Linköping Studies in Science and Technology
Dissertation No 2368

On Infinite-Domain CSPs Parameterized by Solution Cost

George Osipov

LINKÖPING
UNIVERSITY

Linköping Studies in Science and Technology
Dissertations, No. 2368

On Infinite-Domain CSPs Parameterized by Solution Cost

George Osipov

Linköping University
Department of Computer and Information Science
Artificial Intelligence & Integrated Computer Systems
SE-581 83 Linköping, Sweden

Linköping 2024

Typeset using XƎLATEX

Cover picture by Meri Pepanashvili

Printed by LiU-Tryck, Linköping 2024

Edition 1:1

© George Osipov, 2024
ISBN 978–91–8075–496–5 (print)
ISBN 978–91–8075–497–2 (PDF)
https://doi.org/10.3384/9789180754972
ISSN 0345-7524
Published articles have been reprinted with permission from the respective copyright holder.

POPULÄRVETENSKAPLIG SAMMANFATTNING

Datorer påverkar alla aspekter av det moderna samhället och de har djupgående förändrat sätten vi arbetar och kommunicerar på. *Algoritmer* utgör den abstrakta arbetsbeskrivning för en dator som förklarar hur data ska bearbetas och vilket resultat som ska genereras. Målet med *beräkningskomplexitet* är att studera mängden resurser som krävs för att utföra beräkningar: Hur mycket tid kräver en viss beräkning och hur mycket minne behöver vi för att utföra den? För att studera algoritmer använder man inom beräkningskomplexitet en abstrakt beräkningsmodell istället för specifik hårdvara. Abstraktion är nyckeln här - den gör det möjligt att fokusera på generella algoritmiska idéer istället för oviktiga detaljer. Detta underlättar användningen av matematiska metoder och ger generellt tillämpbara resultat.

Algoritmer för ett givet problem kan ha mycket varierande prestanda och ibland kan enkla idéer betydligt minska tidsåtgången. Låt oss fundera kring problemet att söka efter en persons adress i en adressbok. Vi tar namnet ”Mats Matsson” som exempel. Posterna antas vara sorterade efter efternamn så vi kan börja från första sidan och kontrollera sidorna en efter en tills efternamn som börjar med ”M” hittats, sedan de som börjar med ”Ma” och så vidare. En alternativ metod är att öppna boken ungefär i mitten: om efternamnen på mitten av sidan börjar med ”K”, vet vi att ”Mattson” kommer att visas efter det, och vi kan undvika den första halvan helt. I en bok med 1000 sidor där Mats Matsson finns på sida 600 minskar då sökningen från 600 sidor till 100 genom att börja från sida 500. Vi kan tillämpa metoden igen på den andra halvan av 500-sidorsboken, sedan igen på den 250-sidiga halvan av den, och så vidare. På detta sätt kan vi hitta *vilket* namn som helst i den 1000-sidiga boken efter att ha granskat cirka 10 sidor. Den här idén gör det möjligt för datorer att söka i databaser med miljarder poster och hämta information inom några sekunder.

Sökuppgiften ovan indikerar att vissa beräkningar kan utföras med effektiva algoritmer. Naturligtvis finns det problem för vilka vi inte känner till om det finns någon effektiv algoritm. Ett exempel är att planera en kortaste rutt för en leveransbil med flera destinationer; detta problem brukar kallas *handelsresandeproblemet*. Körtiden för de snabbaste exakta algoritmerna man känner till för handelsresandeproblemet växer exponentiellt med antalet destinationer. Detta gör dem mycket långsamma och i praktiken föredras heuristiska lösningar som inte garanterar optimalitet. Men är det bara en tidsfråga innan vi upptäcker någon smart idé för att påskynda även denna beräkning, eller finns det någon inbyggd svårighet som inte kan kringgås? Ett av de viktigaste resultaten inom datavetenskapen ger ett svar på denna fråga. Cook och Levin upptäckte att ett grundläggande problem inom logik (som kallas SAT) är universellt i den mening att en effektiv algoritm för SAT också skulle ge effektiva algoritmer för en generell klass av problem. Senare visade Karp att 21 framstående problem (bland annat handelsresandeproblemet) från många delområden inom datavetenskapen är ekvivalenta med SAT - att lösa ett av dem effektivt skulle också lösa dem alla. Man har inte kunnat bevisa att SAT inte har effektiva algoritmer (detta är det berömda P=NP-problemet) men denna teori förklarar de resultatlösa ansträngningarna att visa att SAT (och en mängd ekvivalenta problem) är svåra. Det är lätt att se de praktiska konsekvenserna av effektiva algoritmer, men även att utesluta sådana algoritmer har mycket viktiga tillämpningar. Faktum är att hela fältet med modern kryptografi och säker kommunikation bygger på antagandet att ingen algoritm effektivt kan bryta de krypteringssystem vi använder, t.ex. när vi skickar våra kreditkortsuppgifter till en onlinebutik.

Cook-Levins resultat och Karps arbete medför att vissa problem är beräkningsmässigt svåra men fortfarande finns naturligtvis ett behov av att kunna lösa sådana problem i praktiken. Vi kan inte förvänta oss effektiva algoritmer som löser alla indata till dessa problem men det kan finnas algoritmer som effektivt löser vissa klasser av intressanta indata. Teorin om parameteriserad komplexitet, även känt som multivariat analys av algoritmer, ger oss verktyg för att studera detta. Den grundläggande idéen är att mäta algoritmers körtid i termer av en eller flera parametrar tillsammans med indatas storlek. Detta gör att vi kan separera olika faktorer som påverkar algoritmens körtid och därigenom studera den på ett djupare plan. Om vi går tillbaka till handelsresandeproblemet kan vi nu börja analysera hur olika egenskaper hos indata (till exempel kartans storlek och antalet destinationer) påverkar en algoritms körtid.

Denna avhandling behandlar multivariat analys av *villkorsproblem* (*constraint satisfaction problems* på engelska och med förkortningen CSP). Ett Sudoku-pussel är ett bra exempel på ett sådant problem. Här har man en 9x9-bräda uppdelad i nio 3x3 rutor. Målet är att placera siffror från 1 till 9 i brädans celler så att varje rad, varje kolumn och varje 3x3 ruta innehåller alla olika siffror. Dessutom innehåller vissa rutor redan siffror, och målet är att fylla i de återstående, tomma cellerna så att den resulterande brädan uppfyller ovanstående begränsningar. CSP är en generalisering av detta: vi får en uppsättning variabler (som tomma celler), en uppsättning tillåtna värden (här 1 till 9) för varje variabel och en uppsättning begränsningar, och målet är att tilldela värden till variablerna (fylla i tomma celler med siffror 1 till 9) så att alla begränsningar är uppfyllda. Medan Sudokus vanligtvis är utformade för att ha lösningar, har vissa CSP:er det inte. I så fall kan man istället vilja hitta en tilldelning av värden som bryter så få begränsningar som möjligt. Detta generella problem kallas MinCSP.

Tyvärr är majoriteten av intressanta MinCSP:er beräkningsmässigt lika svåra som SAT så vi förväntar oss inte effektiva algoritmer för dem alla. I denna avhandling studerar vi MinCSP ur ett multivariat perspektiv. Den relevanta parametern är antalet villkor som inte uppfyllts (tänk på det som en kostnad vi måste betala för att bryta dessa begränsningar—då vill vi betala så lite som möjligt). Till exempel är det ibland omöjligt att utforma ett schema som undviker alla konflikter. Då är det önskvärt att utforma ett som minimerar antalet konflikter. Detta allmänna problem av detta slag kallas MinCSP.

Vi vill utforma effektiva algoritmer för specialfallen av problemet när parametern är liten. I avhandlingen undersöker vi enkla tidsmässiga, intervall-, linjära och likhetsbegränsningar.

- *Enkla tidsmässiga (STP) och intervallbegränsningar* används för att resonera om tid, t.ex. i planerings- och schemaläggningsapplikationer. STP begränsningar tillåter användning av numeriska värden (t.ex. kan de uttrycka påståenden som ”Föreläsningen ska börja och sluta *mellan 8:15 och 11:00*”), medan intervallbegränsningar är utformade för att uttrycka kvalitativa relationer (”Den första sessionen ska schemaläggas *under* den andra sessionen för att ha en gemensam fikapaus *efteråt*.”)
- *Linjära ekvationer* är av grundläggande betydelse inom matematik, teknik och datavetenskap. Några av de äldsta kända algoritmerna var utformade för att hitta lösningar till system av linjära ekvationer. Exempel på tillämpningar finns i antika kinesiska matematiska texter. Metoden som lärs ut i dagens algebra-klasser introducerades av Isaac Newton och systematiserades för datorer av Carl Friedrich Gauss år 1810. Vid den tiden var datorer människor som utförde beräkningar för teknikprojekt, men i stort sett används samma metod fortfarande i digitala datorer idag. Denna metod kan hitta lösningar för konsistenta ekvationssystem eller rapportera att ekvationerna är inkonsistenta. Vi studerar problemet med att beräkna lösningar för nästan konsistenta system, dvs. system som blir konsistenta efter att ha tagit bort några ekvationer.
- *Likhetsbegränsningar* kan användas för att koda talrika egenskaper, t.ex. anslutning i nätverk, likhet och olikhet, vara vänner på sociala medier, osv. Frågan vi ställer kan förstås på följande sätt: om vi har ett stort nätverk och några anslutnings- och

avskiljningsbegränsningar som inte är uppfyllda, kan vi göra några reparationer på nätverket för att uppfylla begränsningarna?

Våra resultat är dikotomi-teorem: för varje specialfall utformar vi antingen effektiva algoritmer för instanser där parametern är liten eller bevisar osannolikheten att sådana algoritmer existerar (likt snabba algoritmer för SAT). Från ett bredare perspektiv främjar de vår förståelse av effektiv beräkning inom området för villkorsproblem och optimeringsproblem.

ABSTRACT

In this thesis we study the computational complexity of MinCSP - an optimization version of the *Constraint Satisfaction Problem* (CSP). The input to a MinCSP is a set of variables and constraints applied to these variables, and the goal is to assign values (from a fixed domain) to the variables while minimizing the solution cost, i.e. the number of unsatisfied constraints. We are specifically interested in MinCSP with infinite domains of values. Infinite-domain MinCSPs model fundamental optimization problems in computer science and are of particular relevance to artificial intelligence, especially temporal and spatial reasoning. The usual way to study computational complexity of CSPs is to restrict the types of constraints that can be used in the inputs, and either construct fast algorithms or prove lower bounds on the complexity of the resulting problems.

The vast majority of interesting MinCSPs are NP-hard, so standard complexity-theoretic assumptions imply that we cannot find *exact* solutions to *all* inputs of these problems in *polynomial time* with respect to the input size. Hence, we need to relax at least one of the three requirements above, opting for either *approximate* solutions, solving *some* inputs, or using *super-polynomial time. Parameterized algorithms* exploits the latter two relaxations by identifying some common structure of the interesting inputs described by some parameter, and then allowing super-polynomial running times with respect to that parameter. Such algorithms are feasible for inputs of any size whenever the parameter value is small. For MinCSP, a natural parameter is optimal solution cost. We also study *parameterized approximation algorithms*, where the requirement for exact solutions is also relaxed.

We present complete complexity classifications for several important classes of infinite-domain constraints. These are *simple temporal constraints* and *interval constraints*, which have notable applications in temporal reasoning in AI, *linear equations* over finite and infinite fields as well as some commutative rings (e.g., the rationals and the integers), which are of fundamental theoretical importance, and *equality constraints*, which are closely related to connectivity problems in undirected graphs and form the basis of studying first-order definable constraints over infinite domains. In all cases, we prove results as follows: we fix a (possibly infinite) set of allowed constraint types $\mathcal{C}$, and for every finite subset Γ of $\mathcal{C}$, determine whether $\textsc{MinCSP}\Gamma$, i.e., MinCSP restricted to the constraint types in Γ, is *fixed-parameter tractable*, i.e. solvable in $f(k) \cdot \mathrm{poly}(n)$ time, where k is the parameter, n is the input size, and f is any function that depends solely on k. To rule out such algorithms, we prove lower bounds under standard assumptions of parameterized complexity. In all cases except simple temporal constraints, we also provide complete classifications for fixed-parameter time constant-factor approximation.

Acknowledgments

There are many people I would like to thank for making my doctoral studies delightful and rewarding, some of them in several capacities. First of all, I would like to thank my main supervisor Peter Jonsson for his constant support in all matters related to research, for being always open for discussions (often outside the standard working hours) and for his generous feedback. I hope I have learned his way of navigating the balance between keeping the research open-minded and fun while managing to get things done in time for deadlines. Further, I would like to thank my secondary supervisor Victor Lagerkvist for the discussions and feedback, and for being exemplary in the way he approaches every task from teaching to giving feedback and writing.

I have been very lucky with collaborators from the very first joint project. I would like to thank all my co-authors: Konrad K. Dabrowski, Leif Eriksson, Peter Jonsson, Victor Lagerkvist, Sebastian Ordyniak, Marcin Pilipczuk, Fredrik Präntare, Roohani Sharma, and Magnus Wahlström. A special thanks goes out to Sebastian and Magnus for introducing me to parameterized algorithms and being my unofficial mentors. It has been a pleasure working with you and learning from all of you!

For having a lively and friendly work environment, I would like to thank my colleagues at TCSLAB: Abhijat, Biman, Christer, Johannes, Jorke, Leif, Peter and Victor. I would also like to thank my colleagues at AIICS for the joint fikas, seminars, and many shared laughs, as well as Amanda and Jonas for making the WASP courses fun and for the many lunches together. Many thanks to Anne and Karin for helping with all administrative matters and putting up with the extra work I generated.

For hosting my numerous research visits and inviting me to workshops I would like to thank Radu Curticapean, Sebastian Ordyniak, Marcin Pilipczuk, Michał Pilicpzuk, Paweł Rzążewski, Stefan Szeider, Roohani Sharma, and Magnus Wahlström. Special thanks to the Amálka, Jana, Karolina, and Tomáš for the warm welcome in Warsaw. I would also like to thank my friends Sophie Gvasalia, Alex Shatberashvili, Giorgi Giglemiani and Mari Khundzakishvili for hosting me during my work trips. For the beautiful cover picture I thank Meri Pepanashvili.

On a more personal note, I would like to start in an unusual way – this might as well be the first precedent in the IDA theses history – by thanking my landlord Mikael Tegler for making us feel more like extended family rather than tenants.

I want to thank my friends and family for their support, especially my parents Tanya and Vova, my brother Roni, my grandma Liana and my uncle Datunka for their love and for putting up with my absence during these years. I thank my parents for encouraging my interests from an early age and for supporting my education, at times at the expense of their own leisure and pleasure, and endorsing my choice to devote my working hours to something I enjoy. It is both delightful and wistful to say that they have succeeded in making my life easier than their own.

Last but not least, I want to thank my beloved wife Salome, who has followed me into the dark Swedish winterland and has been by my side through all the ups and downs of this journey, offering unwavering support and a healthy dose of humour. I could not have wished for a more loving and reliable partner by my side. I also thank Mate for arriving in 2020, turning our world upside down and bringing so much light and joy into our lives.

This work was generously supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

George Osipov
April 2024, Linköping

List of Papers

This thesis is based on the following publications:

1. Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, and George Osipov. "Resolving Inconsistencies in Simple Temporal Problems: A Parameterized Approach." In: *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022).* 2022, pp. 3724–3732

2. Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström. "Almost Consistent Systems of Linear Equations." In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023).* 2023, pp. 3179–3217

3. George Osipov and Magnus Wahlström. "Parameterized Complexity of Equality MinCSP." in: *Proceedings of the 31st Annual European Symposium on Algorithms (ESA 2023).* 2023, 86:1–86:17

4. Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, Marcin Pilipczuk, and Roohani Sharma. "Parameterized Complexity Classification for Interval Constraints." In: *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC 2023).* 2023, 11:1–11:19

Contents

1 Introduction

In this chapter, we frame and motivate the work comprising this thesis in the context of computational complexity, parameterized complexity, constraint satisfaction and approximation algorithms. Formal definitions and technical details are deferred to Chapter 2. We assume familiarity with the basic notions from graph theory (e.g., what is a vertex, an edge, and a path).

1.1 Computational Complexity

We start with two informal examples of *computational problems*: Clique and Vertex Cover. These two will be our protagonists throughout the chapter.

Imagine you are working on designing a new product for your favourite industry. The product consists of k parts, and you have long lists of choices for each. Furthermore, for every pair of choices, you know whether they are compatible or not. Can you determine whether a design is possible, i.e. whether there is a choice of k parts that are pairwise compatible? This problem can be modeled in graph-theoretic terms: create a vertex for every choice, and connect two choices by an edge if they are compatible. Now the goal is to find a set of k vertices in the resulting graph such that all pairs of vertices are edge-connected. Such a set is called a *clique* of size k.

Clique

Instance:	A graph G and an integer k.
Question:	Does G contain a clique of size k?

Now consider a second scenario: your company is hired by the city council to monitor car traffic on the major roads. The city can place an omnidirectional (360-degree) camera at any junction, and with this camera you can monitor all roads meeting at this junction. Can you monitor all major roads by placing k cameras in a strategic way? This problem can also be modeled in graph-theoretic terms: create a vertex for each junction (i.e., each place where a camera can be installed) and add an edge between two vertices if the corresponding junctions are connected by a major road. The goal is to find a set of k vertices in this graph such that every edge has at least one endpoint in the set, i.e., every major road is monitored by at least one camera. Such a set is called a *vertex cover* of size k.

Vertex Cover	
Instance:	A graph G and an integer k.
Question:	Does G contain a vertex cover of size k?

Computational complexity theory studies the amount of resources (e.g. time, space, or communication) required to solve computational problems, like Clique and Vertex Cover. Our main focus will be time complexity. The problems similar in their computational complexity are grouped into *complexity classes*. The main technical tool used to relate complexity of computational problems is *reductions*, which are procedures translating instances of one problem into instances of the other. Importantly, the reductions are subject to the same resource limitations as the problems in the class. Providing a reduction from one problem to another establishes that solving the latter problem is at least as hard as solving the former. Sometimes there are reductions in both directions, which imply that the problems are *equivalent*. For instance, we will see that Clique and Vertex Cover are equivalent under certain efficient reductions. Thus, if we can solve one of them efficiently, then we can solve both.

The most prominent complexity classes are P and NP. The first class contains problems for which solutions can be *found* in polynomial time with respect to the input size, while the second one contains problems for which solutions can be *verified* in polynomial time. Our example problems – Clique and Vertex Cover – are both in NP: if somebody proposed a set of k vertices claiming that this is a clique or a vertex cover, it is straightforward to verify whether the provided set is a solution or not by looking at the edges of the graph. It is quite apparent from formal definitions that P is a subset of NP, i.e. finding a solution is no harder than verifying one. The widely believed hypothesis P≠NP asserts that the inclusion is strict, i.e. there exist problems for which verifying a solution is easier than finding one. The P versus NP question is one of the central challenges in mathematics, and one of the six Millennium Problems [36]. Cook [18] (see also [19]) and Levin [46] independently discovered NP-*complete* problems, which are, informally speaking, the hardest problems in NP. More specifically, these are members of NP such that every other problem in NP reduces to them in polynomial time. The first testament to the explanatory power of NP-completeness is the seminal work of Karp [38], showing that 21 seemingly unrelated computational problems

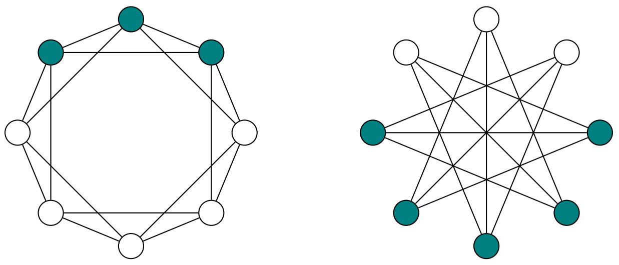
Figure 1.1: A clique in a graph and a vertex cover in its complement.

prominent in their respective application domains are all NP-complete. The book by Garey and Johnson [33] contains many more classic examples.

Example 1.1. Clique and Vertex Cover are both featured on Karp's list. A reduction from Clique to Vertex Cover takes a pair (G, k) of a graph G and an integer k, and returns a new pair $(\overline{G}, |V(G)| - k)$, where $\overline{G}$ is the complement of G, i.e. a graph with the edges and non-edges flipped, and $V(G)$ is the set of vertices in G. The correctness follows by observing that a subset X of vertices in G is a clique if and only if every non-edge of G, or, equivalently, every edge of $\overline{G}$, has one endpoint in $V(G) \setminus X$. The same reduction and proof also work in the other direction, so we establish that these two problems are polynomial-time equivalent. See Figure 1.1 for an illustration. ◂

The theory of NP-completeness explains hardness of a wide range of computational problems, revealing a deep common reason behind it. However, NP-completeness is not a final verdict – we still need to solve instances of hard problems in practice. For this, it is valuable to have a more fine-grained understanding of the relative complexity of computational problems. We believe that NP-complete problems are hard, but it is also useful to know that some are harder than others. *Parameterized complexity* [26] provides us with tools to obtain such an understanding. It offers a multivariate approach towards algorithm analysis: instead of measuring time complexity solely as a function of the input size, parameterized complexity suggests also taking some structure of the inputs into account. Two quintessential problems in this branch are Vertex Cover and Clique parameterized by the solution size k. Let n denote the size of the input instance. A simple branching algorithm can solve Vertex Cover in $O(2^k \cdot n)$ time, while the best algorithms for Clique runs in $n^{O(k)}$ time. For small values of k, e.g. $k = 15$, and moderate values of n, an algorithm with running time $2^k \cdot n$ is reasonable, while n^k is completely infeasible. Problems that can be solved in $f(k) \cdot n^{O(1)}$ time for some computable function f that depends only on the parameter k are called *fixed-parameter tractable*. In contrast to Vertex Cover, we do not expect algorithms with such running time for Clique, and the hard problems for parameterized complexity are those at least as hard as Clique.

1.2 Constraint Satisfaction Problems

The topic of this thesis is parameterized complexity of *constraint satisfaction problems (CSPs).* An instance of a CSP is a set of constraints applied to (possibly overlapping) subsets of variables. The problem asks to check if there is an assignment of values to the variables that satisfies all constraints. The computational problem CSP(Γ) is defined by a constraint language Γ, which restricts the domain of values for each variable as well as the kinds of *relations* that the constraints can impose on the variables of the instance. The framework of CSP is very expressive and allows modeling a wide range of computational problems.

Example 1.2. 3-SAT is the problem of deciding whether a propositional formula in conjunctive normal form, i.e. a conjunction of ternary disjunctive clauses is satisfiable. We can model it as a CSP with variables being the variables of the formula and clauses imposing ternary constraints. For example, a clause $(x \lor y \lor \neg z)$ can be viewed as a constraint imposed on (x, y, z) that allows every assignment except $(0, 0, 1)$. ◂

Example 1.3. A system of linear equations over a ring R can be cast as an instance of CSP with domain R. An equation $ax + by + cz = d$ for any $a, b, c, d \in R$ is a ternary constraint saying that the assignment to (x, y, z) should be equal to one of the triples $(\alpha, \beta, \gamma) \in R^3$ such that $a\alpha + b\beta + c\gamma = d$. ◂

Example 1.4. 3-Coloring is another problem on Karp's list. An instance is a graph, and the question is whether every vertex in the graph can be assigned one of three distinct colors so that no pair of adjacent vertices receives the same color. We can cast it as a CSP with domain $\{1, 2, 3\}$ and Γ containing binary relation $\neq$. Intuitively, an edge uv in the input graph imposes the constraint $u \neq v$ on the colors assigned to the vertices u and v. ◂

Example 1.5. Digraph Acylicity asks, given a directed graph as input, is it acyclic, i.e. does it contain no directed cycle? The problem can be equivalently stated as follows: given a directed graph as input, does it admit a topological ordering, which is a mapping π that assigns (e.g. rational) numbers to the vertices so that for every arc (u, v), we have $\pi(u) < \pi(v)$? This suggests casting Digraph Acylicity as a CSP with domain $\mathbb{Q}$ and constraint language $\Gamma = \{<\}$. ◂

From the theoretical point of view, CSPs form a nicely behaved subset of NP. More specifically, even though the Cook-Levin Theorem provides us with a way to classify problems as tractable (in P) and intractable (NP-complete), we cannot hope for a *dichotomy*: Ladner's Theorem [45] asserts that if P≠NP, there are problems of intermediate complexity, so we cannot classify every NP problem as polynomial-time solvable or NP-complete. On the other hand, Bulatov [9] and Zhuk [57] independently proved a dichotomy theorem for finite-domain CSPs, confirming the celebrated conjecture of Feder and Vardi [29] and the algebraic conjecture of Bulatov, Jeavons and Krokhin [8]. As a bonus, the tractability criterion for CSPs is decidable and algebraically meaningful: informally, the fact that a finite-domain CSP admits an efficient algorithm is witnessed by certain symmetries of Γ, and the lack of these symmetries implies NP-hardness. For example, suppose the domain of Γ is an Abelian group with operation +, and the following property holds: if we take any three assignments $(a_1, \dots, a_n)$, $(b_1, \dots, b_n)$ and $(c_1, \dots, c_n)$ that satisfies an instance I of CSP(Γ), then the assignment $(d_1, \dots, d_n)$ defined as $d_i = a_i - b_i + c_i$ for all $1 \le i \le n$, also satisfies I. Then CSP(Γ) is solvable in polynomial time by a variant of Gaussian elimination. The constraint language of linear equations from Example 1.3 admits such a symmetry, while the language of 3-Coloring from Example 1.4 does not.

1.3 Optimization with Constraints

Define the *cost of an assignment* to an instance of CSP to be the number of constraints it does not satisfy. In the optimization version called MinCSP, the instance is a pair (I, k), where I is a CSP instance and $k \in \mathbb{N}$ is a budget. The problem asks if I admits an assignment of cost at most k. MinCSP generalizes CSP (which is a special case with $k = 0$)

Example 1.6. Many optimization problems can be phrased as MinCSP.

1. Let $\mathbb{Q}$ be the domain of Γ and binary $<$ be the allowed relation. MinCSP(Γ) is equivalent to Directed Feedback Arc Set that asks to delete k arcs from a directed graph to make it acyclic.

2. Let $\{0,1\}$ be the domain of Γ and binary disequality ($\neq$) be the allowed relation. CSP(Γ) is equivalent to checking if a graph is bipartite. MinCSP(Γ) is equivalent to Minimum Bipartization that asks to delete k edges from a graph to make it bipartite. The dual of this problem is the problem of finding a maximum bipartite subgraph also known as Maximum Cut.

3. Let $\mathbb{N}$ be the domain of Γ and binary $=$ and $\neq$ be the allowed relations. MinCSP(Γ) is equivalent to Multicut which takes a graph and a set of vertex pairs called cut requests as input. A request st is satisfied in the graph if s and t are in distinct connected components. The problem asks to delete k edges so that every cut request is satisfied.

Moreover, MinCSP(Γ) is also equivalent to a generalization of the Correlation Clustering problem [1]: the input is a dataset which comes with a qualitative similarity measure: for every pair of data points, the measure tells us that they are either similar, dissimilar, or the relation is unclear. The goal is to assign data points to clusters while minimizing dissimilarity within each cluster and similarity between distinct clusters. To model this problem as MinCSP(Γ), for every pair of data points p_1 and p_2, we add a constraint $p_1 = p_2$ if p_1 and p_2 are similar, $p_1 \neq p_2$ if they are dissimilar, and no constraint if the relation is unclear. ◂

In turn, MinCSP is a special case of the *Valued Constraint Satisfaction Problem* (VCSP), where relations are replaced by *cost functions* that determine the cost of every local assignment to a constraint. The goal is to find an assignment that minimizes the total cost. The costs may be finite or infinite. For example, if every cost is 0 or ∞, then we are effectively solving a CSP. MinCSP corresponds to the case with costs 0 and 1. Negative costs (which can be thought of as rewards for certain assignments) are also allowed in the VCSP.

Like CSP, MinCSP and more generally VCSP also lends itself to dichotomy theorems [43, 56]. For example, Thapper and Živný proved that every tractable VCSP with a finite domain and finite costs can be solved in polynomial time using the basic linear programming relaxation, and otherwise, it is NP-hard. It is natural to ask, how does the borderline look like if we allow more running time, namely fixed-parameter tractable time in the budget k?

Which MinCSPs parameterized by solution cost are fixed-parameter tractable and which are as hard as Clique?

This question is practically motivated since the class of problems solvable by linear programming is quite meager, and many important optimization problems are on the NP-hard side of the classification. However, in practice, we can often assume that the budget k is small and utilize this for efficient computation. For example, one special case of MinCSP is the problem of correcting few errors/inconsistencies occurring in a dataset. This could be the case if the dataset contains few outliers, comes from faulty measurements, or from merging datasets that almost agree.

From the point of view of parameterized complexity, MinCSP parameterized by solution cost is an expressive framework for parameterized deletion problems in which the goal is to delete few elements (e.g. edges from a graph) to achieve the desired property (e.g. bipartiteness). See Table 1.1 for examples. The study of MinCSP-related problems has been very fruitful in terms of discovering influential algorithmic techniques in parameterized complexity. The search for dichotomies has been successful in highlighting limitations of existing methods and led to the introduction of new and powerful methods. One success story started with the project of Chitnis, Egri, and Marx [15, 16]

Problem name	MinCSP	Status
Minimum st-Cut	$\{0,1\}$; 0, 1, $x \to y$	P [32]
Bipartization	$\{0,1\}$; $x \neq y$	FPT [54]
Multiway Cut	$\mathbb{N}$; $x = y$, $x = i$ for $i \in \mathbb{N}$	FPT [49]
DFAS	$\mathbb{Q}$; $x < y$	FPT [12]
Almost 2-SAT	$\{0,1\}$; $x \neq y$, $x \to y$	FPT [53]
Paired st-Cut	$\{0,1\}$; 0, 1, $x_1 \to y_1 \wedge x_2 \to y_2$	W-h [50]
Multicut	$\mathbb{N}$; $x = y$, $x \neq y$	FPT [7, 51]
Subset DFAS	$\mathbb{Q}$; $x < y$, $x \leq y$	FPT [14]
Unique Label Cover	$\{1, \ldots, d\}$; all permutations	FPT [13]
Chain SAT	$\{0,1\}$; 0, 1, $x_1 \to x_2 \to x_3 \to x_4$	FPT [40]
Min-2-Lin	$\mathbb{F}_{p^q}$, $\mathbb{Q}$ or $\mathbb{Z}$; $ax + by = c$	FPT [24]

Table 1.1: Examples of deletion problems modeled as MinCSP. The first column contains the domain and the description of the relations separated by a semicolon. DFAS stands for Directed Feedback Arc Set. 0 and 1 stand for unary assignments of 0 and 1, respectively. Permutation relations are $\{a, \pi(a) : a \in S\}$ for a subset S of $\{1, \ldots, d\}$ and a bijection π on $\{1, \ldots, d\}$. $\mathbb{F}_{p^q}$ is a finite field, i.e., p is a prime and q is an arbitrary positive integer.

who identified Chain-SAT as the missing piece in the full classification for certain graph homomorphism problems. In the quest to solve Chain-SAT, Kim, Kratsch, Pilipczuk, and Wahlström [42, 40, 41] invented a new technique called *flow augmentation* for solving intricate directed cut problems. Not only did they complete the project of Chitnis, Egri, and Marx, but also resolved parameterized complexity of all Boolean CSPs parameterized by solution cost [41]. Furthermore, flow augmentation was the missing ingredient for solving Directed Multicut with Three Terminal Pairs [34], which was open for more than a decade. It has also been immensely useful for several algorithms presented in this thesis.

Another takeaway from Table 1.1 is that many useful properties such as acyclicity and undirected connectivity are naturally modeled by *infinite-domain* CSPs. Such CSPs have applications in artificial intelligence, scheduling, computational linguistics, optimization and other subfields of computer science. However, there is no hope of obtaining a dichotomy theorem for infinite-domain CSPs since they can have arbitrary complexity [3, 37]. This is an obstacle to studying MinCSP for infinite domains since MinCSP(Γ) is only interesting when the decision version CSP(Γ) is solvable in polynomial time (otherwise, the problem is already hard for $k = 0$). A better-behaved class of infinite-domain CSPs are *reducts of finite homogeneous structures*; we refer to Bodirsky's definitive book [2] for technical details. For now, we only mention that this class of CSPs includes a host of interesting examples from the interface with other fields, including many problems that have been previously mentioned, and there is a dichotomy conjecture for this class.

The conjecture has been confirmed for many special cases, e.g. for equality constraints [4] and temporal constraints [5]. MinCSPs with equality and temporal constraints generalize Multicut and DFAS, respectively, so they are worthy targets for a parameterized complexity study. The main focus of this thesis is on infinite-domain MinCSPs.

1.4 Approximation Algorithms

As discussed in the previous section, many prominent MinCSPs are NP-hard. Thus, assuming P≠NP, there is no algorithm that can compute optimal solutions to worst-case instances of these problems in polynomial time. Three common ways to relax these requirements are

(1) opting for *approximate* solutions,

(2) restricting attention to *a subset of instances*, or

(3) allowing *more than polynomial time.*

The field of *polynomial-time approximation* algorithms deals with the relaxation of the first kind – instead of insisting on obtaining optimal solutions, which can be prohibitively slow to compute, one opts for solutions that are guaranteed to be within a factor of the optimum and feasible to compute. Parameterized complexity can be used with the latter two relaxations – if we can identify a meaningful parameter and design an algorithm that runs in fixed-parameter time with respect to the parameter, then we can solve arbitrary instances with small parameter values.[1]

There are problems that are hard for both polynomial-time approximation within any constant factor and exact fixed-parameter tractable algorithms, but amenable to the combination of these approaches, i.e. approximation algorithms running in fixed-parameter tractable time. The surveys by Marx [48] and Feldmann, Lee, and Manurangsi [30] provide a good introduction to this line of inquiry.

Example 1.7. Here are some examples illustrating the varying complexity of MinCSPs in terms of polynomial-time approximation and parameterized complexity.

1. MinCSP over domain $\mathbb{N}$ with relations $x = 0$ and $x \neq y$ is very close to Vertex Cover [52]. It admits a folklore polynomial-time 2-approximation algorithm and a folklore fixed-parameter tractable algorithm.

[1] As a side note, in the context of MinCSP viewed as the problem of fixing inconsistencies in data, one can think of a polynomial-time approximation as treating multiplicative errors (e.g. $\approx 0.1\%$ of ten thousand data points are scrambled), while the parameterized complexity treats additive error (e.g. ≈ 20 data points in a million are corrupt).

2. MinCSP over domain $\mathbb{N}$ with relations $x = y$ and $x \neq y$ is equivalent to Multicut. It admits no constant-factor approximation in polynomial time unless the Unique Games Conjecture (UGC) fails [10, 39]. In contrast, this problem is in FPT [51].

3. MinCSP over domain $\{0, 1\}$ with relations $x_1 \to y_1 \wedge x_2 \to y_2$ and $1 \to x$, $x \to 0$ is W[1]-hard [50] but 2-approximable in polynomial time by considering each implication in the conjunction separately and running a Minimum *st*-Cut algorithm with twice the budget of the original MinCSP instance.

4. MinCSP over domain $\{0, 1\}$ with relations $x = 0$ and $x + y + z = 1 \bmod 2$ expressed Odd (Hitting) Set. It is NP-hard and W[1]-hard to approximate within any constant factor [6]. ◀

FPT-approximation is well-motivated for infinite-domain MinCSPs: arguably the simplest NP-hard MinCSP, which is MinCSP(Γ) with domain $\mathbb{N}$ and relations $=$ and $\neq$, is equivalent to Multicut, and thus admits no polynomial-time constant-factor approximation under the UGC. Thus, allowing more time to compute exact or approximate solutions makes sense in this setting. Moreover, there are examples of prominent problems including Multicut [47] for which we can trade-off solution quality for efficiency: currently known approximation algorithms are much faster and simpler than the exact algorithms. FPT-approximation is one of the topics explored in the thesis.

2 Background

The aim of this chapter is to provide a reader with the technical background required for the rest of the thesis. Although each upcoming paper contains a specific section on technical preliminaries, the streamlined presentation with more space for details and examples may benefit the reader. This chapter also complements Chapter 1 with the missing formal definitions.

Section 2.1 contains basic definitions from computational and parameterized complexity and can be safely skimmed or skipped by a reader familiar with the topics. Section 2.2 includes formal definitions of the constraint satisfaction problem (CSP) and related optimization problems VCSP and MinCSP, as well as a discussion of approximation algorithms for MinCSP. Section 2.3 introduces the techniques from the parameterized complexity toolbox used in solving MinCSP problems.

2.1 Computational Complexity

Computational complexity is a branch of theoretical computer science that studies the amount of resources (time, space, communication, etc.) required to solve computational problems, and strives to compare problems with respect to their complexity. To have a mathematical discussion on this topic, we first need to agree on the model of computation, the definition of a computational problem and the measures of complexity. This section starts by briefly providing these definitions; a reader interested in a comprehensive treatment is referred to the textbook by Sipser [55]. The first part is followed by definitions of the fundamental computational classes P and NP, and the notion of polynomial-time reductions. The basics of parameterized complexity are treated in the last section.

Computational Problems and Time Complexity

Recall the Clique problem from Section 1.1: given a graph G and an integer k, does G contain a clique of size k? A computational problem like Clique is formally modeled as a set of yes-instances. Fix an alphabet, say $\Sigma = \{0,1\}$, and let Σ^* be the set of (finite) strings over this alphabet. For example, (G,k) can be encoded in Σ by listing all edges of G (edges are vertex pairs; vertices can be indexed in an arbitrary way, and then the indices can be written in binary) followed by k written in unary or binary. Let A_{Clique} be the set of encodings of (G,k) such that G contains a clique of size k. The decision problem for a set $A \subseteq \Sigma^*$ is the following: given an input string over Σ, decide whether it belongs to A or not. Clearly, the decision problem for A_{Clique} is the same problem as Clique.

To reason about the complexity of a problem, we need to fix a model of computation. The standard choice in theoretical computer science is a simple yet universal[1] *Turing machine*. This abstract machine consists of a tape representing memory and an algorithm, i.e. a finite set of instructions. The machine works step by step, accessing one symbol from its memory at a time, and takes an action based on its internal state, the currently retrieved symbol and the instructions of the algorithm. A Turing machine can write on its tape, i.e. commit information to its memory. The computation is finished when one of the two final states (accept or reject) is reached.

We can safely fix our alphabet $\Sigma = \{0,1\}$ for the rest of the thesis. The only remaining part in the definition of a Turing machine is the algorithm, so we will refer to them interchangeably. The running time of a Turing machine on an input $w \in \Sigma^*$ is the number of steps it takes to reach a final state starting with w written on its tape. The *time complexity* is a function $t : \mathbb{N} \to \mathbb{N}$ such that the running time is at most $t(|w|)$ on every input $w \in \Sigma^*$. Here $|w|$ denotes the number of symbols (e.g. bits if $\Sigma = \{0,1\}$) in the encoding of w. It is common to denote $|w|$ by n.

In theoretical computer science we are usually less interested in the exact values of the function t, but rather in its asymptotic behaviour. The big-O notation is very convenient for this purpose. For example, we say that an algorithm (i.e. a Turing machine) runs in $O(n)$ time if there exists a constant c such that $t(n) \le c \cdot n$ for all $n \in \mathbb{N}$.

Polynomial Time and Reductions

Polynomial and polylogarithmic running times (like $O(n)$, $O(n \log n)$, $O(n^2)$) arise naturally in the analysis of algorithms. This has led Cobham [17] and Edmonds [27] to postulate that polynomial time captures the notion of efficient computation. The class P contains all problems that admit polynomial-time algorithms. The class NP is defined similarly, but the computational device

[1]Assuming that the Extended Church-Turing Thesis holds.

is replaced with a *non-deterministic Turing machine*, which can branch into (finitely many) states at each step instead of following one deterministic computational path like the classical Turing machine.

It is more intuitive to think about P and NP as the classes of problems for which the solution can be *found* and *verified* in polynomial time, respectively. For our running example Clique, given an instance (G, k) together with an intended solution $X \subseteq V(G)$, one can easily verify whether $|X| = k$ and whether there exists an edge in G between every pair of vertices in X. Moreover, this verification only requires polynomial time. However, finding such a set X in polynomial time seems more difficult. Thus, Clique is in NP, but it is unknown whether it is in P or not. However, what is known is that if Clique is in P, then in fact $\mathsf{P} = \mathsf{NP}$. This is a corollary of the Cook-Levin theorem and Karp's pioneering work [38]. To understand it, we need to introduce *reductions* between computational problems.

A *mapping reduction* $f : \Sigma^* \to \Sigma^*$ from a problem A to B is a computable function such that

$$w \in A \iff f(w) \in B$$

for all $w \in \Sigma^*$. The term 'computable' means that there exists a Turing machine such that, starting with w written on its tape, it reaches a final state with $f(w)$ on the tape. In other words, there is an algorithm that produces $f(w)$ given w as input. Thus, if we want to solve problem A and we have an algorithm for problem B together with an algorithm for f, we can pipeline the algorithm for f with the algorithm for B, obtaining an algorithm for A. Observe that the final running time depends on the running time of the algorithms for f and B. For instance, it follows immediately from the definitions that the class P is closed under polynomial-time reductions.

A problem $B \in \mathsf{NP}$ is NP-*complete* if there is a polynomial-time reduction from every problem in NP to B. In a sense, these are the hardest problems in the class: if one obtains a polynomial-time algorithm for any NP-complete problem, then $\mathsf{P} = \mathsf{NP}$ by pipelining the reduction with the algorithm. The quintessential NP-complete problem is SAT – the problem asking if a propositional Boolean formula is satisfiable. The problems Clique and Vertex Cover are two more examples of NP-complete problems. Example 1.1 contains a description of polynomial-time reductions between these problems.

Basics of Parameterized Complexity

Parameterized complexity [26] takes a multivariable approach towards time complexity. Instead of a univariable function that only depends on the size of the input, we can measure the running time with a multivariate function $t : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, where the first argument is still the size of the instance, while the second can encode any parameter describing the structure of the instance. For example, we might be interested in solving Clique or Vertex Cover for very small values of k, say $k = 5$. Then it is instructive to parameterize the problem by k. Formally, a *parameterized problem* is a pair (Q, p), where $Q \subseteq \Sigma^*$ is the instance and $p \in \mathbb{N}$ is the parameter. To clarify the example above, in case of Clique the instance Q is the encoding of (G, k) in binary, while the parameter p is k.

One can solve Clique in $n^{O(k)}$ time by enumerating all k-subsets of vertices and checking if they form a clique in the graph. For very small values of k and small values of n, this algorithm can be practical. However, it is more desirable to have a running time of the form $f(k) \cdot n^{O(1)}$, where the dependence on the parameter k is decoupled from the dependence on the instance size. The running time of this form is called *fixed-parameter tractable*, or *fpt time* for short. The class of problems that admit fpt-time algorithms is called FPT. This class is the parameterized counterpart to the class P. We remark that there is no single counterpart to the class NP in the parameterized world.

Before delving into what it means for a parameterized problem to be hard, let us first define the reductions that preserve FPT. Intuitively, such reductions need to run in fpt time, map yes-instances to yes-instances, map no-instances to no-instances, and keep the parameter small. Formally, an *fpt-reduction* $R : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ between two parameterized problems $A, B \subseteq \Sigma^* \times \mathbb{N}$ is a fpt-time computable function such that

$$(w, k) \in A \iff R(w, k) = (w', k') \in B,$$

and there exists a function $g : \mathbb{N} \to \mathbb{N}$ such that

$$k' \leq g(k).$$

The latter property is crucial in allowing pipelining of a reduction with a fixed-parameter tractable algorithm.

A canonical hard problem for parameterized complexity is Clique parameterized by the solution size. It is complete for the class $\mathsf{W}[1]$, i.e. every problem in $\mathsf{W}[1]$ admits an fpt-reduction to Clique. Hard classes of parameterized problems form the so-called *weft hierarchy*

$$\mathsf{W}[1] \subseteq \mathsf{W}[2] \subseteq \cdots \subseteq \mathsf{W}[P].$$

Formal definitions of the classes and further complexity-theoretic discussions can be found in the books by Downey & Fellows [26] and Flum & Grohe [31]. For our purposes, it is enough to state the central assumption of the field that $\mathsf{FPT} \neq \mathsf{W}[1]$, which is equivalent to saying that Clique cannot be solved in fpt time. An even stronger lower bound follows from the widely believed Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi and Zane [35], which states that Boolean Satisfiability problem cannot be solved in subexponential, i.e. $2^{o(n)}$, time. In other words, ETH implies $\mathsf{FPT} \neq \mathsf{W}[1]$ [20].

2.2 Constraint Satisfaction Problems

Fix a set of values D called the *domain.* A *relation* of *arity* r over D is a subset of tuples $R \subseteq D^r$. A *constraint* $R(X)$ applied to the tuple of variables $X = (x_1, \dots, x_r)$ is *satisfied* by an assignment $\alpha : V \to D$ if $\alpha(X) = (\alpha(x_1), \dots, \alpha(x_r)) \in R$. The tuple X is called the *scope* of the constraint. A *constraint language* Γ is a set of relations over D. Consider the following computational problem.

CONSTRAINT SATISFACTION PROBLEM FOR Γ (CSP(Γ))

INSTANCE:	$I = (V, C)$, where V is a set of variables, and C is a set of constraints of the form $R(X)$, where $R \in \Gamma$ is a relation of arity r and $X \in V^r$.
QUESTION:	Is there an assignment satisfying all constraints in C?

The *cost* of an assignment α to an instance I of CSP is the number of constraints in I that α does not satisfy. The *cost of I* is defined as the minimum cost of any assignment to I. Equivalently, it is the minimum size of a subset $X \subseteq C$ such that $C \setminus X$ is satisfiable. Thus, CSP asks to decide if the input instance has cost 0. An optimization version is defined as follows.

MINIMUM-COST CSP FOR Γ (MINCSP(Γ))

INSTANCE:	An instance $I = (V, C)$ of CSP(Γ) and a positive integer k.
QUESTION:	Is there an assignment to I of cost at most k?

The main object of study in this thesis is MINCSP parameterized by solution cost k. It is convenient to allow two kinds of constraints in our instances: *soft* constraints that incur cost one, and *crisp* constraints that incur infinite cost, and are thus satisfied by every finite-cost assignment.

A generalization of CSP and MINCSP called *valued constraint satisfaction problem (VCSP)* is defined by replacing relations with *cost functions.* Given a relation $R \subseteq D^r$, we define a cost function $f : D^r \to \mathbb{Q} \cup \{\infty\}$ such that $f(d_1, \dots, d_r) = 0$ if $(d_1, \dots, d_r) \in R$. For the tuples $(d_1, \dots, d_r) \notin R$, we can set $f(d_1, \dots, d_r) = \infty$ in the CSP setting, while in MINCSP we can use $f(d_1, \dots, d_r) = 1$ for *soft* constraints and $f(d_1, \dots, d_r) = \infty$ for *crisp* constraints. Then a relational constraint $R(X)$ can then be replaced by a functional constraint $f(X)$. More generally, fix a set of functions $\mathcal{F}$ mapping tuples over D to $\mathbb{Q} \cup \{\infty\}$. An instance of VCSP($\mathcal{F}$) with variables V consists of a set of functional constraints $\{f_i(X_i)\}_{i=1}^m$, where $\mathcal{F} \ni f_i : D^{r_i} \to \mathbb{Q} \cup \{\infty\}$ and $X \in V^{r_i}$. The goal is to minimize the total cost over all assignments from

V to D, i.e. to find

$$\arg\min_{\alpha: V \to D} \sum_{i=1}^{m} f_i(\alpha(X_i)).$$

CSP Reductions and Approximation

Primitive positive definitions (pp-definitions) are a model-theoretic notion that captures combinatorial gadgets in the reductions between CSPs. Let Γ be a constraint language over D and let $\Gamma_=$ be the language obtained by adding the binary equality relation $\{(d,d) : d \in D\}$ to Γ. A *pp-definition* of relation $R \in D^r$ in Γ is an instance $I_R = (V, C)$ of CSP$(\Gamma_=)$ with variables of V partitioned into *primary variables* $X = (x_1, \dots, x_r)$ and *auxiliary variables* Y, such that the following hold.

1. If $\alpha : V \to D$ satisfies I_R, then $\alpha(X) \in R$.
2. For every $(d_1, \dots, d_r) \in R$, there is an assignment $\alpha : V \to D$ satisfying I_R such that $\alpha(x_i) = d_i$ for all $i \in [r]$.

In logical term, a pp-definition of R over Γ is an existentially quantified formula such that

$$R(X) \equiv \exists Y.\ \phi(X, Y),$$

where ϕ is a conjunction of formulas of the form $R'(v_1, \dots, v_r)$ for $R' \in \Gamma_=$ using variables in $X \cup Y$. Intuitively, if R admits a pp-definition in Γ, then we can simulate constraints $R(X)$ by introducing auxiliary variables Y, replacing the constraint with I_R, and eliminating equality constraints by identifying every pair of variable u and v such that $u = v$ is a constraint in I_R. Consequently, we obtain the following.

Proposition 2.1. *If CSP*(Γ) *is in* P *and* Γ *pp-defines a relation R, then CSP*$(\Gamma \cup \{R\})$ *is in* P.

Pp-definitions are of limited use for MinCSP since they do not preserve costs. For example, the relation

$$R = \{(a,b,c,d) \in \mathbb{Q}^4 : a < b \wedge c < d\} \tag{2.1}$$

admits a simple pp-definition in the language $\{<\}$, namely $\{x_1 < x_2, x_3 < x_4\}$. However, if we replace a constraint $R(x_1, x_2, x_3, x_4)$ by $x_1 < x_2$ and $x_3 < x_4$ in an instance of MinCSP, the minimum cost of an assignment to the resulting instance may increase by one because violating $x_1 < x_2$ and $x_3 < x_4$ incurs a cost of two. Another notion particularly useful for parameterized complexity reductions is *(strict) implementations.* A pp-definition I_R in Γ is an implementation of R if the following holds.

3. For every $(d_1, \dots, d_r) \notin R$, there is an assignment $\alpha : V \to D$ of cost 1 such that $\alpha(x_i) = d_i$ for all $i \in [r]$.

The latter can be interpreted as follows: if an assignment does not satisfy $R(X)$, then it can be extended to the auxiliary variables Y so that the resulting assignment does not satisfy only one constraint in I_R.

For example, the relation $<$ on the domain $\mathbb{Q}$ can be implemented in the language $\{\leq, \neq\}$ Namely, one can replace a constraint $u < v$ with $I_< = \{u \leq v, u \neq v\}$ and no auxiliary variables. If $\alpha(u) < \alpha(v)$, then α clearly satisfies $I_<$. On the other hand, if $\alpha(u) \nless \alpha(v)$, then either $\alpha(u) = \alpha(v)$ and α satisfies $u \leq v$ but not $u \neq v$, or $\alpha(u) > \alpha(v)$, in which case α satisfies $u \neq v$ but not $u \leq v$. In both cases, the cost of α is 1.

The utility of implementations is summarized below.

Proposition 2.2. *If* MinCSP(Γ) *is in* FPT *and* Γ *implements* R*, then* MinCSP$(\Gamma \cup \{R\})$ *is in* FPT*.*

Pp-definitions can still be used for special kinds of reductions in the MinCSP setting.

Proposition 2.3. *If* MinCSP(Γ) *is in* FPT *and* Γ *pp-defines* R*, then* MinCSP$(\Gamma \cup \{R\})$ *is in* FPT *on instances that use* R *only in crisp constraints.*

In VCSP-inspired terminology, one could say that pp-definitions preserve 0 and ∞ costs, but no other costs. Thus, they can be used for gadgets that replace crisp constraints and cannot be used to replace soft ones. In contrast, implementations also preserve unit costs, so they can replace soft constraints with costs 0 and 1.

A pp-definition is *equality-free*, or a *eqpp-definition* for short, if it only used the relations of Γ (as opposed to $\Gamma_=$). Eqpp-definitions have further usage in approximation algorithms. Note that in the example with the relation from (2.1) we observed that replacing $R(x_1, x_2, x_3, x_4)$ with $\{x_1 < x_2, x_3 < x_4\}$ can increase the cost by one. Thus, in total, it can increase the cost of a yes-instance from k to at most $2k$, i.e. the cost stays within a constant factor of k. To make the idea precise, we introduce the following problem defined for every $\gamma \geq 1$.

Gap$_\gamma$MinCSP(Γ)

Instance:	An instance I of CSP(Γ) and an integer k.
Goal:	Distinguish between the following cases:
(YES)	The cost of I is at most k.
(NO)	The cost of I is greater than γk.

Note that Gap$_1$MinCSP(Γ) is exactly the same problem as MinCSP(Γ), while for every $\gamma > 1$ it is a relaxation allowing arbitrary answers if the cost is within $(k, \gamma k)$. An algorithm solving Gap$_\gamma$MinCSP(Γ) is called a *γ-factor approximation* or simply a *γ-approximation*. Note that if such an algorithm returns YES, then the instance is guaranteed to have an assignment of cost at most γk. The connection to eqpp-definitions is formulated as follows.

Proposition 2.4 (Lemma 10 in [6]). *Suppose there is an eqpp-definition of relation R in Γ consisting of c constraints. If* Gap$_\gamma$MinCSP(Γ) *is in* P *(*FPT*), then* Gap$_{c\gamma}$MinCSP$(\Gamma \cup \{R\})$ *is in* P *(*FPT*).*

2.3 Parameterized Deletion Toolbox

This section provides an overview of the tools used in parameterized algorithms illustrated by applications to MinCSPs. To motivate the section, we start with an extensive example.

Casting Problems as MinCSP

Many problems including a number of milestones in the development of parameterized complexity can be modeled as MinCSP – see Table 1.1. Here we elaborate further on these examples.

1. Minimum st-Cut in directed graphs can be modeled as Boolean MinCSP, i.e. MinCSP with domain $\{0, 1\}$, with implication and assignment relations: arcs (s, v) translate to assignment constraints $1 \to v$, arcs (v, t) translate to assignment constraints $v \to 0$, and arcs (u, v) translate to implications $u \to v$. Such an instance is consistent if and only if there is no path of implications from a 0-variable to a 1-variable, i.e. the constraints of the instance do not imply $1 \to 0$. Deleting an implication in the resulting instance of MinCSP corresponds to removing an arc from the original directed graph.

Paired st-Cut is a variant of this problem where the arc set is partitioned into pairs, and the goal is to delete k pairs so that no st-path remains. While Minimum st-Cut is famously in P, Paired st-Cut is NP-hard and W[1]-hard [50]. The way to model this problem as a Boolean MinCSP is similar to the one described above: translate arc pairs $((u_1, v_1), (u_2, v_2))$ into constraints $u_1 \to v_1 \land u_2 \to v_2$.

2. Minimum Bipartization is modeled as Boolean MinCSP by translating edges uv of the graph to disequality constraints $u \neq v$. If the original graph has an odd cycle, then the resulting instance has no Boolean solution, e.g. $\{u \neq v, v \neq w, w \neq u\}$ is inconsistent over $\{0, 1\}$. The converse also holds, and correctness follows since having no odd cycles is equivalent to being bipartite. The problem is NP-hard (see Example 1.6) and fixed-parameter tractable by an algorithm of Reed, Smith, and Vetta [54]. They introduced the technique of *iterative compression*, which has become a standard opening step for parameterized algorithms (see Chapter 4 in [20] for numerous examples).

3. Minimum Multiway Cut is the problem in undirected graphs that asks, given a graph and a subset of its vertices called terminals, to delete few edges so that all terminals become disconnected. Minimum st-Cut on undirected

graphs is a special case with only two terminals. With three or more terminals, the problem is NP-hard [25] and fixed-parameter tractable [49]. To model it as MinCSP, enumerate the terminals $t_1, \dots, t_p$, add an undeletable constraint $t_i = i$ for every i, and constraints $u = v$ for every edge uv in the input graph. The resulting CSP instance is consistent if and only if no two terminals are connected by a path of equalities, i.e. they lie in different connected components. In his work on Multiway Cut, Marx [49] introduced *important separators*, which have since become indispensable in graph separation problems and several other contexts (see Chapter 8 in [20]).

4. Directed Feedback Arc Set asks to delete few arcs from a directed graphs to make it acyclic. This problem is NP-hard [38] and fixed-parameter tractable [12]. The algorithm of Chen, Liu, Lu, Razgon, and O'Sullivan [12] uses iterative compression and guessing in fpt time to reduce DFAS to a separation problem called Skew Multicut and then solve it using important separators. This blueprint for parameterized deletion problems has been used in many subsequent works. It is also present in the Bipartization algorithm of [54], where iterative compression and fpt guessing reduce the problem to Minimum st-Cut.

5. In Almost 2-SAT the input is a propositional formula consisting of 2-clauses, and the goal is to delete few clauses to make it satisfiable. It generalizes Minimum st-Cut and Minimum Bipartization, and hence is NP-hard. The fpt algorithm of Razgon and O'Sullivan [53] follows the same blueprint of iterative compression and fpt guessing, reducing Almost 2-SAT to a separation problem.

6. Multicut is discussed in Example 1.6. This problem generalizes Multiway Cut (which is a special case with cut requests forming a clique), is NP-hard and in FPT parameterized by the solution size. Marx and Razgon [50] first gave a 2-approximation fpt algorithm for this problem with a series of reductions ending in Boolean MinCSP with relation $x_1 \to y_1 \wedge x_2 \to y_2$ and unary assignments $x = 0, x = 1$. In the context of the CSP, the constraint of the form $x_1 \to y_1 \wedge x_2 \to y_2$ introduces no extra complexity: it can be viewed as two separate constraints $x_1 \to y_1$ and $x_2 \to y_2$, and the consistency-checking problem is then equivalent to Minimum st-Cut. However, in the context of optimization, allowing deletion of both constraints at cost 1 makes a difference: the resulting MinCSP is NP-hard and W[1]-hard, i.e. as hard as Clique under fpt reductions. On a very high level, double-implication can be used to create gadgets coordinating two choices (e.g. choosing two endpoints of an edge), and thus allows a reduction from Clique. Essentially the same relation occurs in several subsequent hardness reductions, e.g. in [41, 24, 52].

FPT status of Multicut was an open problem resolved in the positive independently by Bousquet, Daligault & Thomassé [7] and Marx & Razgon [51]. The latter enriched the toolbox of parameterized complexity with *random sampling of important separators*.

7. Subset Directed Feedback Arc Set is a generalization of DFAS in which the input graph comes with a subset of distinguished arcs, and the goal is to delete few arcs in order to obtain a graph with no directed cycle containing a distinguished arc. Note that directed cycles without distinguished arcs are allowed. To cast it as MinCSP with domain $\mathbb{Q}$, translate every arc (u, v) into a constraint $u \leq v$ and every distinguished arc (u, v) into a constraint $u < v$. Observe that a directed cycle of $\leq$-constraints is satisfiable as an instance of CSP (by setting all variables along the cycle to the same value), while having one $<$-constraint on the cycle renders it unsatisfiable. Chitnis, Cygan, Hajiaghayi, and Marx [20] provided the first FPT algorithm for Subset DFAS. They generalized random sampling of important separators to directed graphs, and applied it to reduce the problem to the so-called *shadowless* variant. While simpler, solving this variant still requires significant problem-specific insights.

8. Unique Label Cover (ULC) is the problem at the heart of the Unique Games Conjecture (UGC) of Khot [39]. It can be modeled as MinCSP with a finite domain $\{1, \dots, d\}$ and binary constraints $R(x, y)$ such that fixing the value of either x or y leaves at most one way to extend the assignment to the second variable so that the constraint $R(x, y)$ is satisfied. One example of such constraint is $x + y = 0 \mod 3$ or any other two-variable equation modulo a prime. Another example could be a constraint $(x, y) \in \{(0, 1), (2, 0)\}$ over domain $\{0, 1, 2, 3\}$. ULC is NP-hard and does not admit constant-factor approximation in polynomial time under Khot's conjecture. Chitnis, Cygan, Hajiaghayi and Pilipczuk [13] showed that ULC is fixed-parameter tractable by developing the technique of *randomized contractions*, which has also been instrumental in showing that a challenging open problem called Minimum Bisection is fixed-parameter tractable [21]. The input of the latter problem is an undirected graph G, and the goal is to partition its vertices into two sets A and B of nearly equal size (i.e., $||A| - |B|| \leq 1$) so that at most k edges have one endpoint in A and another in B.

9. Chain SAT is a Boolean MinCSP with the relation $x_1 \to x_2 \to x_3 \to x_4$ and unary assignments $x = 0$ and $x = 1$. It is similar to the double-implication relation $x_1 \to x_2 \wedge x_3 \to y_4$ but additionally links x_2 and x_3. In terms of st-cuts in directed graphs, we are no longer allowed to pair up arbitrary arcs to be deleted at cost one, but triples of consecutive arcs. The importance of this problem was first identified by Chitnis, Egri and Marx [16] in their study of List H-Coloring by Vertex Deletion. A graph G is homomorphic to a graph H if there exists a mapping $h : V(G) \to V(H)$ that preserves the edge relation, i.e. if $uv \in E(G)$, then $h(u)h(v) \in E(H)$. The H-Coloring problem asks whether an input graph G is homomorphic to H. Note that this problem is equivalent to CSP(H), where H is viewed as a binary symmetric relation. List H-Coloring is a generalization in which the input lists come with $L_v \subseteq V(H)$ for all $v \in V(G)$, and the mapping $h : V(G) \to V(H)$ is required

to choose value $h(v)$ from list L_v. Feder, Hell and Huang [28] classified the graphs H for which the problem in P vs NP-complete. List H-Coloring by Vertex Deletion (LHomVD(H)) asks to delete k vertices from an input graph G to make it homomorphic to H. Chitnis, Marx, and Egri conjectured that LHomVD(H) is fixed-parameter tractable whenever List H-Coloring is in P. They solved several cases and gave a reduction from the problem to Chain SAT, conjecturing that the latter is fixed-parameter tractable. Kim, Kratsch, Pilipczuk and Wahlström [40] showed that Chain SAT is indeed in FPT.

Cutting Tools

We review the basics of graph cuts and present more advanced cutting tools used in parameterized complexity. All concepts below can be defined for both directed and undirected graphs. We restrict our attention to the directed counterparts since they are more general in the context of this thesis.

Minimum Cuts

Let G be a directed graph with vertex set $V(G)$ and arc set $A(G)$. To keep the notation concise, we will use uv to denote an arc from u to v, and similarly, vu to denote an arc from v to u. A *cut* in G is a subset X of the vertices $V(G)$. A cut X is an *st-cut* for a pair of vertices $s,t \in V(G)$ if $s \in X$ and $t \notin X$. More generally, X is an ST-cut for a pair of subsets $S,T \subseteq V(G)$ if $S \subseteq X$ and $T \cap X = \varnothing$. Let $\Delta(X)$ denote the set of arcs $uv \in A(G)$ such that $u \in X$ and $b \notin X$, and let $\delta(X) = |\Delta(X)|$. The arcs of G may have positive capacities associated with them via function $\mathrm{cap} : A(G) \to \mathbb{Q}^+$. The *capacity* of a cut X denoted by $\mathrm{cap}(X)$ is the sum of capacities of all arcs in $\Delta(X)$. Note that if all arcs have unit capacity, then $\mathrm{cap}(X)$ equals $\delta(X)$. A cut X is a *minimum cut* if it has minimum capacity.

Consider the following problem: given a directed graph G with a distinguished *source* vertex s and *sink* vertex t, and arc capacities $\mathrm{cap} : A(G) \to \mathbb{Q}^+$, we want to push as much flow from s to t as possible without exceeding the capacity of any arc. Formally, *flow* is a function $f : A(G) \to \mathbb{Q}_{\geq 0}$ fulfilling the following constraints:

- $f(uv) \leq \mathrm{cap}(uv)$ for all $uv \in A(G)$, i.e. flow cannot exceed the capacity of any arc;
- $\sum_{uv \in A(G)} f(uv) = \sum_{vw \in A(G)} f(vw)$ for all $v \in V(G) \setminus \{s,t\}$, i.e. the incoming flow equals the outgoing flow in every interval vertex of the graph, so no flow is lost along the way.

Define *value of f* to be $\sum_{su \in A(G)} f(su) = \sum_{vt \in A(G)} f(vt)$, i.e. the amount of flow leaving the source, which, by the conservation constraints, equals the amount of flow reaching the sink.

A *maximum flow* is a flow of maximum value. Let $\lambda_{st}(G)$ denote the maximum flow value in (G, s, t, cap). We omit the subscript st when it is clear from the context. The celebrated theorem of Ford and Fulkerson [32] asserts that $\lambda_{st}(G)$ equals the capacity of a minimum st-cut in G. Moreover, a flow of maximum value and a cut of minimum capacity can be computed in polynomial time.

Closest and Furthest Cut

Let G be a directed graph with a source s and a sink t, and let $\mathrm{cap} : A(G) \to \mathbb{Q}^+$ be the arc capacities. As before, we can generalize cap to st-cuts. One useful property of the generalized capacity function $\mathrm{cap} : 2^{V(G)} \to \mathbb{Q}^+$ is *submodularity*, i.e.

$$\mathrm{cap}(X) + \mathrm{cap}(Y) \geq \mathrm{cap}(X \cap Y) + \mathrm{cap}(X \cup Y)$$

holds for all st-cuts X, Y in G.[2]

Now suppose X and Y are *minimum* st-cuts. Then $X \cap Y$ and $X \cup Y$ are minimum cuts as well: by submodularity, $\mathrm{cap}(X \cap Y) + \mathrm{cap}(X \cup Y) \leq 2\,\lambda_{st}(G)$, while $\mathrm{cap}(X \cap Y) \geq \lambda_{st}(G)$ and $\mathrm{cap}(X \cup Y) \geq \lambda_{st}(G)$. Moreover, unless $X = Y$, we also have that $X \cap Y \subsetneq X$, $X \cap Y \subsetneq Y$, $X \subsetneq X \cup Y$ and $Y \subsetneq X \cup Y$. Thus, there exists a unique *closest* and a unique *furthest* minimum cut, i.e. a minimum cut that is *inclusion-wise minimum* and *inclusion-wise maximum*, respectively. The Ford-Fulkerson algorithm computes the unique *closest minimum st-cut.* By running the algorithm on the reversed graph (arc directions switched, t becomes source and s becomes sink), we can also compute the unique *furthest minimum st-cut* in polynomial time.

Example 2.5. As an application of closest cuts, let us solve a problem called Digraph Paired Cut. Cast as MinCSP, it is a problem with domain $\{0, 1\}$, soft constraints of the form $1 \to x$, $x \to y$, and crisp constraints of the form $\bar{p} \vee \bar{q}$.[3] In graph-theoretic terms, it can be stated as follows: given a directed graph G with a source vertex s, and a set $\mathcal{P}$ of vertex pairs $\{p, q\} \in \binom{V(G)}{2}$, we want to find a cut $X \subseteq V(G)$ of capacity at most k such that $s \in X$, and for every pair $\{p, q\} \in \mathcal{P}$, either $p \notin X$ or $q \notin X$. The connection to the MinCSP problem stated above goes as follows: source s represents value 1, arcs sx represent assignment constraints $1 \to x$, arcs xy represent implication constraints $x \to y$, and pairs $\{p, q\} \in \mathcal{P}$ stand for the constraints $\bar{p} \vee \bar{q}$. We require that the cut to separate s from either p or q, which, in CSP terms, allows us to set one of them to 0.

Kratsch and Wahlström [44] present an elegant $O^*(2^k)$ algorithm for Digraph Pair Cut. Initialize $T := \varnothing$ and proceed as follows.

[2]One way to see this is to expand the sums and observe that all arcs appearing on the right-hand side also appear on the left-hand side, while the arcs between $X \setminus Y$ and $Y \setminus X$ appear only on the left-hand side.

[3]This problem is almost Almost 2SAT, with the difference being that the constraints of the form $p \vee q$ are not available.

1. Compute the closest minimum sT-cut X in G.
2. If the capacity of X exceeds k, reject.
3. If $p \notin X$ or $q \notin X$ holds for all pairs $\{p,q\} \in \mathcal{P}$, then accept.
4. Otherwise, pick an arbitrary pair $\{p,q\} \in \mathcal{P}$ such that $p \in X$ and $q \in X$.
5. Branch in two directions:
    - Run the algorithm with $T \leftarrow T \cup \{p\}$.
    - Run the algorithm with $T \leftarrow T \cup \{q\}$.
6. Accept if either branch accepts, otherwise reject.

To argue correctness, let $X^\star$ be an optimal solution and $T^\star$ be the set of vertices in $\bigcup_{\{p,q\}\in\mathcal{P}}\{p,q\}$ separated from s by $X^\star$. It suffices to observe that the following invariant is maintained throughout the algorithm: there is at least one branch with $T \subseteq T^\star$.

For the running time, observe that the flow increases in each branch: indeed, if there exists a minimum sT-cut in G that separates p (respectively, q) from s, then so does the closest sT-cut. The recursion tree has depth at most k and branching factor 2, resulting in 2^k branches in total. ◂

Important Cuts

We start with a motivating example.

Example 2.6. Consider Edge 3-Multiway Cut problem. An instance is an undirected graph G with three *terminal* vertices t_1, t_2, t_3, and an integer k. The goal is to delete at most k edges from G so that no two terminals remain connected. Equivalently, the problem can be stated as partitioning $V(G)$ into three sets T_1, T_2, T_3 so that $t_i \in T_i$ for all $i \in \{1,2,3\}$ so that

$$\delta(T_1) + \delta(T_2) + \delta(T_3) \le 2k$$

since every deleted edge is counted twice.

To design an fpt algorithm for this problem, we proceed in two stages: first, we find a cut T_1 that separates t_1 from $\{t_2,t_3\}$, and then compute a minimum t_2t_3-cut in the subgraph induced by $V(G) \setminus T_1$. The second step requires polynomial time, so the difficulty lies in the first step of the algorithm. Indeed, the number of cuts separating t_1 and $\{t_2,t_3\}$ is unbounded in k, so we need to understand the search space better. Guess $k_1 = \delta(T_1)$, i.e. the part of the budget k that we spend in the first stage of the algorithm, and observe that the search for T_1 can be restricted to *inclusion-wise maximal* cuts of capacity k_1. Intuitively, pushing T_1 closer to t_2 and t_3 cannot increase the t_2t_3-flow in the remaining graph – it may only intersect more t_2t_3-paths.

A *replacement argument* helps us formalize this intuition: if a hypothetical solution (T_1, T_2, T_3) with k crossing edges is such that T_1 is not inclusion-wise minimal, then, by definition, there exists another cut $X \subseteq V(G)$ that separates t_1 and $\{t_2, t_3\}$ and satisfied $T_1 \subseteq X$ and $\delta(X) = \delta(T_1) = k_1$. Furthermore, replacing T_1 with X yields a solution of cost at most k. We will see how to leverage this observation in the sequel. ◂

The example above shows that, in the course of designing an fpt algorithm on a directed graph G, we may want to enumerate inclusion-wise maximal cuts X such that $\delta(X)$ is bounded by the parameter. This notion is closely related to furthest cuts considered in the previous section. However, unless we assume that the cut has *minimum* capacity, there is no unique candidate. On the other hand, Marx [49] remarkably showed that the number of inclusion-wise maximal cuts of capacity at most k is bounded by a function of k alone. To formalize, we need the following definition.

Definition 2.7. Let G be a directed graph and $s, t \in V(G)$. An st-cut X is *important* if there exists no st-cut X' such that $X \subseteq X'$ and $\delta(X') \leq \delta(X)$.

A consequence of the definition is that every superset of an important cut X has higher capacity.

Theorem 2.8. *Let G be a directed graph and $s, t \in V(G)$. There are at most 4^k important st-cuts in G.*

Marx [49] introduced *important cuts* and gave an upper bound of 4^{k^2} on their number. Later, the bound was improved to the essentially tight[4] 4^k by Chen, Lu and Liu [11].

An algorithm listing all important cuts (and proving Theorem 2.8 along the way) works as follows. Initialize $S = \{s\}$ and $T = \{t\}$.

1. Let S be the furthest ST-cut in G.
2. If $\delta(S) > k$, stop.
3. Output S as an important cut.
4. Pick an arbitrary arc uv in $\Delta(S)$ with $u \in S$ and $v \notin S$, and branch in two directions:
    - Delete uv from G, decrease k by 1, add v to T, and run the algorithm on the new graph.
    - Add v to S and run the algorithm on the same graph G, keeping k unchanged.

[4]An example of a graph with $\Theta(4^k/k^{3/2})$ important st-cuts is a complete binary tree with k leaves, source s being the root, and a sink t connected to every leaf, with all arcs oriented from s to t. See Figure 8.5 in [20, Section 8.2] for an illustration and the caption for further discussion.

Observe that in the first branch the parameter decreases by one, while the maximum ST-flow in the new graph decreases by at most one. In the second branch, the ST-flow increases by one since we are computing furthest cuts. Thus, the measure $2k - \lambda_{ST}$ decreases in each branch. Our search tree has branching factor 2 and depth at most $2k$, yielding at most $2^{2k} = 4^k$ branches.

Branch, Cut, and Iterate

Many parameterized deletion algorithms follow the paradigm "branch-cut-iterate". The usual opening step in such algorithms is *iterative compression.* Introduced originally by Reed, Smith and Vetta [54] in their fpt algorithm for Odd Cycle Transversal (OCT) (given an undirected graph, delete k vertices to make it bipartite), it has since become an indispensable tool. We illustrate this simple and powerful idea in an algorithm for the edge version of OCT called Graph Bipartization: given an undirected graph G, delete k edges to make it bipartite.[5] This problem can also be modeled as a Boolean MinCSP with a single relation $\neq$. On the conceptual level, the algorithm is mimicking the steps of an inductive proof, building up the graph one edge at a time. Let us enumerate the edges of G as $e_1, \dots, e_m$, and for every $1 \leq i \leq m$, let G_i be the graph with vertices $V(G)$ and edges $\{e_1, \dots, e_i\}$. In particular, $G_m = G$. Note that G_1 is a yes-instance, so $X = \varnothing$ is a solution for G_1. By analogy with an inductive proof, this is our base case. The algorithm will work its way through $G_1, \dots, G_m$, while maintaining a solution X of size at most k. If at any step it detects that no solution of size at most k exists, then the instance is rejected immediately: indeed, if (G_i, k) is a no-instance for any i, then (G, k) is a no-instance as well[6]. For the inductive step, suppose X is a solution to G_i. By the inductive hypothesis, we have $|X| \leq k$. Adding edge e_{i+1} to G_i, we obtain $G' = G_{i+1}$, and observe that deleting $X' = X \cup \{e_{i+1}\}$ from G' make it bipartite since $G_i - X$ and $G' - X'$ are the same graph. The caveat is that in the worst case, we have $|X'| = k + 1$, so our problem reduces to designing a compression routine that takes G' and X' as input with the promise that $G' - X'$ is bipartite and $|X'| = k + 1$, and returns either a solution Z of size at most k, or correctly reports that no such solution exists. Importantly, if this routine runs in fpt time, then we have an fpt algorithm for the original problem. It might seem that conceptually we are 'morally' no closer to a solution. However, having X' provides us with a handle on the problem that we can leverage by fpt computation.

The next step is *branching.* We guess the intersection of X' with the hypothetical optimal solution Z, and remove $X' \cap Z$ from the graph. This step creates at most 2^k branches. The MinCSP formulation of the problem

[5]Note that a minimum set of edges whose deletion leaves a bipartite graph is complementary to a maximum cut in the graph, so Graph Bipartization is NP-hard.

[6]Here we are using the fact that bipartiteness, and, more generally, the satisfiability of a set of constraints is a monotonic property.

becomes helpful at this stage. Since $G' - X'$ is bipartite, there is an assignment $\alpha : V(G) \to \{0,1\}$ that satisfies every edge in $G' - X'$, i.e. assigns distinct values to its endpoints. Moreover, we can find this assignment in polynomial time by a simple propagation algorithm. For the edges in $X' \setminus Z$, we have no guarantee that α satisfies them, and this has to be fixed. Assuming Z is a solution, there also exists an assignment $\beta : V(G) \to \{0,1\}$ that satisfies every edge in $G' - Z$. We have no access to β, but we can guess the values it assigns to the endpoints of the edges in $X' \setminus Z$. Denote this set of vertices by T and observe that $|T| \le 2(k+1)$, hence this guessing step creates at most 2^{k+1} branches.

The final step in the algorithm is *cutting*. Partition T into subsets $A = \{x \in T : \alpha(x) = \beta(x)\}$ and $D = \{x \in T : \alpha(x) \neq \beta(x)\}$, i.e. vertices for which α and β agree and disagree, respectively. The key observation is that a vertex from A cannot be connected to a vertex from D in $G' - Z'$: indeed, if α and β disagree on the value of *one* vertex, then they disagree on the value of *every* vertex in the same connected component. Hence, Z' is an AD-cut in G'. One can show that this condition is not only necessary, but also sufficient, so the problem reduces to computing a minimum AD-cut, which can be done in polynomial time.

The steps of this algorithm are summarized in Algorithm 1.

Algorithm 1 An algorithm for Graph Bipartization

```
procedure GB-Solve($G$, $k$)
    $G' \leftarrow \{V(G), \varnothing\}$                          ▷ Initialize empty graph
    $X = \varnothing$                                               ▷ Maintain solution $X$
    for $e \in E(G)$ do
        $G' \leftarrow G' + e$
        $X = X + e$
        $Z \leftarrow$ GB-Compress($G', X, k$)
        if compression failed then
            reject
        else
            $X \leftarrow Z$
    accept

procedure GB-Compression($G$, $X$, $k$)                ▷ Assume $Z$ is a smaller
solution to $(G, k)$
    for $Y \subseteq X$ do                                ▷ Guess intersection $Y = X \cap Z$
        $G' \leftarrow G - Y$
        $X' \leftarrow X - Y$
        for $A \subseteq V(X')$ do                  ▷ Guess vertices where $\alpha$ and $\beta$ agree
            $D \leftarrow V(X') \setminus A$
            $Z \leftarrow$ MinCut($G, A, B$)       ▷ Compute minimum $(A, B)$-cut in $G$
            if $|Z| \leq k$ then
                return $Z$
    return FAIL
```

3 Overview of Contributions

This chapter contains a short overview of the results comprising this thesis. These are four publications on parameterized complexity of infinite domain MinCSPs. We remark that full report versions of all publications are included in the thesis rather than short conference versions. For discussions of future work, we refer the reader to concluding sections of the included papers.

1) Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, and George Osipov. “Resolving Inconsistencies in Simple Temporal Problems: A Parameterized Approach.” In: *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022).* 2022, pp. 3724–3732

The first paper is about the Simple Temporal Problem (STP), which is an expressive AI formalism for temporal reasoning. Here the constraints are of the form

$$\ell \odot_1 x - y \odot_2 u$$

for $\odot_1, \odot_1 \in \{<, \leq\}$ and $\ell, u \in \mathbb{Q} \cup \{-\infty, +\infty\}$. Values $-\infty, +\infty$ are used for imposing one-sided constraints, e.g. $1 < x - y \leq \infty$ essentially enforces lower bound $x - y > 1$ and no upper bound. Note that $\ell = u$ is allowed, so difference equation constraints like $x - y = 3$ can also be enforced via $3 \leq x - y \leq 3$. This problem generalizes DFAS, which is MinCSP with constraints of the form $x - y > 0$, and Subset DFAS which is MinCSP with constraints of the form $x - y > 0$, and $x - y \geq 0$. We classify parameterized complexity of MinCSP for every subset of STP constraints. In the hindsight, we note that our algorithm for equation constraints can be significantly improved using other techniques.

2) Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström. “Almost Consistent Systems of Linear Equations.” In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023).* 2023, pp. 3179–3217

In MinLin we are given an inconsistent system of linear equations, and the goal is to remove few equations so that the system becomes consistent. The parameter is the number of equations to be removed. This problem can be studied over different rings, e.g. finite fields, the rationals or the integers. Min-2-Lin($\mathbb{F}_2$), i.e. the problem restricted to equations over $\mathbb{F}_2$ with two variables per equation, easily reduces to Bipartization, while Min-3-Lin($\mathbb{F}_2$) encompasses Odd Set, which is W[1]-hard. In the same spirit, we extend the dichotomy between Min-2-Lin and Min-3-Lin to fields and Euclidean domains[1], which is an abstract algebraic structure that includes $\mathbb{Q}$, $\mathbb{Z}$, $\mathbb{Z}[i]$, and many other rings. In particular, we devise fpt algorithms for Min-2-Lin($\mathbb{D}$) if $\mathbb{D}$ is a field or a Euclidean domain, and show that Min-3-Lin($\mathbb{D}$) is W[1]-hard for every nontrivial ring $\mathbb{D}$. For the algorithmic results, we introduce important balanced subgraphs, a generalization of important separators formulated in terms of biased graphs. The technique relies on algorithmically beneficial properties of the linear programming formulation of the problem.

3) George Osipov and Magnus Wahlström. "Parameterized Complexity of Equality MinCSP." in: *Proceedings of the 31st Annual European Symposium on Algorithms (ESA 2023).* 2023, 86:1–86:17

We consider Equality MinCSP, in which the domain is $\mathbb{N}$ and the constraints are defined by arbitrary first-order formulas using the predicate $=$. Already with the two simplest relations $=$ and $\neq$ available, this problem models Multicut, and thus is NP-hard and in FPT. We give FPT vs W[1]-hardness dichotomies for solving such problems exactly and computing constant-factor approximations. While doing so, we identify two interesting generalizations of Multicut, namely Multicut with Triples and Disjunctive Multicut. The former extends Multicut with triple cut requests that can be ignored at unit cost, and is in FPT. The latter allows disjunctive cut requests, and is FPT-approximable within a constant factor. Further, we show that these generalizations are 'maximal', in the sense that if a non-trivial Equality MinCSP does not reduce to the first problem, it is W[1]-hard, and if it does not reduce to the second, it is W[1]-hard to approximate within any constant.

4) Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, Marcin Pilipczuk, and Roohani Sharma. "Parameterized Complexity Classification for Interval Constraints." In: *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC 2023).* 2023, 11:1–11:19

Allen's Interval Algebra (IA) is a prominent formalism for qualitative temporal reasoning. Two proper intervals on the real line may relate in thirteen

[1]The algorithms work under reasonable assumptions on the effectiveness of arithmetic in these domains.

different ways (be equal to each other, the first may precede the second or start when the second ends, the first may be contained in the second, etc.). An instance of the computational problem for IA is a set of constraints using these thirteen basic relations applied to pairs of variables, and the goal is to assign proper intervals to the variables so that all relations hold. We study the parameterized complexity of MinCSP for all subsets of basic interval relations, and show for which subsets the problem is in FPT or W[1]-hard. For the positive cases, we introduce and solve MinCSP$(<, \ll, =)$, an extension of DFAS with edges that enforce equality constraints and 'long' arcs that enforce constraints of the form $x \ll y$. The latter is satisfied by assigning y a value that is 'much greater' than x. All tractable cases of MinCSP for basic interval relations reduce to this problem. The intuitive reason why MinCSP$(<, \ll, =)$ is in FPT is that the solution is a linear arrangement of points on the real line. However, an interval constraint in general enforces a pair of pointwise constraints, one for the left endpoints and one for the right endpoints of the intervals. If these constraints are not correlated in a 'linear' fashion, meaning that we cannot reduce to MinCSP$(<, \ll, =)$, we show that the problem is W[1]-hard by carefully crafted gadget reductions.

Bibliography

[1] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. “Correlation clustering.” In: *Machine Learning* 56 (2004), pp. 89–113.

[2] Manuel Bodirsky. *Complexity of Infinite-Domain Constraint Satisfaction*. 2021.

[3] Manuel Bodirsky and Martin Grohe. “Non-dichotomies in constraint satisfaction complexity.” In: *Proceedings of the 35th International Colloquium Automata, Languages and Programming (ICALP 2008)*. 2008, pp. 184–196.

[4] Manuel Bodirsky and Jan Kára. “The complexity of equality constraint languages.” In: *Theory of Computing Systems* 43 (2008), pp. 136–158.

[5] Manuel Bodirsky and Jan Kára. “The complexity of temporal constraint satisfaction problems.” In: *Journal of the ACM* 57 (2010), pp. 1–41.

[6] Édouard Bonnet, László Egri, Bingkai Lin, and Dániel Marx. *Fixed-parameter approximability of Boolean MinCSPs*. 2016. arXiv: `1601.04935 [cs.CC]`.

[7] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. “Multicut is FPT.” In: *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing (STOC 2011)*. 2011, pp. 459–468.

[8] Andrei Bulatov, Peter Jeavons, and Andrei Krokhin. “Classifying the complexity of constraints using finite algebras.” In: *SIAM Journal on Computing* 34 (2005), pp. 720–742.

[9] Andrei A. Bulatov. “A dichotomy theorem for nonuniform CSPs.” In: *Proceedings of the 58th Annual Symposium on Foundations of Computer Science (FOCS 2017)*. 2017, pp. 319–330.

[10] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D. Sivakumar. “On the hardness of approximating Multicut and Sparsest-Cut.” In: *Computational Complexity* 15 (2006), pp. 94–114.

[11] Jianer Chen, Yang Liu, and Songjian Lu. “An improved parameterized algorithm for the Minimum Node Multiway Cut problem.” In: *Algorithmica* 55 (2009), pp. 1–13.

[12] Jianer Chen, Yang Liu, Songjian Lu, Barry O’sullivan, and Igor Razgon. “A fixed-parameter algorithm for the directed feedback vertex set problem.” In: *Journal of the ACM* 55.5 (2008).

[13] Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michał Pilipczuk. “Designing FPT algorithms for cut problems using randomized contractions.” In: *SIAM Journal on Computing* 45 (2016), pp. 1171–1229.

[14] Rajesh Chitnis, Marek Cygan, Mohammataghi Hajiaghayi, and Dániel Marx. “Directed subset feedback vertex set is fixed-parameter tractable.” In: *ACM Transactions on Algorithms* 11 (2015), pp. 1–28.

[15] Rajesh Chitnis, László Egri, and Dániel Marx. “List H-coloring a graph by removing few vertices.” In: *Proceedings of the 21st Annual European Symposium on Algorithms (ESA 2013).* 2013, pp. 313–324.

[16] Rajesh Chitnis, László Egri, and Dániel Marx. “List H-coloring a graph by removing few vertices.” In: *Algorithmica* 78 (2017), pp. 110–146.

[17] Alan Cobham. “The Intrinsic Computational Difficulty of Functions.” In: *Proceedings of the 1964 International Congress on Logic, Methodology and Philosophy of Science.* 1965, pp. 24–30.

[18] Stephen A. Cook. “The Complexity of Theorem-Proving Procedures.” In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971).* 1971, pp. 151–158.

[19] Stephen A. Cook. “The complexity of theorem-proving procedures.” In: *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook.* 2023, pp. 143–152.

[20] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* 2015.

[21] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. “Minimum bisection is fixed parameter tractable.” In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC 2014).* 2014, pp. 323–332.

[22] Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, and George Osipov. “Resolving Inconsistencies in Simple Temporal Problems: A Parameterized Approach.” In: *Proceedings of the 36th AAAI Conference on Artificial Intelligence (AAAI 2022).* 2022, pp. 3724–3732.

[23] Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, Marcin Pilipczuk, and Roohani Sharma. “Parameterized Complexity Classification for Interval Constraints.” In: *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC 2023).* 2023, 11:1–11:19.

[24] Konrad K. Dabrowski, Peter Jonsson, Sebastian Ordyniak, George Osipov, and Magnus Wahlström. “Almost Consistent Systems of Linear Equations.” In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023).* 2023, pp. 3179–3217.

[25] Elias Dahlhaus, David S Johnson, Christos H Papadimitriou, Paul D Seymour, and Mihalis Yannakakis. “The complexity of multiway cuts.” In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC 1992).* 1992, pp. 241–251.

[26] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* 2012.

[27] Jack Edmonds. “Paths, trees, and flowers.” In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467.

[28] Tomás Feder, Pavol Hell, and Jing Huang. “List homomorphisms and circular arc graphs.” In: *Combinatorica* 19 (1999), pp. 487–505.

[29] Tomás Feder and Moshe Y Vardi. “The computational structure of monotone monadic SNP and constraint satisfaction: A study through Datalog and group theory.” In: *SIAM Journal on Computing* 28 (1998), pp. 57–104.

[30] Andreas Emil Feldmann, Euiwoong Lee, and Pasin Manurangsi. “A survey on approximation in parameterized complexity: Hardness and algorithms.” In: *Algorithms* 13 (2020), p. 146.

[31] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory.* 2006.

[32] Lester R. Ford and Delbert R. Fulkerson. “Maximal flow through a network.” In: *Canadian Journal of Mathematics* 8 (1956), pp. 399–404.

[33] Michael R. Garey and David S. Johnson. *Computers and intractability.* 1979.

[34] Meike Hatzel, Lars Jaffke, Paloma T. Lima, Tomáš Masařík, Marcin Pilipczuk, Roohani Sharma, and Manuel Sorge. “Fixed-parameter tractability of Directed Multicut with three terminal pairs parameterized by the size of the cutset: twin-width meets flow-augmentation.” In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023).* 2023, pp. 3229–3244.

[35] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. “Which problems have strongly exponential complexity?” In: *Journal of Computer and System Sciences* 63 (2001), pp. 512–530.

[36] Clay Math Insitite. *Millenium Problems.* `https://www.claymath.org/millennium-problems`.

[37] Peter Jonsson, Victor Lagerkvist, and Gustav Nordh. "Blowing holes in various aspects of computational problems, with applications to constraint satisfaction." In: *Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming (CP 2013).* 2013, pp. 398–414.

[38] Richard M. Karp. *Reducibility among Combinatorial Problems.* 2010.

[39] Subhash Khot. "On the power of unique 2-prover 1-round games." In: *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002).* 2002, pp. 767–775.

[40] Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. "Directed flow-augmentation." In: *Proceedings of the 54th Annual ACM Symposium on Theory of Computing (STOC 2022).* 2022, pp. 938–947.

[41] Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. "Flow-augmentation III: Complexity dichotomy for Boolean CSPs parameterized by the number of unsatisfied constraints." In: *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023).* 2023, pp. 3218–3228.

[42] Eun Jung Kim, Stefan Kratsch, Marcin Pilipczuk, and Magnus Wahlström. "Solving hard cut problems via flow-augmentation." In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA 2021).* 2021, pp. 149–168.

[43] Vladimir Kolmogorov, Andrei Krokhin, and Michal Rolínek. "The complexity of general-valued CSPs." In: *SIAM Journal on Computing* 46 (2017), pp. 1087–1110.

[44] Stefan Kratsch and Magnus Wahlström. "Representative sets and irrelevant vertices: New tools for kernelization." In: *Journal of the ACM* 67 (2020), pp. 1–50.

[45] Richard E. Ladner. "On the structure of polynomial time reducibility." In: *Journal of the ACM* 22 (1975), pp. 155–171.

[46] Leonid Levin. "Universal sequential search problems." In: *Problemy peredachi informatsii* 9 (1973), pp. 115–116.

[47] Daniel Lokshtanov, Pranabendu Misra, MS Ramanujan, Saket Saurabh, and Meirav Zehavi. "FPT-approximation for FPT Problems." In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA 2021).* 2021, pp. 199–218.

[48] Dániel Marx. "Parameterized complexity and approximation algorithms." In: *The Computer Journal* 51 (2008), pp. 60–78.

[49] Dániel Marx. "Parameterized graph separation problems." In: *Theoretical Computer Science* 351 (2006), pp. 394–406.

[50] Dániel Marx and Igor Razgon. "Constant ratio fixed-parameter approximation of the edge multicut problem." In: *Information Processing Letters* 109 (2009), pp. 1161–1166.

[51] Dániel Marx and Igor Razgon. "Fixed-parameter tractability of multicut parameterized by the size of the cutset." In: *SIAM Journal on Computing* 43 (2014), pp. 355–388.

[52] George Osipov and Magnus Wahlström. "Parameterized Complexity of Equality MinCSP." In: *Proceedings of the 31st Annual European Symposium on Algorithms (ESA 2023)*. 2023, 86:1–86:17.

[53] Igor Razgon and Barry O'Sullivan. "Almost 2-SAT is fixed-parameter tractable." In: *Journal of Computer and System Sciences* 75 (2009), pp. 435–450.

[54] Bruce Reed, Kaleigh Smith, and Adrian Vetta. "Finding odd cycle transversals." In: *Operations Research Letters* 32 (2004), pp. 299–301.

[55] Michael Sipser. *Introduction to the Theory of Computation*. 1997.

[56] Johan Thapper and Stanislav Živný. "The complexity of finite-valued CSPs." In: *Journal of the ACM* 63 (2016), pp. 1–33.

[57] Dmitriy Zhuk. "A proof of the CSP dichotomy conjecture." In: *Journal of the ACM* 67 (2020), pp. 1–78.

Papers

FACULTY OF SCIENCE AND ENGINEERING

Linköping Studies in Science and Technology, Dissertation No. 2368, 2024
Department of Computer and Information Science

Linköping University
SE-581 83 Linköping, Sweden

www.liu.se

LINKÖPING
UNIVERSITY