\text{APX}$-complete.

3. If $0 \in A$ and $0 < a < \max A$, then $\text{MAX Sol}(A)$ is $\text{APX}$-complete.

4. If $0 \in A$ and $a = 0$, then $\text{MAX Sol}(A)$ is $\text{poly-APX}$-complete.

**Proof:** It is known [22] that the operation

$$f(x, y) = \begin{cases} x & \text{if } x = y, \\ a & \text{otherwise.} \end{cases}$$

is a term operation of $A$. We immediately see that if $a = \max A$, then $f$ is a generalised max-function and tractability follows from Proposition 3.27(3). We also note that $f$ is a 2-semilattice operation: $f$ is obviously idempotent and commutative. If $x = y$, then $f(x, f(x, y)) = f(x, f(x, x)) = f(x, x) = f(x, y)$, and if $x \neq y$, then $f(x, f(x, y)) = f(x, a) = a = f(x, y)$.
3.5. Strictly simple surjective algebras

If $0 \notin A$, then, since $f$ is a 2-semilattice operation, we know from [19] that $\text{Csp}(\text{Inv}(f))$ is in $\text{P}$ so $\text{Max Sol}(\text{Inv}(f))$ is in $\text{APX}$ (Proposition 3.27(5)). Assume instead that $0 \in A$. First, assume that $0 < a < \max A$. We see that $f(x, y) = x$ if and only if $x = a$ or $x = y$. Thus, $x < y$ implies that $f(x, y) = x$ if and only if $a < y$. Since $a > 0$, it follows that $a^* > 0$ (where $a^*$ is defined as in Proposition 3.27(4)). Proposition 3.27(4) implies that $\text{Max Sol}(\text{Inv}(f))$ is in $\text{APX}$. If $a = 0$, then we need to prove membership in $\text{poly-APX}$. By Proposition 3.27(6), it is sufficient to prove that $\text{Csp}(\text{Inv}(A) \cup C_A)$ is tractable. The operation $f$ is idempotent so $\text{Inv}(A) \cup C_A \subseteq \text{Inv}(f)$ and $\text{Csp}(\text{Inv}(f))$ is tractable since $f$ is a 2-semilattice operation.

Next, we prove hardness. If $2 \leq k < \omega$, then the relation $X^k_\mathcal{A}$ is, by definition, invariant under the operations in $F_k^\mathcal{A} \supseteq F_k^\mathcal{A} \cap \mathcal{R}_{id}(G)$ so $X^k_\mathcal{A} \subseteq \text{Inv}(A)$. If $k = \omega$, then $X^\omega_\mathcal{A}$ is invariant under the operations $F_\omega \supseteq \mathcal{R}_{id}(G) \cap \bigcap_{k \in \mathbb{N}} F_k^\mathcal{A}$. Furthermore, $\mathcal{A}$ is idempotent so $C_A \subseteq \text{Inv}(A)$. It is therefore sufficient to prove $\text{poly-APX}$- and $\text{APX}$-hardness of $\text{Max Sol}(\{X^k_\mathcal{A} \cup C_A\}, 2 \leq k < \omega$, in order to complete the proof. Consider the binary relation $r$ defined by

$$r(x, y) \equiv_{pp} \exists z: X^k_\mathcal{A}(x, y, z, \ldots, z) \wedge \{\max A\}(z).$$

Note that $r = (A \times \{a\}) \cup (\{a\} \times A)$ and that the max-core of $r$ is given by $\{(a, a), (a, \max A), (\max A, a)\}$. Depending on whether $a \neq 0$ or $a = 0$, $\text{APX}$- and $\text{poly-APX}$-hardness now follows from Proposition 2.29.

With Lemmas 3.28, 3.33 and the results from Sections 3.2 and 3.4 it is now straightforward to prove the following classification of the approximability of $\text{Max Sol}(\mathcal{A})$ when $\mathcal{A}$ is a strictly simple surjective algebra.

**Theorem 3.34** Let $\mathcal{A}$ be a finite strictly simple surjective algebra. Then, $\text{Max Sol}(\mathcal{A})$ is either in $\text{PO}$, it is $\text{APX}$-complete, $\text{poly-APX}$-complete, or it is $\text{NP}$-hard to find a solution.

**Proof:** We use Theorem 3.32 to guide a short case-by-case analysis.

- If $\mathcal{A}$ is of type $(a)$ or $(a^*$), then by Lemma 3.28, the discriminator operation $t(x, y, z)$ is a term operation of $\mathcal{A}$. Tractability follows from Proposition 3.27(1).
- If $\mathcal{A}$ is of type $(b^*)$, then $a_\mathcal{A}$ is a term operation of $\mathcal{A}$ so $\text{Max Sol}(\mathcal{A})$ is in $\text{APX}$ by Lemma 3.10. Conversely, if $\mathcal{A}$ is of type $(b)$, then $\text{Max Sol}(\mathcal{A})$ is $\text{APX}$-hard by Corollary 3.19. Hence, $\text{Max Sol}(\mathcal{A})$ is $\text{APX}$-complete in both cases.
- If $\mathcal{A}$ is of type $(c)$ or $(f)$, then $\text{Csp}(\mathcal{A})$ is $\text{NP}$-complete [22].
- If $\mathcal{A}$ is of type $(d)$, then the result follows from Lemma 3.33.
- Finally, if $\mathcal{A} = (\mathcal{A}; F)$ is of type $(e)$, then since $|\mathcal{A}| = 2$, the result follows from Proposition 3.29.
3. Affine Algebras and Approximability

All cases have been shown to fall into the stated classes, which proves the theorem. ■

3.6 Symmetric algebras

An algebra $A$ is symmetric (in the sense of Szafrle [136]) if for every subalgebra $B = (B; F)$ of $A$,

1. the automorphism group of $B$ acts primitively on $B$; and
2. for any set $C \subseteq A$ with $|C| = |B|$, $C$ supports a subalgebra of $A$ isomorphic to $B$.

Examples of symmetric algebras include homogeneous algebras and algebras whose automorphism group contains the alternating group [136].

If $B$ and $C$ support two subalgebras of $A$, then so does their intersection $B \cap C$. Therefore, by taking appropriate intersections of subalgebras, the second condition implies the following property for a symmetric algebra $A = (A; F)$: if $B = (B; F|_B)$ is a proper subalgebra of $A$, then $(C; F|_C)$ is a subalgebra of $A$ whenever $C$ is a subset of $A$ with $|C| \leq |B|$. Consequently, we can assign a number $\nu(A)$, $0 \leq \nu(A) \leq |A| - 1$, to every symmetric algebra such that a proper subset $B \subset A$ is the universe of a subalgebra of $A$ if and only if $|B| \leq \nu(A)$. One may note that $\nu(A) \geq 1$ if and only if $A$ is idempotent.

We will need some additional definitions and notation in order to describe the symmetric algebras. An isomorphism between two subalgebras of an algebra $A$ is called an internal isomorphism of $A$. The set of all internal isomorphisms will be denoted $\text{Iso} A$.

A $k \times l$ cross $(k, l \geq 2)$ is a relation on $A^2$ of the form

$$X(B_1, B_2, b_1, b_2) = (B_1 \times \{b_2\}) \cup (\{b_1\} \times B_2),$$

where $b_1 \in B_1$, $b_2 \in B_2$, $|B_1| = k$, and $|B_2| = l$.

Let $\mathcal{D}_1$ denote the clone of all idempotent operations on $A$, and $\mathcal{E}_1$ the subclone of $\mathcal{D}_1$ consisting of all operations preserving every relation

$$L_{a,b} = \{(a, a, a), (a, b, b), (b, a, b), (b, b, a)\} \ (a, b \in A, \ a \neq b).$$

For $2 \leq m \leq |A|$, let $\mathcal{D}_m$ be the clone of all operations in $\mathcal{D}_1$ preserving every $m \times 2$ cross. For $2 \leq m \leq |A|$, let $\mathcal{E}_m$ be the clone consisting of all operations $f \in \mathcal{D}_1$ for which there exists a projection $p$ agreeing with $f$ on every $m$-element subset $B$ of $A$.

We are now set to state the characterisation of symmetric algebras, and to prove our $\text{Max}$ $\text{Sol}$-classification.

**Theorem 3.35 ([136])** Let $A$ be a finite symmetric algebra.
3.6. Symmetric algebras

- If $A$ is not idempotent, then $|A|$ is prime and there is a cyclic group $\overline{A} = (A; +)$ such that $A$ is term equivalent to one of the following algebras:
  
  1. $(A; R(T(\overline{A})))$;
  2. $(A; \text{Term}_{id}(\overline{A}) \cup T(\overline{A}))$; or
  3. $(A; T(\overline{A}))$.

- If $A$ is idempotent, then $A$ is term equivalent to one of the following algebras:
  
  1. $(A; R(\text{Iso} A) \cap D_m)$ for some $m$ with $1 \leq m \leq \nu(A)$ or $m = |A|$;
  2. $(A; R(\text{Iso} A) \cap E_m)$ for some $m$ with $1 \leq m \leq \nu(A)$ or $m = |A|$;
  3. $(A; \text{Term}_{id}(\kappa \overline{A}))$ for a 1-dimensional vector space $\kappa \overline{A}$ over a finite field $K$; or
  4. $(A; \text{Term}_{id}(\overline{A}))$ for a 4-element abelian group $\overline{A}$ of exponent 2.

**Theorem 3.36** Let $A$ be a symmetric algebra. Then, $\text{MAX SOL}(A)$ is either in $\text{PO}$, it is $\text{APX}$-complete, it is poly-$\text{APX}$-complete, or it is $\text{NP}$-hard to find a solution.

**Proof:** Assume first that $A$ is not idempotent. It is possible to do a case-by-case analysis as the one for Theorem 3.34, or to note that each case is a special case of Theorem 3.32. We choose to show directly that $A$ is a strictly simple surjective algebra, after which the result follows from Theorem 3.34.

Let $h : A \to A$ be an endomorphism on $A$. Define $h^{(0)}(A) = A$ and for $n \geq 0$, let $h^{(n+1)}(A) = h(h^{(n)}(A))$. Then, $h^{(0)}(A) \subseteq h^{(1)}(A)$ and for $n \geq 1$, $h^{(n)}(A) \subseteq h^{(n+1)}(A)$ implies $h^{(n+1)}(A) \subseteq h^{(n)}(A)$. Therefore, there must exist a $k \geq 0$ such that $h^{(k+1)}(A) = h^{(k)}(A)$. It is easy to verify that $h^{(k)}(A)$ must support a subalgebra of $A$. But since $A$ is symmetric and non-idempotent, it has no proper subalgebras, so $k = 0$. Hence, any choice of $h$ is an isomorphism and $A$ is strictly simple. It remains to show that $A$ is surjective. For a finite algebra, this amounts to verifying that each endomorphism on $A$ is surjective. An endomorphism $f$ on $A$ is also an endomorphism of $(A; G)$, where $G$ is the automorphism group of $A$. Since $A$ is symmetric, $G$ acts primitively on $A$ which is equivalent to saying that $G$ is transitive and $(A; G)$ is simple. Thus, either $f$ is surjective or $f(A) = \{a\}$ for some $a \in A$. In the latter case, $a$ is a fixed point of each $\pi \in G$. But, the set of fixed points of any nonidentity isomorphism supports a proper subalgebra of $A$ and since $G$ is transitive, it must contain at least one nonidentity isomorphism. It follows that $f$, and thereby $A$, is surjective.

Assume instead that $A$ is idempotent. We proceed with a case-by-case analysis guided by the second part of Theorem 3.35.
3. Affine Algebras and Approximability

- If \( A \) is term equivalent to \((A; R(\text{Iso } A) \cap D_m)\) with \( m = 1 \), then \( t \in R_d(\text{Iso } A) \), and according to Lemma 3.28, \( \text{MAX Sol}(A) \) is in \( \text{PO} \). Assume that \( m \geq 2 \). It is known (cf. [34]) that the dual discriminator operation \( d \) preserves crosses, so \( d \in D_m \) for all \( m \geq 2 \). Furthermore, by Lemma 3.28, \( d \in R(\text{Iso } A) \). Therefore, \( d \in R(\text{Iso } A) \cap D_m \) and since \( \text{Csp}(\text{Inv}(d)) \) is in \( \text{P} \) (cf. Theorem 5.7 [79]), we have \( \text{MAX Sol}(A) \) in \( \text{APX} \) if \( 0 \notin A \) by Proposition 3.27(5) and \( \text{MAX Sol}(A) \) in \( \text{poly-APX} \) if \( 0 \in A \) by Proposition 3.27(6) \( \langle A \rangle \) is idempotent. \( \rangle \) To show hardness, note that every \( m \times 2 \)-cross is in \( \text{Inv}(A) \). In particular, if we choose \( B \) such that \( \{ \min A, \max A \} \subseteq B \subseteq A \) with \( |B| = m \), then the cross \( X = X(B, \{ \min A, \max A \}) \) is in \( \text{Inv}(A) \). The max-core of \( X \) is \( \{ \min A, \min A \}, \{ \min A, \max A \}, \{ \max A, \min A \} \}, \) from which it follows that \( \text{MAX Sol}(A) \) is \( \text{APX} \)-hard if \( 0 \notin A \), and \( \text{poly-APX} \)-hard if \( 0 \in A \) (Proposition 2.29).

- Let \( A \) be term equivalent to \((A; R(\text{Iso } A) \cap E_m)\) for some \( 1 \leq m \leq v(A) \) or \( m = |A| \). When \( m = 1 \), then the switching operation \( s \) is idempotent, it preserves \( L_{a,b} \) for any \( a, b \in A \), and \( s \in R(\text{Iso } A) \) from Lemma 3.28. Therefore, \( s \in R(\text{Iso } A) \cap E_1 \) and \( \text{MAX Sol}(A) \) is in \( \text{APX} \) by Proposition 3.27(2). Note that if we identify \( a \) with 0 and \( b \) with 1, then \( L_{a,b} \in \text{Inv}(A) \) such that \( a < b \). Assume instead that \( m = 2 \). The definition of \( E_2 \) implies that each operation in \( A \) restricts to a projection on \( \{a, b\} \) for arbitrary, distinct \( a, b \in A \). Therefore, every two-element subalgebra \( A' = A_{\{a, b\}} \) is trivial, and the corresponding decision problem \( \text{Csp}(\text{Inv}(A')) \) is \( \text{NP} \)-hard. Now, \( \text{NP} \)-hardness of \( \text{Csp}(\text{Inv}(A)) \) follows from the inclusion \( \text{Inv}(A') \subseteq \text{Inv}(A) \). Furthermore, since \( E_2 \supseteq E_k \supseteq R(\text{Iso } A) \cap E_k \) for any \( 2 < k \leq v(A) \) as well as for \( k = |A| \), it follows that \( \text{MAX Sol}(A) \) is \( \text{NP} \)-hard for all \( m \geq 2 \).

- Finally, we consider the exceptional cases 3 and 4. A 4-element abelian group \( A \) of exponent 2 is term equivalent to a two-dimensional vector space over a two-element field. Hence, both cases are \( \text{APX} \)-complete by Corollary 3.19.

The proof is complete. ■

A ternary operation \( m(a, b, c) \) is called a Mal'tsev operation if, for all \( a, b \in A \), it holds that \( m(a, b, b) = m(b, b, a) = a \). Examples of Mal'tsev operations include the operations \( t, s, \) and \( b \) for an abelian group \( A \). Bula-tov and Dalmau [21] have proved that \( \text{Csp}(\text{Inv}(m)) \) is in \( \text{P} \) for any Mal'tsev operation \( m \). Using this fact, and following the proof of Theorem 3.36, it is not hard to do the classification of \( \text{Csp}(A) \) when \( A \) is a symmetric algebra. We state it here for comparison with [22, Conjecture 7.5], which suggests that a finite idempotent algebra is \( \text{NP} \)-complete if there exists a non-trivial homomorphic image of one of its subalgebras, all of whose operations are
3.7. Discussion

It has been observed that classifying the complexity properties of $Csp$ for all strictly simple algebras can be seen as a possible "base case for induction" [22]. This is due to the necessary condition that for a tractable algebra, all of its subalgebras and homomorphic images must be tractable. Furthermore, it is sufficient to study surjective algebras with respect to $Csp$ since the application of a unary polymorphism to a set of relations does not change the computational complexity of the set [78]. For Max Sol, however, it is possible to turn a hard problem into an easy one by applying a unary polymorphism.

Example 3.38 A straightforward example is given by the relation

$$R = \{(0,0), (0,1), (1,0)\},$$

over the domain $\{0,1\}$. The problem Max Sol($\{R\}$) is poly-APX-hard. Let $f$ be the unary polymorphism $0(y) = 0$. The image $f(R)$ is the relation $\{(0,0)\}$ over the a single element domain $\{0\}$, which is trivially tractable.

However, there are some unary polymorphisms, $f$, which we cannot apply to a set of relations without changing its approximability (up to $S$-reductions.) If $f \in Pol(\Gamma)$, and $f(a) \geq a$ for all $a \in A$, then $f(\Gamma)$ $S$-reduces to $\Gamma$ (cf. Lemma 2.36.) Differently stated, we can always assume that $\Gamma$ is a max-core. A related notion is investigated in Chapter 6, which provides a possible way to make further use of the classification of Max Sol for strictly simple surjective algebras obtained in this chapter.

We have proved APX-hardness for optimising a linear objective function over some restricted systems of linear equations. The most extensive result
3. Affine Algebras and Approximability

was obtained when the sum of the coefficients in each equation must be 0. A weaker result was obtained for the case of homogeneous linear equations.

Based on the case $\mathbb{F} = \mathbb{Z}_2$, and the additional cases treated in this chapter, we expect the following conjecture to hold, in the general case.

**Conjecture 3.39** Let $\mathcal{A}$ be a finite affine algebra. If $\text{Inv}(\mathcal{A})$ is not max-$\mathcal{A}$-valid, then $\text{MAX SOL} (\text{Inv}(\mathcal{A}))$ is APX-hard.

Note that $\text{Inv}(\mathcal{A})$ being max-$\mathcal{A}$-valid is equivalent to $f \in \mathcal{A}$, where $f(y) = \text{max } \mathcal{A}$. According to Proposition 3.8, this is the case if and only if $M$ contains $(1, \text{max})$, i.e. if $R(1, \text{max}) = \{(r, r(\text{max})) \mid r \in R\}$ is a submodule of $M$.

In addition to looking into extending the results for homogeneous linear equations to general abelian groups, further attention should also be given to the submodules of $0 \times \mathbb{R} \mathcal{A}$. The corresponding linear equations come from subsets of $\Lambda_0(\mathcal{A})$, where, in addition to having the sum of the coefficients be 0, only constants from some subgroup of $\mathcal{A}$ are allowed.
Chapter 4

Minimisation of Integer Linear Programs

We will now change the scale with which we measure the approximability of optimisation problems to that of the approximation ratio which can be achieved by an algorithm for the problem. Specifically, we study the approximability of minimising integer linear programs with positive right-hand sides. Let $n$ and $m$ be positive integers, representing the number of variables and the number of inequalities, respectively. Let $x = (x_1, \ldots, x_n)^T$ be a vector of $n$ variables, $A$ be an integer $m \times n$ matrix, $b \in (\mathbb{Z}^+)^m$, and $c \in (\mathbb{Q}_{\geq 0})^n$. Finally, let $X$ be some given subset of $\mathbb{N}^n$. We consider here various restrictions of the following integer linear program:

\[
\begin{align*}
\text{Minimise} & \quad c^T x \\
\text{subject to} & \quad Ax \geq b, \\
& \quad x \in X. \quad (4.1)
\end{align*}
\]

Typically, $X$ is either $\mathbb{N}^n$ or $\{x \in \mathbb{Z}^n \mid 0 \leq x \leq d\}$ for some $d \in (\mathbb{Z}^+)^n$, where the inequalities are to hold componentwise. A commonly occurring instance of the latter case is when $X = \{0, 1\}^n$, so-called 0-1 programming. In all but very restricted cases, (4.1) is \text{NP}-hard to solve to optimality. Instead, the effort is directed towards finding approximation algorithms and improving the bound within which it is possible to find approximate solutions.

Let $a_j = (a_{j1}, \ldots, a_{jn}) \in \mathbb{Z}^n$ be the $j$th row of $A$. We will use the norm $\|a_j\|_1 = \sum_{i=1}^n |a_{ji}|$ as well as the \textbf{maximum absolute row sum norm} of $A$, defined as $\|A\|_{\infty} = \max_{1 \leq j \leq m} \|a_j\|_1$. Let $\text{IP}_k$ denote the problem of optimising (4.1), when the instances are restricted by $\|A\|_{\infty} \leq k$. We show that $\text{IP}_k$ can unconditionally be approximated within $k$ when $X = \mathbb{N}^n$, but cannot be approximated within $k - \varepsilon$, $\varepsilon > 0$, if Khot's Unique Games Conjecture (cf. Section 2.4.2) holds. We also show that finding a feasible solution to $\text{IP}_k$ is \text{NP}-hard in almost all cases when $X = \{0, \ldots, a-1\}^n$. 
We note that the problem IP_k can be represented as a “Min Sol”-problem over a possibly infinite domain. The constraints can be given by n-ary relations on the form \{x | \mathbf{a}^T x \geq \mathbf{b}\}, where \(b \in \mathbb{Z}^n\) and \(\mathbf{a} \in \mathbb{Z}^n\) is a vector with \(||\mathbf{a}\||_1 \leq k\). Hence, any constraint essentially depends on at most \(k\) of the variables. While this view will not be the focus for this chapter, it is revisited in the discussion in connection to an open problem.

Previous work

The approximability of the program in (4.1) has been extensively studied in the case where \(A\) is restricted to non-negative entries. In this case, the problem is usually referred to as a (generalised, or capacitated) covering problem. Among the problems described by such programs one finds the Minimum Knapsack Problem, Minimum Vertex Cover (and its k-uniform hypergraph counterpart, described in Section 4.1.1) and various network design problems [24]. We will refer to the program (4.1) with non-negative constraint matrix \(A\) as a CIP (covering integer program). Here, \(X\) is often taken to be \(\{x \in \mathbb{Z}^n | 0 \leq x \leq d\}\). Indeed, optimal solutions remain feasible after introduction of the bounds \(x_i \leq \lceil \max_j b_{ij} / a_{ij} \rceil\).

Hall and Hochbaum [68] restrict \(A\) in CIP to a 0/1-matrix and give an \(||A||_\infty\)-approximating algorithm for the case when \(X = \{0, 1\}^n\). Bertsimas and Vohra [11] study the general CIP with \(X = \{0, 1\}^n\) as well as \(X = \mathbb{N}^n\). They use both a randomised rounding heuristic with a nonlinear rounding function and deterministic rounding using information about the dual program. For \(X = \{0, 1\}^n\), they show that CIP can be approximated within \(||A||_\infty\) using both a deterministic rounding function and a dual heuristic. For \(X = \mathbb{N}^n\), they obtain an \(||A||_\infty + 1\) approximating algorithm. Carr, Fleischer, Leung and Phillips [24] lower the integrality gap of CIP by introducing additional inequalities into the program and improve on the constrained case when \(X = \{x \in \mathbb{Z}^n | 0 \leq x \leq d\}\) to obtain an approximation ratio equal to the maximal number of non-zero entries in a row of \(A\). Koufogiannakis and Young [98] presents an approximation algorithm for a general framework of monotone covering problems, with an approximation ratio equal to the maximal number of variables upon which a constraint depends. The constraints must be monotone (closed upwards), but can be non-convex. This framework in particular includes every problem which can be expressed as a CIP.

Chapter outline

We begin, in Section 4.1, with the unbounded case \(X = \mathbb{N}^n\) by showing how to approximate IP_k within \(k\). We also provide the corresponding lower bound conditioned on the Unique Games Conjecture. In Section 4.2, we look at IP_k for bounded domains and show that it is NP-hard to find a solution in almost all cases. The chapter is concluded in Section 4.3 with a discussion on possible extensions of the results.
4. Minimisation of Integer Linear Programs

4.1 Unbounded domain

We assume that $X = \mathbb{N}^n$ throughout this section. Lower bounds for $IP_k$ are discussed in Section 4.1.1. Here, we begin by proving the following result:

**Proposition 4.1** $IP_k$ can be approximated within $k$.

The problem $IP_k$ is solvable in polynomial time: initially, let $x_i = 0$ for all $i$, and for each inequality $x_i \geq b$, update $x_i$ to $\max\{x_i, b\}$. Any inequality of the form $-x_i \geq b$ implies that there are no solutions. In order to prove Proposition 4.1 for $k \geq 2$, we give a deterministic ‘rounding’-scheme, which produces an integer solution from a rational one, while increasing the value of the objective function by at most $k$. For an integer $k \geq 2$, and $x \in \mathbb{Q}_{\geq 0}$, define the following operation:

$$\hat{x} = \begin{cases} 0 & \text{if } 0 \leq x < 1/k \\ 1 & \text{if } 1/k \leq x < 2/k \\ [(k - 1)x] & \text{otherwise}. \end{cases}$$

For a vector $x = (x_1, x_2, \ldots, x_n)^T$, let $\hat{x} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_n)^T$. Note that $c^T \hat{x} \leq k \cdot c^T x$. We will show that in addition, $\hat{x}$ satisfies $A\hat{x} \geq b$ by showing that for any integer $b \geq 1$, we have $a \cdot \hat{x} \geq b$ whenever $a \cdot x \geq b$ for any vector $a = (a_1, \ldots, a_n)^T$ with $\|a\|_1 \leq k$. In order to do this, we first introduce a linear scaling of $\hat{x}$ which will be easier to work with. Let $x' = \hat{x}/(k - 1)$ and extend to vectors, $x' = (x'_1, x'_2, \ldots, x'_n)^T$, as before.

Our first step is to bound the difference $\Delta = a \cdot x - a \cdot x'$ from above. Let $\delta_i = a_i(x_i - x'_i)$ so that $\Delta = \sum_{i=1}^n \delta_i$. Let $t_i = \text{sgn}(a_i) \cdot x_i$ and $t'_i = \text{sgn}(a_i) \cdot x'_i$. Then, $\delta_i = |a_i|(t_i - t'_i)$. Figure 4.1 illustrates how the $t'_i$ are determined from the $t_i$ in the cases which give positive contributions to $\Delta$. Each arrow represents an interval, and for a $t_i$ in a particular interval, $t'_i$ can be found at the arrow head. Note that there are only two such intervals on the positive axis. To the left of $L_5$ follows an infinite sequence of left arrows, each of size equal to that of $L_5$.

![Figure 4.1: The intervals $L_1, \ldots, L_5$ represented by arrows.](image-url)
4.1. Unbounded domain

Formally, the intervals $L_i, i \geq 1$ are defined as follows:
\[
\begin{align*}
L_1 &= \{ x \in \mathbb{Q} \mid 1/(k-1) \leq x < 2/k \} \\
L_2 &= \{ x \in \mathbb{Q} \mid 0 \leq x < 1/k \} \\
L_3 &= \{ x \in \mathbb{Q} \mid -1/(k-1) < x \leq -1/k \} \\
L_4 &= \{ x \in \mathbb{Q} \mid -2/(k-1) \leq x \leq -2/k \} \\
L_i &= \{ x \in \mathbb{Q} \mid -(i-2)/(k-1) \leq x \leq -(i-3)/(k-1) \}, \ i \geq 5.
\end{align*}
\]

When $k = 2$, the interval $L_1$ vanishes while $L_3$ and $L_4$ become adjacent. Let $L = \bigcup_{i \geq 1} L_i$. Now, $\delta_i$ can be bounded as follows, given the value of $t_i$:
\[
\begin{align*}
0 \leq \frac{\delta_i}{|a_i|} < \frac{(k-2)k-1}{k(k-1)} & \quad \text{if } t_i \in L_1 \\
0 \leq \frac{\delta_i}{|a_i|} < \frac{1}{k} & \quad \text{if } t_i \in L_2 \\
0 \leq \frac{\delta_i}{|a_i|} \leq \frac{1}{k(k-1)} & \quad \text{if } t_i \in L_3 \\
0 \leq \frac{\delta_i}{|a_i|} \leq \frac{2}{k(k-1)} & \quad \text{if } t_i \in L_4 \\
0 \leq \frac{\delta_i}{|a_i|} < \frac{1}{k-1} & \quad \text{if } t_i \in L_j, j \geq 5 \\
\delta_i \leq 0 & \quad \text{if } t_i \notin L.
\end{align*}
\]

Note that when $k = 2$, the upper bound on $\delta_i/|a_i|$ for $t_i \in L_4$ is actually strict. In particular, $\delta_i < |a_i|/(k-1)$ for all $i \geq 1$.

**Lemma 4.2** Let $b \geq 1$ and $k \geq 2$ be integers. If $a \cdot x \geq b$ and $\|a\|_1 \leq k$, then $\Delta < 1$.

**Proof:** Assume that there is an index $l$ such that $t_l \notin L$. Then, $|a_i| > 0$ so $\sum_{i \neq l} |a_i| \leq k-1$. We then have
\[
\Delta \leq \sum_{i \neq l} \delta_i < \sum_{i \neq l} \frac{|a_i|}{k-1} \leq \frac{k-1}{k-1} = 1.
\] (4.2)

Therefore, we can assume that for all $1 \leq i \leq n$ we have $t_i \in L$. Let $X_1 = \{ i \mid t_i \in L_1 \}$ and $X_2 = \{ i \mid t_i \in L_2 \}$. We will bound $\Delta$ by separately bounding the three parts of the sum with index sets $X_1, X_2$ and $\{1, \ldots, n\} \setminus (X_1 \cup X_2)$. Let $p = \sum_{i \in X_1} |a_i|$ and $q = \sum_{i \in X_2} |a_i|$. Since $t_i \geq 0$ if and only if $i \in X_1 \cup X_2$, we must have that $\sum_{i \in X_1 \cup X_2} |a_i| t_i \geq a \cdot x \geq b \geq 1$, hence $p \cdot 2/k + q \cdot 1/k > 1 \Leftrightarrow 2p + q > k$. Upper bounding the three parts yields:
\[
\Delta < p \cdot \frac{k-2}{k(k-1)} + q \cdot \frac{1}{k} + (k-p-q) \cdot \frac{1}{k-1} = \frac{k^2-2p-q}{k(k-1)} < 1.
\]

The lemma follows. 

We can now use Lemma 4.2 to prove the following lemma, and complete the proof of Proposition 4.1.

**Lemma 4.3** Let $b \geq 1$ and $k \geq 2$ be integers. If $a \cdot x \geq b$ and $\|a\|_1 \leq k$, then $a \cdot \hat{x} \geq b$.
4. Minimisation of Integer Linear Programs

**Proof:** From Lemma 4.2, we have $\Delta = a \cdot x - a \cdot x' < 1$ which can be rearranged to $a \cdot x' > a \cdot x - 1 \geq b - 1$. Multiplication by $k - 1$ now yields

$$a \cdot \hat{x} = (k - 1)a \cdot x' > (k - 1)(b - 1). \quad (4.3)$$

When $k \geq 2$ and $b \geq 1$, we have $(k - 1)(b - 1) \geq b - 1$, so (4.3) implies that $a \cdot \hat{x} > b - 1$. It now follows from the integrality of $a \cdot \hat{x}$ that $a \cdot \hat{x}$ must in fact be greater than or equal to $b$.

**Proof:** [Proposition 4.1] It remains to prove the statement for $k \geq 2$. For an instance of $\text{IP}_k$, we solve the LP-relaxation and obtain a solution $\hat{x}$ such that $Ax \geq b$, and $c^T \hat{x}$ is less than or equal to the optimum of the corresponding integer program. In particular, $a \cdot \hat{x} \geq b$ for every row vector $a^T$ of $A$. By Lemma 4.3, it follows that $a \cdot \hat{x} \geq b$, and therefore $A\hat{x} \geq b$. The value of the objective function for the solution $\hat{x}$ is $c^T \hat{x} \leq k \cdot c^T \hat{x}$, hence we have approximated $\text{IP}_k$ within $k$.

### 4.1.1 Lower bounds

A $k$-uniform hypergraph $H$ is a pair $(V, E)$, where $V$ is a set of vertices and each hyperedge $e \in E$ is a $k$-element subset of $V$. The $k$-Vertex-Cover problem is that of finding a minimum size vertex cover in a $k$-uniform hypergraph. Note that $2$-Vertex-Cover is identical to the well-known Minimum Vertex Cover for ordinary graphs. Given $H$, there is an immediate reduction to a CIP with one variable $x_i$ for each vertex $v_i \in V$ and one inequality of the form

$$x_{i_1} + x_{i_2} + \ldots + x_{i_k} \geq 1$$

for each hyperedge $e_j = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\} \in E$. Here, the domain $X$ may be any superset of $\{0, 1\}^n$ since we can always obtain a feasible solution $x' = \min\{1, x\}$ (componentwise) with at least as small measure. Consequently, if $k$-Vertex-Cover is not approximable within a constant $\alpha$, then $\text{IP}_k$ is not approximable within $\alpha$ either.

The best lower bounds currently known under the assumption of $P \neq \mathsf{NP}$ is $1.3606$ for Minimum Vertex Cover by Dinur and Safra [40] and $k - 1 - \varepsilon$ for $k$-Vertex-Cover by Dinur et al. [39]. Stronger bounds are obtainable by exploiting stronger complexity theoretical assumptions such as Khot's Unique Games Conjecture: Khot and Regev [96] show that modulo the truth of this conjecture, $k$-Vertex-Cover cannot be approximated within $k - \varepsilon$ for any $\varepsilon > 0$. Thus, we have good reasons to believe that it may in fact be $\mathsf{NP}$-hard to approximate $k$-Vertex-Cover within $k - \varepsilon$ for any $\varepsilon$. Combined with Proposition 4.1, this bound yields a (conjectured) tight approximation constant of $k$ for $\text{IP}_k$. This bound also matches the upper bound on CIP by Carr et al. [24].
4.2 Bounded domain

Let \( X = \{0, 1, \ldots , a - 1\}, a \geq 2 \) and \( k \geq 3 \), with at least one of the inequalities strict. In this case, we show that it is \( \text{NP} \)-hard to find any feasible solution to \( IP_k \). The exceptional case \( a = 2 \) and \( k = 3 \) turns out to be approximable within \( 3 \). We will reduce from the problem \textsc{One-in-three SAT}, which was shown to be \( \text{NP} \)-hard by Schaefer [119]. It is defined as follows:

**Definition 4.4** \textsc{One-in-three SAT} is the decision problem with:

**Instance:** A set of clauses \( \{C_1, \ldots , C_m\} \) over variables \( U = \{u_1, \ldots , u_n\} \), where each clause is a disjunction \( u_i \lor u_j \lor u_k \) of exactly three variables.

**Question:** Is there a satisfying assignment \( \varphi : U \rightarrow \{0,1\} \) such that precisely one variable in each clause is assigned the value \( 1 \)?

Note that we do not allow negations of the variables. This is in agreement with Schaefer’s original formulation.

**Proposition 4.5** Let \( a \geq 2 \) and \( X = \{0, 1, \ldots , a - 1\}^n \). When \( k \geq 4 \), it is an \( \text{NP} \)-hard problem to find feasible solutions to \( IP_k \).

**Proof:** Let \( \{C_1, \ldots , C_m\} \) be an \textsc{One-in-three SAT}-instance over \( U \). For each instance variable \( u_i \in U \), create a variable \( x_i \), and for each clause \( C = u_i \lor u_j \lor u_k \), add the following inequalities:

\[
\begin{align*}
  x_i, x_j, x_k & \geq a - 2 \\
  x_i + x_j + x_k & \geq 3(a - 2) + 1 
\end{align*}
\]

(4.4)

The first equation restricts the variables \( x_i, x_j, x_k \) to the set \( \{a - 2, a - 1\} \). The second equation ensures that at least one of the variables \( x_i, x_j \) and \( x_k \) is assigned the value \( a - 1 \). Furthermore, we add a (unique) new variable \( y \), and the following inequalities:

\[
\begin{align*}
  y & \geq a - 1 \\
  x_i + x_j & \leq 2y - 1 \Leftrightarrow 2y - x_i - x_j \geq 1 \\
  x_i + x_k & \leq 2y - 1 \\
  x_j + x_k & \leq 2y - 1 
\end{align*}
\]

(4.5)

Since \( y \) must be \( a - 1 \), the last three inequalities, together with the fact that \( x_i, x_j, x_k \in \{a - 2, a - 1\} \) implies that at most one variable from \( \{x_i, x_j, x_k\} \) takes the value \( a - 1 \). We can thus solve the original \textsc{One-in-three SAT}-instance by assigning \( \varphi(u_i) = 0 \) if \( x_i = a - 2 \) and \( \varphi(u_i) = 1 \) if \( x_i = a - 1 \). It follows that finding a solution to \( IP_k \) is \( \text{NP} \)-hard.

It turns out that we can reduce the number of variables per inequality by one if we allow at least three values in the domain.

**Corollary 4.6** If \( X = \{0, 1, \ldots , a - 1\}^n \) with \( a > 2 \), then the problem of finding a feasible solution to \( IP_3 \) is \( \text{NP} \)-hard.
Proof: The proof of Proposition 4.5 can be altered in the following way to produce the result in the corollary. First, replace the equations (4.4) with \( x_i + x_j + x_k \geq 1 \). Then, replace \( 2y - 1 \) in equations (4.5) with \( y - (a - 2) \). Finally, let \( \varphi(u_i) = 0 \) if \( x_i = 0 \) and \( \varphi(u_i) = 1 \) otherwise.

One case is left out from the two previous reductions, namely IP3 with \( X = \{0,1\}^n \). It turns out that this remaining case can be approximated using a straightforward rounding scheme.

**Proposition 4.7** The problem IP3 with domain \( X = \{0,1\}^n \) can be approximated within 3.

**Proof:** Let \( x \) be an optimal solution to the LP-relaxation of IP3. We round \( x \) to an integer solution \( \hat{x} \) as follows:

\[
\hat{x}_i = \begin{cases} 
0 & \text{if } x_i < 1/3, \\
1 & \text{otherwise.}
\end{cases}
\]

This increases the value of the objective function by at most 3. The proof follows a similar strategy as that of Proposition 4.1. Define \( t_i = \text{sgn}(a_i) \cdot x_i \) and \( \hat{t}_i = \text{sgn}(a_i) \cdot \hat{x}_i \). We note again that it suffices to show that

\[
\Delta = a \cdot x - a \cdot \hat{x} = \sum_{i=1}^{n} |a_i|(|t_i - \hat{t}_i| < 1,
\]

since we then have \( a \cdot \hat{x} > a \cdot x - 1 \geq b - 1 \), and the result follows from the integrality of \( a \cdot \hat{x} \).

Let \( X_1 = \{i \mid -1 \leq t_i \leq -1/3\} \) and \( X_2 = \{i \mid -1/3 < t_i < 1/3\} \), and let \( p = \sum_{i \in X_1} |a_i| \) and \( q = \sum_{i \in X_2} |a_i| \). The values of \( p \) and \( q \) must satisfy

\[
b \leq a \cdot x \leq p \cdot (-1/3) + q \cdot 1/3 + (3 - p - q) \cdot 1
\]

which implies \( p \cdot 2/3 + q \cdot 1/3 \leq 1 \), where we have used 1 as a lower bound for \( b \). In fact, we have \( p \cdot 2/3 + q \cdot 1/3 < 1 \); if \( q = 0 \), this follows immediately from the non-strict inequality, and if \( q > 0 \), then the term \( q \cdot 1/3 \) in (4.6) is a strict upper bound on \( \sum_{i \in X_2} t_i \).

To finish the proof, note that \( \{i \mid t_i - \hat{t}_i > 0\} \subseteq X_1 \cup X_2 \). We therefore have the bound

\[
\Delta \leq \sum_{i \in X_1 \cup X_2} |a_i|(|t_i - \hat{t}_i| \leq p \cdot 2/3 + q \cdot 1/3 < 1,
\]

and the proposition follows.
4.3 Discussion and future work

We have obtained a tight approximation of a general class of integer linear programs under the parameterisation $\|A\|_\infty \leq k$. It is however important to note, that the result in Section 4.1 is tight only with respect to this particular parameterisation. It is still imaginable that there exists an approximation algorithm which approximates IP, within, for example, the maximum number of non-zero entries in any row of $A$ as was the case for the CIP problems. The approach of Carr et al. [24] is based on adding so-called Knapsack Cover (KC) inequalities to the program, which lowers the integrality gap. The exponentially many inequalities are then handled using a separation oracle. We note that for (4.1), there seems to be no natural counterpart to the KC-inequalities. The main obstacle is that the validity of the inequalities of (4.1) does not uniformly improve upon increasing individual variables, as is the case for ordinary covering problems.

A first step in this direction could be to look for a 2-approximation of (4.1) with at most two variables per inequality (or, indeed, prove that such an algorithm is unlikely to exist.) An algorithm is known for arbitrary right-hand sides when the variables are bounded, see Hochbaum et al. [72]. The idea behind the proof is to reduce the problem to program with only monotone inequalities ($ax - by \geq c$, where $a, b > 0$.) This system can then be solved in pseudo-polynomial time, depending on the square of the upper bounds of the variables. The value of the final solution can then easily be seen to be off by at most a factor of 2. To use a similar approach, one would like to prove that polynomial time is retained for monotone inequalities, when arbitrary right-hand sides and bounded domain is substituted with positive right-hand sides and unbounded domain. We note that this can be seen as a constraint satisfaction problem over an infinite domain, and that the constraint language of monotone inequalities is invariant under the operations min and max.

Dobson [41] and Fisher and Wolsey [54] both analyse greedy algorithms for CIP and derive bounds of $O(\log d)$, where $d$ is the maximum column sum of $A$. As for the KC-inequalities, the correctness of these algorithms crucially uses the non-negativity of the $A$-matrix, and a direct generalisation to IP fails. Nevertheless, it seems reasonable to assume that some kind of column-sum bound for IP should exist.
Chapter 5

Approximation Distance

In this chapter, we continue to study approximation ratios for optimisation problems. We introduce a novel method designed to extend known approximation ratios for one problem to bounds on the ratio for other problems. We start out in a quite general setting, but will keep things relatively informal before we concretise. The added generality is only meant to highlight the underlying idea—our results reside almost exclusively in the concrete setting.

General formulation

Let $\mathcal{P}$ be a family of maximisation problems parameterised by a family $\mathcal{M}$ of structures, so that $\mathcal{P}(M)$ is an optimisation problem for each $M \in \mathcal{M}$. Assume that the set of instances, $I_{\mathcal{P}(M)}$, is the same for every structure $M$, and call this set $I_{\mathcal{P}}$. Denote by $\text{Opt}_M(I)$ the optimal measure of a solution to $I$ as an instance of the problem $\mathcal{P}(M)$. We will assume that all measures are non-negative. Finally, we will assume that there is a quasi-order (a reflexive and transitive binary relation) $\preceq$ on the set of structures such that if $M \preceq N$, then there exists a polynomial-time computable function $F_{M,N} : \text{Sol}_{\mathcal{P}(M)}(I) \rightarrow \text{Sol}_{\mathcal{P}(N)}(I)$ such that $m_M(f) \leq m_N(F(f))$. In particular, $M \preceq N$ implies $\text{Opt}_M(I) \leq \text{Opt}_N(I)$.

The basic idea is to ask how far off we would be if, instead of solving an instance $I \in I_{\mathcal{P}}$ for $\mathcal{P}(N)$, we solve it for the problem $\mathcal{P}(M)$. We will take the following quantity as the answer to this question:

$$s(M, N) = \inf_{I \in I_{\mathcal{P}}} \frac{\text{Opt}_M(I)}{\text{Opt}_N(I)}.$$  \hfill (5.1)

Some care must be taken to ensure that $\text{Opt}_N(I)$ is not 0, for example by excluding all such instances from the set $I_{\mathcal{P}}$. Assuming $M \preceq N$, we have $0 \leq s(M, N) \leq 1$, and if there are instances for which $\text{Opt}_M$ and $\text{Opt}_N$ are very different, then $s(M, N)$ is close to 0. On the other hand, if $s$ is close to 1, then for all instances, $\text{Opt}_M$ and $\text{Opt}_N$ are similar, and hence we expect
that a bound for the approximation ratio of $\mathcal{P}(M)$ should translate into some bound for $\mathcal{P}(N)$. Indeed, this is precisely what happens. From (5.1), it follows that for any $\mathcal{I} \in I_P$, we have

$$\text{Opt}_M(\mathcal{I}) \geq s(M, N) \cdot \text{Opt}_N(\mathcal{I}).$$

(5.2)

Using this observation, we can show the following:

**Lemma 5.1** Let $M \leq N$ be two structures. If $\text{Opt}_M$ can be approximated within $\alpha$, then $\text{Opt}_N$ can be approximated within $\alpha \cdot s(M, N)$.

**Proof:** Let $A(x)$ be the solution returned by an algorithm which approximates $\text{Opt}_M$ within $\alpha$. For any $\mathcal{I} \in I_P$, we have $m_M(A(\mathcal{I})) \geq \text{Opt}_M(\mathcal{I}) \cdot \alpha$, and also (5.2). Consequently,

$$\text{Opt}_N(\mathcal{I}) \geq m_N(A'(\mathcal{I})) \geq m_M(A(\mathcal{I})) \geq \text{Opt}_M(\mathcal{I}) \cdot \alpha \geq \text{Opt}_N(\mathcal{I}) \cdot s(M, N) \cdot \alpha,$$

where $A'(\mathcal{I}) = F_{M,N}(A(\mathcal{I}))$. The conclusion is that if we solve $\mathcal{I}$ for $\mathcal{P}(M)$ using the algorithm $A$, then the measure of the solution $A'(\mathcal{I})$ is within $\alpha \cdot s(M, N)$ of the optimum of $\mathcal{I}$ as an instance of $\mathcal{P}(N)$. \hfill \blacksquare

The contrapositive of Lemma 5.1 is also of interest: if $M \leq N$ and $\text{Opt}_N$ can not be approximated within $\beta$, then $\text{Opt}_M$ cannot be approximated within $\beta/s(M, N)$, unless $P = \text{NP}$. Hence $s(M, N)$ allows us to extend known upper and lower bounds on the approximation ratio of one problem $\mathcal{P}(M)$ to bounds for another problem $\mathcal{P}(N)$, provided that $s(M, N)$ is close to 1. Clearly, to gain some advantage from this method, we must be able to determine $s(M, N)$ in some generality. From the definition (5.1) it is not at all obvious how to do this, or even if it can be done within reasonable time restrictions, but we will see that this is in fact possible in the cases we study.

**Graph homomorphism problems**

We now concretise the method just described by assuming that the family $\mathcal{P}$ is some optimisation variation of a graph homomorphism problem on undirected graphs. We note already here that we could have chosen to work with general homomorphism problems between relational structures without changing much of the results. The two main reasons for choosing undirected graphs are that things tend to be easier to visualise in this setting, and secondly, that graph homomorphisms are understood in much greater detail than their general counterparts. But see the discussion in Section 5.7 for some considerations on possible generalisations of the obtained results.

Let $G$ and $H$ be undirected graphs. If both $G$ is homomorphic to $H$ and $H$ is homomorphic to $G$, then we say that $G$ and $H$ are **homomorphically equivalent** and denote this by $G \equiv H$. The set of all homomorphisms from
5. Approximation Distance

$G$ to $H$ is denoted by $\text{Hom}(G, H)$. The family $M$ of structures, as well as the instances, will consist of undirected, sometimes weighted, graphs. The weights on the graphs will affect the measure $m$ and, in one case, also the relation $\preceq$.

We consider two different problems of this type. In addition to the usual suspect, MAX SOL, we will look at the following problem:

**Definition 5.2** The weighted maximum $H$-colourable subgraph problem, or MAX $H$-Col, for short, is the maximisation problem with

**Instance:** An undirected graph $G$ with edge-weight $\omega : E(G) \to \mathbb{Q}_{\geq 0}$.

**Solution:** A subgraph $G' \subseteq G$ such that $G' \rightarrow H$.

**Measure:** The weight of $G'$ with respect to $\omega$.

The solutions to a MAX $H$-Col instance have an alternative description, which is better suited for our needs: for any vertex map $f : V(G) \rightarrow V(H)$, let $f^\# : E(G) \rightarrow E(H)$ be the (partial) edge map induced by $f$. Each function $f$ then uniquely determines a subgraph $G' = (f^\#)^{-1}(H) \subseteq G$, but two distinct functions, $f$ and $g$, do not necessarily determine distinct subgraphs. In this notation $h : V(G) \rightarrow V(H)$ is a graph homomorphism precisely when $(h^\#)^{-1}(E(H)) = E(G)$ or, alternatively when $h^\#$ is a total function. The set of solutions to an instance $(G, \omega)$ of MAX $H$-Col can then be taken to be the set of vertex maps $f : V(G) \rightarrow V(H)$ with the measure

$$m_H(f) = \sum_{e \in (f^\#)^{-1}(E(H))} \omega(e).$$

(5.3)

We will predominantly use this description of solutions.

The following are short instantiations of the general method for our two problems of interest. The details are worked out in Sections 5.1 through 5.4 for MAX $H$-Col and in Section 5.6 for MAX SOL.

$P(M) = \text{MAX } M$-Col

We use as solution set $\text{Sol}_P(M) = V(M)^{V(G)}$, the set of all functions from $V(G)$ to $V(M)$. The instance graph $G$ is edge-weighted by a function $\omega : E(G) \rightarrow \mathbb{Q}_{\geq 0}$. The measure of $f \in \text{Sol}_P(M)$ is given by (5.3). The relation $\preceq$ is taken to be the homomorphism relation $\rightarrow$, and we must check that it satisfies the requirements. Let $M \rightarrow N$, $f \in \text{Sol}_P(M)$ and $h \in \text{Hom}(M, N)$. Then, $g = h \circ f$ is a solution to MAX $N$-Col, and $(g^\#)^{-1} = (f^\#)^{-1} \circ (h^\#)^{-1}$. Since $(h^\#)^{-1}(E(N)) = E(M)$, we have $(g^\#)^{-1}(E(N)) = (f^\#)^{-1}(E(M))$, which in turn implies $m_N(h \circ f) = m_M(f)$, as required.
\[ \mathcal{P}(M) = \max \text{Sol}(M) \]

In this case \( \text{Sol}_{\mathcal{P}(M)}(G) = \text{Hom}(G, M) \). The graphs \( G \) and \( M \) have vertex weights given by the functions \( \omega : V(G) \to \mathbb{Q}_{\geq 0} \) and \( \nu : V(M) \to \mathbb{Q}_{\geq 0} \), respectively. For a solution \( f \), we have \( m_M(f) = \sum_{v \in V(G)} \omega(v) \cdot \nu(f(v)) \). The relation \( \preceq \) is not completely obvious, and will depend on the vertex weights. We give one possible definition in Section 5.6.

For a couple of reasons we mainly cover the problem \( \max H\text{-Col} \). The definitions turn out to be more natural and easier to deal with for \( \max H\text{-Col} \) than for \( \max \text{Sol}(M) \). More importantly, there has been much more research conducted on the approximability of \( \max H\text{-Col} \) and this is vital to the application of our method.

**Chapter outline**

In Section 5.1, we begin by (re-)introducing \( \max H\text{-Col} \) as a generalisation of \( \max k\text{-cut} \), and as a special case of \( \max \text{Csp} \). We review known results on exact approximation ratios of this problem, including some upper bounds obtained by Khot et al. [95] under the assumption of the Unique Games Conjecture. The basic properties of \( s \) in this setting are worked out in Section 5.2, and we show that the expression for \( s \) can be drastically simplified when \( N \) is an edge-transitive graph. A linear program is presented for the computation of \( s \), and we exemplify this in some small cases. We also introduce a closely related parameter \( d \), which is shown to be a metric on the space of undirected graphs, modulo homomorphic equivalence.

In Section 5.3 we use the tools developed in Section 5.2 to make some hands-on computations of \( s \) for some families of graphs. In Section 5.4, we use \( s \) for studying the approximability of \( \max H\text{-Col} \) and investigate optimality issues, for several classes of graphs. Comparisons are made to the bounds achieved by the general \( \max 2\text{-Csp} \)-algorithm by Håstad [75]. Our investigation covers a spectrum of graphs, ranging from graphs with few edges and/or containing long shortest cycles to dense graphs containing \( \Theta(n^2) \) edges. Dense graphs are considered from two perspectives: first as graphs having a number of edges close to maximal and secondly as graphs from the \( G(n,p) \) model of random graphs, pioneered by Erdős and Rényi [48].

In Section 5.5, we start out studying a seemingly different problem, formalised as a fractional edge-covering problem, and a related “chromatic number”. It turns out that this construction is strongly related to \( s \), and that we can pass effortlessly between the two views, gaining insights into both. This part is highly inspired by work of Šámal on cubical colourings [122, 123, 124]. We work out some details for the \( \max \text{Sol} \) problem as well, in Section 5.6. Finally, we summarise the future prospects and open problems of the method in Section 5.7.
5. Approximation Distance

5.1 Approximation distance for $\text{MAX } H\text{-COL}$

Let $G$ be a simple and undirected graph. Given a subset $S \subseteq V(G)$, a cut in $G$ with respect to $S$ is the set of edges from a vertex in $S$ to a vertex in $V(G) \setminus S$. The MAX cut problem asks for the size of a largest cut in $G$. More generally, a $k$-cut in $G$ is the edges going from $S_i$ to $S_j$, $i \neq j$, where $S_1, \ldots, S_k$ is a partitioning of $V(G)$, and the MAX $k$-cut problem asks for the size of a largest $k$-cut (cf. Section 2.4.1.) The problem is readily seen to be equivalent to finding a largest $k$-colourable subgraph of $G$. As we have noted, MAX $k$-cut is APX-complete for every $k \geq 2$ and consequently does not admit a polynomial-time approximation scheme (PTAS).

Since the 1970s, the trivial approximation ratio of 1/2, obtained by the argument in Example 1.5, was the best known for MAX cut. It was not until 1995 that Goemans and Williamson [58], using semidefinite programming (SDP), achieved a ratio of 0.87856. Until very recently no other method than SDP was known to yield a non-trivial approximation ratio for MAX cut. Trevisan [128] broke this barrier by using algebraic graph theory techniques to reach an approximation guarantee of .531. Frieze and Jerrum [55] determined lower bounds on the approximation ratios for MAX $k$-cut using SDP techniques. Sharpened results for small values of $k$ have later been obtained by de Klerk et al. [38].

Under the assumption that the Unique Games Conjecture (UGC) holds, Khot et al. [55] proved the approximation ratio for $k = 2$ to be essentially optimal and also provided upper bounds on the approximation ratio for $k > 2$. Hästad [72] has shown that SDP is a universal tool for solving the general MAX 2-CSP problem over any domain, in the sense that it establishes non-trivial approximation results for all of those problems. Assuming UGC, Raghavendra’s SDP algorithms [116] have optimal performance for every MAX CSP, but the exact approximation ratios are not yet known and may be hard to compute. In combination with our results, such ratios could be used to confirm or disprove UGC.

Here, we study approximability properties of MAX $H$-Col for undirected graphs $H$, a problem which generalises MAX $k$-cut, and specialises the general MAX CSP problem. Jonsson et al. [81] have shown that whenever $H$ is loop-free, MAX $H$-Col does not admit a PTAS, and otherwise the problem is trivial. Langberg et al. [103] have studied the approximability of MAX $H$-Col when $H$ is part of the input. We present approximability results for MAX $H$-Col, where $H$ is taken from different families of graphs. Many of these results turn out to be close to optimal under UGC.

Denote by $G$ the set of all non-empty, simple, and undirected graphs. For a graph $G \in G$, let $W(G)$ be the set of weight functions $\omega : E(G) \rightarrow \mathbb{Q}_{>0}$. We will further assume that $\omega \in W(G)$ is not identically 0. For $\omega \in W(G)$, we let $\|\omega\|_1 = \sum_{e \in E(G)} w(e)$ denote the total weight of $G$. Given an edge-weighted graph $(G, \omega)$, denote by $mc_H(G, \omega)$ the measure of an optimal solution to the problem MAX $H$-Col. Denote by $mc_k(G, \omega)$ the (weighted)
size of a largest $k$-cut in $(G, \omega)$. This notation is justified by the fact that
\[ mc_k(G, \omega) = mc_{K_k}(G, \omega) \] and in this sense, MAX $H$-Col generalises MAX $k$-cut. The decision version of MAX $H$-Col, the $H$-COLOURING problem, has been extensively studied (see [70] and its many references). Hell and Nešetřil [71] were the first to show that this problem is in $P$ if $H$ contains a loop or is bipartite, and $NP$-complete otherwise.

The parameter $s$ for MAX $H$-Col

We fix the family of problems to $\mathcal{P} = \{ \text{MAX } H\text{-Col} \}_{H \in \mathcal{G}}$. The definition of $s$ in (5.1) then takes the following form:

\[
s(M, N) = \inf_{\omega \in \bar{\mathcal{W}}(G)} \frac{mc_M(G, \omega)}{mc_N(G, \omega)}. \tag{5.4}
\]

Note that $mc_N(G, \omega)$ is always non-zero, as $G$ and $N$ are both non-empty, and $\omega$ is not identically 0.

**Lemma 5.3** Let $M \rightarrow N$ be two graphs. If $mc_M$ can be approximated within $\alpha$, then $mc_N$ can be approximated within $\alpha \cdot s(M, N)$. If it is $NP$-hard to approximate $mc_N$ within $\beta$, then $mc_M$ is not approximable within $\beta/s(M, N)$, unless $P = NP$.

**Example 5.4** The algorithm by Goemans and Williamson for MAX cut[58] is a 0.87856-approximating algorithm for MAX $K_2$-Col. In Section 5.3, we will see that $s(K_2, C_{11}) = 10/11$. We can now apply Lemma 5.3 to $K_2 \rightarrow C_{11}$, and we find that this MAX cut-algorithm approximates MAX $C_{11}$-Col within $0.87856 \cdot s(K_2, C_{11}) \approx 0.79869$.

### 5.2 Properties of the parameter $s$

In this section we start out by establishing some basic properties of the parameter $s$, essentially that it is invariant under homomorphic equivalence. We then move on to finding a simplified expression for $s$, more suitable for computations. The section is concluded by a particularly useful result, Lemma 5.12, which we will use repeatedly throughout this chapter. We also determine how $s$ behaves under categorical product and disjoint union.

**Lemma 5.5** Let $M, N \in \mathcal{G}$ and $M \rightarrow N$. Then, for every $G \in \mathcal{G}$ and every weight function $\omega \in \mathcal{W}(G)$,

\[ mc_M(G, \omega) \leq mc_N(G, \omega). \]

**Proof**: If $G' \rightarrow M$ for some subgraph $G'$ of $G$, then $G' \rightarrow N$ as well. The lemma immediately follows.
5. Approximation Distance

Corollary 5.6 If $M \equiv N$, then $mc_M(G, \omega) = mc_N(G, \omega)$.

Corollary 5.7 Let $M_1 \equiv M_2$ and $N_1 \equiv N_2$ be two pairs of homomorphically equivalent graphs. Then, $s(N_i, M_j) = s(N_k, M_l)$ for $i, j, k, l \in \{1, 2\}$.

Proof: Corollary 5.6 says that for all $G \in \mathcal{G}$ and $\omega \in W(G)$, we have

$$
\frac{mc_M(G, \omega)}{mc_N(G, \omega)} = \frac{mc_M(G, \omega)}{mc_N(G, \omega)}.
$$

Now, take infima over graphs $G$ and weight functions $\omega$ on both sides. ■

5.2.1 Exploiting symmetries

We are interested in finding an expression for $s$ which can be computed more easily than the one in (5.4). In Lemma 5.9, we show that a particular type of weight function provides a lower bound on $mc_M(G, \omega)/mc_N(G, \omega)$, and in Lemma 5.11, we provide a simpler expression for $s(M, N)$ which depends directly on the automorphism group and thereby the symmetries of $N$. This expression becomes particularly simple when $N$ is edge-transitive. An immediate consequence of this is that, for edge-transitive graphs $H$, we have $s(K_2, H) = b(H)$, where $b(H)$ is the bipartite density of $H$: the bipartite density can be defined as the ratio between the maximum cut, and the number of edges in $H$, and is a well-studied graph parameter [2, 10, 16, 73, 105].

The optimum $mc_H(G, \omega)$ is sub-linear with respect to the weight function, as is shown by the following lemma.

Lemma 5.8 Let $G, H \in \mathcal{G}$, $\alpha \in \mathbb{Q}_{\geq 0}$ and let $\omega, \omega_1, \ldots, \omega_r \in W(G)$ be weight functions on $G$. Then,

- $mc_H(G, \alpha \cdot \omega) = \alpha \cdot mc_H(G, \omega)$,
- $mc_H(G, \sum_{i=1}^r \omega_i) \leq \sum_{i=1}^r mc_H(G, \omega_i)$.

Proof: The first part is trivial. For the second part, let $G'$ be an optimal solution to the instance $(G, \sum_{i=1}^r \omega_i)$ of MAX $H$-Col. Then, the measure of this solution equals the sum of the measures of $G'$ as a (possibly suboptimal) solution to each of the instances $(G, \omega_i)$. ■

Let $f : V(G) \to V(H)$ be an optimal solution to the instance $(G, \omega)$ of MAX $H$-Col. Define $\omega_f \in W(H)$ as follows:

$$
\omega_f (e) = \sum_{e' \in (f^{(r)})^{-1}(e)} \frac{\omega(e')} {mc_H(G, \omega)}.
$$

(5.5)

The following lemma is the warm-up for Lemma 5.11.
Lemma 5.11 Let \( M, N \in \mathcal{G} \) be two graphs. Then, for every \( C \subseteq \mathcal{G} \) a graph, \( s(M, N) = \min_{C} \{s(C, N)\} \).

In particular, when \( N \) is edge-transitive,
\[
s(M, N) = \min_{w \in W} s(M, N, w).
\]

Proof. Let \( M, N \in \mathcal{G} \) be two graphs and let \( \Delta = \text{Aut}(N) \) be the (edge) automorphism group of \( N \), i.e., \( \Delta \subseteq E(N) \) by permuting the edges. The

proof of Lemma 5.9 shows that \( s(M, N) \geq \min_{w \in W} \{s(M, N, w)\} \).

This concludes the proof.

Corollary 5.10 Let \( M, N \in \mathcal{G} \) be two graphs. Then,
\[
s(M, N) \leq \min_{w \in W} \{s(M, N, w)\}.
\]

From Lemma 5.9, we have the following corollary which shows that it is possible to eliminate \( C \subseteq \mathcal{G} \) from the minimum in the definition of
\[
\text{mcx}(G, \omega).
\]

Proof. Arbitrarily choose an optimal solution \( f : V(N) \to \{0, 1\} \) to the

instance \((N, \omega)\) of Max MC. That is, \( f \) is a solution to \((G, \omega)\) as an instance of Max MC. Let \( \omega \in W \) and any optimal solution \( f \) to \((G, \omega)\) of Max MC. It holds
\[
\text{mcx}(G, \omega) \geq \text{mcx}(G, \omega).
\]

5.2. Properties of the parameters
5. Approximation Distance

Proof: From Corollary 5.10, we have that

\[
s(M, N) = \inf_{\omega \in \mathcal{W}(N)} \frac{mc_M(N, \omega)}{mc_N(N, \omega)} \leq \inf_{\omega \in \mathcal{W}(N)} mc_M(N, \omega).
\]

To complete the first part of the lemma, it will be sufficient to prove that for any graph \( G \in \mathcal{G} \) and \( \omega \in \mathcal{W}(G) \), there is an \( \omega' \in \mathcal{W}(N) \) such that the following inequality holds:

\[
\frac{mc_M(G, \omega)}{mc_N(G, \omega)} \geq mc_M(N, \omega').
\]

(5.6)

Taking infima on both sides of this inequality then shows that

\[
s(M, N) \geq \inf_{\omega' \in \mathcal{W}(N)} mc_M(N, \omega').
\]

Let \( A = \text{Aut}^*(N) \) be the automorphism group of \( N \) and let \( \pi \in A \) be an arbitrary automorphism of \( N \). If \( f \) is an optimal solution to \( (G, \omega) \) as an instance of \( \text{Max}_N \text{-Col} \), then so is \( f_\pi = \pi \circ f \). By Lemma 5.9, inequality (5.6) is satisfied by \( \omega_{\pi_0f} \). Summing \( \pi \) in this inequality over \( A \) gives

\[
|A| \cdot \frac{mc_M(G, \omega)}{mc_N(G, \omega)} \geq \sum_{\pi \in A} mc_M(N, \omega_{\pi_0f}) \geq mc_M(N, \sum_{\pi \in A} \omega_{\pi_0f}),
\]

where the last inequality follows from Lemma 5.8. The weight function \( \sum_{\pi \in A} \omega_{\pi_0f} \) is determined as follows:

\[
\sum_{\pi \in A} \omega_{\pi_0f}(e) = \sum_{\pi \in A} \frac{\sum_{e' \in (f \pi)^{-1}(e)} \omega(e')}{mc_N(G, \omega)}
\]

\[
= \frac{|A|}{|\text{Orbit}(e)|} \cdot \frac{\sum_{e' \in (f \pi)^{-1}(\text{Orbit}(e))} \omega(e')}{mc_N(G, \omega)},
\]

where \( \text{Orbit}(e) \) denotes the orbit of \( e \) under \( A \). We have now shown that the inequality in (5.6) is satisfied by \( \omega' = \sum_{\pi \in A} \omega_{\pi_0f}/|A| \), and that \( \omega' \) is in \( \mathcal{W}(N) \). The first part of the lemma follows.

For the second part, note that when the automorphism group \( A \) acts transitively on \( E(N) \), there is only one orbit \( Ae = E(N) \) for all \( e \in E(N) \). Then, the weight function \( \omega' \) is given by

\[
\omega'(e) = \frac{1}{e(N)} \cdot \frac{\sum_{e' \in (f \pi)^{-1}(E(N))} w(e')}{mc_N(G, \omega)} = \frac{1}{e(N)} \cdot \frac{mc_N(G, \omega)}{mc_N(G, \omega)},
\]

since \( f \) is optimal.

We see from Lemma 5.11 that when \( H \) is edge-transitive, we have \( s(K_2, H) = mc_2(H, 1/e(H)) = b(H) \), i.e. \( s(K_2, H) \) is the bipartite density of \( H \). We note that while \( s \) is invariant under (isomorphic) equivalence, this is not true for bipartite density.
5.2. Properties of the parameter $s$

5.2.2 A sandwiching lemma

In some cases, for graphs $M$, $H$, and $N$, it may be difficult to determine $s(M, H)$, or $s(H, N)$ directly. However, if we know that $H$ is “homomorphically sandwiched” between $M$ and $N$, so that $M \rightarrow H \rightarrow N$, then we can provide an upper bound on these parameters by using $s(M, N)$. More generally, we have the following lemma:

**Lemma 5.12** Let $M \rightarrow K$ and $H \rightarrow N$. Then,

$$s(M, H) \geq s(M, N) \quad \text{and} \quad s(K, N) \geq s(M, N).$$

**Proof:** Since $H \rightarrow N$, it follows from Lemma 5.5 that $mc_H(G, w) \leq mc_N(G, w)$. Thus,

$$s(M, H) = \inf_{\omega \in \mathcal{O}(G)} \frac{mc_M(G, \omega)}{mc_H(G, \omega)} \geq \inf_{\omega \in \mathcal{O}(G)} \frac{mc_M(G, \omega)}{mc_N(G, \omega)} = s(M, N).$$

The second part follows similarly.

We will see several applications of this lemma in Section 5.3 and Section 5.4, as well as in the following results. First, we give a bound on $s(M, N)$, for a fixed graph $M$.

**Proposition 5.13** Let $M \in \mathcal{G}$ be a fixed graph. Then, for any $N \in \mathcal{G}$,

$$1 \geq s(M, N) \geq \sum_{\{u,v\} \in E(M)} \frac{\deg(u) \deg(v)}{2e(M)^2}.$$

**Proof:** The upper bound is an immediate consequence of Corollary 5.10. For the lower bound, let $n = \chi(N)$ be the chromatic number of $N$, and note that $N \rightarrow K_n$. Therefore, $s(M, N) \geq s(M, K_n) = mc_M(K_n, 1/e(K_n))$ by Lemma 5.12 and the second part of Lemma 5.11, respectively. The result follows from Lemma 2.6.

For a $d$-regular graph $M$, we have

$$s(M, N) \geq \frac{d}{n(M)},$$

and in particular,

$$s(K_m, N) \geq \frac{m-1}{m}.$$

The homomorphism relation $\rightarrow$ defines a quasi-order, but not a partial order on the set $\mathcal{G}$. The failing axiom is that of antisymmetry, since $G \equiv H$ does not necessarily imply $G = H$. To remedy this, let $\mathcal{G}_\equiv$ denote the set of equivalence classes of $\mathcal{G}$ under $\equiv$. The relation $\rightarrow$ is defined on $\mathcal{G}_\equiv$ in the natural way and $(\mathcal{G}_\equiv, \rightarrow)$ is a lattice denoted by $\mathcal{C}_\equiv$. Let $M, N \in \mathcal{G}$ be
5. Approximation Distance

two representatives of elements in the lattice $C_S$. The greatest lower bound (meet) can be found by taking the categorical product, $M \times N$, which has vertex set $V(M) \times V(N)$ and an edge between $(u_1, v_1)$ and $(u_2, v_2)$ if and only if $\{u_1, u_2\} \in E(M)$ and $\{v_1, v_2\} \in E(N)$. The least upper bound (join) is found by taking the disjoint union $M \cup N$. This situation is depicted in Figure 5.1. For a more in-depth treatment of this lattice in particular, and graph homomorphisms in general, see [70].

![Figure 5.1: Meet $(M \times N)$ and join $(M \cup N)$ in $C_S$.](image)

**Lemma 5.14** For graphs $G, G_i, H, H_i \in \mathcal{G}$, it holds that:

- $s(G, H_1 \cup H_2) = \min\{s(G, H_1), s(G, H_2)\}$;
- $s(G, H_1 \times H_2) \geq \max\{s(G, H_1), s(G, H_2)\}$;
- $s(G_1 \cup G_2, H) \geq \max\{s(G_1, H), s(G_2, H)\}$; and
- $s(G_1 \times G_2, H) \leq \min\{s(G_1, H), s(G_2, H)\}$.

**Proof:** Using Lemma 5.11 we get:

\[
s(G, H_1 \cup H_2) = \inf_{\omega \in W(H_1 \cup H_2)} (mc_G(H_1, \omega|_{H_1}) + mc_G(H_2, \omega|_{H_2})) = \\
= \inf_{0 \leq \alpha \leq 1} \inf_{\omega_1 \in W(H_1)} \inf_{\omega_2 \in W(H_2)} (mc_G(H_1, \alpha \cdot \omega_1) + mc_G(H_2, (1 - \alpha) \cdot \omega_2)) = \\
= \min \left\{ \inf_{\omega_1 \in W(H_1)} mc_G(H_1, \omega_1), \inf_{\omega_2 \in W(H_2)} mc_G(H_2, \omega_2) \right\} = \\
= \min \{s(G, H_1), s(G, H_2)\},
\]

which proves the first part. The inequalities are straightforward applications of Lemma 5.12. It is easy to see that the first and second inequalities can be strict: let $H_1$ and $H_2$ ($G_1$ and $G_2$) be incomparable graphs, and choose $G = H_1 \times H_2$ ($H = G_1 \cup G_2$.)
Corollary 5.15 For \( M, N \in G \), \( s(M \times N, N) \leq s(M, N) = s(M, M \cup N) \).

Proof: We have \( s(M \times N) \leq \min \{ s(M, N), s(N, N) \} = s(M, N) \), and \( s(M, M \cup N) = \min \{ s(M, M), s(M, N) \} = s(M, N) \). \qed

5.2.3 The space \((G_\equiv, d)\)

The parameter \( s(M, N) \) enjoys many properties which make it act as some kind of distance between the graphs \( M \) and \( N \). Lemma 5.12 stands out as a particularly clear example. This interpretation is also present in our main application of the parameter, namely Lemma 5.3. As was noted in the introduction, this result essentially says that graphs which are “close” in \( s \), in the sense of having an \( s \)-value close to 1, also have closely related approximation ratios. In the end of the previous section, we defined \( G_\equiv \) to be the set of undirected graphs modulo homomorphic equivalence. Here we endow \( G_\equiv \) with a metric \( d \) defined in the following way: for \( M, N \in G \), let

\[
d(M, N) = 1 - s(M, N) \cdot s(N, M)
\]

Corollary 5.7 shows that \( d \) is well-defined as a function on the set \( G_\equiv \).

We now show that \( d \) is indeed a metric on this space.

Lemma 5.16 The pair \((G_\equiv, d)\) forms a metric space.

Proof: Positivity and symmetry follows immediately from the definition and the fact that \( 0 \leq s(M, N) \leq 1 \) for all \( M \) and \( N \). Since \( s(M, N) = 1 \) if and only if \( N \rightarrow M \), it also holds that \( d(M, N) = 0 \) if and only if \( M \) and \( N \) are homomorphically equivalent. That is, \( d(M, N) = 0 \) if and only if \( M \) and \( N \) represent the same member of \( G_\equiv \). Furthermore, for graphs \( M, N \) and \( K \in G \):

\[
s(M, N) \cdot s(N, K) = \inf_{G \in G} \frac{mcM(G, \omega)}{mcN(G, \omega)} \cdot \inf_{G \in G} \frac{mcN(G, \omega)}{mcK(G, \omega)} \leq \inf_{G \in G} \frac{mcM(G, \omega)}{mcN(G, \omega)} \cdot \frac{mcN(G, \omega)}{mcK(G, \omega)} = s(M, K).
\]

Therefore, with \( a = s(M, N) \cdot s(N, M), b = s(N, K) \cdot s(K, N) \) and \( c = s(M, K) \cdot s(K, M) \geq a \cdot b \),

\[
d(M, N) + d(N, K) - d(M, K) = 1 - a + 1 - b - (1 - c) \geq 1 - a - b + a \cdot b = (1 - a) \cdot (1 - b) \geq 0,
\]

which shows that \( d \) satisfies a triangle inequality. \qed

Finally we show that proximity of graphs \( G \) and \( H \) in \( d \) allows us to determine bounds on the approximability of \( \text{MAX} \ H-COL \) from known bounds on the approximability of \( \text{MAX} \ G-COL \). This is a (slightly) more general version of Lemma 5.3.
5. Approximation Distance

**Lemma 5.17** Let $M$ and $N$ be undirected graphs. If $mc_M$ can be approximated within $\alpha$, then $mc_N$ can be approximated within $\alpha \cdot (1 - d(M, N))$. If it is NP-hard to approximate $mc_N$ within $\beta$, then $mc_M$ is not approximable within $\beta/(1 - d(M, N))$, unless $P = NP$.

**Proof:** Let $f = A(G, \omega)$ be the solution returned by an algorithm which approximates $mc_M$ within $\alpha$. The algorithm $A'$ for approximating $N$ begins by obtaining $f$ in polynomial time from the algorithm $A$. It then proceeds to solve MAX-N-COL for the instance $(M, \omega_f)$ (cf. (5.5)) to produce a solution $f' : V(G) \to V(N)$. This takes constant time since $M$ and $N$ are fixed. For the first part of the lemma, it remains to show that $f'$ is a sufficiently good approximation of $mc_N(G, \omega)$. The measure of $f'$ can be bounded as follows:

$$m_N(f') = m_M(f) \cdot mc_N(M, \omega_f) \geq m_M(f) \inf_{\omega \in W(M)} mc_N(M, \omega) = m_M(f) \cdot s(N, M),$$

where the final equality follows from Corollary 5.10. From (5.4) we have the inequality $mc_M(G, \omega) \geq s(M, N) \cdot mc_N(G, \omega)$ for all $G \in \mathcal{G}$ and $\omega \in \mathcal{W}(G)$. Consequently,

$$mc_N(G, \omega) \geq m_N(f') \geq m_M(f) \cdot s(N, M) \geq mc_M(G, \omega) \cdot \alpha \cdot s(N, M) \geq mc_N(G, \omega) \cdot \alpha \cdot s(N, M) \cdot s(M, N) = mc_N(G, \omega) \cdot \alpha \cdot (1 - d(M, N)).$$

The second part follows immediately from the first. \hfill \blacksquare

### 5.2.4 A linear program formulation

We start out with a linear program for $s$ based on Corollary 5.10. Later we will see how to reduce the size of this program, but it serves as a good first exercise, and it will also be used for comparison with the linear program studied in Section 5.5.

Each vertex map $f : V(N) \to V(M)$ induces an edge map $f^\#$, which provides a lower bound on $s$ of the following form:

$$\sum_{e \in (f^\#)^{-1}(E(M))} \omega(e) \leq s(M, N). \quad (5.7)$$

By Corollary 5.10, we want to find the least $s$ such that for some weight function $\omega \in \mathcal{W}(N)$, $\|\omega\|_1 = 1$, the inequalities (5.7) hold. Let the variables of the linear program be $\{w_e\}_{e \in E(N)}$ and $s$. We then have the following linear program for $s(M, N)$:

Minimise $s$

subject to

$$\sum_{e \in (f^\#)^{-1}(E(M))} w_e \leq s, \quad \text{for } f \in V(M)^{V(N)},$$

$$\sum_{e \in E(M)} w_e = 1,$$

where $w, s \geq 0.$
5.2. Properties of the parameter $s$

Given an optimal solution $\{w_e\}_{e \in E(N)}$, $s$ to (5.8), a weight function which
minimises $m_r M(N, \omega)$ is given by $\omega(e) = w_e$ for $e \in E(N)$. The measure
of this solution is $s = s(M, N)$. The program will clearly be very large with
$|E(N)| + 1$ variables and $|V(M)| |V(N)|$ inequalities. Fortunately it can be
improved upon.

From Lemma 5.11 it follows that in order to determine $s(M, N)$, it is
sufficient to minimise $m_r M(N, w)$ over $\mathcal{V}(N)$. We can use this to describe
a smaller linear program for computing $s(M, N)$. For $i \in \{1, \ldots, r\}$, let $A_i$
be the orbits of $\text{Aut}^*(N)$. The measure of a solution $f$ when $\omega \in \mathcal{V}(N)$
is equal to $\sum_{i=1}^{r} f_i \cdot \omega_i$, where $\omega_i$ is the weight of an edge in $A_i$ and $f_i$
is the number of edges in $A_i$ which are mapped to an edge in $M$ by $f$. Note
that given an $\omega$, the measure of a solution $f$ depends only on the vector
$(f_1, \ldots, f_r) \in \mathbb{N}^r$. Therefore, take the solution space to be the set of such
vectors:

$$F = \{ (f_1, \ldots, f_r) | f \text{ is a solution to } (N, \omega) \text{ of MAX } M\text{-COL} \}.$$

Let the variables of the linear program be $w_1, \ldots, w_r$ and $s$, where $w_i$
represents the weight of each element in the orbit $A_i$ and $s$ is an upper bound
on the solutions.

\[
\begin{align*}
\text{Minimise} & \quad s \\
\text{subject to} & \quad \sum_{i=1}^{r} f_i \cdot w_r \leq s \quad \text{for } (f_1, \ldots, f_r) \in F; \\
& \quad \sum_{i=1}^{r} |A_i| \cdot w_i = 1, \\
& \quad w_i, s \geq 0
\end{align*}
\]

(5.9)

Given an optimal solution $w_i, s$ to this program, a weight function which
minimises $m_r M(N, \omega)$ is given by $\omega(e) = w_i$ for $e \in A_i$. The measure of this
solution is $s = s(M, N)$.

Example 5.18 The wheel graph on $k$ vertices, $W_k$, is a graph that
contains a cycle of length $k - 1$ plus a vertex $v$, which is not in the cycle, such
that $v$ is connected to every other vertex. We call the edges of the $k$-1-cycle
outer edges and the remaining $k - 1$ edges spokes. It is easy to see that the
diameter of $W_k$ is equal to 4 when $k = 4$ (in fact, $W_4 \cong K_4$) and
that it is equal to 3 in all other cases. Furthermore, $W_k$ is 3-colourable if
and only if $k$ is odd, and 4-colourable otherwise. This implies that for odd
$k$, the wheel graphs are homomorphically equivalent to $K_3$.

We will determine $s(K_3, W_n)$ for even $n \geq 6$ using the previously
described construction of a linear program. The graph $W_n$ is shown in Figure
5.2. Note that the group action of $\text{Aut}^*(W_n)$ on $E(W_n)$ has two orbits,
one which consists of all outer edges (solid in the figure) and one which
consists of all the spokes (dashed). If we remove one outer edge or one spoke
from $W_k$, then the resulting graph can be mapped homomorphically onto
$K_3$. Therefore, it suffices to choose $F = \{ f, g \}$ with $f = (k - 1, k - 2)$ and
5. Approximation Distance

\[ g = (k-2, k-1) \] since all other solutions will have a smaller measure than at least one of these. The program for \( W_k \) looks as follows:

\[
\begin{align*}
&\text{Minimise } s \\
&\text{subject to } (k-1) \cdot w_1 + (k-2) \cdot w_2 \leq s \\
&\quad (k-2) \cdot w_1 + (k-1) \cdot w_2 \leq s \\
&\quad (k-1) \cdot w_1 + (k-1) \cdot w_2 = 1 \\
&\text{where } w_1, w_2, s \geq 0
\end{align*}
\]

The optimal solution to this program is given by \( w_1 = w_2 = 1/(2k-2) \), with \( s(K_3, W_k) = s = (2k-3)/(2k-2) \).

**Example 5.19** In the previous example, the two weights in the optimal solution were equal. Here, we provide another example, where the weights turn out to be different for different orbits. The circular complete graph \( K_{8/3} \) has vertex set \( \{v_0, v_1, \ldots, v_7\} \), which is placed uniformly around a circle. There is an edge between any two vertices which are at a distance at least 3 from each other. Figure 5.3 depicts this graph.

We will now calculate \( s(K_2, K_{8/3}) \). Each vertex is at a distance 4 from exactly one other vertex, which means that there are 4 such edges. These edges, which are dashed in the figure, form one orbit under the action of \( \text{Aut}^+(K_{8/3}) \) on \( E(K_{8/3}) \). The remaining 8 edges (solid) form a second orbit. Let \( V(K_2) = \{u_0, u_1\} \). We can obtain a solution \( f \) by mapping \( f(v_i) = u_0 \) if \( i \) is even, and \( f(v_i) = u_1 \) if \( i \) is odd. This solution will map all solid edges to \( K_2 \), but none of the dashed, hence \( f = (0, 8) \). We obtain a second solution \( g \) by mapping \( g(v_i) = u_0 \) for \( 0 \leq i < 4 \), and \( g(v_i) = u_1 \) for \( 4 \leq i < 8 \). This solution will map all but two of the solid edges in \( K_{8/3} \) to \( K_2 \), hence \( g = (4, 6) \). The inequalities given by \( f \) and \( g \) imply the inequalities given by
any other solution, so we have the following program for \( s(K_2, K_{8/3}) \):

\[
\begin{align*}
\text{Minimise} & \quad s \\
\text{subject to} & \quad 0 \cdot w_1 + 8 \cdot w_2 \leq s \\
& \quad 4 \cdot w_1 + 6 \cdot w_2 \leq s \\
& \quad 4 \cdot w_1 + 8 \cdot w_2 = 1 \\
\text{where} & \quad w_1, w_2, s \geq 0
\end{align*}
\]

The optimal solution to this program is given by \( w_1 = 1/20, w_2 = 1/10 \), and \( s(K_2, K_{8/3}) = s = 4/5 \).

### 5.3 Circular complete graphs

The successful application of our method relies on the ability to compute \( s(M, N) \) for various graphs \( M \) and \( N \). In the previous section we saw how this can be accomplished by the means of linear programming. This insight is put to use in this section in the context of **circular complete graphs**. We have already come across examples of such graphs in the form of cycles, (ordinary) complete graphs, and the graph in Figure 5.3. We will now take a closer look at them.

**Definition 5.20** Let \( p \) and \( q \) be positive integers such that \( p \geq 2q \). The **circular complete graph**, \( K_{p/q} \) has vertex set \( \{v_0, v_1, \ldots, v_{n-1}\} \) and edge set \( \{\{v_i, v_j\} \mid q \leq |i-j| \leq p-q\} \).

The image to keep in mind is that of the vertices placed uniformly around a circle with an edge connecting two vertices if they are at a distance at least \( q \) from each other.
5. Approximation Distance

Example 5.21 Some well-known graphs are extreme cases of circular complete graphs:

- The complete graph \( K_n, n \geq 2 \) is a circular complete graph with \( p = n \) and \( q = 1 \).
- The cycle graph \( C_{2k+1}, k \geq 1 \) is a circular complete graph with \( p = 2k + 1 \) and \( q = k \).

These are the only examples of edge-transitive circular complete graphs.

A fundamental property of the circular complete graphs is given by the following theorem:

Theorem 5.22 (see Hell and Nešetřil [70]) For positive integers \( p, q, p', q' \),

\[
K_{p/q} \to K_{p'/q'} \iff p/q \leq p'/q'
\]

Due to this theorem, we may assume that whenever we write \( K_{p/q} \), the positive integers \( p \) and \( q \) are relatively prime.

The main reason for introducing circular complete graphs is that they refine the notion of complete graphs. In particular, that they refine the notion of the chromatic number \( \chi(G) \). Note that an alternative definition of \( \chi(G) \) is given by \( \chi(G) = \inf \{ n \mid G \to K_n \} \). With this in mind, the following is a natural extension of proper graph colouring, and the chromatic number:

Definition 5.23 The circular chromatic number, \( \chi_c(G) \), of a graph \( G \) is defined as \( \inf \{ p/q \mid G \to K_{p/q} \} \). A homomorphism from \( G \) to \( K_{p/q} \) is called a (circular) \( p/q \)-colouring of \( G \).

For more on the circular complete graphs and the circular complete number, see the book by Hell and Nešetřil [70], and the survey by Zhu [134].

Complete graphs

Let \( \omega(G) \) denote the clique number of \( G \). The Turán graph \( T(n, r) \) is a graph formed by partitioning a set of \( n \) vertices into \( r \) subsets, with sizes as equal as possible, and connecting two vertices whenever they belong to different subsets. Turán graphs have the following properties [129]:

- \( e(T(n, r)) = \left( 1 - \frac{1}{r} \right) \cdot \frac{n^2}{2} \);
- \( \omega(T(n, r)) = \chi(T(n, r)) = r \);
- if \( G \) is a graph such that \( e(G) > e(T(\omega(G), r)) \), then \( \omega(G) > r \).
Proposition 5.24 Let $r$ and $n$ be positive integers and let $H$ be a graph such that $\omega(H) = r$ and $\chi(H) \leq n$. Then,

$$s(H, K_n) = e(T(n, r))/e(K_n) = \left(1 - \frac{1}{r}\right) \cdot \binom{n^2}{2} / \binom{n}{2}.$$ 

In particular, this holds with $H = K_r$.

Proof: The graph $K_n$ is edge-transitive. Therefore, by the second part of Lemma 5.11, it suffices to show that $mc_H(K_n, 1/e(K_n)) = e(T(n, r))/e(K_n)$. By definition, $T(n, r)$ is an $r$-partite subgraph of $K_n$, so $T(n, r) \rightarrow H$. Hence, $mc_H(K_n, 1/e(K_n)) \geq e(T(n, r))/e(K_n)$. Conversely, any subgraph $G$ of $K_n$ such that $G \rightarrow H$ must satisfy $\omega(G) \leq \omega(H) = r$. Thus, by the third property of Turán graphs, $e(G) \leq e(T(n, r))$ which implies $mc_H(K_n, 1/e(K_n)) \leq e(T(n, r))/e(K_n)$. $\blacksquare$

Cycle graphs

The even cycles are all bipartite and therefore homomorphically equivalent to $K_2$. The odd cycles, on the other hand, form a chain in the lattice $C_S$ between $K_2$ and $C_3 = K_3$ in the following manner:

$$K_2 \rightarrow \cdots \rightarrow C_{2i+1} \rightarrow C_{2i-1} \rightarrow \cdots \rightarrow C_3 = K_3.$$ 

Note that the chain is infinite on the $K_2$-side. The following lemma gives the value of $s(M, N)$ for pairs of graphs in this chain. The value depends only on the target graph.

Proposition 5.25 Let $m > k$ be positive integers. Then,

$$s(K_2, C_{2k+1}) = s(C_{2m+1}, C_{2k+1}) = \frac{2k}{2k+1}.$$ 

Proof: Note that $C_{2k+1} \not\rightarrow K_2$ and $C_{2k+1} \not\rightarrow C_{2m+1}$. However, after removing one edge from $C_{2k+1}$ the remaining subgraph is isomorphic to the path $P_{2k+1}$, which in turn is isomorphic to both $K_2$ and $C_{2m+1}$. Hence $mc_2(C_{2k+1}, 1/(2k+1)) = mc_{2m+1}(C_{2k+1}, 1/(2k+1)) = 2k/(2k+1)$. Since $C_{2k+1}$ is edge-transitive, the result follows from Lemma 5.11. $\blacksquare$

We now proceed to investigate the general case $s(K_r, K_t)$ for rational numbers $2 \leq r < t \leq 3$. In Section 5.3.1, we fix $r = 2$ and choose $t$ so that $\text{Aut}^+(K_t)$ has few orbits. We will find some interesting properties of these numbers which lead us to look at certain “constant regions”, in Section 5.3.2, and at the case $r = 2+1/k$ in Section 5.3.3. Our method is based on solving a relaxation of the linear program (5.9) that was presented in the last section, combined with arguments that the chosen relaxation in fact finds the optimum in the original program. Most of the calculations, which involve some rather lengthy ad hoc constructions of solutions, are left out. The complete proofs are found in the technical report [44].
5. Approximation Distance

5.3.1 Maps to $K_2$

We consider $s(K_2, K_t)$ for $t = 2 + n/k$ with $k > n \geq 1$, where $n$ and $k$ are integers. The number of orbits of $\text{Aut}^+(K_t)$ then equals $\lceil (n + 1)/2 \rceil$. We will denote these orbits by

$$A_c = \{ \{v, v_j\} \in E(K_p/q) \mid j - i \equiv q + c - 1 \pmod{p} \},$$

for $c = 1, \ldots, \lceil (n + 1)/2 \rceil$. Since the number of orbits determine the number of variables of the linear program (5.9), we choose to begin our study of $s(K_2, K_t)$ using small values of $n$. For $n = 1$ we have seen that the graph $K_{2 + 1}/k$ is isomorphic to the cycle $C_{2k+1}$. For $n = 2$ we can assume that $k$ is odd in order to have $2k + n$ and $k$ relatively prime. We will write this number as $t = 2 + 2/(2k - 1) = 4k - 2 + 2k$. Note that the graph $K_{8/3}$ from Example 5.19 is covered by Proposition 5.26, for $k = 2$, and that the proof is very similar in the general case.

**Proposition 5.26** Let $k \geq 1$ be an integer. Then, $s(K_2, K_{\frac{4k}{2k+1}}) = \frac{2k}{2k+1}$.

**Proof:** Let $V(K_{\frac{4k}{2k+1}}) = \{v_0, v_1, \ldots, v_{4k-1}\}$ and $V(K_2) = \{w_0, w_1\}$. We start by presenting two maps $f, g : V(K_{\frac{4k}{2k+1}}) \to V(K_2)$. The map $f$ sends $v_i$ to $w_0$ if $i$ is even and to $w_1$ if $i$ is odd. Then, $f$ maps all of $A_1$ to $K_2$ but none of the edges in $A_2$, so $f = (4k, 0)$. The solution $g$ sends a vertex $v_i$ to $w_0$ if $0 \leq i < 2k$ and to $w_1$ if $2k \leq i < 4k$. It is not hard to see that $g = (4k - 2, 2k)$. It remains to argue that these two solutions suffice to determine $s$. But we see that any map $h = (h_1, h_2)$ with $h_2 > h_1$ must cut at least two edges in the even cycle $A_1$. Therefore, $h_1 \leq 4k - 2$, so $h \leq g$, componentwise. The proposition now follows by solving the relaxation of (5.9) using only the two inequalities obtained from $f$ and $g$. \hfill \blacksquare

For $n = 3$, $t = 2 + 3/k$, we see that if $k \equiv 0 \pmod{3}$, then $K_t$ is an odd cycle. If $k \equiv 2 \pmod{3}$, then $2 + 1/k \leq t \leq 2 + 2/(2k - 1)$ and by Theorem 5.22, Lemma 5.12, and known values on $s$, we have

$$\frac{2k}{2k+1} = s(K_2, C_{2k+1}) \geq s(K_2, K_t) \geq s(K_2, K_{\frac{4k}{2k+1}}) = \frac{2k}{2k+1}$$

It follows that $s(K_2, K_t)$ must be $2k/(2k + 1)$ as well. Therefore we assume that $t$ is of the form $2 + 1/(3k + 1) = \frac{4k+5}{4k+1}$ for an integer $k \geq 1$.

**Proposition 5.27** Let $k \geq 1$ be an integer. Then,

$$s(K_2, K_{\frac{4k+5}{4k+1}}) = \frac{6k^2 + 8k + 3}{6k^2 + 11k + 5} = 1 - \frac{3k + 2}{(k + 1)(6k + 5)}$$

For $t = 2 + 4/k$, we find that we only need to consider the case when $k \equiv 1 \pmod{4}$. We then have graphs $K_t$ with $t = 2 + 4/(4k + 1) = \frac{8k+16}{4k+1}$ for integers $k \geq 1$. 


Proposition 5.28 Let $k \geq 1$ be an integer. Then,

$$s(K_2, K_{\frac{32k+6}{4k+1}}) = \frac{8k^2 + 6k + 2}{8k^2 + 10k + 3} = 1 - \frac{4k + 1}{(k + 1/2)(8k + 6)}.$$  

The expressions for $s$ in Propositions 5.27 and 5.28 have some interesting similarities, but for $n \geq 5$ it becomes much harder to choose a suitable set of solutions. Using brute force computer calculations, we have determined the first two values (for $k = 1, 2$) in each of the cases $t = 2 + 5/(5k + 1)$ ($t = 17/6, 27/11$) and $t = 2 + 6/(6k + 1)$ ($t = 20/7, 32/13$). These values are summarised in the following table:

<table>
<thead>
<tr>
<th>$s(K_2, K_t)$</th>
<th>$t = 2 + 5/(5k + 1)$</th>
<th>$t = 2 + 6/(6k + 1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k = 1$</td>
<td>322/425 ≈ 0.7576</td>
<td>67/89 ≈ 0.7528</td>
</tr>
<tr>
<td>$k = 2$</td>
<td>5/6 ≈ 0.8333</td>
<td>94/113 ≈ 0.8319</td>
</tr>
</tbody>
</table>

5.3.2 Constant regions

In the previous section we saw that $s(K_2, C_{2k+1}) = s(K_2, K_{\frac{4k}{2k-1}})$ and used it to prove that $s(K_2, K_t)$ is constant in the interval $t \in [\frac{2 + 1/k}{2k-1}, \frac{4k}{2k-1}]$. This is a special case of a phenomenon described more generally in the following proposition:

Proposition 5.29 Let $k \geq 1$, and let $r$ and $t$ be rational numbers such that $2 \leq r < \frac{2k+1}{k} \leq t < \frac{4k}{2k-1}$. Then,

$$s(K_r, K_t) = \frac{2k}{2k+1}.$$  

Proof: From Theorem 5.22, we have the following chain of homomorphisms:

$$K_2 \to K_r \to K_{\frac{2k+1}{k}} \to K_t \to K_{\frac{4k}{2k-1}}.$$  

By Lemma 5.12, this implies

$$s(K_r, K_{\frac{2k+1}{k}}) \leq s(K_2, K_{\frac{2k+1}{k}}) = \frac{2k}{2k+1},$$  

but since $K_{\frac{2k+1}{k}} \not\cong K_r$ and $K_{\frac{2k+1}{k}}$ is edge-transitive with $2k + 1$ edges, $s(K_r, K_{\frac{2k+1}{k}}) \leq \frac{2k}{2k+1}$ and therefore $s(K_r, K_{\frac{2k+1}{k}}) = \frac{2k}{2k+1}$. Two more applications of Lemma 5.12 show that

$$\frac{2k}{2k+1} = s(K_r, K_{\frac{2k+1}{k}}) \geq s(K_r, K_t) \geq s(K_2, K_{\frac{4k}{2k-1}}) = \frac{2k}{2k+1},$$  

which proves the proposition.

We find that there are intervals $I_k = \{ t \in \mathbb{Q} \mid 2 + 1/k \leq t \leq 2 + 2/(2k - 1) \}$ where the function $s_r(t) = s(K_r, K_t)$ is constant for any $2 \leq r < (2k + 1)/k$. 


In Figure 5.4 these intervals are shown for the first few values of $k$. The intervals $I_k$ form an infinite sequence with endpoints tending to 2.

It turns out that similar intervals appear throughout the space of circular complete graphs. Indeed, it follows from Proposition 5.24 that for a positive integer $n$ and a rational number $r$ such that $2 \leq r \leq n$, we have

$$s(K_r, K_n) = \left(1 - \frac{1}{r^2}\right) \cdot \frac{n^2}{2} \left/ \left(\binom{n}{2}\right)\right..$$

From (5.10) we see that $s(K_r, K_n)$ remains constant for rational numbers $r$ in the interval $k \leq r < k + 1$, where $k$ is any fixed integer $k < n$. Furthermore, for positive integers $k$ and $m$, we have

$$e(T(km - 1, k)) = \left(1 - \frac{1}{k}\right) \cdot \frac{(km - 1)^2}{2} =$$

$$= \left(\frac{(k-1)km^2}{2} - (k-1)m + \frac{k-1}{2k}\right) = \left(\frac{(k-1)km^2}{2} - (k-1)m\right) =$$

$$= \left(\frac{k}{2}\right) m^2 \cdot \left(1 - \frac{2}{km}\right) = e(T(km, k)) \cdot \frac{km - 1}{2} \left/ \left(\binom{km}{2}\right)\right..$$

Thus, $s(K_k, K_{km-1}) = s(K_k, K_{km})$. When we combine this fact with (5.10) and Lemma 5.12, we find that $s(K_r, K_t)$ is constant on each region $(r, t) \in [k, k+1] \times [km-1, km]$.

5.3.3 Maps to odd cycles

It was seen in Proposition 5.29 that $s(K_r, K_t)$ is constant on the region $(r, t) \in [2, 2 + 1/k] \times I_k$. In this section, we will study what happens when $t$ remains in $I_k$, but $r$ assumes the value $2 + 1/k$. A first observation is that the absolute jump of the function $s(K_r, K_t)$ when $r$ goes from being less than $2 + 1/k$ to $r = 2 + 1/k$ must be largest for $t = 2 + 2/(2k-1)$. Let $V(K_{2+2/(2k-1)}) = \{v_0, \ldots, v_{4k-1}\}$ and $V(K_{2+1/k}) = \{w_0, \ldots, w_{2k}\}$, and let the function $f$ map $v_i$ to $w_j$, with $j = \left\lfloor \frac{2k+1}{i}\right\rfloor$. Then, $f$ maps all edges except $\{v_0, v_{2k-1}\}$ from the orbit $A_1$ to some edge in $K_r$. Since the subgraph $A_1$ is isomorphic to $C_{4k}$, any map to an odd cycle must exclude at least one edge from $A_1$. It follows that $f$ alone determines $s$, and we can solve the
linear program (5.9) to obtain \( s(K_{2+1/k}, K_{2+2/(2k-1)}) = (4k-1)/4k \). Thus, for \( r < 2 + 1/k \), we have

\[
s(K_{2+1/k}, K_{2+2/(2k-1)}) - s(K_r, K_{2+2/(2k-1)}) = \frac{2k-1}{4k(2k+1)}.
\]

Smaller \( t \in I_k \) can be expressed as \( t = 2 + 1/(k - x) \), where \( 0 \leq x < 1/2 \). We will write \( x = m/n \) for positive integers \( m \) and \( n \) which implies the form \( t = 2 + n/(kn - m) \), with \( m < n/2 \). For \( m = 1 \), it turns out to be sufficient to keep two inequalities from (5.9) to get an optimal value of \( s \). From this we get the following result:

**Proposition 5.30** Let \( k, n \geq 2 \) be integers. Then,

\[
s(C_{2k+1}, K_{2(kn-1)+n}) = \frac{(2(kn-1) + n)(4k-1)}{(2(kn-1) + n)(4k-1) + 4k - 2}.
\]

There is still a non-zero jump of \( s(K_r, K_t) \) when we move from \( K_r < 2 + 1/k \) to \( K_r = 2 + 1/k \), but it is smaller, and tends to 0 as \( n \) increases. For \( m = 2 \), we have \( 2(kn-m)+n \) and \( kn-m \) relatively prime only when \( n \) is odd. In this case, it turns out that we need to include an increasing number of inequalities to obtain a good relaxation. Furthermore, we are not able to ensure that the obtained value is the optimum of the original (5.9). We will therefore have to settle for a lower bound on \( s \). Brute force calculations have shown that, for small values of \( k \) and \( n \), equality holds in Proposition 5.31. We conjecture this to be true in general.

**Proposition 5.31** Let \( k \geq 2 \) be an integer and \( n \geq 3 \) be an odd integer. Then,

\[
s(C_{2k+1}, K_{2(kn-2)+n}) \geq \frac{(2(kn-2) + n)(\xi_n(4k-1) + (2k-1))}{(2(kn-2) + n)(\xi_n(4k-1) + (2k-1)) + (4k-2)(1 - \xi_n)}
\]

where \( \xi_n = \left( \frac{z_1^{-1}}{z_1^{-1} + z_2^{-1}} \right) / 4 \), and \( z_1^{-1}, z_2^{-1} \) are the roots of \( \frac{2k-3}{4k-2}z^2 - 2z + 1 \).

### 5.4 Approximation bounds for MAX H-Col

In this section we apply our method, consisting of Lemma 5.3 and some of the values determined for \( s \) in Section 5.3, to determine bounds on the approximation ratio of MAX H-Col for various families of graphs.

We address two main points for each family of graphs under consideration. First, we compare the performance of our method with that of some existing, leading, approximation algorithm. Secondly, we investigate near-optimality of our method, i.e. the difference in our obtained upper and lower bounds on approximability. Recall that our method is based on the idea to
5. Approximation Distance

measure how well an algorithm for one problem MAX M-Col performs on another problem MAX N-Col. Our main algorithmic tools will be the following two theorems:

**Theorem 5.32 (Goemans and Williamson [58])** MAX cut can be approximated within

\[ \alpha_{GW} = \min_{0 < \theta < \pi} \frac{\theta / \pi}{(1 - \cos \theta) / 2} \approx 0.87856. \]

**Theorem 5.33 (Frieze and Jerrum [55])** MAX k-cut can be approximated within \( \alpha_k \), where

\[ \alpha_k = \left( 1 - \frac{1}{k} \right) \sim \frac{2 \ln k}{k^2}. \]

Here, the relation \( \sim \) indicates two expressions whose ratio tends to 1 as \( k \to \infty \). We will abuse this notation in the following, commonly occurring way; for two expressions \( f \) and \( g \), we write \( f \sim g \) to indicate that when we move all but the fastest vanishing term of \( g \) to the left-hand side, the ratio of these two quantities tends to 1. As an example, we may write the expression in Theorem 5.33 as \( \alpha_k \sim 1 - \frac{1}{k} + \frac{2 \ln k}{k^2} \). This notation facilitates the use of \( \sim \) when calculations span several lines. We note that de Klerk et al. [38] have presented the sharpest known bounds on \( \alpha_k \) for small values of \( k \); for instance, \( \alpha_3 \geq 0.836008 \).

Håstad [75] has shown the following:

**Theorem 5.34 (Håstad [75])** There is an absolute constant \( c > 0 \) such that \( mc_H \) can be approximated within

\[ 1 - \frac{t(H)}{d^2} \cdot \left( 1 - \frac{c}{d^2 \log d} \right), \]

where \( d = \nu(H) \) and \( t(H) = d^2 - 2 \cdot e(H) \).

We will compare the performance of this algorithm on MAX H-Col with the performance of the algorithms in Theorems 5.32 and 5.33 when analysed using Lemma 5.3 and estimates of the parameter \( s \). For this purpose, we introduce two functions, \( FJ_k \) and \( H \), such that, if \( H \) is a graph, then \( FJ_k(H) \) denotes the best bound on the approximation guarantee when Frieze and Jerrum’s algorithm for MAX k-cut is applied to the problem \( mc_H \), while \( H \) is the guarantee when Håstad’s algorithm is used to approximate \( mc_H \). This comparison is not entirely fair since Håstad’s algorithm was not designed with the goal of providing optimal results—the goal was to beat random assignments. However, it is currently the best known algorithm for approximating MAX H-Col, for arbitrary \( H \in \mathcal{G} \), which also provides an easily computable bound on the guaranteed approximation ratio. This is in contrast with the (conjectured) optimal algorithms of Raghavendra [116] (cf. the discussion in Section 5.7.)
Near-optimality of our approximation method will be investigated using Khot’s Unique Games Conjecture. From hereon, we therefore assume that UGC is true, which gives us the following inapproximability results:

**Theorem 5.35 (Khot et al. [95])** The following holds modulo the truth of the Unique Games Conjecture:

- For every \( \varepsilon > 0 \), it is NP-hard to approximate \( mc_2 \) within \( \alpha_{GW} + \varepsilon \).
- It is NP-hard to approximate \( mc_k \) within
  \[
  1 - \frac{1}{k} + \frac{(2 \ln k)/k^2 + \mathcal{O}((\ln \ln k)/k^2)}{k^2}.
  \]

### 5.4.1 Sparse graphs

In this section, we investigate the performance of our method on graphs which have relatively few edges, and we see that the girth of the graphs play a central role. The girth of a graph \( G \) is the length of a shortest cycle contained in \( G \). Similarly, the odd girth of \( G \) is the length of a shortest odd cycle in \( G \). Hence, if \( G \) has odd girth \( g \), then \( C_g \to G \), but \( C_{2k+1} \not\to G \) for \( 3 \leq 2k+1 < g \).

**Proposition 5.36** Let \( k \geq 3 \) be odd. Then, \( FJ_k(C_k) \geq \frac{k-1}{k} \cdot \alpha_{GW} \) and
\[
\bar{H}_a(C_k) = \frac{k}{k} + \frac{\alpha_{GW}}{k \log k} = \frac{2k - 2}{k} \cdot \left( 1 - \frac{c}{k^2 \log k} \right) = \frac{ck + 2k^2 \log k - 2c}{k^3 \log k} = \frac{2k}{k^2 \log k} - \frac{2c}{k^3 \log k}.
\]

**Proof:** Using Lemma 5.25 we see that \( s(K_2, C_k) = \frac{k-1}{k} \) which implies (using Lemma 5.3) that \( FJ_k(C_k) \geq \frac{k-1}{k} \cdot \alpha_{GW} \). Furthermore, \( mc_2 \) cannot be approximated within \( \alpha_{GW} + \varepsilon' \) for any \( \varepsilon' > 0 \). From the second part of Lemma 5.3, we get that \( mc_{C_k} \) is not approximable within \( \frac{k-1}{k} \cdot (\alpha_{GW} + \varepsilon') \) for any \( \varepsilon' \). With \( \varepsilon' = \varepsilon \cdot \frac{k-1}{k} \) the result follows.

Finally, we see that
\[
\bar{H}_a(C_k) = 1 - \frac{k^2 - 2k}{k^2} \cdot \left( 1 - \frac{c}{k^2 \log k} \right) = \frac{ck + 2k^2 \log k - 2c}{k^3 \log k} = \frac{2k}{k^2 \log k} - \frac{2c}{k^3 \log k}.
\]

Håstad’s algorithm does not perform particularly well on sparse graphs; this is reflected by its performance on cycle graphs \( C_k \) where the approximation guarantee tends to zero when \( k \to \infty \). We will see that this trend is apparent for all graph families studied in this section.

We continue with a result on a class of graphs with large girth:
Proposition 5.37 Let \( m > k \geq 4 \). If \( H \) is a graph with odd girth \( g \geq 2k + 1 \) and minimum degree \( \delta(G) \geq \frac{2m-1}{2k+1} \), then \( FJ_2(H) \geq \frac{2k}{2k+1} \cdot \alpha_{GW} \).

Asymptotically, \( H\alpha(H) \) is bounded by

\[
\frac{c}{n^2 \log n} + \frac{2(n^{\gamma/(g-1)+1})^3}{n^4 n^{1/(g-1)}} \geq \frac{2n^{\gamma/(g-1)+1}}{n^4 \log n},
\]

where \( n = n(H) \). Finally, \( mc_H \) cannot be approximated with \( \frac{2k+1}{2k} \cdot \alpha_{GW} + \varepsilon \) for any \( \varepsilon > 0 \).

**Proof:** Lai & Liu [102] have proved that if \( H \) is a graph with the properties stated in the proposition, then there exists a homomorphism from \( H \) to \( C_{2k+1} \). Thus, \( K_2 \rightarrow H \rightarrow C_{2k+1} \) which implies that \( s(K_2, H) \geq s(K_2, C_{2k+1}) = \frac{2k}{2k+1} \). By Lemma 5.3, \( FJ_2(H) \geq \frac{2k}{2k+1} \cdot \alpha_{GW} \), but \( mc_H \) cannot be approximated within \( \frac{2k+1}{2k} \cdot \alpha_{GW} + \varepsilon \) for any \( \varepsilon > 0 \).

Dutton and Brigham [42] show an upper bound on \( e(H) \) of asymptotic order \( n^{1+2/(g-1)} \). This lets us determine \( H\alpha(H) \) as follows:

\[
H\alpha(H) \approx 1 - \frac{n^2 - 2 \cdot n^{1+2/(g-1)}}{n^2} \left( 1 - \frac{c}{n^2 \log n} \right) =
\]

\[
= \frac{cn^2 + 2n^{(3g-1)/(g-1)} \log n - 2n^{(g+1)/(g-1)} c}{n^4 \log n} =
\]

\[
= \frac{c}{n^2 \log n} + \frac{2(n^{\gamma/(g-1)+1})^3}{n^4 n^{1/(g-1)}} - \frac{2n^{\gamma/(g-1)+1}}{n^4 \log n}.
\]

If we restrict the scope to planar graphs, then it is possible to prove tighter bounds:

**Proposition 5.38** Let \( H \) be a planar graph with girth at least \( g = \frac{20k-2}{3} \).

With \( n = n(H) \), \( FJ_2(H) \geq \frac{2k}{2k+1} \cdot \alpha_{GW} \) and

\[
H\alpha(H) \leq \frac{6}{n} - \frac{12}{n^2} + \frac{c}{n^2 \log n} - \frac{6c}{n^4 \log n} + \frac{12c}{n^4 \log n},
\]

where \( n = n(H) \). Finally, \( mc_H \) cannot be approximated within \( \frac{2k+1}{2k} \cdot \alpha_{GW} + \varepsilon \) for any \( \varepsilon > 0 \).

**Proof:** Borodin et al. [17] have proved that \( H \) is \((2 + \frac{1}{k})\)-colourable which is equivalent to saying that there exists a homomorphism from \( H \) to \( C_{2k+1} \). The proof of the first part, and the statement of inapproximability, then follows as in the proof of Proposition 5.37.

A planar graph \( H \) cannot have more than \( 3n - 6 \) edges so \( H\alpha(H) \) is bounded from above by

\[
1 - \frac{n^2 - 2(3n - 6)}{n^2} \left( 1 - \frac{c}{n^2 \log n} \right) =
\]
5.4. Approximation bounds for $\text{MAX } H\text{-COL}$

\[
\frac{cn^2 - 6ac + 12c + 6n^3 \log n - 12n^2 \log n}{n^4 \log n} = \\
= \frac{6}{n} - \frac{12}{n^2} + \frac{c}{n^2 \log n} - \frac{6c}{n^3 \log n} + \frac{12c}{n^4 \log n}.
\]

(In fact, $H$ contains no more than $\max \{g(n - 2)/(g - 2), n - 1\}$ edges, but using this only makes for an even more convoluted expression, and the one we have derived suffices for comparison.)

Proposition 5.38 can be strengthened and extended in several ways. For $K_4$-minor-free graphs, Pan and Zhu [109] have given odd girth-restrictions for $2k + 1$-colourability which is better than the girth-restriction in Proposition 5.38.

Dvořák et al. [43] have proved that every planar graph $H$ of odd girth at least 9 is homomorphic to the Petersen graph $P$ (Figure 1.2.) The Petersen graph is edge-transitive and its bipartite density is known to be $4/5$ (cf. [10]). In other words, $s(K_5, P) = 4/5$. Consequently, $mc_H$ can be approximated within $\frac{1}{5} \cdot \alpha_{GW}$ but not within $\frac{1}{5} \cdot \alpha_{GW} + \epsilon$ for any $\epsilon > 0$. This is an improvement on the bounds in Proposition 5.38 for planar graphs with girth strictly less than 13.

We can also consider graphs embeddable on higher-genus surfaces. For instance, Proposition 5.38 is true for graphs embeddable on the projective plane, and it is also true for graphs of girth strictly greater than $\frac{20k - 2}{k}$ whenever the graphs are embeddable on the torus or Klein bottle. These bounds are direct consequences of results in Borodin et al. [17].

We conclude the section by looking at a class of graphs with small girth. Let $0 < \beta < 1$ be the approximation threshold for $mc_3$, i.e. $mc_3$ is approximable within $\beta$ but not within $\beta + \epsilon$ for any $\epsilon > 0$. Currently, we know that $a_3 \leq 0.836008 \leq \beta \leq \frac{10}{11}$ [38, 88]. The wheel graphs $W_k$ from Example 5.18 are homomorphically equivalent to $K_3$ for odd $k$ and we conclude (by Lemma 5.3) that $mc_{W_k}$ has the same approximability properties as $mc_3$ in this case. For even $k \geq 6$, we have the following result:

**Proposition 5.39** For $k \geq 6$ and even, $FJ_3(W_k) \geq a_3 \cdot \frac{2k-2}{2k-3}$ but $mc_{W_k}$ is not approximable within $\beta \frac{2k-2}{2k-3}$. $H\bar{a}(W_k) = \frac{4}{k^2} - \frac{4}{k^3} + \frac{k^2}{k^3 \log k} - \frac{k}{k^4 \log k} + \frac{k^4}{k^4 \log k}$.

**Proof:** We know from Example 5.18 that $K_3 \rightarrow W_k$ and $s(K_3, W_k) = \frac{2k-3}{2k-2}$. The first part of the result follows by an application of Lemma 5.3:

\[
H\bar{a}(W_k) = 1 - \frac{t(W_k)}{d^2} \cdot \left(1 - \frac{c}{d^2 \log d}\right) = \frac{1}{d} = k, c(W_k) = 2k - 2 =
= 1 - \frac{k^2 - 4(k - 1)}{k^2} \cdot \left(1 - \frac{c}{k^2 \log k}\right) =
= \frac{k^2 c + 4k^3 \log k - 4kc - k^2 \log k + 4c}{k^4 \log k}.
\]
5. Approximation Distance

\[ \frac{4}{k} - \frac{4}{k^2} + \frac{c}{k^2 \log k} \leq \frac{4c}{k^2 \log k} + \frac{4c}{k^4 \log k} \]

We see that \( FJ_3(W_k) \rightarrow a_3 \) when \( k \rightarrow \infty \) while \( H_3(W_k) \) tends to 0.

5.4.2 Dense and random graphs

We will now study dense graphs, i.e. graphs \( H \) containing \( \Theta(n(H)^2) \) edges. For a graph \( H \) on \( n \) vertices, we obviously have \( H \rightarrow K_n \). If we assume that the clique number, \( \omega(H) \), is greater than or equal to \( r \), then we also have \( K_r \rightarrow H \). From Proposition 5.24 we know the value of \( s(H, K_n) \), which we can use to bound \( FJ_n(H) \):

**Proposition 5.40** Let \( n = n(H) \) and pick \( r \in \mathbb{N} \) and \( \sigma \in \mathbb{Q} \) such that

\[ \left( 1 - \frac{1}{r} \right) \cdot \frac{n^2}{2} \leq \sigma \cdot n^2 = e(H) \leq \frac{n(n-1)}{2}. \]

Then,

\[ FJ_n(H) \geq \alpha_n \cdot \frac{2 \left( 1 - \frac{1}{r} \right) \cdot \frac{n^2}{2}}{n \cdot (n-1)} \sim 1 - \frac{1}{r} - \frac{1}{n} + \frac{2 \ln n}{n(n-1)} \text{, and} \]

\[ H_3(H) = 2\sigma + \frac{c}{n^4 \log n} - \frac{2\sigma \cdot c}{n^4 \log n}. \]

**Proof:** We have \( K_r \rightarrow H \) due to Turán and \( H \rightarrow K_n \) holds trivially since \( n = n(H) \). By Proposition 5.24,

\[ s(K_r, K_n) = \left\lfloor \left( 1 - \frac{1}{r} \right) \cdot \frac{n^2}{2} \right\rfloor / \binom{n}{2}. \]

By Lemma 5.12, \( s(H, K_n) \geq s(K_r, K_n) \), so the first part of the result follows from Lemma 5.3. The second part is obtained by the following calculation:

\[ H_3(H) = 1 - \frac{n^2 - \sigma \cdot n^2}{n^2} \cdot \left( 1 - \frac{c}{n^2 \log n} \right) = \]

\[ = \frac{c + 2\sigma \cdot n^2 \log n - 2\sigma \cdot c}{n^2 \log n} = \frac{c}{n^2 \log n} + 2\sigma - \frac{2\sigma \cdot c}{n^2 \log n}. \]

Note that when \( r \) and \( n \) grow, \( FJ_n(H) \) tends to 1. This means that, asymptotically, we cannot do much better. If we compare the expression for \( FJ_n(H) \) with the inapproximability bound for \( m_{C_n} \) (Theorem 5.35), we see that all we could hope for is a faster convergence towards 1. As \( \sigma \) satisfies \( \left( 1 - \frac{1}{r} \right) \cdot \frac{1}{2} \leq \sigma \leq \left( 1 - \frac{1}{n} \right) \cdot \frac{1}{2} \), we conclude that \( H_3(H) \) also tends to
1 as \( r \) and \( n \) grow. To get a better idea of how \( H \hat{\alpha}(H) \) behaves, we look at two extreme cases.

For a maximal \( \sigma = \left( 1 - \frac{1}{n} \right) \cdot \frac{1}{2} \), \( H \hat{\alpha}(H) \) becomes

\[
1 - \frac{1}{n} + \frac{c}{n^3 \log n}.
\]

On the other hand, this guarantee, for a minimal \( \sigma = \left( 1 - \frac{1}{n} \right) \cdot \frac{1}{2} \) is

\[
1 - \frac{1}{r} + \frac{c}{r n^2 \log n}.
\]

At the same time, it is easy to see that Frieze and Jerrum’s algorithm makes these points approximable within \( \alpha_n \) (since, in this case, \( H \cong K_n \)) and \( \alpha_r \) (since Turán’s theorem tells us that \( H \rightarrow K_r \) holds in this case), respectively. Our conclusion is that Frieze and Jerrum’s and Håstad’s algorithms have similar performance on these graphs.

Another way to study dense graphs is via random graphs. Let \( G(n, p) \) denote the random graph on \( n \) vertices in which every edge is chosen uniformly at random, and independently, with probability \( p = p(n) \). We say that \( G(n, p) \) has a property \( A \) asymptotically almost surely (a.a.s.) if the probability that it satisfies \( A \) tends to 1 as \( n \) tends to infinity. Here, we let \( p = c \) for some \( 0 < c < 1 \).

For \( G \in G(n, p) \) it is well known that a.a.s. \( \omega(G) \) assumes one of at most two values around \( \frac{2 \ln n}{\ln (1/p)} [15, 108] \). It is also known that, almost surely

\[
\chi(G) \sim \frac{n}{2 \ln (np)} \ln \left( \frac{1}{1 - p} \right),
\]

as \( np \to \infty \) \([14, 106] \). Let us say that \( \chi(G) \) is concentrated in width \( s \) if there exists \( u = u(n, p) \) such that a.a.s. \( u \leq \chi(G) \leq u + s \). Alon and Krivelevich \([3] \) have shown that for every constant \( \delta > 0 \), if \( p = n^{-1/2 - \delta} \) then \( \chi(G) \) is concentrated in width \( s = 1 \). That is, almost surely, the chromatic number takes one of two values.

**Proposition 5.41** Let \( H \in G(n, p) \). When \( np \to \infty \),

\[
F J_m(H) \sim 1 - \frac{2}{m} + \frac{2 \ln m}{m^2} + \frac{1}{m^2} - \frac{2 \ln m}{m^3},
\]

where \( m = \omega(H) \).

\[
H \hat{\alpha}(H) = p - \frac{p}{n} + (1 - p) \cdot \frac{c}{n^2 \log n} + \frac{pc}{n^3 \log n}.
\]

**Proof:** Let \( k = \chi(H) \).

\[
F J_m(H) \geq \alpha_m \cdot s(K_m, K_k) \sim \left( 1 - \frac{1}{m} + \frac{2 \ln m}{m^2} \right) \cdot \frac{2 \left( 1 - \frac{1}{m} \right)^{k^2}}{k(k - 1)} \sim
\]
5. Approximation Distance

\[ \sim \frac{(m^2 - m + 2 \ln m)(m - 1)k}{m^3(k - 1)} = \]

\[ = \frac{k}{k - 1} - \frac{2k}{m(k - 1)} + \frac{k}{m^2(k - 1)} + \frac{2k \ln m}{m^2(k - 1)} - \frac{2k \ln m}{m^3(k - 1)} \quad (** \text{)} \]

If \( n \) is large, then \( k \gg m \) and

\[ (** \text{)} \sim 1 - \frac{2}{m} + \frac{2 \ln m}{m^2} + \frac{1}{m} = \frac{2 \ln m}{m^3}. \]

The expected number of edges for a graph \( H \in \mathcal{G}(n, p) \) is \( \binom{n}{2} p \), so

\[ H\hat{\alpha}(H) = 1 - \frac{\lambda(G)}{d^2} \cdot (1 - \frac{e}{d^2 \log d}) = \frac{n}{d} = n, e(G) = \binom{n}{2} p/ = \]

\[ = 1 - \frac{n^2 - (n^2 - n)p}{n^2} \cdot (1 - \frac{e}{n^2 \log n}) = \]

\[ = 1 - (1 - p + \frac{p}{n}) \cdot (1 - \frac{e}{n^2 \log n}) = \]

\[ = \frac{mn^3 \log n + nc - pmc - pm^2 \log n + pc}{n^3 \log n} = \]

\[ = p - \frac{p}{n} + (1 - p) \cdot \frac{c}{n^2 \log n} + \frac{pc}{n^3 \log n} \]

\[ \blacksquare \]

We see that, in the limiting case, \( H\hat{\alpha}(H) \) tends to \( p \), while \( FJ_m(H) \) tends to \( 1 \). Again, this means that, for large enough graphs, we cannot do much better. With a more careful analysis, one could possibly reach an expression for \( FJ_m(H) \) that has a faster convergence rate.

We conclude this section by looking at what happens for graphs \( H \in \mathcal{G}(n, p) \) where \( np \) does not tend to \( \infty \) when \( n \to \infty \). The following theorem allows us to do this.

**Theorem 5.42 (Erdős and Rényi [48])** Let \( c \) be a positive constant and \( p = \frac{c}{n} \). If \( c < 1 \), then a.a.s. no component in \( \mathcal{G}(n, p) \) contains more than one cycle, and no component has more than \( \frac{\ln n}{c-1-mc} \) vertices.

Now we see that if \( np \to \infty \) when \( n \to \infty \) and \( 0 < \epsilon < 1 \), then \( \mathcal{G}(n, p) \) almost surely consists of components with at most one cycle. Thus, each component resembles a cycle where, possibly, trees are attached to certain vertices of the cycle, and each component is homomorphically equivalent to the cycle it contains. The result in Section 5.4.1 on cycle graphs is therefore applicable in this part of the \( \mathcal{G}(n, p) \) spectrum.
5.5 Fractional covering by $H$-cuts

In the following, we generalise the work of Šámal [122, 123, 124] on fractional covering by cuts to obtain a complete correspondence between a family of chromatic numbers $\chi_H(G)$ and $s(H, G)$. These chromatic numbers are a generalisation of Šámal’s cubical chromatic number $\chi_{c}(G)$, the latter corresponds to the case when $H = K_2$. We believe that this alternative view on our parameter $s$ can provide great benefits to the understanding of its properties. We transfer a result in the other direction, in Section 5.5.1, disproving a conjecture by Šámal on $\chi_{c}$, and settle another conjecture by him in the positive, in Section 5.5.2.

We start by recalling the notion of a fractional colouring of a hypergraph. Let $G$ be a (hyper-) graph with vertex set $V(G)$ and edge set $E(G) \subseteq 2^{V(G)}$. A subset $I$ of $V(G)$ is called independent in $G$ if no edge $e \in E(G)$ is a subset of $I$. Let $I$ denote the set of all independent sets of $G$ and for a vertex $v \in V(G)$, let $I(v)$ denote all independent sets which contain $v$. Let $I_1, \ldots, I_n \in I$ be a collection of independent sets.

**Definition 5.43** An $n/k$ independent set cover is a collection $I_1, \ldots, I_n$ of independent sets in $I$ such that every vertex of $G$ is in at least $k$ of them.

The **fractional chromatic number** $\chi_f(G)$ of $G$ is given by the following expression:

$$\chi_f(G) = \inf \left\{ \frac{n}{k} \mid \text{there exists an } n/k \text{ indep. set cover of } G \right\}$$

The definition of fractional covering by cuts mimics that of fractional covering by independent sets, but replaces vertices with edges and independent sets with certain cut sets of the edges. Let $G$ and $H$ be undirected simple graphs and $f$ be an arbitrary vertex map from $G$ to $H$. The map $f$ induces a partial edge map $f^\# : E(G) \to E(H)$ from $E(G)$ to $E(H)$. We will call the preimage of $E(H)$ under $f^\#$ an $H$-cut in $G$. When $H$ is a complete graph $K_k$, this precisely the standard notion of a $k$-cut in $G$. Let $\mathcal{C}$ denote the set of $H$-cuts in $G$ and for an edge $e \in E(G)$, let $\mathcal{C}(e)$ denote all $H$-cuts which contain $e$. The following definition is a generalisation of cut $n/k$-covers [123] to arbitrary $H$-cuts:

**Definition 5.44** An $H$-cut $n/k$-cover of $G$ is a collection $C_1, \ldots, C_N \in \mathcal{C}$ such that every edge of $G$ is in at least $k$ of them.

The graph parameter $\chi_H$ is defined as:

$$\chi_H(G) = \inf \left\{ \frac{n}{k} \mid \text{there exists an } H \text{-cut } n/k\text{-cover of } G \right\}$$

For $H = K_2$, Šámal [123] called the parameter $\chi_H(G)$ in Definition 5.44, the **cubical chromatic number** of $G$. Both the fractional chromatic number and the cubical chromatic number also have linear programming formulations. This, in particular, shows that the value in the infimum of the corresponding definition is obtained exactly for some $n$ and $k$. For our
5. Approximation Distance

A generalisation of the cubical chromatic number, the linear program is the following:

\[
\text{Minimise } \sum_{C \in \mathbb{C}} f(C) \\
\text{subject to } \sum_{C \in \mathcal{C}(e)} f(C) \geq 1 \quad \text{for all } e \in E(G), \\
\text{where } f : \mathcal{C} \rightarrow \mathbb{Q}_{\geq 0}.
\]  

\[\text{(5.11)}\]

**Proposition 5.45** The graph parameter \(\chi_H(G)\) is given by the optimum of the linear program in (5.11).

**Proof:** The proof is completely analogous to those for the corresponding statements for the fractional chromatic number (cf. [57]) and for the cubical chromatic number (Lemma 5.1.3 in [123]):

Let \(C_1, \ldots, C_n\) be an \(H\)-cut \(n/k\)-cover of \(G\). The solution \(f(C) = 1/k\) if \(C \in \{C_1, \ldots, C_n\}\), and \(f(C) = 0\) otherwise, has a measure of \(n/k\) in (5.11). Thus, the optimal measure of the linear program is at most \(\chi_H(G)\).

For the other direction, note that the coefficients of the program (5.11) are integral. Hence, there is a rational optimal solution \(f^*\). Let \(N\) be the least common multiple of the divisors of \(f^*(C)\) for \(C \in \mathcal{C}\). Assume that the measure of \(f^*\) is \(n/k\). Construct a collection of \(H\)-cuts by including the cut \(C\) a total of \(N \cdot f^*(C)\) times. This collection covers each edge at least \(N\) times using \(\sum_{C \in \mathcal{C}} N \cdot f^*(C) = N \cdot n/k\) cuts, i.e. it is an \(H\)-cut \(n/k\)-cover, so \(\chi_H(G)\) is at most equal to the optimal measure of the linear program. \(\blacksquare\)

For fractional colourings, it is well-known that an equivalent definition is obtained by taking \(\chi_f(G) = \inf\{n/k \mid G \rightarrow K_{n,k}\}\), where \(K_{n,k}\) denotes the Knösen graph, the vertex set of which is the \(k\)-subsets of \([n]\) and with an edge between \(u\) and \(v\) if \(u \cap v = \emptyset\). For \(H = K_2\), a corresponding definition of \(\chi_H(G) = \chi_\mathbb{Q}(G)\) was obtained in [123] by taking the infimum (actually minimum due to the formulation in (5.11)) over \(n/k\) for \(n\) and \(k\) such that \(G \rightarrow Q_{n/k}\). Here, \(Q_{n/k}\) is the graph on vertex set \([0,1]^n\) with an edge between \(u\) and \(v\) if \(d_H(u, v) \geq k\), where \(d\) denotes the Hamming distance.

This type of parameterised graph family which determines a particular chromatic number is often referred to as the scale of the chromatic number. In addition to the previously mentioned fractional colouring \(\chi_f\), where the scale is made up of Knösen graphs, and cubical colouring \(\chi_{\mathbb{Q}}\) where the scale is \(\{Q_{n/k}\}\), another prominent example is the circular chromatic number (Section 5.3) for which the scale is given by the family of circular complete graphs \(K_{n/k}\).

Here we generalise the family \(\{Q_{n/k}\}\) to produce one scale for each \(\chi_H\). To this end, let \(H_k^n\) be the graph on vertex set \(V(H)^n\) and an edge between \((u_1, \ldots, u_n)\) and \((v_1, \ldots, v_n)\) when \(|\{i \mid (u_i, v_i) \in E(H)\}| \geq k\). The proof of the following proposition is straightforward, but instructive.

**Proposition 5.46** For \(G, H \in \mathbb{G}\), we have

\[\chi_H(G) = \inf \left\{ \frac{n}{k} \mid G \rightarrow H_k^n \right\}, \]

\[\text{(5.12)}\]
5.5. Fractional covering by \( H \)-cuts

**Proof:** Both sides are defined by infima over \( n/k \). Therefore, it will suffice to show how to translate each of the parameterised objects (an \( H \)-cut \( n/k \)-cover on the left-hand side and a homomorphism from \( G \) to \( H^n_k \) on the right-hand side) into the other, for some given values of \( n \) and \( k \).

Let \( h : V(G) \rightarrow H^n_k \) be a homomorphism and denote by \( \text{pr}_i \) \( h \) the projection of \( h \) onto the \( i \)-th coordinate. Then, for each edge \( e \in E(G) \), at least \( k \) of the edge maps \((\text{pr}_i h)^\# \) must map \( e \) to an edge in \( H \). Hence, the \( H \)-cuts \( \mathcal{C}_i = \{ e \in E(G) \mid (\text{pr}_i h)^\#(e) \in E(H) \} \), for \( 1 \leq i \leq n \), constitute an \( H \)-cut \( n/k \)-cover of \( G \).

For the other direction, note that each \( H \)-cut \( \mathcal{C}_i \) can be defined by a vertex map \( f_i : V(G) \rightarrow H \). For \( v \in V(G) \), let \( h'(v) = (f_1(v), f_2(v), \ldots, f_n(v)) \). To verify that \( h' \) is a homomorphism, note that for every edge \( \{u, v\} \in E(G) \), at least \( k \) of the \( f_i \) must include \( e \) in their corresponding cut \( \mathcal{C}_i \). Hence \( \{i \mid \{f_i(u), f_i(v)\} \in E(H)\} \geq k \), so by definition \( \{h'(u), h'(v)\} \in E(H^n_k) \).

Sámal further notes that \( \chi_0(G) \) is given by the fractional chromatic number of a certain hypergraph associated to \( G \). Inspired by this, we provide a similar formulation in the general case.

**Proposition 5.47** Let \( G' \) be the hypergraph obtained from \( G \) on vertex set \( V(G') = E(G) \) with edge set \( E(G') \) taken to be the set of minimal subgraphs \( K \subseteq G \) such that \( K \not\twoheadrightarrow H \). Then,

\[
\chi_H(G) = \chi_{f'}(G').
\]

**Proof:** We will let \( \mathcal{I} \) denote the set of independent sets in \( G' \) and \( \mathcal{C} \) the set of \( H \)-cuts in \( G \). The parameter \( \chi_H(G) \) is the infimum of \( n/k \) over all \( n/k \)-covers of \( E(G) \) by sets in \( \mathcal{C} \). Similarly, the parameter \( \chi_{f'}(G') \) is the infimum of \( n/k \) over all \( n/k \)-covers of \( V(G') = E(G) \) by sets in \( \mathcal{I} \). By definition, the independent sets of \( G' \) correspond precisely to the edge sets \( E(K) \) of those subgraphs \( K \subseteq G \) such that \( K \rightarrow H \). Hence, \( \mathcal{C} \subseteq \mathcal{I} \), so \( \chi_{f'}(G') \geq \chi_H(G) \).

On the other hand, assume that \( K \subseteq G \) is a subgraph such that \( K \rightarrow H \), and that the independent set \( E(K) \in \mathcal{I} \) is not in \( \mathcal{C} \). Then, any homomorphism \( h : V(K) \rightarrow V(H) \) induces an \( H \)-cut \( C \) of \( G \), and clearly we must have \( E(K) \subseteq C \). Thus, we can replace all edge sets in a cover by sets from \( \mathcal{C} \), without violating the constraint that all edges are covered at least \( k \) times. The proposition now follows.

Finally, we work out the correspondence to \( s(H, G) \). Consider the dual program of (5.11):

\[
\begin{align*}
\text{Maximise} & \quad \sum_{e \in E(G)} g(e) \\
\text{subject to} & \quad \sum_{e \in X} g(e) \leq 1 \quad \text{for all } H \text{-cuts } X \in \mathcal{C}, \\
\text{where} & \quad g : E(G) \rightarrow \mathbb{Q}_{\geq 0},
\end{align*}
\]
5. Approximation Distance

In (5.13) let \(1/s = \sum_{e \in E(G)} g(e)\) and make the variable substitution \(w = g/s\). This leaves the following program:

\[
\begin{align*}
\text{Maximise} & \quad s^{-1} \\
\text{subject to} & \quad \sum_{e \in E(X)} w(e) \leq s & \text{for all } H\text{-cuts } X \in \mathcal{C}, \\
& \quad \sum_{e \in E(G)} w(e) = 1 \\
& \quad w : E(G) \to \mathbb{Q}_{\geq 0}.
\end{align*}
\]

(5.14)

Since the maximal value of \(s^{-1}\) is the same as 1 over the minimal value of \(s\), a comparison with (5.8) now reveals the following fact:

**Proposition 5.48** For \(G, H \in \mathcal{G}\), we have

\[
\chi_{H}(G) = 1/s(H, G).
\]

(5.15)

We now move on to address two conjectures by Šámal [123] on the cubic chromatic number \(\chi_q = \chi_{K_4}\). In Section 5.5.1 we discuss an upper bound on \(s\) which relates to the first conjecture, Conjecture 5.5.3 [123], which asserts that \(\chi_q(G)\) can be determined by measuring the maximum cut over all subgraphs of \(G\). We show that this is false by providing a counterexample from Section 5.3.1. We then consider Conjecture 5.4.2 [123], which concerns “measuring the scale” of the cubic chromatic numbers, i.e. determining \(\chi_q\) for the scale graphs \(Q_{n/k}\) themselves. We prove that this conjecture is true, and state it as Proposition 5.49 in Section 5.5.2.

5.5.1 An upper bound

In Section 5.3 we obtained lower bounds on \(s\) by relaxing the linear program (5.9). In most cases, the corresponding solution was proved feasible in the original program, and hence optimal. Now, we take a look at the only known source of general upper bounds for \(s\).

Let \(G, H \in \mathcal{G}\), with \(H \rightarrow G\) and let \(S\) be such that \(H \rightarrow S \rightarrow G\). Then, applying Lemma 5.12 followed by Lemma 5.11 gives

\[
s(H, G) \leq s(H, S) = \inf_{\omega \in W(S)} mc_H(S, \omega) \leq mc_H(S, 1/|E(S)|).
\]

(5.16)

We can therefore upper bound \(s(H, G)\) by the least maximal \(H\)-cut taken over all graphs \(S\) homomorphic to \(G\). For \(H = K_2\), we have

\[
s(K_2, G) \leq \min_{S \subseteq G} b(S),
\]

where \(b(S)\) denotes the bipartite density of \(S\). Šámal [123] conjectured that this inequality, expressed on the form \(\chi_q(S) \geq 1/(\min_{S \subseteq G} b(S))\), can be replaced by an equality. We answer this in the negative, using \(K_{11/4}\) as our counterexample. Lemma 5.27 with \(k = 1\) gives \(s(K_2, K_{11/4}) = 17/22\). If \(s(K_2, K_{11/4}) = b(S)\) for some \(S \subseteq K_{11/4}\) it means that \(S\) must have at least 22 edges. Since \(K_{11/4}\) has exactly 22 edges it follows that \(S = K_{11/4}\). However, a cut in a cycle must contain an even number of edges. Since the
edges of $K_{11/4}$ can be partitioned into two cycles, we have that the maximum cut in $K_{11/4}$ must be of even size, hence $|E(K_{11/4})| \cdot b(K_{11/4}) \neq 17$. This is a contradiction.

5.5.2 Confirmation of a scale

As a part of his investigation of $\chi_q$, Šámal [123] set out to determine the value of $\chi_q(Q_{n/k})$ for general $n$ and $k$. This question is suggested by the guise of $\chi_q$ as a chromatic number. For the fractional chromatic number and the circular chromatic number, results for such measuring of the scale exist and provide very appealing formulae: $\chi_f(K_{n,k}) = \chi_c(G_{n/k}) = n/k$. For $\chi_q(Q_{n/k}) = 1/s(K_2, Q_{n/k})$, we are immediately out of luck as $1/2 < s(K_2, G) \leq 1$, i.e. $1 \leq \chi_q(G) < 2$ for all non-empty graphs. For $1 \leq n/k < 2$, however, Šámal gave a conjecture (Conjecture 5.4.2 in [123]). We complete the proof of his conjecture to obtain the following result:

Proposition 5.49 Let $k,n$ be integers such that $k \leq n < 2k$. Then,

$$\chi_q(Q_{n/k}) = \begin{cases} 
\frac{n}{k} & \text{if } k \text{ is even, and} \\
\frac{(n+1)(k+1)}{k} & \text{if } k \text{ is odd.}
\end{cases}$$

Corollary 5.50 Let $k,n$ be integers such that $1/2 < k/n \leq 1$. Then, $s(K_2, Q_{n/k}) = k/n$ if $k$ is even, and $(k+1)/(n+1)$ if $k$ is odd.

If we make sure that $k$ is even, possibly by multiplying both $k$ and $n$ by a factor two, we get the following:

Corollary 5.51 For any rational number $r$, with $1/2 < r \leq 1$, there exists a graph $G$ such that $s(K_2, G) = r$.

Šámal [123] provides the upper bound for Proposition 5.49, and an approach to the lower bound of Proposition 5.49 using the largest eigenvalue of the Laplacian of a subgraph of $Q_{n/k}$. The computation of this eigenvalue boils down to an inequality (Conjecture 5.4.6 in [123]) involving some binomial coefficients. We first introduce the necessary notation and then prove the remaining inequality in Proposition 5.53, whose second part, for odd $k$, corresponds to one of the formulations of the conjecture. Proposition 5.49 then follows from Theorem 5.4.7 in [123], conditioned on the result of this proposition.

Let $k,n$ be positive integers such that $k \leq n$, and let $x$ be an integer such that $1 \leq x \leq n$. For $k \leq n < 2k$, let $S_x(n,k,x)$ denote the set of all $k$-subsets of $[n]$ that have an odd-sized intersection with $[n] \setminus [n-x]$. Define $S_{x}(n,k,x)$ analogously as the $k$-subsets of $[n]$ that have an even-sized intersection with $[n] \setminus [n-x]$, i.e. $S_{x}(n,k,x) = \binom{n-x}{k} \setminus S_x(n,k,x)$. Let $N_x(n,k,x) = |S_x(n,k,x)|$ and $N_{x}(n,k,x) = |S_{x}(n,k,x)|$. Then,

$$N_x(n,k,x) = \sum_{t \text{ odd}} \binom{x}{t} \binom{n-x}{k-t}.$$
5. Approximation Distance

\[ N_e(n, k, x) = \sum_{\text{event}} \binom{x}{t} \binom{n-x}{k-t}. \]

When \( x \) is odd, the function \( f : S_e(2k, k, x) \rightarrow S_e(2k, k, x) \), given by the complement \( f(\sigma) = [n] \setminus \sigma \), is a bijection. Since \( N_e(n, k, x) + N_o(n, k, x) = \binom{n}{k} \), we have

\[ N_o(2k, k, x) = N_e(2k, k, x) = \frac{1}{2} \binom{2k}{k}. \quad (5.17) \]

**Lemma 5.52** Let \( 1 \leq x < n = 2k - 1 \) with \( x \) odd. Then, \( N_e(n, k, x) = N_e(n, k, x + 1) \) and \( N_o(n, k, x) = N_o(n, k, x + 1) \).

**Proof:** First, partition \( S_e(n, k, x) \) into \( A_1 = \{ \sigma \in S_e(n, k, x) \mid n-x \not\in \sigma \} \) and \( A_2 = S_e(n, k, x) \setminus A_1 \). Similarly, partition \( S_e(n, k, x + 1) \) into \( B_1 = \{ \sigma \in S_e(n, k, x + 1) \mid n-x \not\in \sigma \} \) and \( B_2 = S_e(n, k, x + 1) \setminus B_1 \). Note that \( A_1 = B_1 \). We argue that \( |A_2| = |B_2| \). To prove this, define the function \( f : 2^{[n]} \rightarrow 2^{[n-1]} \) by

\[ f(\sigma) = (\sigma \cap [n-x-1]) \cup \{s-1 \mid s \in \sigma, s > n-x\}. \]

That is, \( f \) acts on \( \sigma \) by ignoring the element \( n-x \) and renumbering subsequent elements so that the image is a subset of \( [n-1] \). Note that \( f(A_2) = S_e(2k-2, k-1) \) and \( f(B_2) = S_e(2k-2, k-1, x) \). Since \( x \) is odd, it follows from (5.17) that \( |f(A_2)| = |f(B_2)| \). The first part of the lemma now follows from the injectivity of the restrictions \( f|_{A_2} \) and \( f|_{B_2} \). The second equality is proved similarly.

**Proposition 5.53** Choose \( k, n \) and \( x \) so that \( k \leq n < 2k \) and \( 1 \leq x \leq n \). For odd \( k \),

\[ N_e(n, k, x) \leq \binom{n-1}{k-1}, \]

and for even \( k \),

\[ N_o(n, k, x) \leq \binom{n-1}{k-1}. \]

**Proof:** We will proceed by induction over \( n \) and \( x \). The base cases are given by \( x = 1, x = n, \) and \( n = k \). For \( x = 1 \),

\[ N_o(n, k, x) = \binom{n-1}{k-1} \text{ and } N_e(n, k, x) = \binom{n-1}{k} \leq \binom{n-1}{k-1}, \]

where the inequality holds for all \( n < 2k \). For \( x = n \) and odd \( k \), we have \( N_e(n, k, x) = 0 \), and for even \( k \), we have \( N_o(n, k, x) = 0 \). For \( n = k \),

\[ N_e(n, k, x) = 1 - N_o(n, k, x) = \begin{cases} 1 & \text{if } x \text{ is even,} \\ 0 & \text{otherwise.} \end{cases} \]
5.6. Approximation distance for Max Sol

Let $x > 1$ and consider $N_\omega(n, k, x)$ for odd $k$ and $k < n < 2k - 1$. Partition the sets $\sigma \in S_\omega(n, k, x)$ into those for which $n \in \sigma$ on the one hand and those for which $n \notin \sigma$ on the other hand. These parts contain $N_\omega(n-1, k-1, x-1)$ and $N_\omega(n-1, k, x-1)$ sets, respectively. Since $k - 1$ is even, and since $k \leq n - 1 < 2(k - 1)$ when $k < n < 2k - 1$, it follows from the induction hypothesis that

$$N_\omega(n, k, x) = N_\omega(n-1, k-1, x-1) + N_\omega(n-1, k, x-1) \leq \binom{n-2}{k-2} + \binom{n-2}{k-1} = \binom{n-1}{k-1}.$$ 

The case for $N_\omega(n, k, x)$ and even $k$ is treated identically.

Finally, let $n = 2k - 1$. If $x$ is odd, then Lemma 5.52 is applicable, so we can assume that $x$ is even. Now, as before

$$N_\omega(2k-1, k, x) = N_\omega(2k-2, k-1, x-1) + N_\omega(2k-2, k, x-1) \leq \frac{1}{2} \binom{2k-2}{k} + \binom{2k-3}{k-1} = \binom{n-1}{k-1},$$

where the first term is evaluated using (5.17). The same inequality can be shown for $N_\omega(2k-1, k, x)$ and even $k$, which completes the proof. ■

5.6 Approximation distance for Max Sol

In this section, we will retrace some of the steps taken in the analysis of the parameter $s$ for Max H-Col, when we replace this problem by Max Sol. Our main objective is to find out how much can be carried over to Max Sol, and what changes need to be done. Since our knowledge of the approximability of Max Sol is so limited, we have no direct applications. Instead, we end our investigation after obtaining a linear programming formulation for $s$.

The first change will be to allow undirected graphs with loops, since these are no longer trivial for Max Sol. For such graphs $G$, let $\Omega(G)$ be the set of weight functions $\omega : V(G) \rightarrow \mathbb{Q}_{\geq 0}$. Let $\mathcal{M}$ be the set of vertex weighted undirected graphs with loops allowed, i.e. if $(G, \omega) \in \mathcal{M}$, then $G$ is an undirected graph, possibly with loops, and $\omega \in \Omega(G)$.

For Max Sol$(M, \nu)$, with $(M, \nu) \in \mathcal{M}$, it is NP-hard even to find a solution, unless $M$ either contains a loop, or $M$ is bipartite. There are also some problems in this family that are in PO, and some that are in poly-APX. For such problems, our results, which ultimately relates constant approximation ratios, do not make much sense. We will keep this in mind, but for now it will not make any difference whether we keep these problems in our definitions or not. We define $\preceq$ as follows:
5. Approximation Distance

**Definition 5.54** Let \((M, \nu) \preceq (N, \nu')\), if for all \(\omega \in \Omega(M)\), there exists a homomorphism \(h \in \text{Hom}(M, N)\) such that

\[
\sum_{u \in V(M)} \omega(u) \cdot \nu(u) \leq \sum_{u \in V(M)} \omega(u) \cdot \nu'(h(u)).
\]  
(5.18)

It will be convenient at this point to introduce a bit of notation. We will use the same notation in Chapter 6, and it will be explained there in a little more detail. Let \((M, \nu), (M', \nu') \in M\), and assume that \(M\) and \(M'\) are defined on the same vertex set \(V\). Then,

\[
\langle \nu, \nu' \rangle = \sum_{v \in V} \nu(v) \cdot \nu'(v).
\]

The notation hides the set over which the summation takes place, but it will always be clear from the context. For an instance \((G, \omega)\) of \(\text{MaxSol}(M, \nu)\), and a solution \(f \in \text{Hom}(G, M)\), we can now write

\[
m_M(f) = \langle \omega, f(\nu) \rangle,
\]

where \(f(\nu) \in \Omega(G)\) is defined as the composition \(\nu \circ f\).

Let \(M\) and \(N\) be undirected graphs with loops allowed, and let \(g \in \text{Hom}(M, N)\). We then define the function \(g^t : \Omega(M) \to \Omega(N)\) by:

\[
g^t(\nu)(u) = \sum_{v \in g^{-1}(u)} \nu(v),
\]

for \(\nu \in \Omega(M)\), and \(u \in V(N)\). It is easy to check that following identity holds:

\[
\langle \omega, g(\nu) \rangle = \langle g^t(\omega), \nu \rangle.
\]

We are now ready to see if our definition of \(\preceq\) will suffice.

**Proposition 5.55** Let \((M, \nu) \preceq (N, \nu')\). Then, there is a function \(F : \text{Sol}_M(G, \omega) \to \text{Sol}_N(G, \omega)\), where \((G, \omega) \in \mathcal{M}\), and computable in time polynomial in \(G\), such that for any \(f \in \text{Sol}_M(G, \omega)\),

\[
m_M(f) \leq m_N(F(f)).
\]

**Proof:** Let \(f : V(G) \to V(M)\) be a solution to \((G, \omega)\). Since \(M \preceq N\) we can find a homomorphism \(h \in \text{Hom}(M, N)\) such that (5.18) holds for \(f^t(\omega) \in \Omega(M)\). Let \(F\) be defined by \(F(f) = h \circ f\). We have,

\[
m_M(f) = \langle \omega, f(\nu) \rangle = \langle f^t(\omega), \nu \rangle = \langle \omega, h(\nu) \rangle = \langle \omega, (h \circ f)(\nu') \rangle = m_N(F(f)).
\]

Since \(\text{Hom}(M, N)\) does not depend on \(G\), an appropriate \(h\) can be found in time polynomial in \(G\).
Example 5.56 Let \((M, \nu) \in M\), and let \(h\) be an endomorphism on \(M\) (i.e. \(h\) is a homomorphism from \(M\) to \(M\)) such that \(\nu(h(v)) \geq \nu(v)\) for all \(v \in V(M)\). Let \((N, \nu') = (h(M), h'(\nu))\). In this case we have \((M, \nu) \leq (N, \nu')\), but clearly also \((N, \nu') \leq (M, \nu)\) since the identity function on \(N\) can be seen as a homomorphism from \(N\) to \(M\). We conclude that \((M, \nu)\) and its max-core are equivalent under \(\leq\).

We have obtained a reasonable definition of \(\leq\), and we can attempt the following definition of \(s\). Let \((M, \nu) \leq (N, \nu')\), and define

\[
s(M, N) = \inf_{(G, \omega) \in M} \frac{\text{Opt}_M(G, \omega)}{\text{Opt}_N(G, \omega)}.
\]

(5.19)

We must consider two things in relation to this definition. The first is that for some instances \((G, \omega)\), there may be no solutions at all. We can solve this issue by assuming that the graphs \(M\) and \(N\) are either bipartite or contains at least one loop. The instances with solutions are then easily recognisable, and we will assume that the infimum is taken only over such instances. The second potential problem is that one, or both, of the optima may be 0. If \(\text{Opt}_N(G, \omega) = 0\), then \(\text{Opt}_M(G, \omega)\) must be 0 too, since \((M, \nu) \leq (N, \nu')\). We may safely ignore such instances \((G, \omega)\) in the infimum, since we are sure to make a sufficiently good approximation in any case. If \(\text{Opt}_M(G, \omega) = 0\), but \(\text{Opt}_N(G, \omega)\) is not, then we must simply accept that our method will not provide an approximation guarantee for \(\text{Max Sol}(N, \nu')\), as indicated by the fact that \(s(M, N) = 0\). This will not be a problem when \(M\) and \(N\) are both bipartite, but it is harder to determine what will happen in other cases. Further progress on the approximability of \(\text{Max Sol}\) for graphs may very well resolve this issue completely.

With these considerations, we can now use Lemma 5.1 with the parameter \(s\) defined as in (5.19). We would also like to find a reasonable way to compute \(s\), and therefore proceed as in Section 5.2.1. As we move on, it will prove crucial to be able to map solutions back from \(N\) to \(M\) as well. For this reason, we will only be able to compare problems \(\text{Max Sol}(M, \nu)\) and \(\text{Max Sol}(N, \nu')\) when either both \(M\) and \(N\) are bipartite, or both contain at least one loop. In the following results, we will assume that the pair of graphs, \(M\) and \(N\), fulfills this requirement.

Lemma 5.57 Let \((M, \nu), (N, \nu') \in M\) be such that \(M \equiv N\). Then, for every \((G, \omega) \in M\) such that \(\text{Opt}_N(G, \omega) > 0\), and any optimal solution \(f\) to the instance \((G, \omega)\) of \(\text{Max Sol}(N, \nu')\), there is a \(\mu \in \Omega(N)\) such that \(\langle \mu, \nu' \rangle = 1\), and

\[
\frac{\text{Opt}_M(G, \omega)}{\text{Opt}_N(G, \omega)} \geq \text{Opt}_{M, \nu}(N).
\]

Proof: Let \(m^* = \langle \omega, f(\nu) \rangle\) be the measure of the optimal solution \(f\), and let \(\mu = f^*(\omega)/m^*\). First, we check that \(\mu\) satisfies the requirement:

\[
\langle \mu, \nu' \rangle = (f^*(\omega)/m^*, \nu') = \langle \omega, f(\nu') \rangle/m^* = 1.
\]
5. Approximation Distance

We now proceed as in the proof of Lemma 5.9. Let \( g \in \text{Hom}(N, M) \) be any optimal solution to \((N, \mu)\) of \(\text{MaxSol}(M, \nu)\). Then, \( g \circ f \) is a solution to \((G, \omega)\) as an instance of \(\text{MaxSol}(M, \nu)\). Hence,

\[
\text{Opt}_M(G, \omega) \geq \langle \omega, (g \circ f)(\nu) \rangle = (f^t(\omega), g(\nu)) = \langle \mu \cdot m^*, g(\nu) \rangle = \langle \mu, g(\nu) \rangle \cdot m^* = \text{Opt}_M(N, \mu) \cdot \text{Opt}_N(G, \omega),
\]

where the final equality follows since \( g \) is an optimal solution. \( \blacksquare \)

For any \((M, \nu) \in \mathcal{M}\), we define

\[
\hat{\Omega}(M, \nu) = \{ \mu \in \Omega(M) \mid \langle \mu, \nu \rangle = 1 \}.
\]

By specialising the set over which we take the infimum in (5.19), we find that

\[
s(M, N) = \inf_{(G, \omega) \in \mathcal{M}} \frac{\text{Opt}_M(G, \omega)}{\text{Opt}_N(G, \omega)} \leq \inf_{\omega \in \hat{\Omega}(N, \nu')} \frac{\text{Opt}_M(N, \omega)}{\text{Opt}_N(N, \omega)} = \inf_{\omega \in \hat{\Omega}(N, \nu')} \text{Opt}_M(N, \omega).
\]

Therefore, we have the following corollary from Lemma 5.57:

**Corollary 5.58** For \((M, \nu), (N, \nu') \in \mathcal{M}\) such that \(M \equiv N\), we have

\[
s(M, N) = \inf_{\omega \in \hat{\Omega}(N, \nu')} \text{Opt}_M(N, \omega).
\]

It is possible to improve on the expression for \( s \) in Corollary 5.58, analogously to the improvement carried out in Lemma 5.11, based on the edge automorphism group of \( N \). Here, the ordinary (vertex) automorphism group plays a similar role, but a little extra care is needed, since for \(\text{MaxSol}, \ N\) comes with a weight function \( \nu' \). We omit the details.

Thus far, we have ignored a somewhat troublesome consequence of our definitions, which we will have to address. Consider \((M, \nu) \in \mathcal{M}\), and a rational constant \( c > 0 \). We would like \((M, \nu)\) and \((M, c \cdot \nu)\) to be equivalent in some sense, as they clearly give rise to the same problem. However, from Definition 5.54 it follows that \((M, \nu)\) and \((M, c \cdot \nu)\) are not equivalent under \( \leq \) (unless \( c = 1 \)) and furthermore, that \( s((M, \nu), (M, c \cdot \nu)) = 1/c \).

To resolve this, we note that we only need \( M \equiv N \) in order to obtain Corollary 5.58. The idea is the same as in Section 5.2.3, where we defined a symmetric parameter \( d(M, N) = 1 - s(M, N) \cdot s(N, M) \). That is, the quantity \( s(M, N) \cdot s(N, M) \) does behave as we would like:

\[
s((M, \nu), (M, c \cdot \nu)) \cdot s((M, c \cdot \nu), (M, \nu)) = 1.
\]

For \((M, \nu), (N, \nu') \in \mathcal{M}\), it is then possible to prove a result in the spirit of Lemma 5.17, which holds for all \((M, \nu), (N, \nu') \in \mathcal{M}\) such that \(M \equiv N\)
5.7. Discussion and open problems

(and some additional constraints on when an optimum of 0 is obtained for the two problems.)

We conclude this section by working out the linear program corresponding to the formulation of \( s \) in Corollary 5.58. A first attempt yields the following program:

Minimise \( s \)
subject to \( \langle w, f(v) \rangle \leq s \) for each \( f \in \text{Hom}(N, M) \),
\( \langle w, \nu' \rangle = 1 \)
where \( w \in \Omega(N) \), \( s \geq 0 \).

As was noted in Section 5.5, we can rewrite such a program into an equivalent maximisation program for \( 1/s \) as follows:

Maximise \( \langle w, \nu' \rangle \)
subject to \( \langle w, f(v) \rangle \leq 1 \) for each \( f \in \text{Hom}(N, M) \),
where \( w \in \Omega(N) \).

The dual program is again a fractional covering problem. Here we cover \( \nu' \) with weight functions \( f(v) \) for various \( f \in \text{Hom}(N, M) \). The inequality \( \omega \geq \mu \), where \( \omega, \mu \in \Omega(N) \), indicates that \( \omega(v) \geq \mu(v) \) for all \( v \in V(N) \).

Minimise \( \sum_{f \in \text{Hom}(N, M)} z(f) \)
subject to \( \sum_{f \in \text{Hom}(N, M)} z(f) \cdot f(v) \geq \nu' \)
where \( z : \text{Hom}(N, M) \to \mathbb{Q}_{\geq 0} \).

5.7 Discussion and open problems

What started out as a very simple idea has diverged in a number of directions, with plenty of room in each for further investigation and improvements. We single out two main topics, and discuss their respective future prospects and interesting open problems. These two topics relate to the application of our approach to the approximability of the problem \( \text{MAX } H\text{-COL} \) (and more generally to \( \text{MAX CSP}(\Gamma) \)) and to the computation and interpretation of the parameter \( s \) in this case.

Application to \( \text{MAX } H\text{-COL} \)

As we applied the initial idea to the problem \( \text{MAX } H\text{-COL} \), we arrived at a binary graph parameter \( s \) that, in a sense, measures how well one graph can be embedded in another. While not apparent from the original definition in (5.4), which involves taking an infimum over all possible instances, we have shown that the parameter can be computed effectively by the means of linear programming. Given a graph \( H \) and known approximability properties for \( \text{MAX } H\text{-COL} \), this metric allows us to deduce bounds on the corresponding properties for \( \text{MAX } H'\text{-COL} \) for graphs \( H' \) that are “close” to \( H \). Our approach can be characterised as local; the closer together (in the metric \( d \) two
5. Approximation Distance

graphs are located, the more precise are our bounds. In principle, given a
dense enough “base set” of problems with known approximability properties,
our method could be used to derive good bounds on the approximability of
MAX H-Col for all graphs.

For the applications in Section 5.4, we have used the complete graphs as
our base set of known problems. We have shown that this set of graphs is
sufficient for achieving new, non-trivial bounds on several different classes
of graphs. That is, when we apply Frieze and Jerrum's algorithm [55] to
MAX H-Col, we obtain results comparable to, or better than those guar-
anteed by Håstad's MAX 2-Csp algorithm [75], for the classes of graphs we
have considered. This comparison should however be taken with a grain of
salt. The analysis of Håstad's MAX 2-Csp algorithm only aims to prove it
better than a random assignment, and may leave room for strengthening of
the approximation guarantee. At the same time, we are overestimating the
distance for most of the graphs under consideration. It is likely that both
results can be improved, within their respective frameworks. When con-
sidering inapproximability, we have strongly relied on the Unique Games
Conjecture—hence, we are part of the growing body interested in seeing
UGC settled. Weaker inapproximability results, independent of UGC, exist
for both MAX cut [74] and MAX k-cut [88], and they are applicable in our
setting. We want to emphasise that our method is not per se dependent on
the truth of the UGC.

Our results suggest two clear directions of research. On the one hand,
to progress further, we will need approximability/inapproximability result
pairs for MAX H-Col on a substantially larger class of graphs. The draw-
back of our current lack of a larger base set can be exemplified by the prob-
lem MAX C_5-Col: we have s(K_2, C_5) = 4/5 and s(C_5, K_3) = 2/3, which
gives us the inconsequential inapproximability bounds α_{GW} \cdot 5/4 > 1, and
α_3 \cdot 3/2 > 1. Clearly, the graph C_5 is too far separated from the complete
graphs for our local approach to work. Raghavendra [116] has presented an
algorithm for MAX CSP(Γ) based on semi-definite programming. Under
the Unique Games Conjecture, this algorithm optimally approximates MAX
CSP(Γ) in polynomial-time, i.e. no other polynomial-time algorithm can
approximate the problem substantially better. However, it seems nosi-
ously difficult to determine the approximation ratio implied by this result,
for a given MAX CSP(Γ): Raghavendra and Steurer [117] gives an algorithm
to compute this ratio for any Γ, but the algorithm is doubly exponential in
the size of the domain (which is the number of vertices in our case.) For
our purposes, it would suffice to be able to work out the approximation ra-
tios for some restricted set of graphs, i.e. a set of constraint languages, each
consisting of a single binary and symmetric relation. It is possible that the
analysis of such a restricted case is more manageable.

The second direction of research is to obtain a refined knowledge of how
to calculate, or estimate, the parameter s. In our applications, we usually
do not measure the actual distance from each graph to the closest complete
5.7. Discussion and open problems

graph. Instead, we bound this distance for a class of graphs by the distance from $K_2$ to some cycle, or by the distance between two complete graphs. Endōs [47] has proved that for any positive integers $k$ and $l$ there exists a graph of chromatic number $k$ and girth at least $l$. Such graphs cannot be sandwiched between $K_2$ and a cycle, as was our approach for the graphs of high girth, in Section 5.4.1. Additionally, there are obviously graphs with an arbitrarily large gap between clique number and chromatic number, which causes that approach to fail as well. In general, to apply our method more precisely, we need a better understanding of the structure of $\mathcal{C}_S$ and how this structure interacts with our metric $d$.

Clearly, progress in either one of the directions will influence what type of result to look for in the other. We suggest two interesting candidates for future investigations: the circular complete graphs, for which we have obtained partial results for the parameter $s$ in Section 5.3, and the Kneser graphs, see for example [70]. Both of these classes generalise the complete graphs, and have been subject to substantial previous research. The Kneser graphs contain many examples of graphs with low clique number, but high chromatic number. They could thus prove to be an ideal starting point for studying this phenomenon in relation to our parameter.

We conclude this part of the discussion by considering three possible extensions of our results. We have already noted that MAX $H$-Col is a special case of the MAX CSP($\Gamma$) problem, parameterised by a finite constraint language $\Gamma$. It should be relatively clear that we can extend the parameter $s$ to a function from pairs of constraint languages to $\mathbb{Q}_{\geq 0}$. This would constitute a novel method for studying the approximability of MAX CSP—a method that, hopefully, may cast some new light on the performance of Raghavendra’s algorithm. However, in this general setting, it may be a lot harder to find usable results regarding the homomorphism relation. A different approach is found in Kaporis et al. [89] who show that $mc_2$ is approximable within $0.952$ for any given average degree $d$, and asymptotically almost all random graphs $G$ in $\mathcal{G}(n, m = \left\lfloor \frac{2}{10}n \right\rfloor)$. Here, $\mathcal{G}(n, m)$ is the probability space of random graphs on $n$ vertices and $m$ edges, selected uniformly at random. In a similar vein, Coja-Oghlan et al. [28] give an algorithm that approximates $mc_2$ within $1 - O(1/\sqrt{mp})$ in expected polynomial time, for graphs from $\mathcal{G}(n, p)$. It would be interesting to know if these results could be carried further, to achieve better approximability bounds on MAX $H$-Col.

Parameter-specific concerns

For a graph $G$ with a circular chromatic number $r$ close to 2 we can use Lemma 5.12 to bound $s(K_2, G) \geq s(K_2, K_r)$. Due to Proposition 5.26, we have also seen that with this method, we are unable to distinguish between the class of graphs with circular chromatic number $2 + 1/k$ and the (larger) class with circular chromatic number $2 + 2/(2k - 1)$. Nevertheless, the method is quite effective when applied to sequences of graph classes for which the circular chromatic number tends to 2, as was the case in Proposition 5.38.
5. Approximation Distance

Much of the extensive study conducted in this direction was instigated by the restriction of a conjecture by Jaeger [77] to planar graphs. This conjecture is equivalent to the claim that every planar graph of girth at least $4k$ has a circular chromatic number at most $2 + 1/k$, for $k \geq 1$. The case $k = 1$ is Grötzsch’s theorem; that every triangle-free planar graph is 3-colourable. Currently, the best lower bound on the girth of a planar graph which implies a circular chromatic of at most $2 + 1/k$ is $\frac{4k-2}{k}$, and is due to Borodin et al. [17]. We remark that Jaeger’s conjecture implies a weaker statement in our setting. Namely, if $G$ is a planar graph with girth greater than $4k$, then $G \rightarrow C_k$ implies $s(K_2, G) \geq s(K_2, C_k) = 2k/(2k + 1)$. Deciding this to be true would certainly provide support for the original conjecture, and would be an interesting result in its own right. Our starting observation shows that the slightly weaker condition $G \rightarrow K_{2+2/(2k-1)}$ implies the same result.

For edge-transitive graphs $G$, in our case the cycles and the complete graphs, it is not surprising that the expression $s(K_r, G)$ assumes a finite number of values, as a function of $r$. Indeed, Lemma 5.11 states that $s(K_r, G) = m c_{K_r}(G, 1/e(G))$, which leaves at most $e(G)$ possible values for $s$. This produces a number of constant intervals which are partly responsible for the constant regions of Proposition 5.29, and the discussion in Section 5.3.2. More surprising are the constant intervals that arise from $s(K_r, K_{2+2/(2k-1)})$. They give some hope that the behaviour of $s$ is possible to characterise more generally. One direction could be to identify additional constant regions, perhaps showing that they tile the entire space.

In Section 5.5 we generalised the notion of covering by cuts due to Šámal. In doing this, we have found a different interpretation of our parameter as an entire family of chromatic numbers. It is our belief that these alternate viewpoints can benefit from each other. The refuted conjecture in Section 5.5.1 is an immediate example of this. As a natural next step from Proposition 5.49, we would like to have some result for a general graph $H$. A trivial upper bound of $s(H, H^k) \leq k/n$ is obtained from Proposition 5.46, but we have not identified anything corresponding to the parity criterion which appears in the case $H = K_2$. This leads us to believe that this bound can be improved upon. The approach of Šámal on the lower bound does not seem to generalise. The reason for this is that it uses bounds on maximal cuts obtained from the Laplacian of (a subgraph of) $Q_n/k$. We know of no such results for maximal $k$-cuts, with $k > 2$, much less for general $H$-cuts.
Part III

Computational Complexity
Chapter 6

Constant Production

Remember that $C_A$ is the set of all unary constant relations over the domain $A$. For the decision problem $Csp(\Gamma)$, with a finite constraint language $\Gamma$, it is known that if all unary polymorphisms of $\Gamma$ are permutations, then $Csp(\Gamma)$ is in $P$ (NP-hard) if and only if $Csp(\Gamma \cup C_A)$ is in $P$ (NP-hard) [22]. The precondition can be reformulated into requiring that $\Gamma$ is a core. Since the computational complexity of $Csp$, with respect to $\Gamma$, is directly determined by that of the core of $\Gamma$, this allows for the incorporation of all unary constant relations when studying $Csp$. For $\text{Max Sol}(\Gamma)$, taking the core of a constraint language may change its computational complexity, if instead, we consider max-cores, then we cannot always introduce all constant unary relations without changing the computational complexity. In this chapter, we derive a property which is a slight restriction of the max-core condition, so as to allow the addition of constants. We also show that when constants cannot be added in this way, we can derive the computational complexity of $\text{Max Sol}(\Gamma)$ from that of its strictly smaller endomorphic images, i.e. from the constraint languages $g(\Gamma)$ obtained by applying non-surjective unary polymorphisms $g$. The problem $\text{Max AW Sol}(\Gamma)$ is defined similarly to $\text{Max Sol}(\Gamma)$ with the difference that arbitrary rational weights are allowed in the input instances. While the results of this chapter are stated for $\text{Max Sol}$, our method also applies to $\text{Max AW Sol}$, with minor modifications. (In particular, the proof of Lemma 6.2 must be altered slightly to accommodate for the possibility of negative weights.)

We begin by illustrating one of the key ideas with an example. For a constraint language (or relational structure) $\Gamma$, we let $\text{End}(\Gamma)$ denote the set of all endomorphisms on $\Gamma$, i.e. the set of homomorphisms from $\Gamma$ to itself, and we let $\text{Aut}(\Gamma)$ denote the set of automorphisms (that is, surjective endomorphisms) on $\Gamma$. Let $H$ be the graph in Figure 6.1. $H$ is isomorphic to $C_4$ and it is easy to verify that it is a max-core. Nevertheless, we will see that the complexity of $\text{Max Sol}(H)$ can be naturally derived from that of the strictly smaller endomorphic images of $H$. Let $(G, \omega)$ be an instance
of Max Sol$(H)$ and let $f : V(G) \rightarrow V(H)$ be an optimal solution to this
instance. For $i = 1, \ldots, 8$, let $\mu_i = \sum_{v \in f^{-1}(i)} \omega(v)$ be the total weight of
those nodes in $G$ which are mapped to $i$. Pick a non-surjective endomor-
phism $\pi \in \text{End}(H)$. Then, $\pi(H)$ is an irreflexive path, so Max Sol$(\pi(H))$
can be solved in polynomial time (cf. Proposition 6.27.) Now, assume that
\[
\sum_{i=1}^{8} \mu_i \cdot i = \sum_{i=1}^{8} \mu_i \cdot \pi(i). \tag{6.1}
\]
We could then have solved $(G, \omega)$ on $\pi(H)$ instead, and obtained the same
optimal measure. But of course, we cannot a priori know what the values
of $\mu_i$ will be. It is even possible that some other optimal solution exists, for
which (6.1) does not hold. However, since $H$ is fixed, $\text{End}(H)$ is of constant
size, and we can try to solve Max Sol$(\pi(H))$ for every non-surjective $\pi \in
\text{End}(H)$. Since every solution to Max Sol$(\pi(H))$ in particular is a solution
to Max Sol$(H)$, we can choose the maximal one in hope that it will be
maximal also for Max Sol$(H)$. There could exist instances for which this
approach fails if there were constants $\mu_1, \ldots, \mu_8$ such that
\[
\sum_{i=1}^{8} \mu_i \cdot i > \sum_{i=1}^{8} \mu_i \cdot \pi(i) \tag{6.2}
\]
for every $\pi \in \text{End}(H) \setminus \text{Aut}(H)$. If this was the case, then we will see
that it would be possible to produce all constant unary relations, and that
this in turn would imply NP-hardness for Max Sol$(H)$. Conversely, the
approach will succeed, i.e. it will produce a polynomial-time algorithm for
Max Sol$(H)$ if, for every choice of constants $\mu_1, \ldots, \mu_8$, there is at least
one $\pi \in \text{End}(H) \setminus \text{Aut}(H)$ such that (6.2) does not hold. We will see in
Proposition 6.34 that for our example graph $H$, the latter is the case, so
that Max Sol$(H)$ is in PO.

Figure 6.1: The graph $H$ is an 8-cycle and a max-core.
Chapter outline

The general structure of this chapter is as follows. In Section 6.1 we start by introducing a number of concepts which will be used in the proof of the main result. In particular, in Section 6.1.1 we extend the notion of pp-definability to weighted pp-definability, in order to be able to produce constants in certain situations. Section 6.1.2 introduces some notation which simplifies the manipulation of expressions for the measure of solutions to Max Sol instances, and Section 6.1.3 extends the idea of connectedness to relational structures. Section 6.2 is dedicated to the proofs of Lemma 6.20 and Lemma 6.21, which help establish the main theorem, Theorem 6.22. Some of the tools developed are then used, in Section 6.3, in order to classify the computational complexity of Max Sol$(H)$, for some simple classes of graphs $H$. The chapter ends with a discussion, in Section 6.4, on possible future extensions of the results.

6.1 Preliminaries

In this chapter we will mainly use the viewpoint of constraint satisfaction as a homomorphism problem between relational structures. We will assume that all relational structures are finite, i.e. the signature $\tau$ is assumed to be finite. However, we will occasionally use the term constraint language in order to emphasise that we are speaking of the fixed relational structure on the right-hand side. In the same spirit, we will often keep the calligraphic typeface, $\mathcal{I}, \mathcal{J}$, for the left-hand side instances, as opposed to the capital Greek letters, $\Gamma, \Delta$, used for constraint languages and relational structures.

The weight function $\omega$ of an instance $\mathcal{I}$, can be viewed as a valuation of the relational structure $\mathcal{I}$. We will call both $(\mathcal{I}, \omega)$ and $(\Gamma, \nu)$ valued relational structures. The problem Max Sol$(\Gamma, \nu)$ is then a homomorphism problem between valued relational structures, with preference given to solutions of high measure, with respect to the valuations.

Formally, let $\Gamma$ be a $\tau$-structure with valuation $\nu : A \to \mathbb{Q}_{\geq 0}$. The definition of Max Sol$(\Gamma)$, or Max Sol$(\Gamma, \nu)$ then takes the following form:

**Instance:** A valued relational structure $(\mathcal{I}, \omega)$, where $\mathcal{I}$ is a $\tau$-structure over a domain $V$, and $\omega : V \to \mathbb{Q}_{\geq 0}$.

**Solution:** A homomorphism $f$ from $\mathcal{I}$ to $\Gamma$.

**Measure:** The measure of a solution $f$ is given by the following sum:

$$m(f) = \sum_{v \in V} \omega(v) \cdot \nu(f(v)).$$

(6.3)

To keep the notation less cumbersome, we will make some simplifications. An instance $\mathcal{I}$ of Max Sol$(\Gamma)$ must have the same signature as $\Gamma$. 

in order for the problem to be defined. For this reason, we will usually not mention the signatures, but simply assume that they exist, and agree for the relational structures we discuss. Similarly, the relational structures which we consider will always come equipped with valuations. Sometimes these valuations do not change over the course of an argument, or may simply be assumed to exist, but not necessary to mention explicitly. In such cases, we will often write $\Gamma$ for $(\Gamma, \nu)$, $I$ for $(I, \omega)$, and Max Sol($\Gamma$) for Max Sol($\Gamma, \nu$), respectively. When nothing else is stated, $A$ will denote the domain of $\Gamma$.

### 6.1.1 Weighted pp-definability

Let $I$ be an instance of Max Sol($\Gamma$) over a domain $A$, and with variables $V = \{v_1, \ldots, v_n\}$. A solution $f$ to $I$ can be represented as the $n$-tuple $(f(v_1), \ldots, f(v_n)) \in A^n$. With this in mind, we define Opt Sol($I$) as the set of all $n$-tuples which represent optimal solutions to $I$, i.e. Opt Sol($I$) is a subset of $A^n$. For any such subset $S = \{(a_1^i, \ldots, a_n^i)\}_{1 \leq i \leq m} \subseteq A^n$, and a sequence of indices $(i_1, \ldots, i_k) \subseteq [n]^k$, we will let $\pi_{i_1, \ldots, i_k} S$ denote the set $\{(a_1^{i_1}, \ldots, a_n^{i_k})\}_{1 \leq j \leq m}$. We then have the following extension of pp-definability (cf. Section 2.3):

**Definition 6.1** A $k$-ary relation $g$ is weighted pp-definable over $\Gamma$ if there is an instance $I$ of Max Sol($\Gamma$) with optimal solutions Opt Sol($I$) and $g \equiv \pi_{i_1, \ldots, i_k} \text{Opt Sol}(I)$. The set of all relations which are weighted pp-definable from $\Gamma$ is denoted by $(\Gamma)_w$.

**Lemma 6.2** Let $\Gamma$ be a constraint language and $\Gamma' \subseteq (\Gamma)_w$ a finite subset. Then, Max Sol($\Gamma'$) is polynomial-time reducible to Max Sol($\Gamma$).

**Proof:** Let $I' = (V', A, C', \omega')$ be an instance of Max Sol($\Gamma'$). First, we will assume that there is only one constraint application $((u_1, \ldots, u_k), g)$ in $I'$ such that $g \in \Gamma'$, but $g \not\in \Gamma$. We modify $I'$ to an instance $I = (V, A, C, \omega)$ of Max Sol($\Gamma$) as follows: let $J_0$ be an instance of Max Sol($\Gamma$) with variables $V(J_0)$ disjoint from $V'$, and such that $g \equiv \pi_{i_1, \ldots, i_k} \text{Opt Sol}(J_0)$. Now substitute the variable $v_i$ by $u_j$, for $1 \leq j \leq k$, to obtain an instance $J_0'$ which shares some variables with $I'$. Let

$$V = V' \cup V(J_0') \cup C = (C' \setminus \{(u_1, \ldots, u_k), g\}) \cup C(J_0'),$$

and define

$$\omega(v) = \begin{cases} 
\omega'(v) & \text{if } v \in V \setminus V(J_0'), \\
M \cdot \mu(v) + \omega'(v) & \text{if } v \in V \cap V(J_0'), \text{ and} \\
M \cdot \mu(v) & \text{if } v \in V(J_0') \setminus V,
\end{cases}$$

where $\mu$ is the weight function of $J'_0$, and $M$ is a constant.
The idea is now to choose $M$ large enough, so that if $T'$ is satisfiable, then in any optimal solution $s$ to $I$, the restriction of $s$ to the set $V(J'_v)$ is forced to be an optimal solution to the instance $J'_v$. It then follows that we will have $(s(u_1), \ldots, s(u_k)) \in \rho$, and that we can recover an optimal solution to $T'$ from $s$. This is achieved as follows: if $J'_v$ does not have any sub-optimal solutions, we can let $M = 0$. Otherwise, let $\Delta$ be the minimal difference in measure between an optimal solution, and a sub-optimal solution to $J'_v$, and let $M = (U + 1)/\Delta$, where $U = \sum_{v \in V(T')} \omega(v) \cdot \max_{a \in A} \nu(a)$. Note that $U$ is a bound, not only on the measure of an optimal solution to $T'$, but on the optimum of any instance over $A$ with valuation $\nu$ which has the same set of variables and weights as $I'$. In particular, we will use that it is a bound on the instance obtained from $I'$ by removing the application of $\rho$.

Next, we show how to obtain an optimal solution to $T'$ by solving $I$. Assume first that $T'$ is satisfiable, and let $s'$ be any satisfying assignment to $T'$. Then, we can define a satisfying assignment $s$ to $I$ by letting $s(v) = s'(v)$ if $v \in V$, and otherwise choose $s$ so that its restriction to $V(J'_v)$ is an optimal solution to $J'_v$. The measure of this solution will be $m(s) = M \cdot \text{Opt}(J'_v) + m(s')$, hence

$$\text{Opt}(I) \geq M \cdot \text{Opt}(J'_v) + \text{Opt}(I').$$

The construction of $s$ also shows that if $I$ is unsatisfiable, then so is $I'$. So let us assume that $I$ is satisfiable, and let $s$ be any solution to $I$. If the restriction of $s$ to $V(J'_v)$ is an optimal solution to $J'_v$, then we have $(s(u_1), \ldots, s(u_k)) \in \rho$, so the restriction $s|_{V}$ is a solution to $T'$. The measure of $s|_{V}$ is $m(s) - M \cdot \text{Opt}(J'_v)$, so

$$\text{Opt}(I') \geq \text{Opt}(I) - M \cdot \text{Opt}(J'_v),$$

and we conclude that $s|_{V}$ is an optimal solution to $T'$.

On the other hand, if the restriction of $s$ to $V(J'_v)$ is not an optimal solution to $J'_v$, then $\Delta$ must be defined, and we have

$$m(s) \leq M \cdot (\text{Opt}(J'_v) - \Delta) + U = M \cdot \text{Opt}(J'_v) - (U + 1) + U.$$
The weight of a variable \( v \in V \) which is in the scope of a set of applications, \( (s_1, \theta_1), \ldots, (s_r, \theta_r) \), will be \( \omega(v) = \omega'(v) + M \cdot \sum_{i=1}^{r} \mu_i(v) \).

We must also argue that the size of \( \mathcal{I} \) is polynomial in the size of \( \mathcal{I}' \). The number of new variables and constraint applications introduced into \( \mathcal{I}' \) depends on the sizes of the instances \( \{ J_{\theta} \}_{\theta \in \Gamma'} \), and on the number of applications in \( \mathcal{I}' \). The latter is clearly at most linear in the size of \( \mathcal{I}' \), and since \( \Gamma' \) is finite, the set \( \{ J_{\theta} \}_{\theta \in \Gamma'} \) can be computed and fixed ahead of time, so the sizes of these instances are constant with respect to \( \mathcal{I}' \). Finally, the constant \( M \) will depend on the sum of all weights in \( \mathcal{I}' \), hence at most linearly on the size of \( \mathcal{I}' \).

**Example 6.3** To illustrate the use of Lemma 6.2, we show \( \text{NP} \)-hardness of \( \text{Max Sol}(H) \), where \( H \) is the graph in Figure 6.2a on the vertex set \( A = \{0, 1, 2\} \).

![Figure 6.2: The graph \( H \), and the instance \( J_{\theta} \) in Example 6.3.](image)

Let \( \varrho \) denote the constraint \( \{0, 1\}^2 \setminus \{(1, 1)\} \), and let \( \mathcal{I}_\varrho = (V, A, C, \omega) \) be the instance in Figure 6.2b, with \( V = \{v_1, v_2, v_3\} \), the constraint applications \( H(v_i, v_j) \) for \( i \neq j \), and weights given by \( \omega(v_1) = 1 \), and \( \omega(v_2) = \omega(v_3) = 0 \). For this instance, \( \text{OptSol}(\mathcal{I}_\varrho) = \{(2, 0, 0), (2, 0, 1), (2, 1, 0)\} \), i.e. an optimal solution maps \( v_1 \) to 2, and \( (v_2, v_3) \) to the set \( \{0, 1\}^2 \setminus \{(1, 1)\} \). Hence, \( \varrho \equiv \pi_{2,3} \text{OptSol}(\mathcal{I}_\varrho) \) is in \( \{H\}_\omega \). By Lemma 6.2, with \( \Gamma = \{H\} \) and \( \Gamma' = \{\varrho\} \), we have a polynomial-time reduction from \( \text{Max Sol}(\varrho) \) to \( \text{Max Sol}(H) \). Since the problem \( \text{Max Independent Set} \) is polynomial-time reducible to \( \text{Max Sol}(\varrho) \), it follows that both \( \text{Max Sol}(\varrho) \) and \( \text{Max Sol}(H) \) are \( \text{NP} \)-hard.

Next, we sketch how the polynomial-time reduction works in this case. We assume that the instance of \( \text{Max Sol}(\varrho) \) is an undirected graph \( G \), i.e. each edge \( \{u_i, u_j\} \in G \) represents two applications, \( g(u_i, u_j) \) and \( g(u_j, u_i) \), of the constraint \( \varrho \). Since \( \varrho \) is symmetric, we only need to consider one of these applications. An example is illustrated in Figure 6.3. The graph \( G \) is drawn to the left, with white vertices. For each edge of \( G \), we add a copy of the instance \( \mathcal{I}_\varrho \), and multiply the weights of this instance by a large constant \( M \). Each such instance is a triangle on some vertices \( v_1, v_2, \) and \( v_3 \), and we identify (indicated by dotted lines in the figure) \( v_2 \) with \( u_i \) and \( v_3 \) with \( u_j \), and add the weight of \( v_2 \) (\( v_3 \)) to that of \( u_i \) (\( u_j \)). In this case, the weights of \( v_2 \) and \( v_3 \) are both 0, so the total weight of \( u_i \) and \( u_j \) are given by their original weights, while the weight of the (new) variable...
6. Constant Production

\( v_1 \) is \( M \). The result of this procedure is shown to the right in Figure 6.3, where we also give the total weights, with \( \omega_i = \omega(u_i) \). To determine \( M \), we note that \( \Delta \), i.e. the least difference between an optimal solution to \( J_\omega \), and any sub-optimal solution, is 1. Hence, we can let \( M = U + \frac{1}{U} \), where \( U = (\omega_1 + \omega_2 + \omega_3 + \omega_4) \cdot 2 \). In any optimal solution, we have now forced \( f(v_1) = 2 \) for all copies of \( v_1 \) so that for any pair of adjacent vertices \( u_i \) and \( u_j \), we have \( (f(u_i), f(u_j)) \in \varrho \), as intended. Thus, any optimal solution has a measure equal to \( 8M \) plus the maximal weight of an independent set in \( G \). Note that in this particular example we could have identified all the copies of \( v_1 \) as well.

Figure 6.3: An instance \( G \) of \( \text{Max Sol}(\varrho) \) and four copies of \( J_\varrho \) (left), and the corresponding instance of \( \text{Max Sol}(H) \) with weights (right.)

6.1.2 The measure \( m(f) \) as a bilinear form

In order to work more easily with expressions such as (6.2) and (6.3), it will be convenient to introduce a bit of notation for the measure. We start in a more general setting of functionals on vector spaces over \( \mathbb{Q} \) and linear maps between the dual spaces. This is eventually reduced to functions from finite sets to \( \mathbb{Q} \), and functions between these sets. While the generality first introduced is by no means necessary for our application, it provides some motivation for the notation used. The essential definitions are found in (6.6) and (6.7) and the identity (6.8) shows two ways of representing the measure of a solution.

Let \( U \) and \( W \) be two finite-dimensional vector spaces over \( \mathbb{Q} \). Denote by \( U^* \) (\( W^* \)) the dual space of \( U \) (\( W \)), consisting of all linear maps from \( U \) (\( W \)) to \( \mathbb{Q} \). Define \( \langle \cdot, \cdot \rangle_U : U^* \times U^* \rightarrow \mathbb{Q} \) to be the following symmetric, bilinear form:

\[
\langle \alpha, \beta \rangle_U = \sum_{i=1}^{\dim U} \alpha(e_i) \cdot \beta(e_i),
\]

(6.4)
where \( \{e_i\} \) is an orthonormal base for \( U \) and \( \alpha, \beta \in U^* \).

Now, let \( \gamma \in W^* \) and let \( F : W^* \to U^* \) be a linear map. Then, the transpose of \( F \), denoted \( F^t : U^* \to W^* \), can be defined as the unique linear map which satisfies
\[
\langle \alpha, F(\gamma) \rangle_U = \langle F^t(\alpha), \gamma \rangle_W.
\] (6.5)

As the notation suggests, \( F^t \) can also be defined as the map whose matrix representation is the transpose of the matrix representation for \( F \).

We now restrict this idea to our current setting. Let \( A \) and \( V \) be two finite domains. If we identify the elements of \( A \) with the basis \( \{e_i\} \), then maps in \( U^* \) can be identified with maps from \( A \) to \( Q \). We then have a definition of \( \langle \cdot, \cdot \rangle_A \) from (6.4):
\[
\langle \mu, \nu \rangle_A = \sum_{a \in A} \mu(a) \cdot \nu(a),
\] (6.6)
where \( \mu, \nu : A \to Q \). Now, we can let a function \( f : V \to A \) act on \( \nu \) by \( f(\nu) = \nu \circ f \). Thus, \( f \) maps functions from \( A \) to \( Q \) into functions from \( V \) to \( Q \). The equality (6.5) now corresponds to
\[
\langle \omega, f(\nu) \rangle_V = \langle f^t(\omega), \nu \rangle_A,
\]
where \( f^t \) is given by
\[
f^t(\omega)(a) = \sum_{v \in f^{-1}(a)} \omega(v), \quad \text{for all } a \in A.
\] (6.7)

Now we take the final step, and replace \( Q \) by \( Q_{\geq 0} \). We then move away from the setting of vector spaces, as we can no longer subtract vectors (weight functions) freely, but this will never actually pose any problem. Let \( (\Gamma, \nu) \) be a valued constraint language and \( (\mathcal{I}, \omega) \) an instance of \( \text{MAX SOL}(\Gamma, \nu) \), with \( A \) being the domain of \( \Gamma \) and \( V \) the domain of \( \mathcal{I} \). If \( f \) is a homomorphism from \( \mathcal{I} \) to \( \Gamma \), then by combining (6.3) and (6.6) we obtain two expressions for the measure of \( f \), namely
\[
m(f) = \langle \omega, f(\nu) \rangle = \langle f^t(\omega), \nu \rangle.
\] (6.8)

From here on, we will drop the subscripts \( A \) and \( V \) on \( \langle \cdot, \cdot \rangle \), as they will be evident from the context. We can expand the last expression in (6.8) to see that what we have done is essentially a rearrangement of terms in the sum representing the measure:
\[
\langle f^t(\omega), \nu \rangle = \sum_{a \in A} f^t(\omega)(a) \cdot \nu(a) = \sum_{a \in A} \sum_{v \in f^{-1}(a)} \omega(v) \cdot \nu(f(v)) = \sum_{v \in V} \omega(v) f(\nu(v)).
\]

We mention an additional manipulation of \( \langle \mu, \nu \rangle \), which is obtained by applying an arbitrary permutation \( \pi : A \to A \) to both arguments. Since
this corresponds to a change in the summation order, we get the following identity:

$$ \langle \pi(\mu), \pi(\nu) \rangle = \langle \mu, \nu \rangle. $$

(6.9)

Example 6.4 Recall the example in Figure 6.1. In order to introduce a valuation on the vertices, instead of the explicit numbers 1 to 8, we start by relabelling the graph. Let $A = \{a_1, \ldots, a_8\}$ and let $\Gamma = \{H\}$ be a relational structure over $A$, where $H$ is the binary, symmetric relation given by the graph in Figure 6.4. Let the valuation of $\nu$ be given by the integers on the corresponding vertices in Figure 6.1: $\nu = \{a_1 \mapsto 8, a_2 \mapsto 1, \ldots, a_8 \mapsto 3\}$. Let $(I, \omega)$ be an instance, where $I = \{G\}$ and $G$ is a graph on vertex set $V$.

![Graph](attachment:image.png)

Figure 6.4: A relabelling of the graph from Figure 6.1.

A solution to MAX Sot.$(\Gamma, \nu)$ for this instance is a (graph) homomorphism $f : V \to A$, and by (6.8) we can write $m(f) = \langle \omega, f(\nu) \rangle$. Let $\mu(a_i) = \sum_{v \in f^{-1}(a_i)} \omega(v)$ for $i = 1, \ldots, 8$, or equivalently, let $\mu = f^*(\omega)$. Then, (6.2) takes the form

$$ \langle \mu, \nu \rangle > \langle \mu, \pi(\nu) \rangle, $$

(6.10)

where $\pi$ is some unary polymorphism of $H$. Using the identity (6.8), we see that the right-hand side of (6.10) is equal to $\langle \pi^* (\mu), \nu \rangle$. This value can be interpreted as the measure, $m(id)$, of the identity solution to the instance $(\Gamma, \pi^* (\mu))$. This idea is used in the proof of Lemma 6.16.

Let $id$ be the identity operation on $A$. In Section 6.2, we will want to determine when $C_A$ is a subset of $(\Gamma)_w$. For this, we use the following result, which is easily derived from Lemma 6.2.

Lemma 6.5 Let $\Gamma$ be a finite constant language with valuation $\nu$, and assume that there exists a $\mu : A \to \mathbb{Q}_{\geq 0}$ such that for all $g \in \text{End}(\Gamma) \setminus \{id\}$,

$$ \langle \mu, \nu \rangle > \langle \mu, g(\nu) \rangle. $$

Then, $C_A \subseteq (\Gamma)_w$. 


Proof: Consider $(\Gamma, \mu)$ as an instance of $\text{MAX SOL}(\Gamma, \nu)$. The condition (6.11) shows that $id$ is the unique maximal solution to this instance, i.e. $\text{OPT SOL}(\Gamma, \mu) = \{(a_1, a_2, \ldots, a_n)\}$. Consequently, for each $i$, $1 \leq i \leq n$, we have $\{(a_i)\} \equiv \pi_i \text{OPT SOL}(\Gamma, \mu) \in (\Gamma)_{w_i}$. Thus, $C_A \subseteq (\Gamma)_{w_i}$.

6.1.3 Connected substructures

When determining properties, or parameters, of undirected graphs, it is often possible, and convenient, to first decompose the graph under study into its connected components. These components can then be dealt with separately, and the results for the separate components are finally aggregated to produce the answer to the question for the original graph.

Examples include the chromatic number, $\chi$, of a graph, the existence of graph homomorphisms, and, $\alpha$, the maximal size of an independent set. In the case of the chromatic number, one has the identity:

$$\chi(G \sqcup H) = \max\{\chi(G), \chi(H)\},$$

(6.12)

where $\sqcup$ denotes disjoint union. In general, thus, the chromatic number is the maximum of the chromatic number of its connected components. For the other examples, $G \rightarrow H$ if and only if $G_i \rightarrow H$ for all $i$, where $G_i$ are the connected components of $G$, and finally, $\alpha(G) = \sum \alpha(G_i)$.

The notion of connectedness and that of a disjoint union are related in the sense that an undirected graph can be written as a disjoint union of its connected components. This decomposition is unique, up to reordering of the components. If $G$ and $H$ have partially overlapping vertex sets, we can introduce an isomorphic copy, $G'$, of $G$, for which we let $V(G') = V(G) \times \{1\}$, and similarly, for $H$, we can introduce a copy $H'$, with $V(H') = V(H) \times \{2\}$. Finally, we let $G \sqcup H = G' \cup H'$.

Here, we define a notion of connectedness for relational structures. We want to retain two properties from the ordinary undirected graph connectedness. Namely,

1. Every structure can be decomposed into a disjoint union of connected components, and an optimal solution to an instance can be obtained by composing optimal solutions to the separate components.
2. The homomorphic images of a connected structure are connected.

Let $\Gamma$ be a relational structure over a finite set $A$. Define the relation $C_\Gamma \subseteq A \times A$ by letting $(a, b), (b, a) \in C_\Gamma$ if there is a tuple $(\ldots, a, \ldots, b, \ldots) \in \rho$ for some relation $\rho \in \Gamma$. The relation $C_\Gamma$ is turned into an equivalence relation by adding $(a, a)$ for each $a \in A$, and taking the transitive closure.

Definition 6.6 Let $A' \subseteq A$ and let $\Delta = \Gamma|_{A'}$ be the relational structure induced by $A'$. If $A'$ is an equivalence class of $C_\Gamma$, then we say that $\Delta$ is a connected component of $\Gamma$. 


Example 6.7 Let \( D \) be a directed graph. In graph theory, one distinguishes two notions of connectedness for directed graph. A subgraph \( D' \) of \( D \) is called strongly connected, if for every ordered pair of vertices \( u, v \in D' \), there is a directed path from \( u \) to \( v \). Define the underlying graph of \( D \) as the undirected graph \( G \), with \( V(G) = V(D) \) and \( \{ u, v \} \in E(G) \) if either \( (u, v) \) or \( (v, u) \) (or both) is a directed edge in \( D \). A subgraph of \( D \) is called weakly connected if the underlying graph is connected (in the usual, undirected, sense). The connectedness of Definition 6.6 is equivalent to weak connectedness for directed graphs.

Let \( \Gamma \) be a relational structure over \( A \). Up to reordering, there is a unique partition of \( A \) into sets \( A_1, \ldots, A_k \) (i.e., \( A = \bigcup_{i=1}^{k} A_i \), and \( A_i \cap A_j = \emptyset \) for \( i \neq j \)) given by the equivalence classes of \( C_{\Gamma} \), such that \( \{ \Gamma|_{A_i} \}_{i=1}^{k} \) are the connected components of \( \Gamma \). In addition to decomposing structures into their connected components, we need to build larger structures from their individual components. For this purpose, we introduce a disjoint union of relational structures, in the natural way.

Definition 6.8 Let \( I \) be an index set and \( \{ A_i \}_{i \in I} \) be a collection of relational structures of a common signature \( \tau \) over sets \( A_i \). The disjoint union of \( \{ A_i \}_{i \in I} \), denoted \( \bigsqcup_{i \in I} A_i \), is a relational structure of signature \( \tau \) over the set \( \bigcup_{i \in I} (A_i \times \{ i \}) \). For \( j \in I \), let \( \iota_j : A_j \to A_j \times \{ j \} \) be the injection \( \iota_j(a) = (a, j) \), extend it to tuples by \( \iota_j(t) = (\iota_j(t_1), \ldots, \iota_j(t_l)) = (\iota_j(t_1), \ldots, \iota_j(t_l)) \), and to relations by \( \iota_j(r) = \bigcup_{i \in I} \iota_j(r_i) \). Each relation of \( \bigsqcup_{i \in I} A_i \) is given by \( \bigcup_{i \in I} \iota_i(\varrho') \), where \( \varrho' \in A_i \).

The following two propositions verify that connectedness according to Definition 6.6 has the properties we desire; we want to solve instances by decomposing them, and then solve each connected component separately. Additionally, we want the homomorphic images of connected structures to be connected. For convenience, if \( I \) does not have any solutions, then we define \( \text{Opt}(I) = -\infty \).

Proposition 6.9 Let \( (I, \mu) \) be an instance of Max Sol(\( \Gamma \)), for some constant language \( \Gamma \). Then, \( \text{Opt}(I) = \sum_{1 \leq i \leq k} \text{Opt}(I_i) \), where \( I_1, \ldots, I_k \) are the connected components of \( I \). Furthermore, the decomposition \( \{ I_i \}_{i=1}^{k} \) can be found in time which is polynomial in the number of tuples of \( I \).

Proof: For the first part, it suffices to note that if \( (u, v) \notin C_I \), then \( u \) and \( v \) can be assigned values independently of each other, i.e., no assignment to \( u \) can affect the possible assignments to \( v \), and vice versa. More generally, let \( A_i \) be the domain of \( I_i \). Then, \( A_i \times A_j \cap C_I = \emptyset \) for \( i \neq j \), so \( A_i \) and \( A_j \) can be assigned values independently. Hence, the optimal solution to \( I \) is given by the sum, over \( i \), of the optimal solutions to \( I_i \).

For the second part, we assume that \( \Gamma \) is represented by a list of relations, each consisting of a list of the tuples in that relation. To compute the
equivalence classes \( \{ A_i \}_{i=1}^k \) of \( C_\Gamma \), we can use a union-find data structure. We perform a first pass over all tuples and merge all classes corresponding to the domain elements in each given tuple. Using the equivalence classes, it is then straightforward to extract each connected component from \( \Gamma \) in a second pass.

**Proposition 6.10** Let \( \Gamma \) be a connected constraint language. Then, any homomorphic image of \( \Gamma \) is connected.

**Proof:** Let \( f : A \to B \) be a surjective homomorphism from \( \Gamma \) to \( f(\Gamma) \) over domain \( B \). For a connected \( \Gamma \) over \( A \), we have \( C_\Gamma = A \times A \). This implies \( f(C_\Gamma) = B \times B \supseteq C_{f(\Gamma)} \). Hence, it suffices to show that we have \( f(C_\Gamma) \subseteq C_{f(\Gamma)} \).

To this end, pick a tuple \((b, b') \in f(C_\Gamma)\), and some preimage \((a, a') \in C_\Gamma\). A path in \( \Gamma \) from \( a_1 \) to \( a_k \) in \( \Gamma \), is a (finite) sequence of tuples \( t_1, \ldots, t_{k-1} \) from relations in \( \Gamma \) such that \( a_1 \) is an element in \( t_1 \), \( a_k \) is an element in \( t_{k-1} \), and for \( i = 2, \ldots, k-1 \), \( a_i \) is an element both in \( t_{i-1} \) and in \( t_i \). Since \((a, a') \in C_\Gamma\), such a path must exist between \( a = a_1 \) and \( a' = a_k \). The endomorphism \( f \) maps this path to a path in \( f(\Gamma) \) from \( f(a) = b \) to \( f(a') = b' \), consisting of the tuples \( f(t_1), \ldots, f(t_{k-1}) \). Consequently, \((b, b') \in C_{f(\Gamma)}\), which finishes the proof.

One of the main reasons for introducing connectedness of relational structures, apart from decomposing instances into connected components, is the following immediate consequence of Proposition 6.10.

**Corollary 6.11** Let \( \Gamma, \Gamma_1, \ldots, \Gamma_k \) be connected and let \( h : \Gamma \to \prod_{i=1}^k \Gamma_i \) be a homomorphism. Then, \( h(\Gamma) \cap \Gamma_i(\Gamma_i) = \emptyset \) for all but one \( i \in \{1, \ldots, k\} \).

**Remark 6.12** Similarly to weak connectedness for directed graphs discussed in Example 6.7, we can give an alternative definition of a connected substructure, equivalent to Definition 6.6, by introducing an *underlying graph* of an arbitrary relational structure. For a single relation, this graph is obtained by adding a clique for each tuple, connecting all domain elements occurring in that tuple. The graph for the relational structure is then taken as the union of the graphs of the relations appearing in the structure. Then, a substructure is connected (in sense of Definition 6.6) if and only if the underlying graph is connected.

**Remark 6.13** A third, and equivalent, definition of connectedness for relational structures can be obtained from Definition 6.8. Namely, a substructure is connected (in the sense of Definition 6.6) if and only if it cannot be written as a disjoint union of two non-empty structures.

In the following, we will use the same relational structure \( I \) as an instance of the problem \( \text{MAX SOL} (\Gamma) \) for various constraint languages \( \Gamma \). To make the intended problem explicit in the notation, we will denote the optimum of \( I \) by \( \text{Opt}(I, \Gamma) \).
6. Constant Production

**Definition 6.14** Let \( \{ \Gamma_1, \ldots, \Gamma_k \} \) be a non-empty collection of connected constraint languages. Let \( \mathcal{I} \) be a relational structure such that \( \text{Opt}(\mathcal{I}, \Gamma_i) > \text{Opt}(\mathcal{I}, \Gamma_j) \) for all \( j \neq i \). We then say that \( \Gamma_i \) is **essential** in \( \prod_{i=1}^{k} \Gamma_i \), and we call \( \mathcal{I} \) a **witness**.

In the special case that the collection in Definition 6.14 contains only a single constraint language \( \Gamma \), then \( \Gamma \) is always essential, and every instance \( \mathcal{I} \) of \( \text{MAX SOL}(\Gamma) \) which has any solution is a witness. The following proposition provides a first reason for introducing the notion of essentiality:

**Proposition 6.15** Let \( \{ \Gamma_1, \ldots, \Gamma_k \} \) be a collection of connected constraint languages, and assume that for some \( i, \Gamma_i \) is not essential in \( \prod_{j \neq i} \Gamma_j \). Then, \( \text{MAX SOL}(\prod_{j \neq i} \Gamma_j) \) is polynomial-time equivalent to \( \text{MAX SOL}(\prod_{j \neq i} \Gamma_j) \).

**Proof:** Let \( \mathcal{I} \) be a relational structure. By Proposition 6.9 we may assume that \( \mathcal{I} \) is connected which, by Corollary 6.11, implies that \( \text{Opt}(\mathcal{I}, \prod_{j \neq i} \Gamma_j) = \max_{j \neq i} \{ \text{Opt}(\mathcal{I}, \Gamma_j) \} \). Furthermore, \( \max_{j \neq i} \{ \text{Opt}(\mathcal{I}, \Gamma_j) \} = \max_{j \neq i} \{ \text{Opt}(\mathcal{I}, \Gamma_j) \} \), since \( \Gamma_i \) is not essential, i.e. \( \text{Opt}(\mathcal{I}, \Gamma_i) \leq \text{Opt}(\mathcal{I}, \Gamma_j) \) for all \( j \neq i \). Finally, with \( \mathcal{I} \) connected, and again due to Corollary 6.11, \( \max_{j \neq i} \{ \text{Opt}(\mathcal{I}, \Gamma_j) \} = \text{Opt}(\mathcal{I}, \prod_{j \neq i} \Gamma_j) \).

Proposition 6.15 shows that we can discard non-essential structures. We would also like to say something about essential structures, namely that we can reduce from them. This is proved in Proposition 6.17(2), but before we do so, we need an additional lemma, which shows that witnesses can be assumed to have a particular form.

**Lemma 6.16** Let \( \{ \Gamma_1, \ldots, \Gamma_k \} \) be a collection of finite, connected constraint languages, and assume that for some \( i, \Gamma_i \) is essential in \( \prod_{j \neq i} \Gamma_j \). Then, there exists a witness of the form \((\Gamma_i, \mu)\), \( \mu : A_i \to \mathbb{Q}_{\geq 0} \), where \( A_i \) is the domain of \( \Gamma_i \). Furthermore, the identity homomorphism on \( \Gamma_i \) is an optimal solution to this witness.

**Proof:** Since \( \Gamma_i \) is essential, we can find some witness \( \mathcal{I} \) to this fact. Let \( f \) be an optimal solution to \( \mathcal{I} \) as an instance of \( \text{MAX SOL}(\Gamma_i) \), and consider the instance \( \mathcal{J} = (\Gamma_i, f^*(\omega)) \). The identity homomorphism on \( \Gamma_i \) is a solution to \( \mathcal{J} \) of measure \( (f^*(\omega), \nu_i) = (\omega, f(\nu_i)) = \text{Opt}(\mathcal{I}, \Gamma_i) \). Hence, \( \text{Opt}(\mathcal{J}, \Gamma_i) \geq \text{Opt}(\mathcal{I}, \Gamma_i) \). Now, let \( g : \Gamma_i \to \Gamma_j \) be an optimal solution to \( \mathcal{J} \) as an instance of \( \text{MAX SOL}(\Gamma_j) \), possibly with \( j = i \). Then, \( g \circ f \) is a solution to \( \mathcal{I} \) as an instance of \( \text{MAX SOL}(\Gamma_j) \), and the measure of \( g \circ f \) is

\[
\langle \omega, (g \circ f)(\nu_j) \rangle = \langle f^*(\omega), g(\nu_j) \rangle = \text{Opt}(\mathcal{J}, \Gamma_j).
\]

Hence, \( \text{Opt}(\mathcal{I}, \Gamma_j) \geq \text{Opt}(\mathcal{J}, \Gamma_j) \) for all \( j \). When \( j = i \), we therefore have \( \text{Opt}(\mathcal{J}, \Gamma_i) = \text{Opt}(\mathcal{I}, \Gamma_i) \), and the identity homomorphism is one of the optimal solutions to the instance \( \mathcal{J} \). Furthermore, when \( j \neq i \), \( \text{Opt}(\mathcal{J}, \Gamma_j) \leq \text{Opt}(\mathcal{I}, \Gamma_j) = \text{Opt}(\mathcal{J}, \Gamma_i) \), so \( \mathcal{J} \) is indeed a witness.
Proposition 6.17 Let \( \{ \Gamma_1, \ldots, \Gamma_k \} \) be a collection of finite, connected relational structures and let \( \Gamma = \prod_{i=1}^{k} \Gamma_i \).

1. If \( \text{Max Sol}(\Gamma_i) \) is in \( \text{PO} \) for \( 1 \leq i \leq k \), then \( \text{Max Sol}(\Gamma) \) is in \( \text{PO} \).

2. If \( \Gamma_i \) is essential in \( \Gamma \), then \( \text{Max Sol}(\Gamma_i) \) is polynomial-time reducible to \( \text{Max Sol}(\Gamma) \).

Proof: Due to Proposition 6.9, we may assume that the input instance is connected. If not, we can decompose it in polynomial time and treat the parts separately. The first part of the lemma is immediate; for an instance \( I \) of \( \text{Max Sol}(\Gamma) \), solve \( I \) for the \( k \) problems \( \text{Max Sol}(\Gamma_i) \) and return the maximal solution.

For the second part, since \( \Gamma_i \) is essential, we can find a witness \( J \) which maximises strictly on \( \Gamma_i \) and by Lemma 6.16, \( J \) can be chosen as \( (\Gamma_i, \mu) \). Let \( A_i \) be the domain of \( \Gamma_i \) and define the relations \( g_a \equiv \pi_a \text{Opt} \text{sol}(J) \), for \( a \in A_i \). Since the identity map from \( J \) to \( \Gamma_i \) is one of the optimal solutions, we have \( (a) \in g_a \) for all \( a \in A_i \). From Corollary 6.11, we know that an optimal solution to the connected instance \( J \) of \( \text{Max Sol}(\Gamma) \) has an image which lies entirely in \( \Gamma_i \). From this, we get \( g_a \subseteq \{(b) \mid b \in A_i \} \). Now, we fix one variable \( v \) in the input instance \( I \), and create one instance \( I_a \) for each \( a \in A_i \), by taking a copy of \( I \), and adding the constraint \( (v) \in g_a \). By Lemma 6.2, we can solve \( I_a \) as an instance of \( \text{Max Sol}(\Gamma \cup \{g_a\}) \). By another application of Corollary 6.11, and the fact that the image of \( v \) under any solution is in \( \Gamma_i \), we know that the entire image of \( I \) under any solution is in \( \Gamma_i \). For every \( a \), we have \( \text{Opt}(I_a, \Gamma) \leq \text{Opt}(I, \Gamma_i) \), and equality will hold for some \( a \), since \( a \) ranges over the entire set \( A_i \). It follows that we can solve \( I \) for \( \text{Max Sol}(\Gamma_i) \) by taking the maximal of the solutions to the instances \( I_a \).

6.2 Constant production

In this section, we make use of weighted pp-definability and connectedness to show that, in a sense, we can always assume that we have access to all constants. The precise statement of this result is given in Theorem 6.22. It says that \( \text{Max Sol} \) has a dichotomy if and only if \( \text{OAL Max Sol} \) does, where \( \text{OAL Max Sol}(\Gamma) \) is defined as \( \text{Max Sol}(\Gamma \cup \mathcal{C}_A) \). The abbreviation \( \text{OAL} \) stands for One-or-All Lists, which means that, in addition to the constraints in \( \Gamma \), we may choose the domain of each variable either to be the entire domain of \( \Gamma \), or to be just one specific domain value, i.e. we may use any constant, unary constraint. The main goal will be to prove Lemma 6.21, from which this theorem is derived.

Let \( \Gamma \) be a constraint language, and let \( F \subseteq \text{End}(\Gamma) \). Define

\[
\prod_{F} \Gamma = \prod_{f \in F} f(\Gamma),
\]
the disjoint union of all constraint languages obtained by applying an endomorphism from \( F \) to \( \Gamma \). Here we index the disjoint union by the endomorphisms of \( \Gamma \). Let \( a \) be an element from the domain of \( \Gamma \). It will be understood that a valuation \( \nu \) on \( \Gamma \) is extended to \( \prod F \Gamma \) by \( \nu(\psi(a)) = \nu(a) \) for all \( f \in F \).

The following lemma is easy but acts as an important starting point for Lemma 6.21.

**Lemma 6.18** Let \( \Gamma \) be a finite, connected constraint language with valuation \( \nu \), and let \( P \subseteq \text{End}(\Gamma) \setminus \{\text{id}\} \) be a set of endomorphisms on \( \Gamma \). Then, the problems \( \text{Max Sol}(\Gamma) \) and \( \text{Max Sol}(\prod_{P \cup \{\text{id}\}} \Gamma) \) are polynomial-time equivalent.

**Proof:** For any endomorphism \( f \in P \), it is clear that \( f(\Gamma) \) cannot be essential in \( \prod_{P \cup \{\text{id}\}} \Gamma \). Hence, we can remove \( f \) from \( P \) by Proposition 6.15. The lemma follows by repeating this process until \( P \) is empty. \( \blacksquare \)

Note that if \( g \in P \subseteq \text{End}(\Gamma) \setminus \{\text{id}\} \) for some automorphism \( g \) on \( \Gamma \), then \( \Gamma \) can never be essential in \( \prod_{P \cup \{\text{id}\}} \Gamma \). However, we can show that if \( P = \text{End}(\Gamma) \setminus \text{Aut}(\Gamma) \), and \( \Gamma \) is essential in \( \prod_{P \cup \{\text{id}\}} \Gamma \), then all unary constant relations are in \( \langle \Gamma \rangle_w \). This is a consequence of Lemma 6.20, and in order to prove this result, we first need a technical lemma.

**Lemma 6.19** Let \( \Gamma \) be a finite constraint language with an injective valuation \( \nu \), let \( \nu : A \to \mathbb{Q}_{\geq 0} \), and assume that \( S, T \subseteq \text{End}(\Gamma) \) are such that, for all choices of \( f, g \in S \), and \( h \in T \),

\[
\langle \mu, f(\nu) \rangle = \langle \mu, g(\nu) \rangle,
\]

and

\[
\langle \mu, f(\nu) \rangle > \langle \mu, h(\nu) \rangle.
\]

Then, there exists a \( \mu' : A \to \mathbb{Q}_{\geq 0} \), and an \( f' \in S \) such that,

\[
\langle \mu', f'(\nu) \rangle > \langle \mu', g'(\nu) \rangle,
\]

for all \( g' \in (S \cup T) \setminus \{f'\} \).

**Proof:** First, for any \( \omega : A \to \mathbb{Q}_{\geq 0} \), define \( S_{\omega} \) to be the endomorphisms \( f \in S \cup T \) for which \( \langle \omega, f(\nu) \rangle \) is maximised, and define \( T_{\omega} = (S \cup T) \setminus S_{\omega} \). Note that we have \( S_{\mu} = S \) and \( T_{\mu} = T \). To prove the lemma, we will show that it is possible to change \( \mu \) slightly and obtain a \( \mu' \) for which \( S_{\mu'} = \{f'\} \), with \( f' \in S \). To this end, define the “distance”, \( \Delta_{\mu}(S, T) \), between \( S \) and \( T \), with respect to \( \mu \), by

\[
\Delta_{\mu}(S, T) = \min_{f \in S, h \in T} \langle \mu, f(\nu) \rangle - \langle \mu, h(\nu) \rangle.
\]
We let $\mu_0 = \mu$, and for $i = 1, \ldots, n$, we carry out the following construction. Let $\varepsilon_i > 0$, and define the valuation $\mu_i$ as follows:

$$
\mu_i(a_j) = \begin{cases} 
\mu_{i-1}(a_j) & \text{if } i \neq j, \\
\mu_{i-1}(a_j) + \varepsilon_i & \text{otherwise.}
\end{cases}
$$

We now claim that for each $i \geq 1$, we can choose $\varepsilon_i > 0$ in such a way that $S_{\mu_i} \subseteq S_{\mu_{i-1}}$. Note that this follows if we can ensure that $\Delta_{\mu_i}(S_{\mu_{i-1}}, T_{\mu_{i-1}}) > 0$. Let $\gamma = \max_{a \in A} \nu(a) - \min_{a \in A} \nu(a) > 0$. Then,

$$
\Delta_{\mu_i}(S_{\mu_{i-1}}, T_{\mu_{i-1}}) = \min_{f \in S_{\mu_{i-1}}, h \in T_{\mu_{i-1}}} \varepsilon_i \cdot (\nu(f(a_j)) - \nu(h(a_j)) + (\mu_{i-1}, f(\nu)) - (\mu_{i-1}, h(\nu))
$$

$$
> -\varepsilon_i \cdot \gamma + \Delta_{\mu_{i-1}}(S_{\mu_{i-1}}, T_{\mu_{i-1}}).
$$

By definition, we have $\Delta_{\mu_{i-1}}(S_{\mu_{i-1}}, T_{\mu_{i-1}}) > 0$, hence if we pick $\varepsilon_i$ so that $0 < \varepsilon_i < \Delta_{\mu_{i-1}}(S_{\mu_{i-1}}, T_{\mu_{i-1}})/\gamma$, then the claim follows.

With this property established, we find that $S_{\mu_1}$ contains those $f \in S_{\mu_0}$ for which $\nu(f(a_1))$ is maximised, and in general, for $i \geq 1$, $S_{\mu_i}$ contains those $f \in S_{\mu_{i-1}}$ for which $\nu(f(a_i))$ is maximised. This shows that for any two $f_1, f_2 \in S_{\mu_i}$, we have $\nu(f_1(a_j)) = \nu(f_2(a_j))$, for all $j$. Since $\nu$ is assumed to be injective, this is equivalent to $f_1(a_j) = f_2(a_j)$ for all $j$, hence $f_1 = f_2$. Therefore, $S_{\mu_n}$ contains a single endomorphism $f'$, and furthermore $f' \in S_{\mu_n} \subseteq S_{\mu_{n-1}} \subseteq \cdots \subseteq S_{\mu_0} = S$. The lemma follows with this $f'$ and $\mu' = \mu_n$.

**Lemma 6.20** Let $\Gamma$ be a connected constraint language with valuation $\nu$. If there exists a $\mu : A \to \mathbb{Q}_{\geq 0}$ such that for all $f \in \text{End}(\Gamma) \setminus \text{Aut}(\Gamma)$, it holds that

$$
\langle \mu, \nu \rangle > \langle \mu, f(\nu) \rangle,
$$

then $C_A \subseteq \langle \Gamma \rangle_{w}$.

**Proof:** Let $\mathcal{I} = (\Gamma, \mu)$ be an instance of $\text{MaxSol}(\Gamma)$. Then, the set of optimal solutions to $\mathcal{I}$ is a subset of $\text{Aut}(\Gamma)$. Let $S$ be this set, and let $T = \text{End}(\Gamma) \setminus S$. By Lemma 6.19, we can find a $\mu' : A \to \mathbb{Q}_{\geq 0}$, and an $f' \in S$ such that $f'$ is the unique maximal solution to $(\Gamma, \mu')$. We claim that the identity is the unique maximal solution to $\mathcal{I}' = (\Gamma, (f')^{-1}(\mu'))$; since $f'$ is an automorphism, and by (6.9), we have

$$
\langle (f')^{-1}(\mu'), \nu \rangle = \langle \mu', f'(\nu) \rangle > \langle \mu', h(\nu) \rangle = \langle (f')^{-1}(\mu'), (f')^{-1}(h(\nu)) \rangle,
$$

for all solutions $h$ different from $f'$. Since $(f')^{-1}$ is an automorphism, we have $\{h \circ (f')^{-1} | h \in \text{End}(\Gamma) \setminus \{f'\} = \text{End}(\Gamma) \setminus \{1_d\}$. Therefore,
Lemma 6.19 is actually more general than necessary for its use in the proof of Lemma 6.20. It is not hard to make the modifications so that if $S$ consists only of automorphisms, then we can single out a $\mu'$ for which $f' = \text{id}$ is the unique solution, without having to use $(f')^{-1}(\mu')$.

The condition (6.13) will be our replacement for the concept of a core. It ensures that we can add constants to $\Gamma$, and as the following lemma shows, in the case that it does not hold, we can reduce to $\text{Max Sol}(\Gamma)$ from some “simpler” problem. However, unlike for the case of a core (and for the case of a max-core), we cannot find a unique smaller structure, and in fact, we may need a disjoint union of several.

**Lemma 6.21** Let $\Gamma$ be a finite, connected constraint language with an injective valuation $v$. If condition (6.13) does not hold for $\Gamma$, then one of the following holds:

1. There exists a non-surjective endomorphism $g$ on $\Gamma$ such that $g(\Gamma)$ is not in $\text{PO}$, and $\text{Max Sol}(g(\Gamma))$ is polynomial-time reducible to $\text{Max Sol}(\Gamma)$, or

2. There exists a set $P$ of non-surjective endomorphisms on $\Gamma$ such that $g(\Gamma)$ is in $\text{PO}$ for all $g \in P$, and $\text{Max Sol}(\prod P \Gamma)$ is polynomial-time equivalent to $\text{Max Sol}(\Gamma)$.

**Proof:** Let $F = \text{End}(\Gamma) \setminus \text{Aut}(\Gamma)$. Assume that $\Gamma$ is essential in $\prod_{F \cup \{\text{id}\}} \Gamma$. By Lemma 6.16, we can find an instance $\mathcal{I} = (\Gamma, \mu)$ of $\text{Max Sol}(\Gamma)$, with the identity as one of the maximal solutions, and such that all maximal solutions to $\mathcal{I}$ are surjective. But then $\mu$ satisfies (6.13), hence we conclude that $\Gamma$ is not essential in $\prod_{F \cup \{\text{id}\}} \Gamma$. By Lemma 6.18 and Proposition 6.15, we have that $\text{Max Sol}(\Gamma)$ is polynomial-time equivalent to $\text{Max Sol}(\prod P \Gamma)$. We now proceed by reducing the set $F$, one endomorphism at a time, until we can show that either (1) or (2) holds. We use an inductive argument on the size of $F$, which is finite, where in each step, we maintain the property that $\text{Max Sol}(\prod P \Gamma)$ is polynomial-time equivalent to $\text{Max Sol}(\Gamma)$.

Note that unless (6.13) holds, $F$ must be non-empty. Therefore, our base case will be when $F = \{g\}$, in which case $g(\Gamma)$ is trivially essential in $\prod_{\{g\}} g(\Gamma)$. Since we have maintained that $\text{Max Sol}(\prod P \Gamma)$ is polynomial-time equivalent to $\text{Max Sol}(\Gamma)$, either (2) or (1) holds, respectively depending on whether $\text{Max Sol}(g(\Gamma))$ is in $\text{PO}$ or not.

If $|F| > 1$, and $F$ consists entirely of endomorphisms $g$ such that $g(\Gamma)$ is in $\text{PO}$, then by Proposition 6.17(1), we see that $\text{Max Sol}(\prod P \Gamma)$ is in $\text{PO}$, and since it is also polynomial-time equivalent to $\text{Max Sol}(\Gamma)$, (2) holds with $P = F$, and we are done.

Hence, we can assume that $F$ contains a $g$ such that $g(\Gamma)$ is not in $\text{PO}$. If in addition, $g(\Gamma)$ is essential in $\prod P \Gamma$, then by Proposition 6.17(2), we
see that \( \text{Max Sol}(g(\Gamma)) \) is polynomial-time reducible to \( \text{Max Sol}(\prod F \Gamma) \), which in turn is polynomial-time equivalent to \( \text{Max Sol}(\Gamma) \). In this case, (1) holds, and we are done.

Finally, if \( F \) contains a \( g \) such that \( g(\Gamma) \) is not in \( \text{PO} \), and \( g(\Gamma) \) is not essential in \( \prod F \Gamma \), then we can apply the induction hypothesis on the set \( F' = F \setminus \{ g \} \). By Proposition 6.15, \( \text{Max Sol}(\prod F' \Gamma) \) is polynomial-time equivalent to \( \text{Max Sol}(\prod F \Gamma) \), and hence polynomial-time equivalent to \( \text{Max Sol}(\Gamma) \), so this property is preserved.

\[ \text{Theorem 6.22 Max Sol exhibits a dichotomy iff OAL Max Sol does.} \]

**Proof:** Obviously, if \( \text{Max Sol}(\Gamma) \) is either in \( \text{PO} \) or \( \text{NP}-\text{hard} \) for arbitrary finite \( \Gamma \), then in particular this is true for languages of the form \( \Gamma' = \Gamma \cup \mathcal{C}_A \).

To prove the other direction, assume that \( \text{OAL Max Sol} \) exhibits a dichotomy. We proceed by induction over the size of the domain \( A \). The base case, when \( |A| = 1 \), is trivial. We may assume that \( \Gamma \) is connected. Indeed, if it is not, we decompose it into connected components. Since all of the connected components have a domain of size less than \( |A| \), we use the induction hypothesis and apply Proposition 6.17.

If Lemma 6.20 is applicable, then all constants are in \( (\Gamma)_w \), and by Lemma 6.2, it follows that \( \text{Max Sol}(\Gamma \cup \mathcal{C}_A) \) is polynomial-time reducible to \( \text{Max Sol}(\Gamma) \). The other direction is trivial, so in this case the assumed dichotomy of \( \text{OAL Max Sol} \) carries over to \( \text{Max Sol} \).

Otherwise, Lemma 6.21 is applicable, and one of the two listed cases holds. If case 1 holds, then there is a non-surjective endomorphism \( g \in F = \text{End}(\Gamma) \setminus \text{Aut}(\Gamma) \) such that \( g(\Gamma) \) is not in \( \text{PO} \), and \( \text{Max Sol}(g(\Gamma)) \) is polynomial-time reducible to \( \text{Max Sol}(\Gamma) \). By the induction hypothesis, we conclude that \( \text{Max Sol}(g(\Gamma)) \), and thereby \( \text{Max Sol}(\Gamma) \), are in fact \( \text{NP}-\text{hard} \). Finally, if case 2 of the lemma holds, then by Proposition 6.17(1), \( \text{Max Sol}(\Gamma) \) is in \( \text{PO} \).

A remark is in order, before we move on to study some examples on graphs. Namely, Theorem 6.22 does not state that \( \text{Max Sol} \) and \( \text{OAL Max Sol} \) are polynomial-time equivalent. Indeed, it is easy to see that they are not (see for example Proposition 6.34.) Instead, it shows that in order to prove (or disprove) a dichotomy property for \( \text{Max Sol} \), it suffices to do the same for \( \text{OAL Max Sol} \), which is likely to simplify the task substantially. However, given a dichotomy theorem for \( \text{OAL Max Sol} \), it is not clear that the boundary between easy and hard constraint languages will be easily deducible.

### 6.3 Application: graphs

Having considered the general case in the previous sections, we now turn to some concrete examples, namely undirected graphs. Throughout this
section, we will assume that all graphs defining constraint languages are max-cores and connected. Due to Lemma 2.36, Proposition 6.15, and Proposition 6.17, we can do this without loss of generality. We will (usually) avoid the use of an explicit valuation for the constraint language defining graphs, and instead assume that the vertex sets of these graphs are subsets of the natural numbers, to be used for calculating the measure of solutions.

We consider $H = (V, E)$ both as an undirected graph on the vertex set $V$, and as a relation on $V$. For instance, we may write both $\{1, 2\} \subseteq E$ and $(1, 2) \in H$ to indicate an edge between vertices 1 and 2. We also write $H(x, y)$ to denote that the instance variables $x$ and $y$ are constrained by $H$. Since we mostly work with constraint languages consisting of a single constraint, we will often write $\text{MAX SOL}(H)$ instead of $\text{MAX SOL}(\{H\})$.

Several of the hardness results will be established by reduction from the independent set problem, using Proposition 2.29. As in the statement of this proposition, we will let $R_{(a, b)}$ denote the relation $\{(a, a), (a, b), (b, a)\}$.

Two additional problems will play important roles by providing some useful reductions. The first of these two concerns retracts of graphs. Let $G$ be a graph and $H$ be a subgraph of $G$. $H$ is a retract of $G$ if there exists a graph homomorphism $f : G \rightarrow H$ such that $f(v) = v$ for all $v \in V(H)$.

**Definition 6.23** The Retraction Problem, denoted Ret($H$), is the decision problem with:

**Instance:** A graph $G$, which contains $H$ as a subgraph.

**Question:** Is $H$ a retract of $G$?

In the retraction problem, we can assume that $V(H) \subseteq V(G)$, so that it is straightforward to see how $H$, as a subgraph, is contained in $G$. The following establishes the connection between MAX SOL and the retraction problem:

**Lemma 6.24** Let $H$ be an undirected graph on the vertex set $V$. Then, Ret($H$) is polynomial-time reducible to MAX SOL($H \cup C_V$).

**Proof:** We can use the constant relations in $C_V$ to fix the subgraph $H$ in $G$. Then, this instance of MAX SOL has solutions if and only if $H$ is a retract of $G$.

The second problem is an optimisation problem introduced by Gutin et al. [67] to model problems in defence logistics.

**Definition 6.25** The problem Minimum Cost Graph Homomorphism over a graph $H$, denoted MIN HOM($H$), is defined to be the minimisation problem with

**Instance:** a graph $G$ and costs functions $c_i : V(G) \rightarrow Q_{\geq 0} \cup \{\infty\}$, for $i \in V(H)$;
Solution: a graph homomorphism $f : V(G) \to V(H)$;

**Measures**: $m(f) = \sum_{v \in V} c_{f(v)}(v)$.

It is easy to see that we can model the measure of the maximum solution problem by the more general cost functions of the MIN HOM problem. This provides us with the following reduction:

**Lemma 6.26** Let $H$ be a graph with $V(H) \subseteq \mathbb{N}$. Then, MAX SOL$(H)$ is polynomial-time reducible to MIN HOM$(H)$.

**Proof**: Define the cost functions of the MIN HOM instance by $c_i(v) = M - \mu(v) \cdot i$, where $\mu$ is the weight function of the MAX SOL instance, and $M$ is a large enough constant to make all the cost functions non-negative. ■

Gutin et al. [62] established a dichotomy result for MIN HOM on undirected graphs with loops allowed. We will have a closer look at this result in Chapter 7. Here, we will only use one particular polynomial-time solvable class for MIN HOM, to prove the following proposition:

**Proposition 6.27** For every irreflexive path $H$, MAX SOL$(H)$ is in PO.

**Proof**: It is easy to check that paths are proper interval graphs (cf. Definition 7.6.) The proposition then follows from Theorem 1.1 in [62] together with the reduction in Lemma 6.26. ■

The final tool will be weighted pp-definability and Lemma 6.2. These will be used in the form of the following three lemmas:

**Lemma 6.28** Let $\Gamma$ be a constraint language, with a unary relation $U \in \Gamma$. Let $m$ be the maximal element in $U$. Then, MAX SOL$(\Gamma)$ is polynomial-time equivalent to MAX SOL$(\Gamma \cup \{ \langle m \rangle \})$.

**Proof**: Let the instance $\mathcal{I} = (V, A, C, \omega)$ be defined by $V = \{ v_1 \}$, $C = \{ \langle v_1, U \rangle \}$, and $\omega(v_1) = 1$. Then, $\{ \langle m \rangle \} = \pi_1 \text{OPTSOL}(\mathcal{I}) \in (\Gamma)_w$, and the result follows from Lemma 6.2. ■

**Lemma 6.29** Given an undirected graph $H$, let $U = \{ \langle a_i \rangle \mid \{ a_i, a_j \} \in E(H) \text{ and } a_i + a_j \geq a_i' + a_j' \text{ for all } \{ a_i', a_j' \} \in E(H) \}$. Then, MAX SOL$(H)$ is polynomial-time equivalent to MAX SOL$(H \cup \{ U \})$.

**Proof**: Let the instance $\mathcal{I} = (V, A, C, \omega)$ be defined by $V = \{ v_1, v_2 \}$, $C = \{ H(v_1, v_2) \}$, and $\omega(v_1) = \omega(v_2) = 1$. Then, $U = \pi_1 \text{OPTSOL}(\mathcal{I}) \in (H)_w$, and the result follows from Lemma 6.2. ■
6. Constant Production

**Lemma 6.30** Let $H = (V,E)$ be a connected bipartite graph with bipartition $V = A \cup B$. Then, MAX SOL($H$) is polynomial-time equivalent to MAX SOL($H \cup \{(\max A), (\max B)\}$).

**Proof:** For the non-trivial direction, assume that $a = \max A$, $b = \max B$, and without loss of generality that $a > b$. Let

$$\omega(v) = \begin{cases} 2, & \text{if } v = a, \\ 1, & \text{if } v = b, \text{ and} \\ 0, & \text{otherwise.} \end{cases}$$

Let $f$ be a solution to the instance $(H, \omega)$ of MAX SOL($H$). Then,

$$m(\text{id}) - m(f) = 2 \cdot (a - f(a)) + 1 \cdot (\omega(b) - f(b)).$$

Hence, if $f$ is such that $f(a) \in A$, and either $f(a) \neq a$ or $f(b) \neq b$, then $m(\text{id}) > m(f)$. On the other hand, if $f$ is such that $f(a) \in B$, then $m(\text{id}) - m(f) \leq 2 \cdot (a - b) + 1 \cdot (b - a) = a - b > 0$. We conclude that an optimal solution $f$ must fix both $a$ and $b$. Therefore, $\{(a)\}, \{(b)\} \in \langle H, \omega \rangle$, and the result follows from Lemma 6.2.

### 6.3.1 Paths and cycles

There are two types of connected, irreflexive graphs $H$ with $\deg(v) \leq 2$ for all $v \in V(H)$: paths and cycles. We have already seen, in Proposition 6.27, that MAX SOL($H$) is in PO when $H$ is a path.

When $H$ is an odd cycle, we have that CSP($H$) is NP-complete and therefore MAX SOL($H$) is NP-hard. It remains to investigate even cycles. Ignoring multiple edges, $C_2$ can be treated as a single edge, for which MAX SOL is in PO (for example by Proposition 6.27.)

For even cycles of length greater than or equal to 6, it has been shown in [51] that the retraction problem is NP-complete. We will use this with Lemma 6.24 to prove the NP-hard cases. The tractable cases follow from Proposition 6.27 together with the observation that the image of any non-surjective endomorphism on a cycle must be a proper, connected, subset of the cycle itself. This implies that such an image must in fact be a path.

For even cycles $H$, we will assume a bipartition $V(H) = \{a_1, \ldots, a_k\} \cup \{b_1, \ldots, b_k\}$ such that $a_1 < a_2 < \cdots < a_k$ and $b_1 < b_2 < \cdots < b_k$. Furthermore, without loss of generality, we will assume that $a_k > b_k$. In the general case, we have the following proposition:

**Proposition 6.31** Let $H$ be a graph isomorphic to $C_{2k}$, for $k \geq 2$. If there exists a $\mu : A \to \mathbb{Q}_{\geq 0}$ such that for every $f \in \text{End}(H)$,

$$(\mu, \nu) > (\mu, f(\nu)),$$

then MAX SOL($H$) is NP-hard. Otherwise, MAX SOL($H$) is in PO.
6.3. Application: graphs

Proof: Any image of a non-surjective endomorphism on a cycle is a proper, connected, subgraph of the cycle, i.e. a path. By Proposition 6.27, Max Sol on a path is in PO. The proposition now follows from Lemma 6.21.

We now give more explicit forms of Proposition 6.31 for the cases when $H$ is isomorphic to $C_4$, $C_6$, or $C_8$.

Proposition 6.32 If $H$ is isomorphic to $C_4$, then Max Sol($H$) is in PO.

Proof: Let $H$ be the graph in Figure 6.5. The endomorphism $f$ which sends $a_1$ to $a_2$, $b_1$ to $b_2$, and leaves $a_2$ and $b_2$ fixed, is increasing. This shows that the graph consisting solely of the edge $\{a_2, b_2\}$ is the max-core of $H$. The proposition now follows either from Proposition 6.27 directly, or by Proposition 6.31 since $\langle \mu, f(\nu) \rangle > \langle \mu, \nu \rangle$, for any $\mu : V(H) \to \mathbb{Q}_{\geq 0}$ with $\mu(a_1), \mu(b_1) > 0$.

Figure 6.5: The graph $H$ in Proposition 6.32, with max-core $H|_{\{a_2, b_2\}}$.

Proposition 6.33 Let $H$ be isomorphic to $C_6$ and a max-core. Then, Max Sol($H$) is NP-hard.

Proof: If $(a_i, b_i) \in H$ for $i = 2$ or $i = 3$, then $f(a_j) = a_j, f(b_j) = b_j$ for $j \geq i$ and $f(a_j) = a_i, f(b_j) = b_i$ for $j < i$ defines a non-injective, increasing endomorphism $f$ on $H$. So in order for $H$ to be a max-core, we must have $(a_i, b_i) \not\in H$ for all $i$. The graph $H$ can then be drawn as in Figure 6.6.

Figure 6.6: The max-core $H$, and the weights $\mu$ used in Proposition 6.33.
6. Constant Production

We claim that, with constants $M$ and $M'$ chosen appropriately, the weights $\mu : A \to \mathbb{Q}_{\geq 0}$ given on the right in the figure fulfills the condition for NP-hardness in Proposition 6.31. To see this, we consider an optimal solution to $\text{Max Sol}(H)$ of the instance $G$. If we choose $M$ and $M'$ large enough, with $M > M'$, then for any endomorphism $f \in \text{End}(H)$ such that $(\mu, f(\nu)) \geq (\mu, \nu)$, we must have $f(a_3) = a_3$ and $f(b_3) = b_3$. There are now four remaining possibilities for $f$, one of which is the identity. If $f(a_1) = a_2$, $f(b_2) = b_1$, and $f$ acts as the identity on $\{a_1, a_2, a_3, b_3\}$, then $m(f) - m(\text{id}) = \mu(a_1) \cdot (a_2 - a_1) + \mu(b_2) \cdot (b_1 - b_2) = b_1 - b_2 < 0$. The other cases follow similarly, hence we conclude, by Proposition 6.31, that $\text{Max Sol}(H)$ is NP-hard.

**Proposition 6.34** Let $H$ be isomorphic to $C_8$ and a max-core. If $H$ is the graph to the left in Figure 6.7, and

$$(a_4 - a_3)(b_4 - b_3) \geq (a_3 - a_2)(b_3 - b_2),$$

then $\text{Max Sol}(H)$ is in $\text{PO}$. Otherwise it is NP-hard.

![Figure 6.7: The graph $H$ in Proposition 6.34 (left), and the position of $a_4$ and $b_4$ in a max-core isomorphic to $C_8$ (right.)](image)

**Proof:** Since $H$ is a max-core, we know that $(a_4, b_4) \notin H$, hence we can draw $H$ as the graph on the right in Figure 6.7. By Lemma 6.30, we can work with $\text{Max Sol}(H \cup \{(a_4), (b_4)\})$, and for this constraint language, any endomorphism must fix $a_4$, $b_4$, $v_1$ and $v_2$. This also implies that any non-injective endomorphism must map either $\{v_3 \mapsto v_2, v_4 \mapsto v_1\}$ or $\{v_5 \mapsto v_2, v_6 \mapsto v_1\}$. Hence, for $H$ to be a max-core, it is sufficient (and necessary) that

$$v_3 > v_2 \vee v_4 > v_1 \quad \text{and} \quad v_5 > v_2 \vee v_6 > v_1.$$
There are five non-identity endomorphisms on $H \cup \{(a_4), (b_4)\}$:

<table>
<thead>
<tr>
<th></th>
<th>$v_3$</th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>$v_3$</td>
<td>$a_4$</td>
<td>$v_2$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$f_2$</td>
<td>$v_2$</td>
<td>$a_4$</td>
<td>$v_2$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$f_3$</td>
<td>$v_2$</td>
<td>$v_1$</td>
<td>$v_2$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$f_4$</td>
<td>$v_2$</td>
<td>$v_1$</td>
<td>$b_4$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$f_5$</td>
<td>$v_2$</td>
<td>$v_1$</td>
<td>$b_4$</td>
<td>$v_6$</td>
</tr>
</tbody>
</table>

By Proposition 6.31, Max Sol($H$) is NP-hard if and only if we can find a solution $\mu \geq 0$ to the following system of equations:

$$\sum_{i=3}^{6} \mu(v_i) \cdot (\text{id} - f_j)(v_i) > 0, \quad j = 1, \ldots, 5. \quad (6.14)$$

Instead of finding explicit solutions, we will use a variation of the Farkas lemma (cf. [107]) which states that (6.14) has solutions iff the following system does not have any solutions $x = (x_1, \ldots, x_5)^T \geq 0$:

$$\begin{pmatrix}
0 & v_3 - v_2 & v_3 - v_2 & v_3 - v_2 & v_3 - v_2 \\
v_4 - a_4 & v_4 - a_4 & v_4 - v_1 & v_4 - v_1 & v_4 - v_1 \\
v_5 - v_2 & v_5 - v_2 & v_5 - v_2 & v_5 - b_4 & v_5 - b_4 \\
v_6 - v_1 & v_6 - v_1 & v_6 - v_1 & v_6 - v_1 & 0
\end{pmatrix} x \lesssim 0. \quad (6.15)$$

Here, $\lesssim$ indicates componentwise less than or equal, but not identically equal vectors. On this form, we see that if $v_3 > v_2$, then $x_2 = \cdots = x_5 = 0$, which forces $x_1 > 0$, but since either $v_5 > v_2$ or $v_6 > v_1$ in order for $H$ to be a max-core, either the third, or the last equation of (6.15) does not hold. NP-hardness for the case when $v_6 > v_1$ follows similarly.

Hence, we must have $v_3 < v_2, v_4 > v_1, v_5 > v_2$, and $v_6 < v_1$, i.e. the case shown on the left-hand side in Figure 6.7. This gives strict inequalities in either the first or the last equation. We also see that we may take $x_3 = 0$ in any solution. The system (6.15) therefore reduces to the following:

$$\begin{pmatrix}
a_3 - a_4 & a_3 - a_2 \\
b_3 - b_2 & b_3 - b_4
\end{pmatrix} y \leq 0,$$

with $y \geq 0$. This system has solutions if and only if the determinant is non-negative, i.e. if $(a_3 - a_4)(b_3 - b_4) - (a_3 - a_2)(b_3 - b_2) \geq 0$. 

In the previous proof, we could have produced an explicit $\mu$ for each individual case, as we did in Proposition 6.33. However, the alternative approach of studying the system (6.15) may sometimes be preferable. In particular, for Max AW Sol, when we allow negative weights, the alternative system becomes a system of equalities.
6.3.2 Graphs on at most three vertices

In this section we determine the complexity of $\text{MaxSol}(H)$ for arbitrary graphs $H = (V, E)$ on at most 3 vertices, i.e., $|V| \leq 3$. In the process we discover a new tractable class for the $\text{MaxSol}(H)$ problem which is closely related to the critical independent set problem. We will use the notation $\text{id}[a_i \mapsto a_j]$ to denote the function which sends the domain element $a_i$ to $a_j$, and which acts as the identity function on all other elements.

The case $|V| = 1$ is trivial, there are only two graphs, an isolated vertex, and an isolated vertex with a loop, both of which are max-cores, and tractable. For $|V| = 2$ there are also two max-cores, shown in Figure 6.8. The irreflexive path in Figure 6.8a is in $\text{PO}$ by Proposition 6.27, and the graph in Figure 6.8b is $\text{NP}$-hard by Proposition 2.29.

![Figure 6.8: The two max-cores on two vertices, with $a_1 < a_2$.](image)

For $|V| = 3$ we have the following classification:

**Theorem 6.35** There are six (types of) max-cores over \{a_1, a_2, a_3\} where $a_1 < a_2 < a_3$, denoted $H_1, \ldots, H_6$ and shown in Figure 6.9. $\text{MaxSol}(H)$ is $\text{NP}$-hard for all of these except for $H_5$. $\text{MaxSol}(H_5)$ is in $\text{PO}$ if $a_3 + a_1 \leq 2a_2$ and $\text{NP}$-hard otherwise.

![Figure 6.9: The six max-cores on three vertices, with $a_1 < a_2 < a_3$.](image)

**Proof:** We begin by proving that $H_1, \ldots, H_6$ are the only max-cores over...
\( \{a_1, a_2, a_3\} \) where \( a_1 < a_2 < a_3 \). We consider three different cases depending on the neighbourhood, \( N(a_3) \), of \( a_3 \).

\( N(a_3) = \{a_1, a_2\} \): Assume first that \( a_1 \) and \( a_2 \) are not adjacent, so that \( a_1 \rightarrow a_2 \rightarrow a_3 \) forms a path. If \( a_1 \) has no loop, or if \( a_1 \) and \( a_2 \) have loops, then \( \text{id}[a_1 \mapsto a_2] \) is a non-injective, increasing endomorphism on \( H \), so \( H \) is not a max-core. Therefore, we can assume that there is a loop at \( a_1 \), but not at \( a_2 \), hence \( H = H_1 \).

Assume instead that \( a_1 \) and \( a_2 \) are adjacent. If \( a_2 \) has a loop, then \( \text{id}[a_1 \mapsto a_2] \) is a non-injective, increasing endomorphism on \( H \), so we can assume that \( a_2 \) has no loop. If \( a_1 \) has a loop, then \( H = H_2 \), otherwise \( H = H_0 \).

\( N(a_3) = \{a_2\} \): Since \( H \) is connected, \( a_1 \rightarrow a_2 \rightarrow a_3 \) forms a path. If there is no loop at \( a_1 \), then \( \text{id}[a_1 \mapsto a_3] \) is a non-injective, increasing endomorphism on \( H \), and if \( a_2 \) has a loop, then \( \text{id}[a_1 \mapsto a_2] \) is a non-injective, increasing endomorphism. We conclude that there is a loop at \( a_1 \), but not at \( a_2 \), and \( H = H_3 \).

\( N(a_3) = \{a_1\} \): Since \( H \) is connected, \( a_2 \rightarrow a_1 \rightarrow a_3 \) forms a path. If there is no loop at \( a_2 \), then \( \text{id}[a_2 \mapsto a_3] \) is a non-injective, increasing endomorphism on \( H \). Hence, there must be a loop at \( a_2 \), and \( H = H_4 \) or \( H = H_5 \).

The computational complexity of \( \text{Max \ Sol}(H_i) \) for \( 1 \leq i \leq 6 \) is given in Lemma 6.36 through 6.41 below, which finishes the proof.

**Lemma 6.36** \( \text{Max \ Sol}(H_1) \) is \( \text{NP-hard} \).

**Proof:** The unary relation \( U = \{(a_1), (a_3)\} \) can be implemented by the pp-formula \( U(x) \equiv \exists y : E(y, y) \land E(y, x) \). The lemma follows, since \( \text{Max \ Sol}(E)|_{\{(a_1, a_3)\}} \) is \( \text{NP-hard} \) by Proposition 2.29.

**Lemma 6.37** \( \text{Max \ Sol}(H_2) \) is \( \text{NP-hard} \).

**Proof:** This was essentially shown in Example 6.3.

**Lemma 6.38** \( \text{Max \ Sol}(H_3) \) is \( \text{NP-hard} \).

**Proof:** Analogous to the proof of Lemma 6.36.

**Lemma 6.39** \( \text{Max \ Sol}(H_4) \) is \( \text{NP-hard} \).

**Proof:** By Lemma 6.29, we can work with the constraint language \( H_4 \cup \{U\} \), where \( U \) is the unary relation \( U = \{(a_2), (a_3)\} \). The relation \( R(x, y) = V^2 \setminus \{(a_2, a_3)\} \) can be implemented by the pp-formula \( \exists z, w : H_4(x, z) \land H_4(z, w) \land H_4(w, y) \). We conclude that we can reduce from \( \text{Max \ Sol}(R)|_{\{(a_2, a_3)\}} \), which is \( \text{NP-hard} \) by Proposition 2.29.
6. Constant Production

**Lemma 6.40** \( \text{MAX} \ Sol(H_5) \) is in \( \text{PO} \) if \( a_3 + a_1 \leq 2a_2 \) and \( \text{NP-hard} \) otherwise.

**Proof:** The proof of this Lemma requires slightly more intricate ideas and is given in Section 6.3.3.

**Lemma 6.41** \( \text{MAX} \ Sol(H_6) \) is \( \text{NP-hard} \).

**Proof:** \( \text{CSP}(H_6) \) is the \( \text{NP-complete} \) three colouring problem, so \( \text{MAX} \ Sol(H_6) \) is \( \text{NP-hard} \).

### 6.3.3 Proof of Lemma 6.40

In this section we prove Lemma 6.40, regarding the graph \( H_5 \) in Figure 6.9. Let \( G \) denote an undirected simple graph and for \( B \subseteq V(G) \), let \( N(B) \) denote the neighbourhood of \( B \), i.e., the set \( \bigcup_{b \in B} N(b) \). The input graph \( G \) is weighted, so that each \( v_i \in V(G) \) has a nonnegative integer weight \( w_i \). Given a set \( S \subseteq V(G) \), we denote the sum \( \sum_{v_i \in S} w_i \) by \( w(S) \), the total weight of the vertices in \( S \).

**Definition 6.42** For a graph \( G \), define the parameters \( \alpha(G) \) and \( \beta(G) \) of \( G \) as follows.

\[
\alpha(G) = \max \{ w(B) - w(N(B)) \mid B \subseteq V(G) \}
\]

\[
\beta(G) = \max \{ w(I) - w(N(I)) \mid I \text{ is an independent set in } G \}
\]

A set of vertices \( B_C \subseteq V(G) \) is called **critical** if

\[
w(B_C) - w(N(B_C)) = \alpha(G).
\]

An independent set \( I_C \subseteq V(G) \) is called **critical** if

\[
w(I_C) - w(N(I_C)) = \beta(G).
\]

Zhang [133] observed that \( \alpha(G) = \beta(G) \) and managed to prove that \( \alpha(G) \) (and hence \( \beta(G) \)) can be computed in polynomial time. The weighted version of the problem (each vertex \( v_i \) of \( G \) have a nonnegative weight \( w_i \)) was proved to be in \( \text{P} \) by Ageev [1].

We begin by extending the notion of critical independent sets.

**Definition 6.43** For a graph \( G \), and any non-negative constants \( k \) and \( m \), define the parameter \( \beta_{km}(G) \) of \( G \) as follows.

\[
\beta_{km}(G) = \max \{ kw(I) - mw(N(I)) \mid I \text{ is an independent set in } G \}
\]

A \((k,m)\)-critical independent set is an independent set \( I_C \subseteq V(G) \) such that

\[
k \cdot w(I_C) - m \cdot w(N(I_C)) = \beta_{km}(G).
\]
We define $\alpha_{km}(G)$ and $(k, m)$-critical sets in the analogous way. Note that the maximum independent set problem is exactly the problem of finding a (1,0)-critical independent set.

The graph $H_5$ is given in Figure 6.9(e), where $a_1 < a_2 < a_3$ (otherwise the graph is not a max-core). The connection between the critical independent set problem and MAX SO1$(H_5)$ is explained in the following proposition.

**Proposition 6.44** $I_C$ is a $(a_3 - a_2, a_2 - a_1)$-critical independent set in $G$ if and only if the homomorphism $h$ from $G$ to $H_5$, defined by $h^{-1}(a_3) = I_C$ and $h^{-1}(a_1) = N(I_C)$ is an optimal solution for MAX SO1$(H_5)$.

**Proof:** Assume that $I_C$ is a $(a_3 - a_2, a_2 - a_1)$-critical independent set in $G$ but $h$ is not an optimal solution to MAX SO1$(H_5)$, i.e., there exists a homomorphism $g$ from $G$ to $H_5$ such that $m(g) > m(h)$. That is,

$$w(g^{-1}(a_3)) \cdot a_3 + w(g^{-1}(a_1)) \cdot a_1 + w(g^{-1}(a_2)) \cdot a_2 >$$

$$w(h^{-1}(a_3)) \cdot a_3 + w(h^{-1}(a_1)) \cdot a_1 + w(h^{-1}(a_2)) \cdot a_2.$$

Subtracting $w(V(G)) \cdot a_2$ from both sides, we get

$$w(g^{-1}(a_3)) \cdot (a_3 - a_2) - w(g^{-1}(a_1)) \cdot (a_2 - a_1) >$$

$$w(h^{-1}(a_3)) \cdot (a_3 - a_2) - w(h^{-1}(a_1)) \cdot (a_2 - a_1).$$

This contradicts the fact that $I_C$ is a $(a_3 - a_2, a_2 - a_1)$-critical independent set. The proof in the other direction is similar. ■

When $a_3 + a_1 > 2a_2$, then by Lemma 6.29, we get a reduction from MAX SO1$(H_5 \cup \{U\})$, to MAX SO1$(H_5)$, where $U = \{(a_1), (a_3)\}$. Using $H_5$ and $U$ we can implement the relation $R_{a_1, a_3}$, and use Proposition 2.29 to prove NP-hardness. We get the following corollary for the complexity of the $(k, m)$-critical independent set problem.

**Corollary 6.45** The $(k, m)$-critical independent set problem is NP-hard when $k > m$.

We will now prove that MAX SO1$(H_5)$ is in PO in the remaining cases, i.e., when $a_3 + a_1 \leq 2a_2$. This will be done by proving that the $(k, m)$-critical independent set problem is in P when $k \leq m$. The proofs are simple modifications of the proofs presented in [1].

We begin with the observation that when $k \leq m$, then the search for $(k, m)$-critical independent sets can be reduced to the search for $(k,m)$-critical sets.

**Proposition 6.46** When $k \leq m$, we have that $\beta_{km}(G) = \alpha_{km}(G)$. Moreover, if $B_C$ is a $(k, m)$-critical set, then the set of isolated (and non-looped) vertices in $B_C$ is a $(k, m)$-critical independent set (and hence a $(k, m)$-critical set).
6. Constant Production

**Proof:** Obviously, \( \beta_{km}(G) \leq \alpha_{km}(G) \). Assume that \( \beta_{km}(G) < \alpha_{km}(G) \). Then, there exists a \((k, m)\)-critical set \( B' \) containing some edge \( \{x, y\} \) (possibly \( x = y \)) in the subgraph of \( G \) induced by \( B' \). Furthermore assume that \( B' \) is a minimal \((k, m)\)-critical set with respect to the number of induced edges. Denote the set \( B' \) with both \( x \) and \( y \) removed by \( B'' \). Since \( x \) and \( y \) are both in \( B' \) and in \( N(B') \), we get that \( k \cdot w(B') - m \cdot w(N(B')) \leq k \cdot w(B'') - m \cdot w(N(B'')) \) which contradicts the fact that \( B' \) is a minimal \((k, m)\)-critical set.

The problem of finding \((k, m)\)-critical sets (for any \( k, m \)) can be modelled as a MAX AW SOL(G) problem over the domain \([0, 1]\) where \( G \) is the constraint language consisting of the relation \( R = \{(1, 0), (1, 1), (0, 0)\} \). Note that \( G \) is closed under both max and min, and hence, MAX AW SOL(G) is in \( \text{PO} \) by a result in [83]. Given an instance \( G \) of the \((k, m)\)-critical set problem where \( V(G) = \{x_1, \ldots, x_n\} \) and the weights are \( \{w_1, \ldots, w_n\} \) we construct the following instance \((V, [0, 1], C, \omega)\) of MAX AW SOL(G). Let \( V = \{y_1, \ldots, y_n, z_1, \ldots, z_n\} \), \( \omega(y_i) = kw_i \), \( \omega(z_i) = -mw_i \), \( C = \{R(z_i, y_j) \mid (x_i, x_j) \in E(G)\} \). It can be verified that \( s \) is an optimal solution to this instance if and only if \( B' = \{y_i \mid s(y_i) = 1\} \) is a \((k, m)\)-critical set in \( G \). Consult [1] for more details.

6.3.4 Graphs on the vertex set \([0, 1, 2, 3]\)

We now give a complexity classification of \( \text{MAX SOL}(H) \) for all graphs \( H \), with \( V(H) = \{0, 1, 2, 3\} \). Just as in the case where \( |V(H)| \leq 3 \) we make heavy use of the fact that only graphs that are max-cores need to be classified. Our second tool is the following lemma, which states that for such max-cores \( H \), we can add the unary constant relations to the constraint language, without changing its computational complexity.

**Lemma 6.47** Let \( H \) be a max-core over the domain \( A = \{0, 1, 2, 3\} \). Then, \( \text{MAX SOL}(H) \) is in \( \text{PO} \) (\( \text{NP}-\text{hard} \)) if and only if \( \text{MAX SOL}(H \cup C_A) \) is in \( \text{PO} \) (\( \text{NP}-\text{hard} \)).

**Proof:** The proof of the lemma is given in Lemma 6.49 through 6.52.

As an immediate corollary of Lemma 6.47 we get that \( \text{MAX SOL}(H) \) is \( \text{NP}-\text{hard} \) for all max-cores \( H \) on \( A = \{0, 1, 2, 3\} \) where the retraction problem, \( \text{RET}(H) \), is \( \text{NP}-\text{complete} \). Note that the complexity of the retraction problem for all graphs on at most 4 vertices has been classified in [130]. Our classification is completed by considering the remaining max-cores, for which \( \text{RET}(H) \) is in \( \text{P} \), in a similar fashion to what was done for the graphs on 3 vertices. The result is stated in the following theorem.

**Theorem 6.48** Let \( H \) be a max-core on \( A = \{0, 1, 2, 3\} \). Then, \( \text{MAX SOL}(H) \) is in \( \text{PO} \) if \( H \) is an irreflexive path, and otherwise, \( \text{MAX SOL}(H) \) is \( \text{NP}-\text{hard} \).
6.3. Application: graphs

**Proof:** The proof of the theorem is given in Lemma 6.53 through 6.56. ■

**Lemma 6.49** Let $H$ be a max-core over the domain $A = \{0, 1, 2, 3\}$. Assume that $N(3) \cap \{1, 2\} \neq \emptyset$. Then, $\text{Max Sol}(H)$ is polynomial-time equivalent to $\text{Max Sol}(H \cup C_A)$.

**Proof:** Assume that $2 \in N(3)$, the other case is treated similarly. According to Lemma 6.28 we may assume that $\{3\} \in H$. Furthermore, since the neighbourhood of 3 can be implemented by the p-p formula $N_3(x) \equiv \exists y : \{3\}(y) \land H(x, y)$, we may, again by Lemma 6.28, assume that $\{2\} \in H$. Hence, any non-identity endomorphism $\pi$ must have $\pi(1) = 0$. Let $\Gamma = H \cup \{\{2\}, \{3\}\}$, $\mu(0) = \mu(2) = \mu(3) = 0$, and $\mu(1) = 1$ in Lemma 6.5. This finishes the proof. ■

**Lemma 6.50** Assume that $N(3) = \{0\}$, $\{1, 2, 3\} \subseteq N(0)$, and that $(1, 2) \not\in H$. Then $H$ is not a max-core.

**Proof:** If $(2, 2) \not\in H$, then $\text{id}[2 \mapsto 3]$ is an increasing endomorphism, and if $(2, 2) \in H$, then $\text{id}[1 \mapsto 2]$ is an increasing endomorphism. In both cases, $H$ is not a max-core. ■

**Lemma 6.51** Assume that $N(3) = \{0\}$, $\{1, 2, 3\} \subseteq N(0)$, and that $(1, 2) \in H$. Then, $H$ is a max-core if and only if $(2, 2) \not\in H$, and in this case $\text{Max Sol}(H)$ is polynomial-time equivalent to $\text{Max Sol}(H \cup C_A)$.

**Proof:** If $(2, 2) \in H$, then $\text{id}[1 \mapsto 2]$ is a non-injective, increasing endomorphism. Hence, $H$ is not a max-core.

The case when $(2, 2) \not\in H$ is divided into two subcases depending on whether or not $(0, 0) \in H$. If $(2, 2) \not\in H$ and $(0, 0) \not\in H$, then $H$ is a max-core since if $(1, 1) \not\in H$ there are no non-injective endomorphisms (the only non-identical endomorphism swaps 1 and 2) and if $(1, 1) \in H$ the only non-injective endomorphism is $\text{id}[2 \mapsto 1]$. In both cases, let $\mu(3) = \mu(1) = \mu(0) = 0$, and $\mu(2) = 1$, and apply Lemma 6.5 to get a polynomial-time reduction from $\text{Max Sol}(H \cup C_A)$ to $\text{Max Sol}(H)$.

If $(2, 2) \not\in H$, $(0, 0) \in H$, and $(1, 1) \in H$, then $H$ is a max-core, and by considering the set of endomorphisms on $H$ it is easy to check that applying Lemma 6.5 with $\mu(3) = \mu(0) = 0$, and $\mu(2) = \mu(1) = 1$, we again have a polynomial-time reduction from $\text{Max Sol}(H \cup C_A)$ to $\text{Max Sol}(H)$.

Finally, if $(2, 2) \not\in H$, $(0, 0) \in H$, and $(1, 1) \not\in H$, then $H$ is the graph in Figure 6.10. By following the steps in the proof of Lemma 6.21, we find that $\text{Max Sol}(g(H))$ is polynomial-time reducible to $\text{Max Sol}(H)$ for $g = \text{id}[1 \mapsto 0, 2 \mapsto 3]$. Since the former problem is $\textbf{NP}$-hard (Proposition 2.29) it follows that both $\text{Max Sol}(H)$ and $\text{Max Sol}(H \cup C_A)$ are $\textbf{NP}$-hard. ■
6. Constant Production

From the preceding lemmas, we now know that the remaining graphs \( H \) must have \( N(3) = \{0\} \) and either \( 1 \notin N(0) \) or \( 2 \notin N(0) \), i.e., \( H \) must be a (partially reflexive) path.

**Lemma 6.52** If \( N(3) = \{0\} \) and either \( 1 \notin N(0) \) or \( 2 \notin N(0) \), then \( \text{MAX SOL}(H) \) is polynomial-time equivalent to \( \text{MAX SOL}(H \cup C_A) \).

**Proof:** Since \( H \) is connected, it must be a (partially reflexive) path. According to Lemma 6.28 we may assume that \( \{3\} \in H \). Furthermore, since \( N(3) = \{0\} \) we can assume that \( \{0\} \in H \). We divide the remaining analysis into two cases depending on whether there is a loop at 0 or not.

If \((0, 0) \in H\), then the subgraph of \( H \) induced by \( N(0) \) is a three vertex graph \( H' \) on the vertex set \( \{0, c, 3\} \), where \( c \) can be either 1 or 2. If \( c = 2 \), we note that, by the assumption that \( H \) is a max-core, we must have \((2, 2) \notin H'\). Thus, there are three possible graphs \( H' \), namely \( c = 2 \), \( c = 1 \) with \((1, 1) \in H' \), and finally \( c = 1 \) with \((1, 1) \notin H \). By Theorem 6.35, it follows that \( \text{MAX SOL}(H') \), and thereby \( \text{MAX SOL}(H) \) is \( \text{NP} \)-hard for these three graphs.

Now, consider the case where \((0, 0) \notin H \), and assume that \( 2 \in N(0) \). By the assumption that \( H \) is a max-core, we have \((2, 2) \notin H \). Therefore, \( N^3(3) = \{0, 1\} \), and by Lemma 6.28, we can assume that \( \{1\} \in H \). Moreover, \( N(0) \cap N(1) = \{2\} \), which implies that \( \{2\} \) can be assumed to be in \( H \) as well. The case where \( 1 \in N(0) \) can be handled by similar reasoning.

In the proofs of Lemma 6.53 through 6.56, we will always assume that the vertex 3 is not looped, since we are considering max-cores. Furthermore, we will use a general result for the retraction problem, which states that \( \text{RET}(H) \) is \( \text{NP} \)-complete whenever the looped vertices in \( H \) induce a disconnected graph, cf. [130]. Thus, by Lemma 6.24 we know that \( \text{MAX SOL}(H) \) is \( \text{NP} \)-hard whenever the looped vertices of \( H \) induce a disconnected graph. We conclude that it is sufficient to consider the cases where the looped vertices in \( H \) induce a connected graph.

By Lemma 6.28, we can always assume that we have access to the constant relation \( \{3\} \). Using the neighbourhood of 3, and another application of Lemma 6.28, we can also assume that we have the constant relation \( \{(\max N(3))\} \) in \( H \).
Lemma 6.53 Let $H$ be a max-core on $A = \{0, 1, 2, 3\}$ which is a path. Then, $\text{Max Sol}(H)$ is in $\text{PO}$ if $H$ is an irreflexive path, and $\text{NP}$-hard otherwise.

Proof: Proposition 6.27 shows that $\text{Max Sol}(H)$ is in $\text{PO}$ if $H$ is an irreflexive path. Hence, we can assume that $H$ contains a looped vertex. We will use the fact that $H^3 = (V, E^3)$ is the complete graph on four vertices where a vertex $v \in V$ is looped precisely when $v$ itself, or one of its neighbours, is looped.

Assume that $3$ has no looped neighbour. Then, the max-core of $H^3$ is either $R_{2,3}$ (if 2 is looped) or the graph $H_2$ in Figure 6.9, with $a_1 = 1$, $a_2 = 2$, and $a_3 = 3$. Both of these are $\text{NP}$-hard.

Now assume instead that 3 has a looped neighbour $c_3$. If $c_3$ is a leaf, then the graph induced by $N(c_3)$ is $R_{c_3,3}$ which is $\text{NP}$-hard by Proposition 2.29. Otherwise, the graph $H'$ induced by $N(c_1)$ is either $H_1$ or $H_2$ in Figure 6.11. If $H' = H_1$ or $H' = H_2$ with $c_2 < c_3$, then the max-core of $H'$ is again $R_{c_3}$.

\begin{center}
\begin{tabular}{ccc}
$c_2$ & $c_1$ & 3 \\
\hline
\end{tabular} & \begin{tabular}{ccc}
$c_2$ & $c_1$ & 3 \\
\hline
\end{tabular} & \begin{tabular}{ccc}
2 & 0 & 3 & 1 \\
\hline
\end{tabular}
\end{center}

$H_1$ \hspace{1cm} $H_2$ \hspace{1cm} $H_3$

Figure 6.11: $H_1$ and $H_2$ are the two possible graphs induced by $N(c_1)$. $H_3$ is the original max-core when $(c_1, c_2) = (0, 2)$ in $H_2$.

and $\text{NP}$-hardness follows from Proposition 2.29. If $H' = H_2$ with $c_2 > c_3$, the we have the possibilities $(c_1, c_2) \in \{(0, 1), (0, 2), (1, 2)\}$.

- If $(c_1, c_2) = (0, 1)$, we can use the result of Theorem 6.35 for the graph $H_5$ with $a_3 = 3$, $a_2 = 1$ and $a_1 = 0$.
- The case $(c_1, c_2) = (1, 2)$ cannot occur since $H$ is supposed to be a max-core; either $\text{id}|_{0 \mapsto 2}$ or $\text{id}|_{0 \mapsto 1}$ is an increasing endomorphism on $H$, depending on where 0 is attached.
- Finally we consider $(c_1, c_2) = (0, 2)$. In this case, the graph $H$ must be the graph $H_3$ in Figure 6.11. Here, $H^3 = A^2 \setminus \{(1, 1)\}$, which implies that $H^3_{|N(3)} = R_{0,1}$, and we are done.

This finishes the proof of the lemma.

\begin{flushright}
\blacklozenge
\end{flushright}

Lemma 6.54 Let $H$ be a graph on $\{0, 1, 2, 3\}$ which is a tree but not a path. Then, $H$ is not a max-core.
6. Constant Production

Proof: If \( H \) is a tree, then there is a unique centre vertex and three leaves. Either at least two of the leaves have loops, or at least two of the leaves do not have loops. In either case, \( H \) is not a max-core.

We can now assume that \( H \) contains a cycle.

**Lemma 6.55** Let \( H \) be a max-core on \( A = \{0, 1, 2, 3\} \) which is a (partially reflexive) 4-cycle. Then, \( \text{MAX SOL}(H) \) is \( \text{NP-hard} \).

**Proof:** By [130] we only need to consider the case when the looped vertices induce a connected graph. Let the vertices of \( H \) be labelled as in Figure 6.12(a). Vertex \( c \) must have a loop, or else \( \text{id}[c \mapsto 3] \) is an increasing endomorphism, and \( H \) is not a max-core. Similarly, precisely one of the vertices \( a \) and \( b \) must have a loop, and without loss of generality, we may assume that \( a \) has a loop. Furthermore, \( b > a \), or else \( \text{id}[b \mapsto a] \) is an increasing endomorphism.

If \( H \) is the graph shown in Figure 6.12(b), then the unary relation \( U = \{(1), (2), (3)\} \) can be implemented by Lemma 6.29. Now, \( E_{[1,2,3]} \) is the graph \( H_2 \) in Figure 6.9, which is \( \text{NP-hard} \) by Theorem 6.35.

Otherwise, let \( H' \) be the graph induced by \( N_H(a) \). If \( a > c \), then the max-core of \( H' \) is \( R_{a,3} \) which is \( \text{NP-hard} \) by Proposition 2.29. If not, we are left with the case \( a = 0, b = 2 \) and \( c = 1 \), and Theorem 6.35 shows that \( \text{MAX SOL}(H') \) is \( \text{NP-hard} \).

The remaining case is when \( H \) contains a (partially reflexive) 3-cycle.

**Lemma 6.56** Let \( H \) be a max-core on \( A = \{0, 1, 2, 3\} \) which contains a (partially reflexive) 3-cycle. Then, \( \text{MAX SOL}(H) \) is \( \text{NP-hard} \).

**Proof:** By Vikas’ classification [130] of retraction for all 4-vertex graphs and the reduction from retraction to \( \text{MAX SOL} \) in Lemma 6.24 we know that \( \text{MAX SOL}(H) \) is \( \text{NP-hard} \) when \( H \) is reflexive and contains a 3-cycle. Hence, we can assume that at least one of the vertices 0, 1 and 2 is looped.

---

Figure 6.12: a) The labelling of the 4-cycle, and b) the final case considered in Lemma 6.55.
6.4. Discussion

We have given a condition (6.13) on valued constraint languages \( \Gamma \) which for the problem \( \text{Max Sol}(\Gamma) \) mimics certain aspects of the core concept for \( \text{Csp}(\Gamma) \). For a constraint language \( \Gamma \) over \( A \) with an injective valuation \( \nu \), the condition is that there exists a \( \mu : A \to \mathbb{Q}_{\geq 0} \) such that for all non-surjective endomorphisms \( f \), \( \langle \mu, \nu \rangle > \langle \mu, f(\nu) \rangle \). We have shown that if
6. Constant Production

this condition holds, then we can produce all constant unary relations, and
that if the condition does not holds, then we can derive the computational
complexity from that of the strictly smaller endomorphic images of $\Gamma$.

We can make the connection between condition (6.13) and cores more
explicit. Although Lemma 6.20 requires $\nu$ to be injective, we can still consider
what happens if we allow the constant valuation $\nu \equiv 1$ in (6.13). In this
case, we find that the condition is satisfied if and only if all endomorphisms
of $\Gamma$ are surjective, hence we recover the definition of an ordinary core as a
special case. An interesting open question is how to make Lemma 6.20 work
for non-injective valuations as well.

The immediate consequence of our result is that in the search for a di-
chotomy theorem for MAX Sol, we can shift attention to OAL MAX Sol,
i.e. MAX Sol$(\Gamma \cup C_A)$. This provides a close tie-in with the retraction
problem, as for any $\textbf{NP}$-hard problem $\text{Ret}(\Gamma)$, we have that OAL MAX Sol$(\Gamma)$
is also $\textbf{NP}$-hard. However, this only helps as far as the retraction prob-
lem is classified, and moreover, already for small graphs, we find examples
where instances, such as the graph $H_5$ in Theorem 6.35, are easy for the
retraction problem, but poses a subtle problem in our setting; for any injective
valuation, we can produce the constants for $H_5$, but the computational
complexity of MAX Sol$(H_5)$ still depends on the particular valuation.

Another implication of our result relates to Chapter 3. There we studied
the approximability of MAX Sol$(A)$, when $A$ is a strictly simple and sur-
jective algebra. We noted that this restriction is, at first sight, less natural
in the case of MAX Sol than for Csp. In particular, it was not clear how to
naturally impose surjectivity and simplicity. Here, we see that if we are willing
to settle for a classification of languages as solvable in polynomial time,
or $\textbf{NP}$-hard, we can in fact ensure surjectivity by the result of this chapter:
when we have all constants in the constraint language, the corresponding
algebra is even idempotent. For an algebra $A$ with only one-element proper
subalgebras, this also implies simplicity: let $A'$ be a homomorphic image of
$A$ under an algebra homomorphism $\pi$. Then, $\pi$ induces an equivalence relation
$\varrho$ by $\varrho(a, b) \iff \pi(a) = \pi(b)$, and it is not hard to show that $\varrho \in \text{Inv}(A)$.
Let $U$ be an equivalence class of $\varrho$, and let $u$ be an element of $U$. Since $A$ is
idempotent, $\{u\} \in \text{Inv}(A)$, and the unary relation $U$ can be defined by the
pp-formula $U(u) \equiv \varrho(x, y) \land \{u\}(y)$. Hence, $U$ supports a subalgebra of
$A$. By the assumption that all proper subalgebras of $A$ are one-element, we
conclude that $U = \{u\}$, or $U = A$, where $A$ is the domain of $A$. Therefore,
either $\pi$ is an isomorphism, or $A'$ is a one-element algebra.

Finally, we consider the problem of deciding the condition (6.13) for a
given constraint language. The solutions $\mu$ to this equation determine an
open cone in the space of weight functions $\mu : A \rightarrow \mathbb{Q}_{\geq 0}$, and checking
the condition amounts to verifying that this cone is non-empty. In the
proof of Proposition 6.34, we used a variation of the Farkas lemma to do this
for varying $\nu$. We have noted that our method is also applicable to
MAX AW Sol$(\Gamma)$, i.e. when we allow arbitrary weights in the instances.
this case, the variation of the Farkas lemma applicable leads to a system of equalities instead of inequalities. In general, we would like to answer questions such as when a constraint language $\Gamma$ satisfies condition (6.13) for varying $\nu$. The special case for general even cycles $C_{2k}$ is a good starting point. Here, the endomorphisms are easily determined, but the size of the resulting system of inequalities grows exponentially in $k$.\[\textendash\]
Chapter 7

Conservative Constraint Languages and Arbitrary Weights

When facing an seemingly insurmountable task (such as that of completely classifying the computational complexity of MaxSol), there are two frequently applied approaches. The first is to restrict the task to something more manageable, and the second is, somewhat counter-intuitively, to generalise it. The idea of the latter approach is that by introducing some more, but not too much, freedom into the problem at hand, we can remove some rough corners and outliers from the original problem. In this chapter, we apply both of these approaches to the classification task of MaxSol. The generalisation will be to allow arbitrary weights, i.e. to study MaxAWSol in place of MaxSol. As we will see, this does indeed remove some of the rough corners. More specifically, as we have observed in the previous chapters, the choice of valuation on the constraint language often has decisive implications on the complexity of the resulting problem. Combined with the restriction to consider only conservative constraint languages, to be defined below, the freedom of arbitrary weights will in fact remove the dependence on the particular valuation used. As we move on, we will see how these choices suggest an investigation of the close relationship between this variation of MaxSol and the MinHom-problem encountered in Chapter 6, and ultimately, to a proof that the two are in fact polynomial-time equivalent.

Conservative constraint languages

A constraint language which contains all unary relations on the domain is called a conservative constraint language. For an actual instance, it is sufficient to have one unary constraint per instance variable. Indeed, assume an instance which contains two unary constraints, $u_1(v)$ and $u_2(v)$
on the variable $v$. Then, we can replace these two constraints with the single constraint $u(v) = u_1(v) \cap u_2(v)$ and obtain an instance with the same set of solutions. To make this more explicit, we introduce additional list components into the instance, one for each variable. Formally, a list $L(v)$ for a variable $v$ is a subset of the domain. The list $L(v)$ indicates the domain values that are allowed for $v$ and one can simply regard it as a unary constraint with a special name. We define List Max AW Sol, the restriction of Max AW Sol to conservative constraint languages, as follows:

**Definition 7.1** List Max AW Sol($\Gamma$), over the constraint language $\Gamma$, is the maximisation problem with

**Instance:** A tuple $(V, A, C, L, \omega)$, where $A$ is a finite subset of $\mathbb{N}$, $(V, A, C)$ is a CSP instance over $\Gamma$, $L : V \to 2^A$ is a function from $V$ to subsets of $A$, and $\omega : V \to \mathbb{Q}$ is a weight function.

**Solution:** An assignment $f : V \to A$ to the variables such that all constraints are satisfied and such that $f(v) \in L(v)$ for all $v \in V$.

**Measures:** $\sum_{v \in V} \omega(v) \cdot f(v)$

In the case of List Max AW Sol, we can either consider the problem as a restriction to conservative constraint languages, or as the result of introducing a list component in the instance. For other problems, the list view is the more natural one, e.g. in cases when the constraint languages are graphs, as in the list homomorphism problem, L-HOM [50, 51].

Conservative constraint languages are quite well-behaved and have received previous attention in different guises [18, 126]. Since it is possible to induce any subset of the domain, one immediately obtains a set of necessary conditions for tractability; each induced constraint language must be tractable. In particular, since two-element constraint languages are well understood, via Post’s lattice, one can deduce a set of necessary conditions based on these two-element subsets. The computational complexity of the classical CSP decision problem has been classified for conservative constraint languages [18] and for a generalisation of Min Hom, which is inherently conservative, a classification was recently obtained by Takhanov [126, 127].

**Relation to Min Hom**

The graph problem Min Hom was introduced by Gutin et al. [67] to model problems in defence logistics. The computational complexity was determined for undirected graphs with loops allowed [62] and in a series of papers [61, 63, 64, 65, 66]. Gutin and co-authors decided the computational complexity for an increasing number of families of directed graphs. The final dichotomy result on directed graphs was proved by Rafiey and Hell [115].
7. Conservative Constraint Languages and Arbitrary Weights

The study of List Max AW SOL was initiated by Jonsson et al. [85], where they show that for undirected graphs (with loops allowed), the computational complexity coincides with that of Min Hom. This is somewhat surprising, since Min Hom clearly has more general means of constructing instances. The proof of this result is given in Section 7.2.

Since the List Max AW SOL has a natural generalisation to arbitrary constraint languages, Jonsson et al. were led to introduce the following generalisation of Min Hom:

**Definition 7.2** Min Hom$(\Gamma)$, over the constraint language $\Gamma$, is the minimisation problem with

**Instance**: Tuple $(V, A, C, \{c_a\}_{a \in A})$, where $(V, A, C)$ is a CSP instance over $\Gamma$, and $c_a : V \rightarrow Q_{\geq 0} \cup \{\infty\}$.

**Solution**: An assignment $f : V \rightarrow A$ to the variables such that all constraints are satisfied.

**Measure**: $\sum_{v \in V} c_{f(v)}(v)$

It was noted [85] that List Max AW SOL$(\Gamma)$ and Min Hom$(\Gamma)$ are polynomial-time equivalent for $\Gamma$ over a Boolean domain. Lemma 7.5 proves this directly. The following question practically suggests itself:

**([85])** Is it the case that the complexity of List Max AW SOL$(\Gamma)$ and Min Hom$(\Gamma)$ are equal for all constraint languages $\Gamma$?

This question is answered in the affirmative in Section 7.3, using recent progress on Min Hom by Takhanov [126, 127].

7.1 Basic reductions

In this section we give two reductions. Proposition 7.3 is a general reduction from List Max AW SOL to Min Hom. Proposition 7.4 goes in the opposite direction, but puts certain restrictions on the Min Hom-instances. Despite this apparent lack of generality, it turns out that this reduction suffices for the proofs of two following sections.

In the following proposition, note in particular how the valuation $\nu$ on $\Gamma$ is completely absorbed by the more expressive weight functions $\{c_a\}$ of the Min Hom-framework. Conversely, the statement in Proposition 7.4 holds for any injective valuation of the constraint language. This provides the basis for the observation in the introduction, that the simultaneous restriction to conservative constraint languages and generalisation to arbitrary weighted variables remove the impact of specific valuations. Note that the injectivity of the constraint language is necessary. For example, take the relation $\rho = \{(a, a), (a, b), (b, a)\}$. For $\nu(a) \neq \nu(b)$, we have that List Max AW SOL$(\{R\}, \nu)$ is NP-hard. But for $\nu(a) = \nu(b)$, this problem is trivially solvable, simply map all variables to the element $a$. 
Proposition 7.3 Let \((\Gamma, \nu)\) be a valued constraint language. Then, the problem \textsc{List Max AW Sol}(\Gamma, \nu) is polynomial-time reducible to the problem \textsc{Min Hom}(\Gamma).

**Proof:** Let \(\mathcal{I} = (V, C, A, L, \omega)\) be an instance of \textsc{List Max AW Sol}(\Gamma). We modify it to an instance \(\mathcal{J} = (V, C, A, \{c_a\}_{a \in A})\) of \textsc{Min Hom}(\Gamma) as follows. Let \(Z = \max_{v \in V} \omega(v)\) and define

\[
c_a(v) = \begin{cases} 
(Z - \omega(v)) \cdot \nu(a) & \text{if } a \in L(v) \\
\infty & \text{otherwise.} 
\end{cases}
\]  

(7.1)

Thus, if \(\text{Opt}(\mathcal{J})\) is finite, then a solution to \(\mathcal{J}\) yields a solution to \(\mathcal{I}\) with

\[\text{Opt}(\mathcal{J}) = |V| \cdot \sum_{a \in A} \nu(a) \cdot Z - \text{Opt}(\mathcal{J}).\]

The following result is a generalisation of the reductions from \textsc{Min Hom} to \textsc{List Max AW Sol} given by Jonsson et al. [85].

Proposition 7.4 Let \(\Gamma\) be a constraint language. The problem \textsc{Min Hom}(\Gamma) restricted to instances for which each cost function \(c_a\) takes at most two finite values, is polynomial time reducible to \textsc{List Max AW Sol}(\Gamma, \nu), for any injective valuation \(\nu\) on \(\Gamma\).

**Proof:** Let \(\mathcal{J} = (V, C, A, \{c_a\}_{a \in A})\) be an instance of \textsc{Min Hom}(\Gamma). We construct an instance \(\mathcal{I} = (V, C, A, L, \omega)\) of \textsc{List Max AW Sol}(\Gamma, \nu) as follows. For each \(v \in V\), define \(L(v) = \{a \in A \mid c_a(v) \neq \infty\}\) and note that \(|L(v)| \leq 2\) by assumption. For a fixed \(v\), if \(L(v) = \{a_v\}\), then we unconditionally assign \(v\) to \(a_v\) and propagate this value to all constraints involving \(v\). Thus, we will assume that \(L(v) = \{a_v, b_v\}\) for all \(v \in V\). It is clear that an assignment from \(V\) to \(A\) is a feasible solution to \(\mathcal{I}\) if and only if it is a solution to \(\mathcal{J}\) of measure strictly smaller than \(\infty\). Ideally we would like the assignment \(v \mapsto a_v\) to contribute precisely \(c_{a_v}(v)\) to the measure of a solution to \(\mathcal{I}\), and similarly, \(v \mapsto b_v\) to contribute \(c_{b_v}(v)\). The following almost does the trick:

\[
\omega(v) = \frac{c_{a_v}(v) - c_{b_v}(v)}{\nu(a_v) - \nu(b_v)}. 
\]

Here, of course, the injectivity of \(\nu\) is crucial. Under the assumption that \(v \mapsto a_v\), we can express \(v\)'s contribution to the measure as follows:

\[
\omega(v) \cdot \nu(a_v) = \omega(v) \cdot (\nu(a_v) - \nu(b_v)) + \omega(v) \cdot \nu(b_v) = 
\]

\[
c_{a_v}(v) - c_{a_v}(v) + \omega(v) \cdot \nu(b_v). 
\]

Given a solution \(s : V \rightarrow A\) to \(\mathcal{I}\) and a variable \(v \in V\), we thus have \(\omega(v) \cdot \nu(s(v)) = -c_{a_v}(v) + c_{b_v}(v) + \omega(v) \cdot \nu(b_v)\). Let \(m(s)\) denote the measure of \(s\) as a solution to the \textsc{List Max AW Sol}-instance and let \(m'(s)\) denote
7. Conservative Constraint Languages and Arbitrary Weights 165

the measure of s as a solution to the Min Hom-instance. The two measures are then related as follows:

\[ m(s) = \sum_{v \in V} \omega(v) \cdot \nu(s(v)) = \]

\[ -\sum_{v \in V} c_s(v) + \left( \sum_{v \in V} c_{b_s}(v) + \omega(v) \cdot \nu(b_v) \right) = -m'(s) + C, \]

where C can be calculated in polynomial time from \( \mathcal{J} \). Hence, if there is a solution to \( \mathcal{I} \), then \( \text{Opt}(\mathcal{J}) = C - \text{Opt}(\mathcal{I}) \), otherwise \( \text{Opt}(\mathcal{J}) = \infty \). □

**Lemma 7.5** Let \( \Gamma \) be a Boolean constraint language. Then, Min Hom(\( \Gamma \)) and List Max AW Sol(\( \Gamma \)) are polynomial-time equivalent.

**Proof:** Since \( \Gamma \) is a constraint language over a two-element domain, we find that the restriction in Proposition 7.4 is trivially satisfied. Hence, the lemma immediately follows from Proposition 7.3 and Proposition 7.4. □

### 7.2 Undirected graphs with loops

In this section we set out to prove a dichotomy theorem for problems of the form List Max AW Sol(H), where \( H \) is an undirected graph with loops allowed. The result is inspired by the corresponding result for Min Hom [62], and is stated in terms of proper interval graphs and bigraphs.

**Definition 7.6** Let \( F = \{I_1, \ldots, I_k\} \) be a family of intervals on the real line. A graph \( G \) with \( V(G) = F \) and \( (I_i, I_j) \in E(G) \) if and only if \( I_i \cap I_j \neq \emptyset \) is called an interval graph. If the intervals are chosen to be inclusion-free, \( G \) is called a proper interval graph.

**Definition 7.7** Let \( F_1 = \{I_1, \ldots, I_k\} \) and \( F_2 = \{J_1, \ldots, J_l\} \) be two families of intervals on the real line. A graph \( G \) with \( V(G) = F_1 \cup F_2 \) and \( (I_i, J_j) \in E(G) \) if and only if \( I_i \cap J_j \neq \emptyset \) is called an interval bigraph. If the intervals in each family are chosen to be inclusion-free, \( G \) is called a proper interval bigraph.

Interval graphs are necessarily reflexive, while interval bigraphs are ir-reflexive and bipartite. The main result can now be stated as follows:

**Theorem 7.8** Let \( H \) be an undirected graph with loops allowed. Then, List Max AW Sol(H) is tractable if all components of \( H \) are proper interval graphs or proper interval bigraphs. Otherwise, List Max AW Sol(H) is NP-hard.
Theorem 7.8 describes a dichotomy for \( \text{LIST MAX AW SOL} \) which is identical to that of \( \text{MIN HOM} \) [62]. We therefore have the following as an immediate corollary.

**Corollary 7.9** Let \( H \) be an undirected graph with loops allowed. Then, \( \text{LIST MAX AW SOL}(H) \) is polynomial-time equivalent to \( \text{MIN HOM}(H) \).

The tractable cases of Theorem 7.8 follow from Theorem 1.1 in [62] combined with the reduction in Proposition 7.3. The remaining part of this section is devoted to showing the other direction, i.e., if some component of \( H \) is neither a proper interval graph nor a proper interval bigraph, then \( \text{LIST MAX AW SOL}(H) \) is \textbf{NP}-hard. We will assume that the input instance is connected. If it is not, then we can solve each connected component separately and add up the measures of the obtained solutions.

The following lemma is trivial, but essential.

**Lemma 7.10** Let \( H \) be an undirected graph and \( U \subseteq V(H) \). Then, \( \text{LIST MAX AW SOL}(H|_U) \) is polynomial-time reducible to \( \text{LIST MAX AW SOL}(H) \).

**Proof:** Let \( I = (V, A, C, L, \omega) \) be an instance of \( \text{LIST MAX AW SOL}(H|_U) \). We create a new instance \( J = (V, A, C, L', \omega) \) of \( \text{LIST MAX AW SOL}(H) \), where \( L'(v) = L(v) \cap U \). Then, there is a solution to \( J \) of measure \( m \) if and only if there is a solution to \( I \) of the same measure. \( \blacksquare \)

**Lemma 7.11** Let \( H \) be a connected undirected graph in which there exists both loop-free vertices and vertices with loops. Then, \( \text{LIST MAX AW SOL}(H) \) is \textbf{NP}-hard.

**Proof:** We will reduce from the problem of finding a maximal independent set in an undirected graph \( G \), which is \textbf{NP}-hard. The reduction is straightforward to do directly, but we do it using Proposition 7.4 in order to illustrate its use. To this end, we first reduce maximal independent set to \( \text{MIN HOM}(H) \). Let \( V = V(G) \), \( A = V(H) \) and \( C = \{(H(x, y) | (x, y) \in E(G)) \}. Create an instance \( J = (V, A, C, \{c_{a_1}\}_{a \in A}) \) of \( \text{MIN HOM}(H) \) as follows. By assumption, we can find \( a_1, a_2 \in V(H) \) such that \( H|_{\{a_1, a_2\}} = \{(a_1, a_1), (a_1, a_2), (a_2, a_1)\} \). For each \( v \in V \), let

\[
c_a(v) = \begin{cases} 
1 & \text{if } a = a_1, \\
0 & \text{if } a = a_2, \text{ and} \\
\infty & \text{otherwise.}
\end{cases}
\]

It is obvious that any independent set \( I \) of \( G \) yields a solution to \( J \) of measure \( |V \setminus I| \), by assigning the vertices of the independent set to \( a_2 \), and all other vertices to \( a_1 \). Conversely, any solution \( f \) to \( J \) of measure \( m \) induces an independent set \( f^{-1}(a_2) \), with \( |f^{-1}(a_2)| = |V| - m \). Therefore, a
7. Conservative Constraint Languages and Arbitrary Weights

minimal solution to \( J \) induces an independent set of maximal size. Now, we invoke Proposition 7.4 to finish the reduction to \( \text{List Max AW Sol}(H) \).

The following theorem by Hell and Huang characterises proper interval bigraphs in terms of the absence of certain induces subgraphs.

**Theorem 7.12 (P. Hell, J. Huang [69])** A bipartite graph \( H \) is a proper interval bigraph if and only if it does not contain an induced cycle of length at least six, or a bipartite claw, or a bipartite net, or a bipartite tent (cf. Figure 7.1.)

![Figure 7.1: a) bipartite claw, b) bipartite net, c) bipartite tent.](image)

In light of the preceding theorem, it is now clear that in order to prove the part of Theorem 7.8 which refers to proper interval bigraphs, we must show that \( \text{List Max AW Sol}(H) \) is \( \text{NP} \)-hard for cycles of length at least six and for the graphs in Figure 7.1. By Lemma 7.10, we can then induce such a graph in any bipartite graph which is not a proper interval bigraph. We start with the cycles.

**Lemma 7.13** Let \( H \) be a cycle of length at least six. Then, \( \text{List Max AW Sol}(H) \) is \( \text{NP} \)-hard.

**Proof:** We will reduce from the retraction problem \( \text{Ret}(H) \) (Definition 6.23.) This problem is known to be \( \text{NP} \)-hard for cycles of length at least six [51]. Let \( G \) be a graph such that \( H \subseteq G \), and construct the instance \( \mathcal{I} = (V(G), V(H), \{H(x, y) \mid (x, y) \in E(G)\}, L, \omega) \), where

\[
L(v) = \begin{cases} 
\{v\} & \text{if } v \in V(H) \\
V(H) & \text{otherwise},
\end{cases}
\]

and let \( \omega \) be arbitrary. Then, \( H \) is a retract of \( G \) if and only if \( \mathcal{I} \) has any solution. ■
The proofs of the three following lemmas all follow the ideas from [62]. They are done by reduction from the problem of finding a maximum independent set in a 3-partite graph $G$. This problem is known to be NP-hard, even when the three partite sets are known (they can be calculated from $G$ in polynomial time). The proofs by Jonsson et al. [85] goes by reduction directly from the independent set problem to List Max AW Sol, which introduces a good amount of clutter in the choice of weights for the variables. We avoid this by keeping closer to the original proofs for Min Hom, and using Proposition 7.4.

**Lemma 7.14** Let $H$ be the reflexive claw graph shown in Figure 7.2. Then, List Max AW Sol$(H)$ is NP-hard.

![Figure 7.2: The reflexive claw graph.](image)

**Proof:** Let $G$ be a 3-partite graph with partite sets $V_1$, $V_2$, and $V_3$. Let $V = V(G)$, $A = V(H) = \{x_1, x_2, x_3, x_4\}$ and $C = \{(u, v) \mid (u, v) \in E(G)\}$. We create an instance $\mathcal{J} = (V, A, C, \{c_a\}_{a \in A})$ of Min Hom$(H)$ as follows. For $j = 1, 2, 3$, and each $v \in V_j$, let

$$c_a(v) = \begin{cases} 1 & \text{if } a = x_4, \\ 0 & \text{if } a = x_j, \text{ and} \\ \infty & \text{otherwise.} \end{cases}$$

Let $I = I_1 \cup I_2 \cup I_3$ be an independent set of $G$, where $I_j \subseteq V_j$ for $j = 1, 2, 3$. Then, $f(v) = x_1$ if $v \notin I$ and $f(v) = x_j$ if $v \in I_j$ is a solution to $\mathcal{J}$ of measure $|V \setminus I|$. Conversely, any solution $f$ to $\mathcal{J}$ of measure $m$ induces an independent set $f^{-1}(\{x_1, x_2, x_3\})$ of size $|V| - m$. Therefore, a minimal solution to $\mathcal{J}$ induces an independent set of maximal size. An application of Proposition 7.4 finishes the proof.

**Corollary 7.15** Let $H$ be the bipartite claw shown in Figure 7.1(a). Then, List Max AW Sol$(H)$ is NP-hard.

**Proof:** Let $V = \{x_1, x_2, x_3, x_4\}$ and let $H^2$ be the graph with vertex set $V(H)$ and edge set $\{(u, v) \mid \exists w : (u, w), (w, v) \in E(H)\}$. Then, $H^2|_V$ is isomorphic to the reflexive claw.
7. Conservative Constraint Languages and Arbitrary Weights

Lemma 7.16 Let $H$ be the bipartite net graph shown in Figure 7.1(b). Then, List Max AW Sol$(H)$ is NP-hard.

Proof: Let $G$ be a 3-partite graph with partite sets $X, Y$ and $Z$ and we create a new graph $G'$ from $G$ as follows. For each vertex $a$ in $X \cup Y$, we let $a$ be a vertex in $G'$. For each vertex $z$ in $Z$, we create a path $s_i^1 t^z s_2^z$ in $G'$. Finally for each edge $e \in E(G) \cap (X \times Y)$, we create a new vertex $m_e$ in $G'$. The edges in $G'$ are as follows:

$$E(G') = \{(x, m_e), (m_e, y) \mid e = \{x, y\} \in E(G) \cap (X \times Y)\} \cup \{(s_1^1, t^z), (t^z, s_2^z) \mid z \in Z\} \cup \{(x, s_1^1) \mid (x, z) \in E(G) \cap (X \times Z)\} \cup \{(s_2^z, y) \mid (z, y) \in E(G) \cap (Z \times Y)\}$$

Let $V = V(G'), A = V(H) = \{x_1, x_2, x_3, x_4\}$ and $C = \{(H(u, v) \mid (u, v) \in E(G'))\}$. The instance $\mathcal{J} = (V, A, C, \{c_a\}_{a \in A})$ of MIN HOM $(H)$ is completed by the following definitions. For $x \in X, y \in Y, z \in Z$, and $e \in E(G) \cap (X \times Y)$, let

$$c_a(x) = \begin{cases} 1 & \text{if } a = x_4, \\ 0 & \text{if } a = x_1, \\ \infty & \text{otherwise.} \end{cases} \quad c_a(y) = \begin{cases} 1 & \text{if } a = x_4, \\ 0 & \text{if } a = x_2, \\ \infty & \text{otherwise.} \end{cases}$$

$$c_a(m_e) = \begin{cases} 0 & \text{if } a = y_1, \\ 0 & \text{if } a = y_2, \\ \infty & \text{otherwise.} \end{cases} \quad c_a(t^z) = \begin{cases} 0 & \text{if } a = x_3, \\ 1 & \text{if } a = x_4, \\ \infty & \text{otherwise.} \end{cases}$$

$$c_a(s_1^1) = \begin{cases} 1 & \text{if } a = y_3, \\ 0 & \text{if } a = y_1, \\ \infty & \text{otherwise.} \end{cases} \quad c_a(s_2^z) = \begin{cases} 1 & \text{if } a = y_2, \\ 0 & \text{if } a = y_3, \\ \infty & \text{otherwise.} \end{cases}$$

Let $I = I_1 \cup I_2 \cup I_3$ be an independent set of $G$, where $I_1 \subseteq X, I_2 \subseteq Y$ and $I_2 \subseteq Z$. We can turn $I$ into a solution $f$ of $\mathcal{J}$ by assigning each variable in $X$ to $x_1$ if it is in the independent set and to $x_4$ otherwise and similarly for the variables in $Y$. The variables in the path corresponding to $z \in Z$ are given the values

$$f(s_1^1) = f(s_2^z) = y_3, \quad f(t^z) = x_4$$

when $z$ is in the independent set and

$$f(s_1^1) = y_1, \quad f(s_2^z) = y_2, \quad f(t^z) = x_3$$

otherwise. Finally, $m_e$ is assigned to either $y_1$ or $y_2$, depending on which of its neighbours is assigned to $x_4$. The measure of $f$ equals $|V| + |Z| - |I|$. Conversely, any solution $f$ to $\mathcal{J}$ of measure $m < \infty$ induces an independent set $f^{-1}(\{x_1, x_2\}) \cup \{z \in Z \mid f(s_1^1) = f(s_2^z) = y_3\}$ of size $|V| + |Z| - m$. Thus,
7.2. Undirected graphs with loops

a minimal solution to $J$ induces an independent set of maximal size. The lemma now follows from Proposition 7.4.

**Example 7.17** This is a good time to take a closer look at the numbers involved in an application of Proposition 7.4. We exemplify by the use in the previous proof, and concretise by fixing some values on the bipartite net graph, cf. Figure 7.3. Assuming we have created $G'$ as in the proof,

![Figure 7.3: The bipartite net graph with a valuation.](image)

we are left with the task of assigning lists and weights to the variables in $V(G')$ to complete a List Max AW Sol($H$)-instance. The lists are easily read off from the definition of the $c_w$-functions: $L(x) = \{x_1, x_4\}$ for $x \in X$, $L(m_e) = \{y_1, y_2\}$ for $e \in E(G) \cap (X \times Y)$, and so on. The weights are given by the difference between the two finite $c_w$-values divided by minus the difference between the corresponding $\nu$-values. They turn out to be

$$\omega(u) = \begin{cases} 
1/(\nu(x_1) - \nu(x_4)) = 1/(6 - 3) = 1/3 & \text{if } u \in X, \\
1/(\nu(x_2) - \nu(x_4)) = 1/(7 - 3) = 1/4 & \text{if } u \in Y, \\
1/(\nu(x_3) - \nu(x_4)) = 1/(1 - 3) = -1/2 & \text{if } u = t^2, \\
-1/(\nu(y_1) - \nu(y_3)) = -1/(5 - 4) = -1 & \text{if } u = s^1_1, \\
-1/(\nu(y_2) - \nu(y_3)) = -1/(2 - 4) = 1/2 & \text{if } u = s^2_2, \\
(0 - 0)/(\nu(y_2) - \nu(y_1)) = 0 & \text{if } u = m_e.
\end{cases}$$

Hence, we can calculate the contribution of a vertex $x \in X$ which is assigned to $x_4$ as $1/3 \cdot 6 = 2$. If, instead, it is assigned to $x_4$, then this contribution is $1/3 \cdot 3 = 1$. Similarly, for a vertex $y \in Y$, the corresponding contributions are $1/4 \cdot 7 = 7/4$ and $1/4 \cdot 3 = 3/4$. Note that the differences $(2 - 1, \text{and } 7/4 - 3/4, \text{respectively})$ are equal. This difference is the positive contribution of letting $x$ (or $y$) be in the independent set. We can also sum up the contributions for the possible assignments of $s^1_1, s^2_2$ and $t^2$. There are four assignments to consider when $z$ is left out of the independent set. Three of these contribute $-9/2$, and one contributes $-11/2$ (and never occurs in an optimal solution.) For a $z$ in the independent set, i.e. $\{s^1_1 \mapsto 4, t^2 \mapsto 3, s^2_2 \mapsto 4\}$, the contribution is $-7/2$.

Finally, let us calculate the constant $C = \sum_{v \in V(G')} c_w(v) + \omega(v) \cdot \nu(b_v)$, where, as before, we choose $a_v$ as the domain value of the first case listed.
7. Conservative Constraint Languages and Arbitrary Weights

in the definition of the \( c_\alpha \)-functions (and \( b_\alpha \) as the second.) Then, \( C = 2 \cdot |Z| + |X| \cdot 1/3 + 6 + |Y| \cdot 1/4 + 7 \cdot (-1/2 \cdot 1 + (-1) \cdot 5 + 1/2 \cdot 2) + 0 = 2 \cdot |X| + 7/4 \cdot |Y| - 5/2 \cdot |Z| \). Let \( f \) be a solution corresponding to the independent set \( I = I_x \cup [I_y \cup I_z] \). Then, \( m(f) = |I_x| \cdot 2 + |X \setminus I_x| \cdot 1 + |I_y| \cdot 7/4 + |Y \setminus I_y| \cdot 3/4 + |I_z| \cdot (-7/2) + |Z \setminus I_z| \cdot (-9/2) = |I_x| + |I_y| + |I_z| + (|X| + 3/4 \cdot |Y| - 9/2 \cdot |Z|). \) The same measure is obtained from (7.1) using \( m'(f) = |V| \cdot |Z| - |I| \) and the previously determined value of \( C \).

The proof of the following lemma is similar to those of Lemma 7.14 and 7.16, and is therefore omitted.

**Lemma 7.18** Let \( H \) be the bipartite test graph shown in Figure 7.1(c). Then, \( \text{LIST MAX AW SOL}(H) \) is \( \text{NP} \)-hard.

We are now equipped to finish the proof of Theorem 7.8.

**Proof:** [Theorem 7.8] It remains to show that \( \text{LIST MAX AW SOL}(H) \) is \( \text{NP} \)-hard if some component of \( H \) is neither a proper interval graph nor a proper interval bigraph. By Lemma 7.10, we can assume, without loss of generality, that \( H \) is itself connected. If \( H \) is neither loop-free nor reflexive, then \( \text{LIST MAX AW SOL}(H) \) is \( \text{NP} \)-hard by Lemma 7.11.

If \( H \) is loop-free and non-bipartite, we can reduce from the graph homomorphism problem which is \( \text{NP} \)-hard for non-bipartite loop-free graphs [71].

So assume that \( H \) is bipartite. If \( H \) is not a proper interval bigraph, then, due to Theorem 7.12, \( H \) has either an induced cycle of length at least 6, an induced bipartite claw, an induced bipartite net or an induced bipartite tent. In either case, we can use Lemma 7.10 to induce the particular graph. \( \text{NP} \)-hardness now follows from Lemma 7.13, Corollary 7.15, Lemma 7.16, or Lemma 7.18, respectively.

Finally, if \( H \) is reflexive, then it is either not an interval graph, or a non-proper interval graph. If \( H \) is not an interval graph, then we can reduce from the list homomorphism problem, \( \text{L-HOM}H \), which has been shown to be \( \text{NP} \)-hard for reflexive, non-interval graphs [50]. In the second case, Roberts [118] has shown that \( H \) must contain an induced (reflexive) claw, and Lemma 7.14 shows that the problem is then \( \text{NP} \)-hard.

**7.3 A complete classification**

Despite the additional constraint of only two finite values per function \( c_\alpha \) in Proposition 7.4, we saw in the previous section that it was nevertheless strong enough in order to show polynomial-time equivalence between \( \text{MIN HOM}(H) \) and \( \text{LIST MAX AW SOL}(H) \) for undirected graphs \( H \) with loops allowed. In fact, a recent classification of \( \text{MIN HOM}(\Gamma) \) by Takanov [126, 127] for an arbitrary constraint language \( \Gamma \) allows us to prove
the full conjecture, that \textsc{List Max AW Sol}(\Gamma) and \textsc{Min Hom}(\Gamma) are indeed polynomial-time equivalent. Takhanov originally studied the problem from the viewpoint of supervised learning. In this setting, the right-hand side can be seen as a priori knowledge and the measure is given by a sum over a set of tuples \((v, a, w)\), called the \textit{training set}, where the assignment \(v \mapsto a\) is given weight \(w\). Of course, given an arbitrary number of tuples, this is equivalent to a "\textsc{Max Hom}"-problem, which was also soon realised. Furthermore, the minimising and maximising versions of \textsc{Min Hom} are polynomial-time equivalent.

![Diagram](image)

**Figure 7.4:** The conservative relational clones on two elements.

In this section, we review the main points of Takhanov's proof, in order to properly work out the equivalence to \textsc{List Max AW Sol}. We stress that the proof is not ours. Takhanov's proof is based on the algebraic approach, and in particular uses relational clones \((\Gamma)\) in place of \(\Gamma\). For \textsc{Min Hom} and \textsc{List Max AW Sol} it is obvious that one can produce all unary constraints in polynomial time. Indeed, in the latter case we could define \textsc{List Max AW Sol}(\Gamma) as \textsc{Max AW Sol}(\Gamma \cup \mathcal{C}_A), where \(A\) is the domain of \(\Gamma\). Takhanov shows that for any finite subset \(\Gamma' \subseteq (\Gamma)\), the problem \textsc{Min Hom}(\Gamma') is polynomial-time reducible to \textsc{Min Hom}(\Gamma). For \textsc{List Max AW Sol}, we have Theorem 2.34. This forms the foundation of
the algebraic approach. The relational clone IR₂ is the unique minimal conservative relational clone. The lattice of all conservative relational clones are shown in Figure 7.4. This figure and the statement of the first theorem, which establishes necessary conditions for tractability of Min Hom and List Max AW Sol uses the naming convention for relational clones given in Section 2.6. The statement of the original theorem [126] uses a different convention (and also does not mention IR₂ as it is contained in the two other relational clones.)

The proof consists of four main components. The first is the classification of two-element subalgebras (Corollary 7.20) which in turn provides necessary conditions for tractability (Definition 7.21). The second is the construction of an associated graph, (M°, P), and a proof that unless this graph is bipartite, then List Max AW Sol is NP-hard (Proposition 7.22). Thirdly, it is shown that if the necessary conditions for tractability are met and the graph (M°, P) is bipartite, then the majority operation is a polymorphism of the constraint language. It is known (see for example [79]) that this implies that we only need to consider the binary constraints of the relational clone. The final step uses a graph associated to the remaining constraints, which we will call the microstructure graph (see Chapter 8) and proves that under the aforementioned conditions, the microstructure graph is a perfect graph. This allows an optimal solution to be found in polynomial time. The two last steps are summarised as Proposition 7.23.

**Theorem 7.19 (Theorem 5 in [126])** Over a two-element domain, the conservative relational clones IR₂, IM₂, and ID₁ are tractable for Min Hom. All other conservative relational clones are NP-hard.

Using Lemma 7.5, we have the following corollary:

**Corollary 7.20** Theorem 7.19 holds for List Max AW Sol.

Let B be a two-element domain, let fₓ and fᵧ be the two distinct binary, commutative and conservative operations on B, and let f : B³ → B be such that f(x, x, y) = f(y, x, x) = f(y, x, y) = y. Then, the relational clones IM₂ and ID₁ can be expressed as inv({fₓ, fᵧ}), and inv({}), respectively. Furthermore, from Figure 7.4 we see that IR₂ ⊆ IM₂ and IR₂ ⊆ ID₁. This, together with Corollary 7.20 implies that if some two-element subalgebra of A contains neither of the two sets {fₓ, fᵧ} or {f}, then List Max AW Sol(A) is NP-hard. We therefore have the following necessary conditions for tractability:

**Definition 7.21** Let A be a conservative clone over a domain A. If for every two-element subset B ⊆ A, A|₆ either contains both fₓ and fᵧ, or it contains a function f : B³ → B such that f(x, x, y) = f(y, x, x) = f(y, x, y), then A is said to satisfy the necessary local conditions.
Let $A$ be a conservative clone over a domain $A$ and let $M$ be the set of two-element subsets of $A$ which support a subalgebra containing both $f_A$ and $f_C$. Let $M^o = \{ (a, b) \mid (a, b) \in M \}$ be the corresponding set of ordered two-element subsets. Let $P \subseteq M^o \times M^o$ be defined by $((a, b), (c, d)) \in P$ if there is no $f \in A$ such that $f(a, b) = f(b, a) = b$ and $f(c, d) = f(d, c) = d$.

**Proposition 7.22** If the graph $(M^o, P)$ is not bipartite, then the problem \textsc{List Max AW Sol}$(\lnw(A))$ is \textbf{NP}-hard.

**Proof:** The proof of Theorem 7 in [126] shows that $\lnw(A)$ must contain a set of constraints $\{g_0, \ldots, g_{2k}\}$, where

$$g_i = \{ (a_i, a_{i+1}), (b_i, b_{i+1}), (a_i, b_{i+1}) \},$$

and the elements in $a_0, \ldots, a_{2k}, b_1, \ldots, b_{2k}$ are distinct, cf. Figure 7.5. It is also shown (Lemma 15.1 [126]), that such a constraint language is \textbf{NP}-hard for \textsc{Min Hom}. We must thus argue that the same holds in the case of \textsc{List Max AW Sol}. The proof is a reduction from the problem of finding a maximal independent set in a graph $G$ for which $G \to C_{2k+1}$, where $C_{2k+1}$ is the cycle on $2k + 1$ vertices. This is the extension of the problem in Section 7.2 of finding a maximal independent set in a 3-partite graph $G$, i.e., when $G \to C_3$. Takhanov shows that the generalised problem is still \textbf{NP}-hard even when the homomorphism to the cycle is known. Hence, we can assume that the preimages $V_0, \ldots, V_{2k}$ of the vertices in $C_{2k+1}$ under such an homomorphism are known.

Let $A = \{ a_0, \ldots, a_{2k} \}$ and $B = \{ b_0, \ldots, b_{2k} \}$. We construct an instance $J$ of \textsc{Min Hom}$(\{g_i\})$. Identify the variables of $J$ with the vertices of $G$, let the domain be $A \cup B$, and define the cost functions as follows:

$$e_a(v) = \begin{cases} 1 & \text{if } v \in V_i \text{ and } a = a_i, \\ 0 & \text{if } v \in V_i \text{ and } a = b_i, \\ \infty & \text{otherwise.} \end{cases}$$

For each $i = 0, \ldots, 2k$, and each (ordered) pair $(v_i, v_{i+1}) \in V_i \times V_{i+1}$, add the constraint $g_i(v_i, v_{i+1})$ to $J$. Here, the indices are assumed to be reduced modulo $2k + 1$.

![Figure 7.5: The relation $g_i$ represented by a directed graph.](image-url)
7. Conservative Constraint Languages and Arbitrary Weights

Let \( s : V(G) \rightarrow A \cup B \) be a solution to \( \mathcal{J} \). If \( m(s) < \infty \), then the set \( J = s^{-1}(B) \) is an independent set in \( G \), and

\[
m(s) = \sum_{v \in V(G)} c_{f(v)} = \sum_{i=0}^{2k} |s^{-1}(a_i)| = |V(G) \setminus J|.
\]

Conversely, any independent set \( I \) in \( G \) induces a solution by assigning \( f(v_i) = b_i \) if \( v_i \in I \) and \( f(v_i) = a_i \) otherwise. The measure of such a solution equals \( |V(G) \setminus I| \). Therefore, a minimal solution to \( \mathcal{J} \) produces a maximal independent set in \( G \). It remains to observe that the cost functions once again fulfill the condition of Proposition 7.4, and we are done.

The following proposition deals with the remaining tractable cases, which we get for free by Proposition 7.3. It shall be noted that the proof for the \( \text{Min Hom} \)-case is non-trivial; the necessary local conditions of Definition 7.21 and the bipartiteness property of \( (M^o, P) \) are used twice—first to prove that the constraint language contains a majority operation, which implies that the problem is polynomial-time equivalent to finding a maximal weighted, maximal sized clique in the microstructure of the binary constraints of the constraint language. The second time the necessary conditions and bipartiteness are used to show that the microstructure is a perfect graph. Finally, a modification of an algorithm for finding a maximal weighted clique in a perfect graph can be applied.

**Proposition 7.23** If \( A \) satisfies the necessary local conditions, and \( (M^o, P) \) is bipartite, then \( \text{List Max AW Sol}(\text{Inv}(A)) \) is tractable.

**Proof:** The corresponding result for \( \text{Min Hom} \) is shown in [126], hence the result follows by Proposition 7.3.

We now combine Corollary 7.20, Proposition 7.22 and Proposition 7.23 to show the following dichotomy of \( \text{List Max AW Sol} \):

**Theorem 7.24** If \( A \) satisfies the necessary local conditions, and \( (M^o, P) \) is bipartite, then \( \text{List Max AW Sol}(\text{Inv}(A)) \) is tractable. In all other cases \( \text{List Max AW Sol}(\text{Inv}(A)) \) is \( \text{NP-hard}. \)

Additionally, we have:

**Theorem 7.25** \( \text{List Max AW Sol}(\Gamma) \) and \( \text{Min Hom}(\Gamma) \) are polynomial-time equivalent for all constraint languages \( \Gamma \).

### 7.4 Conclusions

We started out by generalising and restricting the \( \text{Max Sol} \) problem in the hope of finding some new, simplifying structure of the problem. As it turned
out, this led us to the surprising link to the seemingly much more expressive problem Min Hom. For graphs, and in the end for arbitrary constraint languages, the restricted power of Proposition 7.4 was sufficient for proving the polynomial-time equivalence between the problems.

Takhanov’s proof of a dichotomy for Min Hom, and its immediate applicability to List Max AW Sol could be seen as the final word on the subject. Furthermore, in private communication, Takhanov has been able to modify his proof to work for the conservative Max Sol case, i.e. when only non-negative weights are allowed in the instances. This indicates that the crucial simplifying ingredient was for the constraint language to be conservative, as previously indicated by the successful classification of CSP for such languages [18]. However, for conservative Max Sol, and for Max AW Sol, the computational complexity for a constraint language is still sensitive to the particular valuation used, unlike the case of List Max AW Sol.

We conclude by mentioning that, in spite of the great leap forward in our understanding of List Max AW Sol provided by Takhanov’s result, the polynomial-time equivalence between Min Hom and List Max AW Sol remains intriguing, and it is still an open problem to find a “simple” reason for the existence of a reduction from Min Hom to List Max AW Sol, one which does not take the detour via a complete classification.
Chapter 8

Exponential-Time Algorithms

The general, non-parameterised Max Sol problem has no efficient (i.e. polynomial-time) algorithm, unless $P = NP$. In previous chapters, we have taken the approach of studying hardness at different granularities, for a constraint language parameterisation. Despite the conjectured theoretical obstacle to finding general, efficient algorithms for NP-complete problems, there has been significant progress on designing exponential-time algorithms. Actually, one may argue that this is a natural approach to take; although we a priori consider exponential-time algorithms inefficient, we also strongly believe that $P \neq NP$, hence this may very well be the only option.

As usual, one can approach this problem either from a practical, or a theoretical point of view. On the practical side, NP-hard problems need to be solved every day, and improvement in algorithms and hardware keep pushing the limit on the sizes of instances which can be handled. However, these algorithms are usually based on heuristics, and provide little guarantee on the running time. From the theoretical point of view, which is where our interest lies, one studies upper bounds on the worst case running time, but these bounds may hide large constant factors which make them unsuitable for immediate implementation.

It is common, when dealing with exponential-time algorithms, to not only ignore constant factors in the expressions for the time complexity, but also polynomial factors. To this end, let $O^*(e^n)$ denote the class of functions which asymptotically grows no faster than $e^n$, up to a polynomial factor in $n$. The goal is to find a low constant $c$ for a particular problem, in this case Max Sol. This can be achieved either by constructing a new algorithm, or by tightening the analysis of some already existing one. In this chapter, we will be using a different parameterisation of our problems, in which we impose an upper bound on the domain size and constraint arity. Such parameterisations are more convenient to work with when constructing
this type of algorithms.

**Definition 8.1** A \((d, l)\)-constraint satisfaction instance, \(I\), is a 3-tuple 
\((V, D, C)\), where

- \(V\) is a finite set of variables,
- \(D\) is a finite set of values, with \(|D| \leq d\), and
- \(C\) is a finite set of constraint applications \(\{(s_1, \varrho_1), \ldots, (s_q, \varrho_q)\}\), where \(\varrho_i\) is an \(l_i\)-ary relation, with \(l_i \leq l\) for all \(1 \leq i \leq q\).

Under this parameterisation, the decision problem of finding a satisfying assignment to a \((d, l)\)-Csp instance is still \(NP\)-hard, except for the case where \(d = l = 2\), which is the 2-SAT problem. The currently best known running times of exact, exponential-time algorithms for the decision problem for the case when \(l = 2\) and \(d \geq 3\) are due to Eppstein for \(3 \leq d \leq 10\) [46], with a running time of \(O^*(0.4518d^n)\), Feder & Motwani for \(11 \leq d \leq 45\) [52], with a running time of \(O^*((d!)^{n/d})\), and Williams for \(d \geq 46\) [132], with a running time of \(O^*(d^{\omega n/3})\), where \(\omega < 2.376\) is the matrix multiplication exponent. For constraints with arities greater than 2, the algorithm by Iwama & Tamaki [76] is the fastest for \((2, 3)\)-Csp and \((2, 4)\)-Csp, and for \((2, 5)\)-Csp, the algorithm by Paturi et al. [111] runs in \(O^*(1.5692^n)\) time. For general \((d, l)\)-Csp, the algorithm by Schöning [121] has a running time of \(O^*((d - d/l + \varepsilon)^n)\). These results are summarised in Table 8.1. Except for the \((2, 2)\), \((3, 2)\), \((4, 2)\)-Csp algorithms, and the \((d, 2)\)-Csp algorithm by Williams, these are all probabilistic.

<table>
<thead>
<tr>
<th>(d)</th>
<th>(l = 2)</th>
<th>(l = 3)</th>
<th>(l = 4)</th>
<th>(l = 5)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d = 2)</td>
<td>(poly(n))</td>
<td>1.3645(^n)</td>
<td>1.8072(^n)</td>
<td>4.5180(^n)</td>
</tr>
<tr>
<td>(d = 3)</td>
<td>1.3323(^n)</td>
<td>2(^n)</td>
<td>2.6667(^n)</td>
<td>6.6667(^n)</td>
</tr>
<tr>
<td>(d = 4)</td>
<td>1.474(^n)</td>
<td>2.25(^n)</td>
<td>3(^n)</td>
<td>7.5(^n)</td>
</tr>
<tr>
<td>(d = 5)</td>
<td>1.5692(^n)</td>
<td>2.4(^n)</td>
<td>3.2(^n)</td>
<td>8(^n)</td>
</tr>
</tbody>
</table>

Table 8.1: Time complexities of the currently best known Csp-algorithms.

The problem \(\text{MaxSol}(d, l)\) is that of maximising a solution (in the ordinary \(\text{MaxSol}\) sense) of a \((d, l)\)-Csp instance. In some cases, we will solve the slightly more general problem \(\text{MaxHom}(d, l)\). This is the maximisation counterpart of \(\text{MinHom}\), restricted to \((d, l)\)-Csp instances \((V, D, C)\), i.e., in addition the the \((d, l)\)-Csp instance, we are given weight functions \(\{c_a\}_{a \in D}\), where each \(c_a\) is a function from \(V\) to \(\mathbb{Q}\), and the goal is to find a satisfying assignment \(f: V \rightarrow D\) such that

\[
m(f) = \sum_{v \in V} c_{f(v)}(v)
\]

is maximised.
We will use three main tools in the construction of our algorithms. The first is an algorithm for the following problem:

**Definition 8.2 (Dahlöf et al [35])** Let $F$ be a 2-SAT formula, and let $L$ be the set of all literals for all variables occurring in $F$. Given a weight function $w$ on $L$, and a satisfying assignment $f$ for $F$, let $m(f)$ be defined as

$$m(f) = \sum_{l \in L \mid \text{ l is true in } f} w(l)$$

The problem of finding a maximum satisfying assignment to $F$ is called the weighted 2-SAT problem.

It should be clear that this problem is equivalent to MAX HOM$(2, 2)$. Wahlström [131] presents the current best algorithm for counting the number of maximum weighted solutions to 2-SAT instances. This algorithm has a running time of $O^*(1.2377^n)$, and it can easily be modified to return one of the solutions. We will denote this modified algorithm by 2-SAT$_w$.

**Example 8.3** Any $(2, 2)$-CSP instance can be represented as a 2-SAT instance. For a MAX SOL$(2, 2)$ instance $\mathcal{I} = (V, \{0, 1\}, C, \omega)$ with a valuation $\nu : \{0, 1\} \rightarrow \mathbb{Q}_{\geq 0}$, we can define a weight vector $w$ as follows:

$$w(l) = \begin{cases} \omega(\nu) \cdot \nu(0) & \text{when } l = \neg v, \\ \omega(\nu) \cdot \nu(1) & \text{when } l = v. \end{cases}$$

Therefore, we can solve the MAX SOL$(2, 2)$ instance $\mathcal{I}$ in $O^*(1.2377^n)$ time by running 2-SAT$_w$ on the instance $(V, \{0, 1\}, C, w)$.

The second tool is called the microstructure graph, and was introduced by Jégou [80]. It provides a representation of a $(d, 2)$-CSP instance on $n$ variables as an undirected graph, $G$, on $dn$ variables. The microstructure graph makes it easier to visualise local properties of the instance, and we can also apply known graph algorithms directly to $G$.

The third and final tool is a general method for obtaining algorithms for various $(d, l)$-CSP problem, provided the existence of an algorithm for some corresponding $(d', l)$-CSP problem, with $d' < d$. We call this the covering method and it has successfully been applied to a large range of decision, counting, and optimisation problems, cf. Ángelmark [5].

In the analysis of the algorithms, we will encounter recursions on the form

$$T(n) = \sum_{i=1}^{k} T(n - r_i) + p(n),$$

where $p(n)$ is a polynomial in $n$, and $r_i \in \mathbb{Z}^+$. These equations satisfy $T(n) \in O^*(\lambda(r_1, \ldots, r_k)^n)$, where $\lambda(r_1, \ldots, r_k)$ is the largest real-valued root
8.1. The microstructure

to $1 - \sum_{i=1}^{k} x_i^{r_i}$ (see Kullmann [100]). Note that this bound does not depend on neither $p(n)$ nor the boundary conditions $T(1) = b_1, \ldots, T(k) = b_k$. We will usually call $\lambda$ the work factor (in the sense of [46]).

In addition to the Max $\text{Sol}(d, l)$ problem, we also give algorithms for a different optimisation problem on $(d, l)$-$\text{Csp}$ instances. This problem asks us to find two solutions that are as far away from each other as possible; i.e. we want to find two satisfying assignments that disagree on the values for as many variables as possible. It is known as the Max Hamming Distance problem, and was first introduced in Crescenzi & Rossi [33], where it was suggested as a domain independent measure of ignorance, quantifying how much we do not know of the world.

Chapter outline

We begin by formally defining the microstructure graph, in Section 8.1, and presenting the covering method, in Section 8.2, where we also give an immediate application of this method, in which we derive a Max $\text{Sol}(d, 2)$ algorithm from 2-$\text{SAT}_w$. In Section 8.3, we first give an algorithm for the general Max $\text{Sol}(d, l)$ problem. We then use the 2-$\text{SAT}_w$ algorithm to construct a specialised Max $\text{Sol}(3, 2)$ algorithm, to which we again apply the covering method. In Section 8.4, we look the Max Hamming Distance problem, and derive a number of algorithms for different domain sizes and constraint arities. We end the chapter, in Section 8.5, with some suggestions for future work.

8.1 The microstructure

The following is the original definition of the microstructure graph:

**Definition 8.4 (Jégou [80])** Given a binary $\text{Csp} I = (V, D, C)$, the microstructure graph of $I$ is an undirected graph $G$ defined as follows:

1. For every variable $v \in V$, and domain value $a \in D$, there is a vertex $v[a] \in V(G)$.
2. There is an edge $\{u[a_1], v[a_2]\} \in E(G)$ iff $(a_1, a_2)$ satisfies the constraint between $u$ and $v$.

We assume that there is exactly one constraint between any pair of variables, and any variables with no explicit constraint is assumed to be constrained by the universal constraint which allows all values.

Satisfying assignments to a binary $\text{Csp}$ instance are encoded as cliques of size $|V|$ in the microstructure graph $G$. For technical reasons, we actually consider the complement of the microstructure graph, in which there is an edge between two partial assignments $u[a_1]$ and $v[a_2]$ if and only if both $u[a_1]$ and $v[a_2]$ may not appear simultaneously in a satisfying assignment.
In this complement, satisfying assignments appear as independent sets of size $|V|$. An example for the case instance $V = \{x, y\}$, $D = \{0, 1\}$, and $C = \{\neg (x \land y)\}$ is shown in Figure 8.1.

![Microstructure Graph](image)

Figure 8.1: The (complement) microstructure graph of $\neg (x \land y)$.

For the rest of this chapter, whenever we use the microstructure, it is this complement graph that we have in mind. Obviously, the microstructure of a finite domain problem will use polynomial space, since there will be $dn$ vertices in the graph.

By adding weights to the vertices of the microstructure, we get a graph which we call the **weighted microstructure**. When the graph represents a weighted 2-SAT instance, each vertex corresponds to a variables being assigned a specific value, and we can efficiently move from a weighted 2-SAT instances to its weighted microstructure, and back.

### 8.2 The covering method

We will now formally introduce the covering method. Let $n$, $d$ and $d'$ be positive integers, with $d' \leq d$, and let $D$ be a set of size $d$. Define the set

$$Q_{D,d'}^n = \prod_{i=1}^{n} \binom{D}{d'}.$$

A subset $X \subseteq Q_{D,d'}^n$ is called a $d'$-**covering** of $D^n$ if $\bigcup X = D^n$. Let $C_{d,d'}^n$ denote the smallest size of a $d'$-covering of $D^n$ for a set $D$ of size $d$. Obviously, $C_{d,d'}^n = (d/d')^n$ if $d'|d$. This is also a lower bound for arbitrary $d$ and $d'$, which can be strengthened to

$$C_{d,d'}^n \geq \left\lfloor \frac{d}{d'} C_{d,d'}^{n-1} \right\rfloor.$$

We now prove an upper bound for arbitrary $d$ and $d'$.

**Theorem 8.5** For any $\varepsilon > 0$, there exists an integer $n_0$ such that

$$C_{d,d'}^n \leq \left( \frac{d}{d'} + \varepsilon \right)^n$$

whenever $n \geq n_0$. 

8.2. The covering method

Proof: Let $D$ be a set of size $d$. Pick a random set $D' = \prod_{i=1}^{n} D_i \in \mathcal{Q}_{D,d'}^n$, with uniform distribution, and let $v$ be an arbitrary vector in $D^n$. Then, $Pr(v_i \in D_i) = d'/d$ implies $Pr(v \notin D') = 1 - (d'/d)^n$. Now, with uniform distribution, randomly pick $X$ from the set of $t$-subsets of $\mathcal{Q}_{D,d'}^n$. For each $v \in D^n$, let $A_v$ be the event that $v \notin \bigcup X$. Then,

$$Pr(A_v) = \left(1 - \left(d'/d\right)^n\right)^t < e^{-t(d'/d)^n}.$$

To see this, note that $(1 - 1/x)^x < e^{-1}$ when $1/x \leq 1$. Then, let $1/x = (d'/d)^n$, raise both sides of the equation to the power of $t(d'/d)^n$, and the inequality follows.

The event that $X$ will fail to be a covering is equal to the union of the events $A_v$ over all $v$. By the union bound,

$$Pr\left(\bigcup_{v \in D^n} A_v\right) \leq \sum_{v \in D^n} Pr(A_v) < d^n e^{-t(d'/d)^n}.$$ 

If we let $t = cn(d/d')^n$, $c \geq \ln d$, then we have $d^n e^{-t(d'/d)^n} \leq 1$. This proves that there exists a covering of size $cn(d/d')^n$. We note that, for sufficiently large $n_0$, we have

$$\left(\frac{d}{d'} + \varepsilon\right)^n \geq cn(d/d')^n \geq C_{d,d'},$$

for all $n \geq n_0$. This completes the proof. 

The covering asserted by Theorem 8.5 is thus optimal (up to $\varepsilon$). The second ingredient in the covering method is problem specific. For $\text{MAX } \text{Hom}(d,l)$, it can be stated as the following lemma:

Lemma 8.6 Let $d'$ and $d$ be positive integers with $d' < d$, and assume that $\text{MAX } \text{Hom}(d',l)$ can be solved in $\mathcal{O}(a^n)$ time. Let $\mathcal{I}_d$ denote the set of $(d,l)$-CSP instances satisfying the following restriction: for every $(V,D,C) \in \mathcal{I}_d$, and every $x \in V$, there exists a unary constraint $(x;S) \in C$ such that $|S| \leq d'$. Then, the problem $\text{MAX } \text{Hom}(d,l)$, restricted to instances in $\mathcal{I}_d$ can be solved in $\mathcal{O}(a^n)$ time.

Proof: Let $(V,D,C)$ be a $(d,l)$-CSP instance and let $\{c_a\}_{a \in D}$ be the given weight functions. For each variable $x$ in $(V,D,C)$, we know that it can be assigned at most $d'$ out of $d$ value, due to the constraint $(x;S)$. Thus, we can modify the constraints so that $x$ takes values in the set $D' = \{1, \ldots, d'\}$, and so that $\varphi_x : D' \to D$ is an injective function with $S \subseteq \varphi_x(D')$ which gives back the original domain values of $x$. The weight functions of the new instance are given by $c'_{\varphi_x}(a) = c_{\varphi_x(a')} \text{ for } a' \in D'$. This transformation can be done in polynomial time, and the result is a instance of the MAX $\text{Hom}(d',l)$ problem, which can be solved in $\mathcal{O}(a^n)$ time.
Using Lemma 8.6, we can produce a simple probabilistic algorithm for MAX HOM\( \langle d, 2 \rangle \): randomly and independently restrict each variable to some 2-element subdomain of \( D \), and solve the resulting instance. The probability of failure, i.e. of not finding a globally maximal solution, will be bounded by a constant if this is repeated \( O^\ast(\langle d/2 \rangle^n) \) times. The running time of this algorithm is \( O^\ast((1.2377d/2)^n) = O^\ast(0.6189d^n) \). The covering method is essentially a way to derandomise this type of probabilistic construction, with the penalty of an arbitrarily small, but fixed, \( \varepsilon \). Next, we exemplify this use of Theorem 8.5.

**Proposition 8.7** The exists a deterministic algorithm for MAX HOM\( \langle d, 2 \rangle \) (MAX SAT\( \langle d, 2 \rangle \)) which runs in \( O^\ast(\langle 0.6189d + \varepsilon \rangle^n) \) time.

**Proof:** First, let \( \varepsilon' > 0 \) be a constant, depending on \( \varepsilon \), to be fixed later. By Theorem 8.5, there exists an \( n_0 \) such that \( C_{d,2}^{n_0} \leq (d/d' + \varepsilon')^{n_0} = N \), thus let \( X \subseteq Q_{d,2}^{n_0} \) be a covering of this size. Since \( n_0 \) and \( d \) are fixed, \( X \) is of constant size with respect to \( n \), and can be hard-coded into the algorithm. Assume, without loss of generality, that \( |V| \equiv 0 \pmod{n_0} \), and let \( V = V_1 \cup \cdots \cup V_K \) be a partition of \( V \) into \( K \) disjoint subsets \( V_i \) of \( V \) such that \( |V_i| = n_0 \). Let the variables in \( V_i \) be \( \{v_i^j\}_{j=1}^{n_0} \).

Let \( X = \{X_1, \ldots, X_N\} \), and note that each member \( X_i \in X \) is of the form \( X_i = \prod_{j=1}^{n_0} X_i^j \), where \( |X_i^j| = 2 \). Let \( \varphi : [K] \to X \) be an arbitrary function. Intuitively, \( \varphi \) assigns, to each variable \( v_i^j \), in each of the subsets \( V_i \), the unary constraint \( (v_i^j; X_i^j) \). Define a set of constraints, \( C_{\varphi} \), as follows:

\[
C_{\varphi} = C \cup \left\{ (v_i^j; X_i^j) : 1 \leq i \leq K, 1 \leq j \leq n_0 \right\}
\]

The instance \( I_{\varphi} = (V, D, C_{\varphi}) \) now satisfies the requirements of Lemma 8.6, with \( d' = 2 \). Hence, we can use the algorithm 2-SAT\( w \) to solve it in time \( O^\ast((1.2377)^n) \).

Our algorithm for MAX HOM\( \langle d, l \rangle \) works as follows: for every function \( \varphi \), solve the corresponding instance \( C_{\varphi} \). Finally, pick, among the solutions found, one of maximal measure. Since \( X \) is a covering, each satisfying assignment in \( (V, D, C) \) will be satisfying in at least one of the instances \( I_{\varphi} \), hence the algorithm will eventually find a maximal assignment for the original instance. The number of instances which need to be solved is equal to \( |X|^K \leq (d/2 + \varepsilon')^{n_0} = (d/2 + \varepsilon')^n \), and the total running time becomes \( (1.2377 \cdot (d/2 + \varepsilon'))^n = (0.6189d + \varepsilon)^n \), if we let \( \varepsilon' = \varepsilon/1.2377 \).

From the proof of Proposition 8.7, we see that, in addition to the penalty of an \( \varepsilon \) in the running time, we must also be able to find an appropriate covering \( X \). Theorem 8.5 asserts the existence of such an object, but we have no efficient procedure of finding one. However, since it is of constant size with respect to \( n \), it could be calculated ahead of time using an exhaustive search of all possible coverings. It is clear that the constant involved is huge, and prevents any practical application of the method. Finding an efficient
way of producing coverings is an important open problem. A probabilistic approach is briefly discussed in Section 8.5

8.3 Algorithms for Max Sol

We begin by studying the Max Hom(2, l) problem. Let $\gamma_i$ denote the largest real-valued root of the equation $1 - 1/x - 1/x^2 - \ldots - 1/x^l = 0$; $\gamma_i \approx 1.6180, 1.8393, 1.9276, \ldots$ when $i = 2, 3, 4, \ldots$.

Lemma 8.8 The Max Hom(2, l) problem can be solved in $O^*(\gamma_l^n)$ time.

Proof: Let $(V, \{0, 1\}, C, \{c_i\}_{i \in \{0, 1\}})$ be an instance of Max Hom(2, l). A recursive algorithm $A$ for this problem can be outlined as follows: pick one constraint $(x_1 \lor \ldots \lor x_m)$ in $C$ and return the maximum of the following $m$ values:

$$
c_1(x_1) + A(C[x_1 = 1]),
$$

$$
c_0(x_1) + c_1(x_2) + A(C[x_1 = 0, x_2 = 1]),
$$

$$
\vdots
$$

$$
\sum_{i=1}^{m-1} c_0(x_i) + c_1(x_m) + A(C[x_1 = 0, \ldots, x_{m-1} = 0, x_m = 1])
$$

Here, $C[\alpha]$ represents the result of applying the partial assignment $\alpha$ to the constraint $C$. Note that all possible satisfying assignments are covered by these $m$ recursive branches. Since the arity of a constraint is at most $l$, the running time $T(n)$ satisfies $T(n) \leq \sum_{i=1}^l T(n - i)$. Using the previously described results by Kullmann, it follows that $T(n) \in O^*(\gamma_l^n)$. \qed

By combining this lemma with Lemma 8.6 and Theorem 8.5, we get the following result:

Proposition 8.9 There exists an algorithm that solves the Max Hom($d, l$) (Max Sol($d, l$)) problem in $O^*((d/2 \cdot \gamma_l + \varepsilon)^n)$ time.

The fact that $\gamma_l < 2$ for all $l$ implies that this algorithm is faster than the $O^*(d^n)$ exhaustive search algorithm.

For binary constraints ($l = 2$), the algorithm in Proposition 8.9 runs in time $O^*((d/2 \cdot 1.6180 + \varepsilon)^n) \subseteq O^*((0.8091d + \varepsilon)^n)$. As we found in Section 8.2, we actually get an $O^*((0.6189d + \varepsilon)^n)$ by using the 2-SAT algorithm together with the covering theorem. We will now present a further improvement on this problem. We begin by constructing an algorithm for Max Hom(3, 2)
with a running time of $O(1.7458^n)$, to which we then apply Theorem 8.5 in order to obtain an algorithm for MAX Hom($d$, 2) (MAX Sol($d$, 2).) The idea behind the algorithm is to use the microstructure graph and recursively branch on variables with three possible values until all variables are either assigned a value or only have two possible values left. We then transform the remaining microstructure graph to a weighted 2-SAT instance and solve this instance using the 2-SAT algorithm.

Before we start, we need some additional definitions: A variable having three possible domain values will be called a 3-variable, and a variable having two possible domain values will be called a 2-variable. In order to analyse the algorithm we define the size of an instance $I$ as $|I| = n_2 + 2n_3$. Here, $n_2$ and $n_3$ denote the number of 2- and 3-variables in $I$, respectively. This means that the size of an instance can be decreased by 1 by either removing a 2-variable or removing one of the possible values for a 3-variable, thus turning it into a 2-variable.

For a variable $x \in V$ with possible values $\{d_1, d_2, d_3\}$ ordered so that $c_{d_1}(x) > c_{d_2}(x) > c_{d_3}(x)$, let $\delta(x) := (\delta_1, \delta_2, \delta_3)$ where $\delta_i = \deg_G(x[d_i])$. $G$ being the microstructure graph. If $x$ is a 2-variable then, similarly, we define $\delta(x) := (\delta_1, \delta_2)$. We will use $\geq i$ and $(\cdot)$, (dot) as wildcards, to denote sets of possible degree vectors. As an example, $\delta(x) \in (\geq i, \cdot, \cdot)$ should be interpreted as $\delta(x) \in \{(n_1, n_2, n_3) \in \mathbb{N}^3 | n_1 \geq i\}$. The maximal value of a variable $x$, i.e. the domain value $a$ for which $c_a(x)$ is maximal, will be denoted $x_{\text{max}}$. The algorithm is presented as a series of lemmas. Applying these as shown in Algorithm 1 solves the MAX Hom(3, 2) problem.

**Lemma 8.10** For any instance $I$, we can find an instance $I'$ with the same optimal measure as $I$, with size smaller or equal to that of $I$ and to which neither of the following cases apply.

1. There is a 2-variable $x$ for which $\delta(x) \in (2, \geq 1)$.
2. There is a variable $x$ for which the maximal value is unconstrained.

**Proof:** The transformation in Figure 8.2 takes care of the first case, removing one 2-variable (and therefore decreasing the size of the instance). For the second case, we can simply assign the maximal value to the variable, leaving us with a smaller instance.

**Lemma 8.11** If there is a variable $x$ with $\delta(x) \in (\geq 3, \geq 2)$ then we can reduce the instance with a work factor of $\lambda(3, 2)$.

**Proof:** We branch on the two possible values of $x$ and propagate the chosen value to the neighbours of $x$. In one of the branches, the size will decrease by at least 3 and in the other by at least 2.
Lemma 8.12 If there is a variable $x$ for which $\delta(x) \in (3, \cdot, \cdot)$ then we can reduce the instance with a work factor of $\lambda(3, 2)$.

Proof: In one of the branches, we choose $x = x_{\text{max}}$ and propagate this value, decreasing the size by at least 3. In the other branch, choosing $x \neq x_{\text{max}}$ implies that the value of its exterior neighbour must be chosen in order to force a non-maximal value to be chosen in $x$. Therefore, in this branch, the size decreases by at least 2. \hfill \blacksquare

Lemma 8.13 If there is a variable $x$ with $\delta(x) \in (\geq 5, \cdot, \cdot)$ then we can reduce the instance with a work factor of $\lambda(5, 1)$.

Proof: Choosing $x = x_{\text{max}}$ decreases the size of the instance by at least 5. In the other branch, we choose $x \neq x_{\text{max}}$, turning a 3-variable into a 2-variable and thereby decreasing the size by 1. \hfill \blacksquare

If none of Lemma 8.10 to Lemma 8.13 is applicable, then every 3-variable must satisfy $\delta(x) \in (4, \cdot, \cdot)$ and every 2-variable must satisfy $\delta(x) \in (\geq 3, 1)$.

Lemma 8.14 If none of Lemma 8.10 to Lemma 8.13 is applicable, then we can remove every remaining 3-variable with a work factor of $\lambda(4, 4, 4)$.

Proof: Let $x_1$ be a 3-variable with maximal value $d_1$, and let $x_2[d_2]$ and $x_3[d_3]$ be the external neighbours of $x_1[d_1]$, as in Figure 8.3. First note that if both $x_2[d_2]$ and $x_3[d_3]$ have degrees 4, then we can branch on $x_i = d_i$ for $1 \leq i \leq 3$, since either $x_1$ is assigned its maximal value, or at least one of the assignments to $x_2$ and $x_3$ must prevent this. This decreases the size by 4 in each branch. Therefore, we may assume that at least one of $x_2[d_2]$ and
8. Exponential-Time Algorithms

$x_3[d_3]$ have degree 3. If one of $x_1$'s neighbours, say $x_2$, is a 2-variable, then either $\delta(x_2) = (3, 1)$, or $\delta(x_2) \in (\geq 4, 1)$. If $\delta(x_2) = (3, 1)$ (see Figure 8.3a), then we can let one of the branches be $x_1 = d_1$ and the other be $x_1 \neq d_1$. This makes $\delta(x_2) = (2, 1)$ in the latter branch, and $x_2$ can be removed by Lemma 8.10. This means we can decrease the size by 2 in this branch, giving a work factor of $\lambda(4, 2) < \lambda(4, 4)$. Otherwise, if $\delta(x_2) \in (\geq 4, 1)$, or if $x_2$ is a 3-variable and $x_2[d_2]$ has degree 4, then $x_3[d_3]$ must have three neighbours, and since it cannot be a 2-variable (as we would again be in the first case), we conclude that $x_3$ must be a 3-variable. This implies that $d_3$ cannot be the maximal value in $x_3$. The situation when $x_2$ is a 2-variable is depicted in Figure 8.3b. We can now branch on $x_1 = d_1$, $x_2 = d_2$ and $\{x_3 = d_3, x_2 \neq d_2\}$, and decrease the size by 4 in each branch. In the final case, if both $x_2[d_2]$ and $x_3[d_3]$ are of degree 3 (Figure 8.4), and it is not possible to choose a variable $x_1$ without this property, then for every remaining 3-variable, the maximal value can be assigned without branching at all.

![Figure 8.3: The first 2 cases in Lemma 8.14. (Note that $x_i$ denotes a variable, while $d_j$ denotes a value.]

The input to Algorithm 1 is the microstructure graph $G$ and a set of weight function $\{v_a\}_{a \in D}$. The algorithm returns an assignment of maximal measure. Note that in order to actually get a solution to the original problem, one has to a) keep track of the variables eliminated by Lemma 8.10, and b) extract them from the solution returned on line 4.

**Proposition 8.15** There exists an algorithm that solves the MAX $\text{Hom}(3,2)$ problem in $O^*(1.7548^n)$ time.

**Proof:** We claim that Algorithm 1 correctly solves MAX $\text{Hom}(3,2)$. First, we show the correctness of the algorithm. Lemma 8.10 does not remove
Figure 8.4: The final case in Lemma 8.14. (Again, $x_i$ denotes a variable, and $d_j$ denotes a value.)

**Algorithm 1** Algorithm for Max Hom$(3, 2)$.

$MaxHom(G, \{c_a\}_{a \in D})$

1. **if** at any time, the domain of a variable becomes empty, that branch can be pruned
2. Apply the transformations in Lemma 8.10, keeping track of eliminated variables
3. **if** applicable, **return** the maximum result of the branches described in Lemma 8.11 to 8.13
4. **else if** applicable, **return** the maximum result of the branches described in Lemma 8.14
5. **else return** 2-SAT$_w(G, w)$
any solutions to the problem, it merely reduces the number of vertices in
the microstructure. Lemma 8.11 branches on both possible values for the
chosen variable, while Lemma 8.12 and Lemma 8.13 try all three possible
assignments, as is the case for Lemma 8.14. This, together with the proof
of correctness of the 2-SAT algorithm by Wallström [131], shows the
correctness of the algorithm.

Now, apart from the call to 2-SAT, the highest work factor found in
the algorithm is \( \lambda(3, 2) = \lambda(1, 5) \approx 1.3247 \). Recall that we measure
the size of \( I \) by \( |I| = n_2 + 2n_3 \) which, for a \((3, 2)\)-Csp, is \( 2n \), where \( n \)
is the number of variables in \( I \). If we can solve weighted 2-SAT in \( O^*(c^n) \),
then the entire algorithm will run in \( O^*(\max(1.3247, c)2^n) \). Using the construction with
weighted microstructures mentioned earlier, a weight function \( w(x_i[d]) = c_d(x_i) \),
together with 2-SAT, we get \( c \approx 1.2377 \), and the result follows.

Analogously to how we extended the 2-SAT algorithm to the problem
Max Hom\((d, 2)\) in Proposition 8.7, we can now use Theorem 8.5 and
Lemma 8.6 to extend Algorithm 1.

**Proposition 8.16** For \( d > 3 \), there exists an algorithm for solving the
Max Hom\((d, 2)\) (Max Sol\((d, 2)\)) problem which has a running time of
\( O^*(0.5849d + \varepsilon)^n \).

**Proof:** The proof is identical to that of Proposition 8.7 with \( d' = 3 \) and
Algorithm 1 in place of \( d' = 2 \) and the algorithm 2-SAT. Since Algorithm 1
has a running time of \( O^*(1.7548^n) \), the running time for the obtained Max
Hom\((d, 2)\) algorithm is \( O^*((1.7548d/3 + \varepsilon)^n) = O^*((0.5849d + \varepsilon)^n) \).

### 8.4 The Max Hamming Distance problem

In this section we study the problem of finding two satisfying assignments, \( f \)
and \( g \), to a \((d, l)\)-Csp instance so that the number of variables, \( x \), for which
\( f(x) \neq g(x) \) is maximised. This Hamming distance is defined by

\[
d_H(f, g) = |\{x \in V \mid f(x) \neq g(x)\}|
\]

The following formalises the problem:

**Definition 8.17** The Max Hamming Distance \((d, l)\)-Csp problem is the
following maximisation problem:

**Instance:** A \((d, l)\)-Csp instance \( I = (V, D, C) \).

**Solution:** A pair of satisfying assignments \((f, g)\) of \( I \).

**Measure:** The Hamming distance, \( d_H(f, g) \), between \( f \) and \( g \).

A naïve enumeration algorithm for this problem would have a time complexity
of \( O^*(d2^n) \). In the following sections we will present ways to significantly
improve this upper bound.
8.4. The Max Hamming Distance problem

8.4.1 Algorithm for Max Hamming Distance(2, 2)-Csp

Before we describe our Max Hamming Distance(2, 2)-CSP algorithm, we introduce the main idea by giving a slight improvement over the naïve enumeration algorithm for the case of (2, 2)-CSP. For each satisfying assignment $f$ of $\mathcal{I}$, we solve an instance, $(V, \{0, 1\}, C, w)$, of the weighted 2-SAT problem. The weights $w$ are given as follows: $w(x[0]) = f(x)$ and $w(x[1]) = 1 - f(x)$. That is, we give preference to the assignment of $x$ which is opposite to that of $f$. Since a complete enumeration of all solutions $f$ takes $O^*(2^n)$ time and the weighted 2-SAT problem can be solved in $O^*(1.2377^n)$ time, this algorithm runs in $O^*(2.4754^n)$ time, which is already a clear improvement over the naïve $O^*(4^n)$ time algorithm. Our algorithm further refines this idea by making a more clever enumeration of the assignments $f$ and by using some additional power of the weights handed to the weighted 2-SAT solver.

Algorithm 2 The main algorithm for Max Hamming Distance(2, 2)-CSP, $MH1(\alpha, G, \mathcal{I})$

1. if $\delta(x) \in \{(4, 1), (3, 1), (2, 2), (2, 1), (1, 1)\}$ for all variables $x$ in $G$ then
2. return $MH2(\alpha, G, \mathcal{I})$
3. end if
4. Choose a variable $x$ in $G$ with $\delta(x) \in \{(\geq 3, \geq 2), (\geq 5, 1)\}$
5. $(\alpha_0, \beta_0) = MH1(\alpha \cup \{x[0]\}, G - N_G(x[0]) - \{x[0]\}, \mathcal{I})$
6. $(\alpha_1, \beta_1) = MH1(\alpha \cup \{x[1]\}, G - N_G(x[1]) - \{x[1]\}, \mathcal{I})$
7. return $(\alpha_i, \beta_i), i \in \{0, 1\}$ maximising $d_H(\alpha_i, \beta_i)$

As in the previous section, we will use $\delta(x)$ to denote a tuple of degrees for the two vertices of the microstructure associated with the variable $x$. Here, however, we will let $\delta(x) = (\deg(x[i]), \deg(x[1 - i]))$, where $x[i]$ is the vertex with highest degree. We will again write expressions such as $\delta(x) \in (\geq 3, \geq 2)$ for $\delta(x) \in \{(n_1, n_2) \in \mathbb{N}^2 \mid n_1 \geq 3, n_2 \geq 2\}$. We will say that a variable $x$ with $\delta(x) = (a, b)$ is an $(a, b)$-variable.

The main algorithm, $MH1$ (Algorithm 2), takes as input a partial assignment $\alpha$, a microstructure graph $G$, and the original problem instance $\mathcal{I}$. First, the algorithm checks if every variable $x$ in the microstructure has $\delta(x)$ in the set $\{(4, 1), (3, 1), (2, 2), (2, 1), (1, 1)\}$. If this is the case, then the helper function $MH2$ is called. Otherwise, there is a variable $x$ with $\delta(x) \in (\geq 3, \geq 2) \cup (\geq 5, 1)$. We note that for $\delta(x) \in (\geq 3, \geq 2)$, there will be at least 3 variables less in one branch and 2 variables less in the second branch, and for $\delta(x) \in (\geq 5, 1)$, there are at least 5 and 1 variables less, respectively. Thus the time complexity is described by the following two recurrences:

$$T_{(3,2)}(n) = T(n - 3) + T(n - 2) + p(n)$$
8. Exponential-Time Algorithms

\[ T_{(5,1)}(n) = T(n - 5) + T(n - 1) + p'(n) \]

where \( p(n) \) and \( p'(n) \) are polynomials. Using the method by Kullmann [100] described earlier, the two cases have running times of \( T_{(3,2)} \in \mathcal{O}^*(\tau(3, 2)^n) \) and \( T_{(5,1)} \in \mathcal{O}^*(\tau(5, 1)^n) \). Both of these are in \( \mathcal{O}^*(1.3247^n) \).

**MH2 (Algorithm 3)**

The first helper function, MH2 (Algorithm 3), takes over when all variables have degrees in the set \{ (4, 1), (3, 1), (2, 2), (2, 1), (1, 1) \}. Apart from lines 1 to 3, which we will come back to later, the algorithm first checks if there is a variable of degree (4,1). If not, then the algorithm will look for cycles. Note that cycles may consist of zero or more (3,1)-variables, with the remaining variables all being (2,2)-variables. The algorithm ignores the all-(2,2)-variable cycles, and leaves them for MH3 to handle.

**Chains**

To fully analyse how the algorithm behaves on cycles, we will first look at what happens when there are no cycles, i.e. when the algorithm reaches line 12. All (3,1)-variables are then contained in chains. A chain is either an isolated (1,1)-variable, or consists of two (2,1)-variables, one in each end of the chain, and a number of (3,1)-variables, connected by paths of (2,2)-variables, cf. Figure 8.5. The length of a chain is the number of variables it contains. We say that a chain is proper if it contains at least one (3,1)-variable. Non-proper chains, i.e. paths of (2,2)-variables with one (2,1)-variable in each end, are ignored by MH2. They are instead handled without further branching by MH3.

We will distinguish between the case when a proper chain contains more than one (3,1)-variable, and the case when it contains a single (3,1)-variable. In the first case, we find the first (3,1)-variable, \( y \), from one of the ends, and let \( x \) denote the variable following \( y \). We now have two possibilities: \( \delta(x) = (3,1) \), or \( \delta(x) = (2,2) \), as shown in Figure 8.6, top and bottom, respectively. In the first case, when we assign the value to \( x \) as indicated in the top part of the figure, \( x, y, \) and \( x \)'s right neighbour are removed. Additionally, a new non-proper chain is created from \( z \) to \( y \)'s left neighbour.
8.4. The Max Hamming Distance problem

Algorithm 3 The helper function MH2.

\( MH2(\alpha, G, I) \)
1. if \( \delta(x) \in \{(2, 2), (2, 1), (1, 1)\} \) for all variables \( x \) in \( G \) then
2. \( \text{return } MH2(\alpha, G, I) \)
3. end if
4. if \( \delta(x) \neq (4, 1) \) for all variables \( x \) in \( G \) then
5. if \( G \) contains a cycle \( C \) such that \( \delta(x) = (3, 1) \)
6. for some \( x \) in \( C \) then
7. if there is a variable \( z \) in \( C \) s.t. \( \delta(z) = (2, 2) \) then
8. Choose \( x \) in \( C \) s.t. \( \delta(x) = (3, 1) \) and \( x[i] \) has a
9. neighbour \( y \) with \( \delta(y) = (2, 2) \)
10. else
11. end if
12. Choose \( x \) in \( C \) arbitrarily
13. end if
14. \( (\alpha_0, \beta_0) = MH1(\alpha \cup \{x[0]\}, G - N_G(x[0]) - \{x[0]\}, I) \)
15. \( (\alpha_1, \beta_1) = MH1(\alpha \cup \{x[1]\}, G - N_G(x[1]) - \{x[1]\}, I) \)
16. \( \text{return } (\alpha_i, \beta_i), i \in \{0, 1\} \text{ maximising } d_H(\alpha_i, \beta_i) \)
17. end if
18. if \( \{x[i], y[j]\} \) in \( E(G) \) s.t. \( \delta(x) = (4, 1) \) and \( \delta(y) \neq (4, 1) \) then
19. Branch according to Figure 8.8
20. else if \( G \) contains a \( (4, 1) \)-component with a triangle \( x-y-z \) then
21. Branch according to Figure 8.9
22. else if \( G \) contains a triangle-free \( (4, 1) \)-component
23. Branch according to Figure 8.10
24. end if
8. Exponential-Time Algorithms

Figure 8.6: The variable $x$ follows the first $(3, 1)$-variable, $y$.

(which may coincide with $z$.) This chain is later removed by $MH3$, and thus we remove at least 4 variables in this branch. In the other branch, the same variables are removed, except for $x$’s right neighbour, i.e. at least 3 variables. The bottom case is similar, but here the new non-proper chain is from $z$ to $y$. The work factor for this step is therefore at most $\tau(3, 4) \approx 1.2207$. For each case, the black vertices in the figure indicate the assignments chosen in the first branch.

In the second case, when the chain contains a single $(3, 1)$-variable, $x$, we branch on this variable. In both branches, $x$ is removed, and a number of non-proper chains remain, which are later removed by $MH3$. Thus, we can analyse this case as if we removed the entire chain in both branches. Since a proper chain must contain at least 3 variables, we have a work factor of at most $\tau(3, 3) \approx 1.2599$ for this step.

Cycles

Getting back to discussing cycles; when we reach line 17 of algorithm $MH2$, the algorithm considers two cases. If there is some $(2, 2)$-variable in the chosen cycle, then we branch on a $(3, 1)$-variable $x$ which has a neighbour $y$ with $\delta(y) = (2, 2)$. If the cycle consists of only 2 variables, then $x$ has a unique assignment, and no branching is necessary. If the cycle consists of 3 variables, then all of them are removed in both branches. Otherwise, we remove at least 4 variables in one branch ($x$, its two neighbours, and one more neighbour of $y$), and 1 variable in the other. In the second branch, a chain of length at least 3 is created, and a future step of the algorithm will reduce this chain with a work factor of at most $\tau(3, 3)$. This results in a work factor of $\tau(4, 4, 4)$.

If there are only $(3, 1)$-variables in the cycle, we branch on one of them. If the cycle contains 3 variables, then in both branches, the entire cycle is removed, for a work factor of $\tau(3, 3)$. If the cycle contains 4 variables,
then in one branch, the entire cycle is removed, and in the other, a chain of length 3 is created. This, in turn, is removed in its entirety by one additional branching, for a total work factor of $\tau(4, 4, 4)$ for both steps. Otherwise, in one branch 3 variables are removed, and in the other 1 variable is removed. However, in the second branch, we have now created a chain of length at least 4, and one branching step on this chain removes at least 4 and 3 variables, respectively. In total, for these two steps, we have therefore removed at least 3, 4, and 5, variables, respectively, for a work factor of $\tau(3, 4, 5) \approx 1.3247$.

**Degree (4, 1)-vertices**

We now turn to lines 18 through 24. To improve the clarity of our figures and make the reasoning about the microstructure easier, we will make the following simplifications. For each variable $x$ with $\delta(x) \in (\geq 1, 1)$, we suppress the vertex of degree 1. Additionally, if there is a path $x = x_0, x_1, \ldots, x_{k-1} = y$, $k > 1$, such that $\delta(x_0), \delta(x_{k-1}) \in (\geq 1, 1)$ and $\delta(x_i) = (2, 2)$ for $1 \leq i < k-1$, then we remove all vertices for the variables $\delta(x_i)$, $1 \leq i < k-1$ and replace the path with a thick edge $[x, y]$. This "reduced microstructure" is thus an undirected graph with vertices of degree less than or equal to 3, each representing a distinct variable, and with either thin or thick edges, cf. Figure 8.7. We stress that, although they are no longer represented by vertices in our figures, we cannot yet remove the $(2, 2)$-variables from consideration, and some care must be taken when we branch on a vertex which is incident on a thick edge. As an example, say that we branch on $x$ in Figure 8.7. In the branch where we make the assignment $x[1]$, this value is propagated as expected. From the reduced microstructure, we can deduce that we will remove at least 5 variables (remember that a thick edge represents a non-proper chain of length at least one.) However, in the branch where we make the assignment $x[0]$, we remove $x$, but after this removal, $\delta(z) = (2, 1)$, so $z$ now appears as a vertex in the reduced microstructure, connected to $y$.

When line 18 is reached, there is at least one variable of degree $(4, 1)$,
8. Exponential-Time Algorithms

i.e. at least one vertex in the reduced microstructure graph of degree 3. If we were to branch on a degree 3 vertex only, we would in general remove 4 vertices in one branch and 1 in the other, for a work factor of $\tau(1,4) = 1.3803$. To reduce the graph more efficiently, and obtain a better work factor, we will have to analyse a large number of sub cases. The main branching technique is however the same in each case; we choose a sequence of vertices, $x_1, \ldots, x_k$, to branch on. In the first branch, we can remove $x_1$ and its neighbours by assigning the value to $x_1$ which is involved in some constraint(s). In the second branch, we can only remove $x_1$, but in addition, we branch on $x_2$ in order to remove more vertices. In the third branch, we remove $x_1$, $x_2$, and branch further on $x_3$, etc. In total, we get $k + 1$ branches with a different number of vertices removed in each branch. The success of this technique relies on the fact that we eventually manage to remove vertices so as to produce isolated components with vertices of degrees $\{(2, 2), (2, 1), (1, 1)\}$. These can then also be removed (they are handled by MH3 without extra branching.) and it is these additional removals which help to improve the overall work factor.

Informally, the algorithm works by first removing degree 3 vertices which have some neighbour of lower degree. We are then left with a graph $G$ which contains a number of 3-regular connected components in addition to a number of components in which each vertex has degree strictly smaller than 3. At this point, we start to remove triangles from the 3-regular components, after which the structure of the remaining 3-regular components is simple enough to overview, and exploit. In each call to MH2, the algorithm checks the cases in order, and applies the first applicable one; later cases may change the structure of $G$ and make earlier cases applicable again.

Obtaining 3-regular components

Formally, the algorithm first checks if there is a vertex $x$ of degree 3 ($\delta(x) = (4, 1)$) which is incident on a thick edge, see Figure 8.8a. If so, it branches on $x$, and removes at least 5 vertices in one branch, and at least 1 in the other, for a work factor of $\tau(5,1) \approx 1.3247$. Otherwise, the algorithm checks if there is a vertex $x$ of degree 3 which has a neighbour $y$ of degree 1 (Figure 8.8b.) If so, when we branch on $x$, 4 variables are removed in one branch, and 2 ($x$ and $y$) in the other, implying a work factor of $\tau(2,4) \approx 1.2720$.

Otherwise, if there is a vertex $x$ of degree 3 which has a neighbour $y$ of degree 2, then the algorithm checks the neighbour $z \neq x$ of $y$. If $N_G(z) \subseteq \{x,y\}$, then we branch on $x$, which removes 4 and 3 variables, respectively, for a work factor of $\tau(3,4) \approx 1.2207$. The edge $\{y,z\}$ may be thick, but this only improves the performance of the algorithm, as the entire non-proper chain between $y$ and $z$ gets removed in both cases. Otherwise, $z$ has some additional neighbour $w \notin \{x,y\}$. If $z$ is incident on a thick edge, then by branching first on $x$, and then on $z$, we will remove at least 4 ($x$ and its neighbours), 5 (all named variables plus one on the thick edge,) and 3 ($x$, $y$, and $z$) variables, respectively, for a work factor of $\tau(3,4,5) \approx (1, 3247)$. From
now on, we can thus assume that all edges from $z$ are thin. If $\deg(z) = 2$ (Figure 8.8d), or if $\deg(z) = 3$ and $N_G(z) = \{ x, y, w \}$ (Figure 8.8e), then in one branch we remove $x$ and its neighbours, and in the other branch, we remove $x$ and further branch on $w$. This removes $w$ and its neighbours (which include $z$), and thereby $y$, in one sub-branch, and $\{ x, y, z, w \}$ in the other. In total, we get 3 branches, each removing at least 4 vertices, for a work factor of $\tau(4, 4, 4) \approx 1.3161$. If, on the other hand, $\deg(z) = 3$ but $x \notin N_G(z)$ (Figure 8.8f), then we can branch first on $x$, and then on $z$, and remove at least 4, 5 and 3 vertices, respectively, with a work factor of $\tau(3, 4, 5) \approx 1.3247$.

Triangle removal

We are now left with the cases when all neighbours of every degree 3 vertex are themselves of degree 3. That is, $G$ consists of some connected, 3-regular components, and otherwise only vertices of degree 2 or less. We may additionally assume that the 3-regular components contain no thick edges, since such vertices were removed in the case of Figure 8.8a. First, we show that we can make the 3-regular components triangle-free. If we have two triangles connected by an edge, as in Figure 8.9a, then we can branch on $y$ and $w$, and in each of the three branches remove at least 4 vertices. This implies a work factor of $\tau(4, 4, 4) \approx 1.3161$. Otherwise, for a triangle $x-y-z$, each of its neighbours $x'$, $y'$ and $z'$ must be distinct. We have two cases, depending on whether or not all of the edges $\{ x', y' \}, \{ x', z' \}$, and $\{ y', z' \}$ are present in $G$. If they are, then the entire component must be as shown in Figure 8.9b. Here, we can branch on any vertex to remove all 6 vertices in one branch, and 1 in the other for a work factor of $\tau(1, 6) \approx 1.2852$. If there is at least one edge missing, say $\{ x', y' \}$ (Figure 8.9c), then we can branch on $x'$, $y'$, and $z$, in order, for a work factor of $\tau(4, 5, 5, 6) \approx 1.3247$. 

Figure 8.8: Branching rules for obtaining 3-regular components.
8. Exponential-Time Algorithms

Reducing 3-regular triangle-free components

We can now assume that all degree 3 vertices in \( G \) reside in 3-regular components, and that all 3-regular components are triangle-free. We distinguish four additional cases. In each case, the algorithm tries to find a vertex \( x \) which satisfies some particular condition. We will denote the neighbours of this \( x \) by \( y_1, y_2, \) and \( y_3 \).

First, if there is a vertex \( z \in N_G(y_1) \cap N_G(y_2) \cap N_G(y_3) \) such that \( z \neq x \) (Figure 8.10a), then we can branch on \( y_1, y_2, \) and \( y_3, \) to remove 4, 5, 6, and 5 vertices, respectively, for a work factor of \( \tau(4, 5, 5, 6) \approx 1.3247 \). This crucially depends on the fact that there are no edges between the vertices \( \{y_1, y_2, y_3\} \).

Second, if we can label the neighbours of \( x \) so that \( N_G(y_1) \cap N_G(y_2) = N_G(y_1) \cap N_G(y_3) = \{x\} \) (Figure 8.10b), then we branch on \( z_1, z_2, y_2, \) and \( y_3, \) where \( N_G(y_1) = \{x, z_1, z_2\} \). This removes 4, 5, 7, 8, and 6 vertices for a work factor of \( \tau(4, 5, 6, 7, 8) \approx 1.3247 \).

Third, if we can label the neighbours of \( x \) so that \( N_G(y_1) \cap N_G(y_2) = \{x\} \), but \( N_G(y_1) \cap N_G(y_3) = \{x, z_{13}\} \), then we must also have \( N_G(y_2) \cap N_G(y_3) = \{x, z_{23}\} \). (Otherwise, we would be in the second case with \( y_2 \) as the vertex \( y_1 \).) This situation is depicted in Figure 8.10c. We now branch on \( z_2, z_{13}, y_2, \) and \( y_3, \) The first two branches remove 4 and 5 vertices, respectively. (Note that there is no edge between \( z_2 \) and \( z_{13} \) since the component is triangle-free.) In the third, we additionally remove \( y_2 \) and its neighbours, which include \( z_{23} \) and \( x \), but cannot include \( z_2 \) nor \( z_{13} \). Since we have previously removed \( z_2 \) and \( z_{13} \), both \( y_1 \) and \( y_3 \) are also removed, for a total of 8 vertices. In the last two branches, 7 and 6 vertices are removed, and again we have
the work factor of $\tau(4, 5, 6, 7, 8) \approx 1.3247$. Note that this branching rule is the same as in Figure 8.10b, but the analyses differ.

In the fourth and last case, each pair of distinct $y_i$ and $y_j$ must have two neighbours: $x$ and $z_{ij}$. Furthermore, the vertices $z_{ij}$ are distinct (or else we would be in the first case with a common neighbour of all of the $y_i$) and there are no edges among $\{z_{12}, z_{13}, z_{23}\}$, since this would induce a triangle. This situation is depicted in Figure 8.10d. Here, we branch on $z_{12}$, $z_{13}$, $z_{23}$, and $x$, and remove 4, 5, 8, 7, and 7 vertices respectively. We have $\tau(4, 5, 7, 7, 8) = 1.3138$. In summary, the maximal work factor for MHI is 1.3247.

**MH3 (Algorithm 4)**

When algorithm MH3 is called, the graph $G$ only contains variables of degree (2, 2), (2, 1), or (1, 1), i.e. every variable will be of one of the forms found in Figure 8.11. The weights shown in the figure are now added to the corresponding assignments in the original problem, $I$, and the resulting weighted 2-SAT problem is given to a weighted 2-SAT solver. If the solution $\beta$ returned by the solver has weight $W$, this means that we can add assignments (i.e. vertices) to $\alpha$ to create a solution which differs from $\beta$ on $W$ assignments. First of all, since all assignments in $\alpha$ are given weight 0, if any of these are chosen, they will not add anything to the distance, while the other possible value for all these variables will add one to the distance (and are consequently given a weight of 1 on line 3.) For the variables $x$ with $\delta(x)$, we can choose freely which value they should assume, and thus
we can always find an assignment which adds one to the distance from $\beta$ by choosing the other value for $\alpha$.

For components of the form seen in Figure 8.11b, we can reason as follows. If $\beta$ contains both assignments with weight 1/2, then obviously, we have to add one of them to $\alpha$, since not both assignments with weight 3/2 are allowed simultaneously—and thus we get a distance of 1, which is the sum of the weights in $\beta$. On the other hand, if $\beta$ contains one 3/2 and one 1/2 assignment, then we can choose the opposing value for both of these and get a distance of 2. A similar argument applies for the components of the form seen in Figure 8.11c, and finally, for the cycles (Figure 8.11d), there are only 2 mutually exclusive assignments. Therefore, the sum of the weights in $\beta$ is equal to the number of variables participating in the cycle, and we can let $\alpha$ be the opposite assignment on these variables. Consequently, the pair returned on line 12 will have a Hamming distance equal to the weight of $\beta$, and with $\alpha$ and $G$ given, no pair with greater Hamming distance can exist. Except for the call to $2\text{-SAT}_w$ on line 6, every step of algorithm $MH3$ can be carried out in polynomial time, thus the time complexity is fully determined by that of the $2\text{-SAT}_w$ algorithm.

To summarise this section, we state the following proposition, and its immediate corollary:

**Proposition 8.18** Algorithm $MH1$ correctly solves the problem MAX HAMMING DISTANCE(2, 2)-CSP and has a running time of $O^*(n^{30})$, where $n$ is the number of variables in the problem, and $O^*(n^{30})$ is the time complexity of solving a weighted 2-SAT problem.

**Proof:** The correctness follows from the previous discussion, and among the steps in the algorithm, $O^*(1.3247^n)$ dominates. Since the $2\text{-SAT}_w$ algorithm
is called for every leaf in the search tree, we get a total time complexity of $O^*(n)$. ■

![Weights assigned by the algorithm MHI3.](image)

**Corollary 8.19** The problem Max Hamming Distance (2, 2)-Csp can be solved in $O^*(1.6396^n)$ time.

**Proof:** The algorithm 2-SAT$_w$ by Wahlström [131] with a running time of $O^*(1.2377^n)$, together with Proposition 8.18 gives the result. ■

### 8.4.2 Algorithm for Max Hamming Distance (d, l)-Csp

For problems where the arity of the relations is greater than 2, the microstructure graph becomes a hypergraph, and is not as convenient to work with. Here, we find a different approach for the general case.

Let us first consider the following problem: given a CSP instance $I = (V, D, C)$, can we find a pair of solutions with Hamming distance equal to
8. Exponential-Time Algorithms

Algorithm 5 Algorithm sketch for Max Hamming Distance \((d, l)\)-CSP.
\(\text{Max Hamming Distance}(d, l) - \text{CSP}(I = (V, D, C))\)

1. Pick a subset \(U\) of \(V\) with \(|U| = k\)
2. Create a copy \(I' = (V', D, C')\) of \(I\), where each
   \(x \in V\) is renamed to \(x' \in V'\)
3. for each \(x \in U\) do
4.   add a constraint \(x \neq x'\) to \(C'\)
5. for each \(x \notin U\) do
6.   add a constraint \(x = x'\) to \(C'\)
7. if \(I'\) is satisfiable with solution \(f\) then
8.   for each \(x \in V\) do
9.     add \(f(x)\) to \(\alpha\)
10. for each \(x' \in V'\) do
11.     add \(f(x')\) to \(\beta\)
12. return \((\alpha, \beta)\) with distance \(k\)
13. end if

\(k\)? One way of doing this is sketched in Algorithm 5. There are \(2^n\) ways to choose \(U\) on line 1, so if we can solve the satisfiability problem for \(I'\) on line 7 in time \(\Omega^*(h(n))\), then, since the number of variables in \(I'\) is twice that of \(I\), we can find a pair of solutions with maximum Hamming distance in time \(\Omega^*(2^n h(2n))\). For example, since 2-SAT can be solved in polynomial time, we would, using this approach, get a running time of \(\Omega^*(2^n)\) for the Max Hamming Distance \((2, 2)\)-CSP. This does give a slower running time than the algorithm we presented in the previous section, but it can be applied to CSP instances with domain size and constraint arity greater than 2.

Note that in this algorithm, it is actually unnecessary to make a copy of all the variables. Having selected \(k\) variables that should be different in the two solutions, we only need to make copies of those, leaving the remaining \(n - k\) variables unchanged. With this improvement, we get Algorithm 6 for Max Hamming Distance \((d, l)\)-CSP.

Theorem 8.20 If we can solve \((d, l)\)-CSP in \(\Omega^*(a^n)\), then there exists an algorithm for the Max Hamming Distance \((d, l)\)-CSP problem which runs in \(\Omega^*((a(1 + a))^n)\).

Proof: In Algorithm 6, the instance \(I''\) will contain \(2n - k\) variables, and there are \(\binom{n}{k}\) ways of choosing \(\chi\). Consequently, given that we can solve \((d, l)\)-CSP in \(\Omega^*(a^n)\), the algorithm has a total running time of
\[\Omega^*(\sum_{k=0}^{n} \binom{n}{k} a^{2n-k}) = \Omega^*(a^n \sum_{k=0}^{n} \binom{n}{k} a^{n-k}) = \Omega^*(a^n(1 + a)^n),\]
and the result follows. \(\blacksquare\)
Algorithm 6 Algorithm for MAX HAMMING DISTANCE\((d,l)\)-Csp.

\[
\text{MAX HAMMING DISTANCE}(d, l) - \text{Csp}(I = (V, D, C))
\]

1. for \(k := |V| \) down to 0 do
2. for each \(U \subseteq V \) with \(|U| = k\) do
3. \(L' = (V', D, C')\) be a copy of \(I\)
4. \(\gamma \subseteq C\) be all constraints involving variables from \(U\)
5. Create \(\gamma'\) by exchanging all variables not in \(U\) with their counterparts from \(V'\)
6. \(C' := C' \cup \gamma'\)
7. for each \(x \in U\) do
8. \(C' := C' \cup \{x \neq x'\}\)
9. if \(I'' = (V \cup V', D, C')\) is satisfiable with solution \(f\) then
10. \(\alpha, \beta\) be the two assignments found in \(f\)
11. return \((\alpha, \beta)\)
12. end if
13. end for
14. end for

If we use a probabilistic algorithm for some \((d,l)\)-Csp in Algorithm 6, we similarly get a probabilistic algorithm for MAX HAMMING DISTANCE\((d,l)\)-Csp. In this case, the probability of failure, i.e. the probability that we return a pair of satisfying assignments, \(f\) and \(g\), for which \(d_H(f,g)\) is not maximal or that we do not return any satisfying assignments at all, even though one exists, is bounded by the probability that the \((d,l)\)-Csp algorithm fails to find solutions to all of the satisfiable instances \(I''\) for the optimal value of \(k\). Since there is at least one such instance (in the case when satisfying assignments exist), the probability of failure of Algorithm 6 is bounded by that of the corresponding \((d,l)\)-Csp algorithm. Using the general, probabilistic, \((d,l)\)-Csp algorithm by Schöning [121] with a running time of \(O^+(d/d(l+\varepsilon)^n)\), we have the following corollary of Theorem 8.20:

Corollary 8.21 For \(\varepsilon > 0\), there exists a probabilistic algorithm for solving MAX HAMMING DISTANCE\((d,l)\)-Csp in time

\[
O^+\left((d/d^2 + d/d^2 + \varepsilon)^n\right).
\]

In addition, there exist a number of algorithms for special cases of \((d,l)\)-Csp, in particular for binary constraints and for \(l\)-SAT. The running times of the currently best ones are given in Table 8.1.

For problems with domain size 2 and \(l\)-ary constraints, i.e. \(l\)-SAT, we can improve further on Algorithm 6. First, consider the following formula:

\((x \lor y) \land (\neg y \lor z)\)
8. Exponential-Time Algorithms

If we want to find out if there are two solutions which differ on the assignment on \( x \) and agree on \( y \) and \( z \), we would, using the algorithm in Figure 6, get the following formula:

\[(x \lor y) \land (\neg y \lor z) \land (x' \lor y) \land (x \lor \neg x') \land (\neg x \lor x')\]

But there is a better way. Since there are only two possible domain values, and we force \( x' \) to always assume the opposite of \( x \), there is no reason to create new variables. Instead, we duplicate the clauses containing variables on which the two solutions should differ, and among these clauses, we replace every literal containing one of these variables with its negation. In our example, we would get:

\[(x \lor y) \land (\neg y \lor z) \land (\neg x \lor y)\]

This formula has a solution \( \{ x \mapsto 0, y \mapsto 1, z \mapsto 1 \} \), and from this we can derive two solutions to the original formula which differ on the assignment of \( x \), and agree on \( y \) and \( z \).

Algorithm 7 The MAX HAMMING DISTANCE(2, l)-CSP algorithm.

\[
\text{MAX HAMMING DISTANCE}(2, l) - \text{CSP}(V, D, C)
\]

1. for \( k := |V| \) down to 0 do
2.  for each \( U \subseteq V \) with \( |U| = k \) do
3.  Let \( \gamma \) be all the clauses of \( C \) containing variables from \( U \)
4.  Create \( \tilde{\gamma} \) by negating all occurrences of a variable \( x \in U \) in \( \gamma \)
5.  Let \( C' := C \cup \{ \tilde{\gamma} \} \)
6.  if \((V, D, C')\) is satisfiable with solution \( f \) then
7.  Let \( \alpha, \beta \) be the two assignments found in \( f \).
8.  return \((\alpha, \beta)\)
9. end if
10. end for
11. end for

Algorithm 7 for MAX HAMMING DISTANCE(2, l)-CSP is similar to the one for the general case, but it does not add any variables to the problem.

Theorem 8.22 If we can solve \((2, l)\)-CSP in \( O^*(a^n) \) time, then there exists an algorithm for solving MAX HAMMING DISTANCE(2, l)-CSP which runs in \( O^*((2a)^n) \) time.

Proof: The algorithm in Figure 7 considers all subsets of variables of the problem, as discussed in this section. Consequently, it will deliver a solution in \( O^*((2a)^n) \) time. \( \blacksquare \)
8.5. Discussion and open problems

**Corollary 8.23**  For every $\varepsilon > 0$, there exists a probabilistic algorithm for solving the MAX HAMMING DISTANCE$(2, l)$-CSP in $O^*((4 - 4/l + \varepsilon)^n)$ time. Additionally, the special case of MAX HAMMING DISTANCE$(2, 3)$-CSP can be solved (by a probabilistic algorithm) in $O^*(2.6476^n)$ time.

**Proof:** Using the probabilistic $O^*(((2 - 2/l + \varepsilon)^n)$ algorithm for $l$-SAT found in [121] on line 7 of the algorithm in Figure 7 gives the first result, and replacing it with the (similarly probabilistic) 3-SAT algorithm from [76] gives the second.

As could be expected, the algorithms we get from Corollary 8.23 offer a large improvement over the more general Corollary 8.21; rather than an $O^*(3.1111^n)$ time algorithm for MAX HAMMING DISTANCE$(2, 3)$-CSP, we get a running time of $O^*(2.6476^n)$.

8.5 Discussion and open problems

We have presented algorithms for two different optimization problems on $(d, l)$-CSP instances: MAX SOL$(d, l)$, (or, more generally, MAX HOM$(d, l)$) and MAX HAMMING DISTANCE$(d, l)$-CSP. In both cases, we have worked with the microstructure graph, and in the first case, we have employed a general technique, called the covering method, to extend the results to arbitrary domain sizes.

In comparison to the $O^*(1.2377^n)$ time analysis of the 2-SAT$_w$ algorithm by Wahlström [131], the analysis of our MAX HOM$(3, 2)$ algorithm is quite simplistic. It would be interesting to see if the more sophisticated techniques used in Wahlström’s paper could be extended to the $(3, 2)$-CSP case. Ultimately, the aim should be to find general techniques to prove upper bounds $O^*(c(d)^n)$ on MAX HOM$(d, 2)$-CSP for which $c(d)$ grows sublinearly, thereby eliminating the need to apply the covering method.

It is also interesting to note how the MAX HAMMING DISTANCE$(2, 2)$-CSP algorithm was able to benefit from the 2-SAT$_w$ algorithm. Perhaps one can find a general way of reducing any MAX HAMMING DISTANCE$(d, 2)$-CSP instance to a (small enough) number of calls to a MAX HOM$(d, 2)$ algorithm. This could possibly be used to improve on the running time given in Theorem 8.20 in the case of $l = 2$. We can sketch the following comparison, using a naïve approach: enumerate all $d^n$ satisfying assignments $f$, and for each variable $x \in V$, give the pair $(x, f(x))$ weight 0 and $(x, a)$ weight 1 for $a \neq f(x)$. Then, solve the resulting MAX HOM$(d, 2)$ instances using the $O^*((0.5849d + \varepsilon)^n)$ algorithm in Proposition 8.16. We can use a probabilistic approach, instead of the covering method, to get rid of the $\varepsilon$. The resulting running time is $O^*((0.5849d^2)^n)$, while the one we get from Theorem 8.20 together with Eppstein’s $O^*((0.4518d)^n)$ algorithm [46] has a running time of $O^*((0.4518d^2 + 0.4518d)^n)$. For $d = 3$, the former is $O^*(5.2641^n)$ and the latter is $O^*(3.1925^n)$. In order to make the suggested
approach works, it seems likely that we would both have to improve the Max Hom(3, 2) algorithm, and find a good branching scheme in place of the naïve enumeration.

Finally, we will address two interesting problems related to the covering method and Theorem 8.5. The first is combinatorial: can we relate the numbers $C'_{d,d}$ to some known sequences of numbers, or in some other way find better upper and lower bounds? Is it possible to get rid of the $\varepsilon$ in Theorem 8.5, and replace it by some polynomial factor? Another approach is to determine $C'_{d,d}$ in some special cases, for example by fixing $d$ and $d'$ to some small integers. A special case with $d' = n = 2$ was used by Angelfmark et al. [6] to count the number of solutions to binary Csp instances, and to count the number of 3-colourings of a given graph. Here $C^2_{d,2} = (d^2 + d + 4)/4$ if $d \equiv 1 \pmod{4}$, and $C^2_{d,2} = (d^2 + d)/4$ if $d \equiv 3 \pmod{4}$.

The second problem related to coverings is an algorithmic question, and was briefly mentioned in Section 8.2. Is there a way to efficiently find a covering of a given size? A probabilistic approach works, by repeatedly selecting subsets of $Q'_{d,d}$, until a covering is found. The probability of failure decreases exponentially in the number of repetition, which makes this approach feasible. However, we still do not know of any deterministic procedure for finding coverings.
Bibliography


Bibliography


Bibliography


Department of Computer and Information Science  
Linköpings universitet  
Dissertations  
Linköping Studies in Science and Technology

No 14  

No 17  

No 18  

No 22  

No 33  

No 51  

No 54  

No 55  

No 58  

No 69  

No 71  

No 77  

No 94  

No 97  

No 109  

No 111  

No 155  

No 165  

No 170  

No 174  

No 192  

No 213  


