RESOLUTION COMPLEXITY OF RANDOM CONSTRAINT SATISFACTION PROBLEMS

by

Chun Kong Yung

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

© Copyright 2016 by Chun Kong Yung
Abstract

Resolution Complexity of Random Constraint Satisfaction Problems

Chun Kong Yung
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2016

The resolution complexity of random constraint satisfaction problems is a widely studied topic. This line of research started with a seminal paper by Chvátal and Szemerédi. They showed that for any $k \geq 3$, w.h.p. an unsatisfiable random $k$-SAT instance has exponentially high resolution complexity when the clause density is a constant. The result was later extended to settings with super-constant clause density.

The random $(d, k, t)$-CSP model is another well-studied random CSP model. It is a natural generalization of the random $k$-SAT model by allowing a more general domain of $d \geq 2$ variable values instead of \{TRUE, FALSE\}, and allowing $t \geq 1$ restrictions in each constraint (clause) instead of only one. Earlier results give the whole picture for the resolution complexity of random $(d, k, t)$-CSP instances for every constant triple of $(d, k, t)$ and every constant constraint density $\Delta$. However, very little is known for the resolution complexity when the constraint density grows beyond constant. In this thesis, we generalize the resolution complexity results for random $(d, k, t)$-CSP to settings with super-constant constraint density, just like the later studies extended the $k$-SAT result of Chvátal and Szemerédi to settings with super-constant clause density.

We introduce a general approach for studying the resolution and tree resolution complexity of random $(d, k, t)$-CSP with super-constant constraint density. We are particularly interested in the ranges of constraint density where the resolution and tree resolution complexity drop from superpolynomial to polynomial. By applying our approach, we obtain new lower and upper bounds on these constraint density ranges. In particular, the bounds for the tree resolution complexity are tight up to a $o(1)$ term in the exponent. The settings we consider include two generalizations of the random $k$-SAT model, and the bounds we obtain almost match the best known bounds for the random $k$-SAT model. Therefore, our results can be regarded as generalizations of the resolution complexity results for random $k$-SAT.

Finally, it seems possible to apply our approach to other models of random CSPs as well.
Acknowledgements

I would like to extend my sincerest gratitude towards people who have helped me through the years of work that has led to this thesis.

My supervisor, Michael Molloy, provided me invaluable supports and advices during my time at University of Toronto. He did not only guide me all the time during the research and the writing of this thesis, but also allowed me to work in my own pace. From bringing the topic of random constraint satisfaction problems to my attention, to going through my work repeatedly, this thesis could not have been finished without his help. I am very lucky to have him as my supervisor.

Heartfelt thanks to my other committee members: Toniann Pitassi, Charles Rackoff, Alasdair Urquhart and Allan Borodin, and my examiners: Cristopher Moore and Aleksandar Nikolov. They provided me insightful ideas and advices at various points along the path. I am extremely grateful for the time they spent on my work. They made this work better than it would have been without their help.

I also want to thank my friends, my colleagues, and the scholars and staffs in my department. My time at University of Toronto is pleasurable and beneficial because of each of them.

Finally, I am immensely grateful to my family for their love, support and everything they did for me. I dedicate this thesis to my mother Fung Ying Yu.
# Contents

1 Introduction 1

2 Background 6
   2.1 Resolution Proof System 6
   2.2 Some Resolution Complexity Results 8
   2.3 Random Constraint Satisfaction Problems 12
   2.4 Some Results in Resolution Complexity of Random CSPs 16
   2.5 Techniques in Resolution Complexity of Random CSPs 19
   2.6 Formal Definitions and Notations 21

3 Our Results 25

4 A General Approach 28
   4.1 A Reduction to a Non-Random Problem 28
   4.2 The Special \((d, k, t)\)-CSP instance \(H^*\) 30
   4.3 Proofs of the Upper and Lower Bounds 31
   4.4 The Token Allocation Problem 35
   4.5 Algorithm **TA-SA** and Proof of Lemma 4.22 39
   4.6 Chapter Summary 41

5 The Special Forcer \(H^*\) 42
   5.1 An Outline on the Construction of \(H^*\) 42
   5.2 Forbidders and Forcers 43
5.3 Some Properties of $H^*$ and $G^*$ ................................................................. 48
5.4 The Value of $\Lambda(d, k, t)$ ............................................................................. 51
5.5 Chapter Summary ............................................................................................. 53

6 The Token Function $g(c)$ ................................................................................. 55

6.1 An Outline on the Design of $g(c)$ ................................................................. 55
6.2 A Valid $(\alpha, \beta)$-Token Allocation on $(V(H^*) - I(H^*), E(H^*))$ ............ 56
6.3 The Design of the Token Function $g(c)$ ....................................................... 58
6.4 Proof of Claim 4.24 ......................................................................................... 62
6.5 Proof of Claim 4.25 ......................................................................................... 68
6.6 Chapter Summary ............................................................................................. 71

7 Concluding Remarks ......................................................................................... 73

Bibliography ........................................................................................................... 76

A Proofs using the First and the Second Moment Methods ................................ 81

A.1 Proof of Lemma 4.13 by the First Moment Method ....................................... 81
A.2 Proof of Lemma 4.17 by the First Moment Method ....................................... 82
A.3 Proof of Lemma 4.12 by the Second Moment Method ................................... 83

B Sparsity of Forcing Blocks ................................................................................ 87

B.1 Proof of Claim 5.4 ........................................................................................... 87
B.2 Proof of Claim 5.5 ........................................................................................... 88
B.3 Proof of Claim B.3 .......................................................................................... 89

C The Setting with $t = 1$ ..................................................................................... 93

C.1 The Special Forcer Subgraph $H^*$ ............................................................... 93
C.2 Values of $\Lambda(d, k, t), \alpha$ and $\beta$ ............................................................. 94
C.3 Valid $(\alpha, \beta)$-Token Allocations ............................................................. 94
C.4 Token Function $g(c)$ .................................................................................... 94
D The Setting with \( d = 2 \)

D.1 The Special Forcer Subgraph \( H^* \) .................................................. 95

D.2 Values of \( \Lambda(d, k, t), \alpha \) and \( \beta \) .................................................. 96

D.3 Valid \( (\alpha, \beta) \)-Token Allocations .................................................. 96

D.4 Token Function \( g(c) \) .................................................. 96

E Results for the Random Model CSP\( ^{d,k,t}_{n,p} \) 97
Chapter 1

Introduction

Constraint Satisfaction Problems

Constraint Satisfaction Problems (CSPs) are extensively studied in many areas of computer science. A CSP instance consists of a set of variables, each of which can take a value on a finite domain of values; and a set of constraints, each of which forbids certain tuples of values from being assigned to a subset of variables. One objective is to determine the existence of a satisfying assignment - a value assignment on the variable set is called a satisfying assignment if it is not forbidden by any of the constraints.

Constraint satisfaction problems capture a wide range of famous combinatorial problems, such as SAT, the problem of determining the satisfiability of a given boolean formula; k-Colourability, the problem of determining the existence of a k-colouring on a given graph; and planning, scheduling and resource allocation problems. Although there are sub-cases of CSPs that are polynomial-time solvable, solving a constraint satisfaction problem is an NP-complete problem in general, which can be seen easily from the fact that CSPs capture the well-known NP-complete problem SAT. It is widely believed that P ≠ NP, so it seems the existence of a polynomial time CSP algorithm is quite unlikely. However, it is still of interest to find efficient practical algorithms because of the various applications of CSPs. A natural step in this research direction is to identify hard instances for the well-known algorithms.

On the theoretical side, if there exist hard instances with no poly-time proof of unsatisfiability, then P ≠ NP. A natural approach for achieving this distant goal is to study stronger and stronger proof systems (algorithms) and show the existence of instances that are hard to all of them. A variety of proof systems were studied in the past decades. However, we may not fully understand even the relatively simple ones. It is still of theoretical interest to study hard instances for those weaker and simpler proof systems.
Resolution Proof System and Resolution Complexity

Resolution is one of the most popular propositional proof systems. Many practical complete solvers for SAT and CSP use resolution-based algorithms. It is a simple proof system, with a single inference rule, the Resolution Rule:

\[
\frac{(A \lor x), (B \lor \overline{x})}{(A \lor B)},
\]

for some variable \(x\) and clauses \((A \lor x)\) and \((B \lor \overline{x})\). Sometimes, we use the non-essential Weakening Rule as well:

\[
\frac{(A)}{(A \lor B)}.
\]

By the resolution rule, if we can derive the empty clause \(\emptyset\) (i.e. FALSE), then the given boolean formula must be unsatisfiable. A resolution proof or resolution refutation of a given boolean formula is a derivation of the empty clause. The resolution proof system is both sound and complete, i.e. a boolean formula has a resolution refutation if and only if it is unsatisfiable.

We define the resolution complexity of a boolean formula \(F\) to be the number of clauses in the shortest resolution refutation of \(F\).\(^1\) Hence, the runtime complexity of any resolution-based algorithm on an unsatisfiable boolean formula is lower bounded by the resolution complexity of the given formula.

Hard Instance for Resolution

A lot of work has been done to study hard instances for the resolution proof system. The first super-polynomial lower bound on resolution complexity was presented by Tseitin in 1968 [74]. He proved that the resolution complexity of some Tseitin formula is superpolynomial, when regular resolution (which is a restricted version of resolution) is used. The first lower bound result for general resolution was proved by Haken in 1985 [55]. He showed that any resolution refutation of Pigeonhole Principle has superpolynomial length. Urquhart proved the first asymptotically tight lower bound on resolution complexity in 1987 [75]. He considered some Tseitin formulas on expander graphs and showed that every corresponding resolution refutation has length exponential in the number of variables. The study of resolution complexity was continued in different formula families [32, 31, 24, 69, 70, 71, 14, 4, 1, 16, 13, 33].

Properties of these formula families in some sense explain why they are hard for the resolution proof system. This may give some clues on designing better algorithms, or constructing hard instances for other proof systems. One formula family we are particularly interested in is the Random Constraint Satisfaction Problems.

\(^1\) For convenience, we define the proof complexity of satisfiable instances to be \(\infty\).
Resolution Complexity of Random Constraint Satisfaction Problems

The study of resolution complexity of Random Constraint Satisfaction Problems (random CSPs) started with a seminal paper by Chvátal and Szemerédi [37]. They showed that for any $k \geq 3$, when the clause density is a constant, w.h.p. a random $k$-SAT instance has exponentially high resolution complexity.\(^2\) The result was later extended to settings with super-constant clause density [54, 15, 24, 18, 14, 20].

The best known results on the resolution complexity of random $k$-SAT were proved by Beame et al. [14] and Ben-Sasson and Wigderson [24, 18]. They showed that for any $k \geq 3$ and any constant $\epsilon > 0$, the resolution complexity of a random $k$-SAT instance is w.h.p. superpolynomial when the clause density is $n^{k-2-\epsilon}$; and is w.h.p. at most polynomial when the clause density is $\Omega(n^{k-2}/\log^{k-2} n)$. In other words, the resolution complexity drops from superpolynomial to polynomial when the clause density grows from $n^{k-2-\epsilon}$ to $\Omega(n^{k-2}/\log^{k-2} n)$. There is still a gap of $\frac{k-2}{2} + o(1)$ in the exponent.

On the other hand, previous works proved a threshold-like phenomenon for the tree resolution complexity - tree resolution is a famous restricted version of resolution and corresponds to the widely used DPLL procedure. Beame et al. [14] and Ben-Sasson and Wigderson [24, 18] showed that for any $k \geq 3$ and any constant $\epsilon > 0$, the tree resolution complexity of a random $k$-SAT instance is w.h.p. superpolynomial when the clause density is $n^{k-2-\epsilon}$; and is w.h.p. at most polynomial when the clause density is $\Omega(n^{k-2}/\log^{k-2} n)$. In other words, the tree resolution complexity drops from superpolynomial to polynomial in a narrow range between $n^{k-2-\epsilon}$ and $\Omega(n^{k-2}/\log^{k-2} n)$. These bounds are tight up to a $o(1)$ term in the exponent.

The random $(d, k, t)$-CSP model is another well-studied random CSP model. It is a natural generalization of the random $k$-SAT model by allowing a more general domain of $d \geq 2$ variable values instead of $\{\text{TRUE, FALSE}\}$, and allowing $t \geq 1$ restrictions in each constraint (clause) instead of only one. Previous results [62, 2, 63, 64] give the whole picture for the resolution complexity of random $(d, k, t)$-CSP instance for every constant triple of $(d, k, t)$ and every constant constraint density $\Delta$. However, very little is known for the resolution complexity when the constraint density grows beyond constant. In this thesis, we attempt to extend resolution complexity results for random $(d, k, t)$-CSP in this direction.

Our Results

In this thesis, we study the resolution complexity and the tree resolution complexity of random $(d, k, t)$-CSP instances with super-constant constraint density. In particular, we are interested in the ranges of constraint density where the resolution and tree resolution complexity drop from superpolynomial to polynomial.

We consider random $(d, k, t)$-CSP instances with constants $d, k \geq 2$ - they are trivial when either $d = 1$ or $k = 1$. Also, we focus on settings with $1 \leq t < (d - 1)d^{k-2}$ - it is known that for $t \geq (d - 1)d^{k-2}$, w.h.p. the resolution complexity and the tree resolution complexity are at most polynomial when the

\(^2\) See footnote 1.
Chapter 1. Introduction

We introduce an approach for proving lower and upper bounds on the resolution and tree resolution complexity, by taking advantage of the relation between the (tree) resolution complexity of random CSP instances and some properties of fixed non-random CSP instances. To be more specific, we prove lower and upper bounds on the resolution and tree resolution complexity by solving the following non-random problem.

- **Key Problem:** how sparse can an unsatisfiable \((d, k, t)\)-CSP instance be?

The (constraint) density of a CSP instance is defined to be the constraint-variable ratio of the instance. Intuitively, dense CSP instances are tending to be unsatisfiable. However, we can actually reduce the density of an unsatisfiable instance arbitrarily close to zero by adding isolated variables. Therefore, a better measure is the density of the densest subproblem - we call a CSP instance \(\Delta\)-sparse if all of its subproblems have constraint densities at most \(\Delta\). It is straightforward to see that an unsatisfiable instance must contain a subproblem that is sufficiently dense. So it is natural to ask how sparse can an unsatisfiable \((d, k, t)\)-CSP instance be. This is a question regarding the basic properties of CSPs. Some people may even find it more interesting than the problem regarding the resolution complexity of random instances.

Roughly speaking, an unsatisfiable CSP instance is easy to refute if it contains a small unsatisfiable subproblem. Therefore, we can prove resolution complexity results on random CSPs by determining the range of constraint density for the appearances of small unsatisfiable subproblems. We consider the sparsest unsatisfiable CSP instance (subproblem) because it is the first to appear w.h.p. in a random CSP instance when the constraint density grows from zero.

Let \(\Lambda(d, k, t)\) be the solution of the above non-random problem. It is straightforward to obtain the lower and upper bounds on the resolution and tree resolution complexity from \(\Lambda(d, k, t)\), by applying some standard techniques. Therefore, a large portion of the technical contents in this thesis is devoted to solving the non-random problem. The novelty in our work is the way we construct an unsatisfiable instance with constraint density arbitrarily close to \(\Lambda(d, k, t)\) by using a special instance \(H^*\) as the building blocks; and the way we prove that every \(\Lambda(d, k, t)\)-sparse \((d, k, t)\)-CSP instance is satisfiable by transforming a CSP instance into a token allocation problem, which is a combinatorial problem that we create for the purpose of this proof.

We solved the non-random problem and determined the value of \(\Lambda(d, k, t)\) for every constant triple of \((d, k, t)\) with \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\). The proof for the general setting is quite complicated and we are still trying to simplify it. So we will focus on special cases with either \(k = 2\) or \(t = 1\) or \(d = 2\) in this thesis. As a remark, a \(k\)-SAT instance is a \((d, k, t)\)-CSP instance with \(t = 1\) and \(d = 2\), and we have \(\Lambda(2, k, 1) = 1\).

With the exact value of \(\Lambda(d, k, t)\), we showed that for any constant \(\epsilon > 0\), the tree resolution complexity of a random \((d, k, t)\)-CSP instance is w.h.p. superpolynomial when the constraint density is
These bounds are tight up to a \( o(1) \) term in the exponent, and they nearly match the best known bounds for random \( k \)-SAT - the bounds for random \( k \)-SAT are also tight up to a \( o(1) \) term in the exponent.

For general resolution, we show that for any constant \( \epsilon > 0 \), the resolution complexity of a random \((d, k, t)\)-CSP instance is w.h.p. superpolynomial when the constraint density is \( n^{k-1 - \frac{1}{\Lambda(d,k,t)} \epsilon} \); and is w.h.p. constant when the constraint density is \( n^{k-1 - \frac{1}{\Lambda(d,k,t)} + \epsilon} \). There is a gap of \( \frac{k-1}{2} - \frac{1}{\Lambda(d,k,t)} + o(1) \) in the exponent, and these bounds are similar to the best known bounds for random \( k \)-SAT - there is a gap of \( \frac{k-2}{2} + o(1) \) in the exponent for random \( k \)-SAT.

Note that the settings with \( t = 1 \) or \( d = 2 \) are two natural generalizations of the random \( k \)-SAT model - the former allows a more general domain of \( d \) variable values instead of \{TRUE, FALSE\}, and the latter allows \( t \) restrictions in each constraint (clause) instead of one. Thus, the results in these settings can be regarded as generalizations of the resolution complexity results for random \( k \)-SAT.

We will discuss these results in Chapter 3. Then, we will introduce our approach in Chapter 4, and see how to apply our approach in the setting with \( k = 2 \) in Chapter 5 and Chapter 6. Finally, we will briefly discuss the applications of this approach to other models of random CSPs in Chapter 7. Before we go into the details, we will first go through some background in the next chapter.
Chapter 2

Background

In this chapter, we will go through some background related to this thesis. Readers familiar with the topics of resolution complexity and random constraint satisfaction problems may go straight to Section 2.6, where we state the definitions and notations required in later chapters.

2.1 Resolution Proof System

2.1.1 Satisfiability Problem and Propositional Proof System

The Satisfiability Problem (SAT) is one of the fundamental problems in computer science. Given a boolean formula, written using only AND, OR, NOT, boolean variables and parentheses, the question is whether there is any satisfying assignment of TRUE / FALSE values to the variables such that the entire formula is evaluated to TRUE. This problem is the first known NP-complete problem [42]. There is always a simple short proof of satisfiability, i.e. a satisfying assignment. However, a proof of unsatisfiability is much more complicated since it is difficult to verify the absence of satisfying assignment.

A propositional proof system is a polynomial time algorithm $\mathcal{A}$ such that for every boolean formula $F$, $F$ is said to be unsatisfiable if there exists a string $s$ such that $\mathcal{A}$ accepts input $(F, s)$; otherwise, $F$ is said to be satisfiable. Examples of propositional proof system include Frege Proofs, Polynomial Calculus, Resolution, Truth Tables, etc.

The complexity of a propositional proof system $\mathcal{A}$ is defined by a function $f : \mathbb{N} \rightarrow \mathbb{N}$ where

$$f(n) = \max_{F : F \text{ unsatisfiable}, |F| = n} \min_{s : \mathcal{A} \text{ accepts } (F, s)} |s|.$$ 

It concerns the length of the shortest proof on the worst input among all unsatisfiable boolean formulas. A proof system $\mathcal{A}$ is polynomially-bounded if $f(n)$ is upper-bounded by a polynomial function of $n$. It is well-known that a polynomially-bounded propositional proof system exists iff $\text{NP} = \text{co-NP}$ [43]. Short proofs of unsatisfiability are hence interesting for both practical and theoretical reasons.
Chapter 2. Background

2.1.2 Resolution Proofs

Resolution is one of the most popular propositional proof systems. Many practical Satisfiability solvers use resolution-based algorithms. It is a simple proof system, with a single inference rule, the Resolution Rule:

\[
\frac{(A \vee x), (B \vee \overline{x})}{(A \vee B)},
\]

for some variable \(x\) and clauses \((A \vee x)\) and \((B \vee \overline{x})\). The earlier clauses \((A \vee x)\) and \((B \vee \overline{x})\) are called assumption clauses, and the new resulting clause \((A \vee B)\) is called a consequence clause. Sometimes, we use the non-essential Weakening Rule as well:

\[
\frac{(A)}{(A \vee B)}.
\]

A resolution derivation of a clause \(C\) from a boolean CNF formula \(F\) is a sequence of clauses \(C_1, \ldots, C_r = C\) such that each \(C_i\) is either an initial clause from \(F\), or is derived from two earlier clauses \(C_j, C_k\) for \(j, k < i\) by one of the two rules. A resolution refutation is a resolution derivation of the empty clause \(\emptyset\) (i.e. FALSE).

The resolution proof system is both sound and complete. The proof of soundness is easy - at each derivation step, if a truth assignment satisfies both assumption clauses, then it must satisfy the consequence clause as well - we say that a truth assignment satisfies a clause if it makes the clause evaluate to TRUE. Therefore, the unsatisfiable empty clause at the end of a refutation implies there is no satisfying assignment for the given formula. The completeness can be proved by an induction on the number of variables. The proof is a little bit long and will not be shown here.

The length of a derivation is the number of clauses in the derivation, and the size of a derivation, i.e. the number of symbols, is polynomially-bounded by the length of the derivation. The resolution complexity of a boolean CNF formula \(F\), denoted \(\text{RES}(F)\), is defined to be the length of the shortest resolution refutation of \(F\). If \(F\) is satisfiable, no such refutation exists and so \(\text{RES}(F) = \infty\).

Resolution can only work with formulas in conjunctive normal form (CNF), as one can notice from the Resolution Rule. However, requiring the input formula to be in conjunctive normal form is not an important limitation. It is well-known that given any boolean formula, one can construct a CNF formula which has size polynomially-bounded by the size of the original formula, and which is satisfiable iff the original formula is.

2.1.3 Resolution-based Algorithms

The DPLL procedure (or Davis-Putnam procedure) was introduced by Davis, Putnam, Logemann and Loveland [45, 44]. It is both a proof system and a class of satisfiability algorithms. This class of algorithms is said to be resolution-based since it corresponds to a special case of resolution proof. The pseudo code of the DPLL procedure is shown in Figure 2.1. The formula \(F'|_{x=0}\) is created from formula \(F'\) by assigning \(x\) to value 0 (i.e. FALSE) and simplifying the formula accordingly; the formula \(F'|_{x=1}\) is created in the same manner. There are a number of possible splitting rules for choosing a variable in
Step 2. This gives a family of different satisfiability algorithms based on the DPLL procedure.

\[
\text{DPLL}(F)
\]

1. Repeatedly satisfy any unit clause (i.e. size one clause) and pure literal, and simplify the formula accordingly.
   - If resulting formula \( F' \) has no clause, return \( TRUE \).
   - If resulting formula \( F' \) has an empty clause, return \( FALSE \).
2. Choose a variable \( x \in F' \).
3. return \( \text{DPLL}(F'|_{x=0}) \lor \text{DPLL}(F'|_{x=1}) \).

Figure 2.1: The DPLL Procedure

2.1.4 Derivation Graph and Tree Resolutions

For any resolution derivation \( \pi \), we define the derivation graph \( G_\pi \) to be a directed acyclic graph where (i) vertices in \( G_\pi \) are the clauses of \( \pi \); (ii) directed edges in \( G_\pi \) are added from the assumption clauses to the consequence clause, for each derivation step.

Tree resolution is a sub-class of resolution, which only includes the derivations with tree-like derivation graphs. In other words, each derived clause is used at most once as an assumption clause in a tree refutation - if a derived clause is used more than once, it has to be re-derived each time it is used. The tree resolution complexity of a boolean CNF formula \( F \), denoted by \( \text{RES}_T(F) \), is the length of the shortest tree resolution refutation of \( F \). If \( F \) is satisfiable, no such refutation exists and so \( \text{RES}_T(F) = \infty \). The DPLL procedure actually corresponds to tree resolution - the execution of a backtracking DPLL algorithm on an unsatisfiable CNF formula forms a binary DPLL refutation tree, and such a DPLL refutation tree can be translated into a tree-like derivation graph easily.

2.2 Some Resolution Complexity Results

2.2.1 Refutation Length

The refutation length is a widely studied resolution complexity measure. It is defined to be the number of clauses in the shortest resolution refutation. Each clause in a refutation contains at most \( n \) literals, so the size of a refutation is polynomially-bounded by the length of the refutation. We defined resolution complexity by using refutation length earlier (although a more formal definition of resolution complexity is the size of the smallest resolution refutation). The first superpolynomial lower bound on refutation length was presented by Tseitin in 1968 [74], and the study was continued in different formula families [37, 75, 71, 4, 16, 13, 77, 61]. In the following, we will look at some results for three popular formula families.
Random $k$-SAT. A random $k$-SAT instance is a random boolean CNF formula such that each clause contains exactly $k$ literals. The study of resolution complexity of random $k$-SAT started with a seminal paper by Chvátal and Szemerédi [37]. They showed that for any $k \geq 3$, as long as the clause density is a constant, w.h.p. a random $k$-SAT instance has exponentially high resolution complexity. The result was later extended to settings with super-constant clause density, and to more general random models. Random $k$-SAT is a special case of the more general random constraint satisfaction problems, which is the main topic of this thesis. More details will be discussed from Section 2.3 to Section 2.5.

Tseitin Formulas. Tseitin formulas are unsatisfiable CNF formulas capturing a basic combinatorial principle: the sum of vertex degrees is even in every graph. Given any graph $G = (V, E)$ and any marking $m : V \rightarrow \{0, 1\}$, the corresponding Tseitin formula is a CNF formula which is satisfiable if and only if there exists a subgraph $G' = (V, E' \subseteq E)$ such that $v \in V$ has odd vertex degree iff $m(v) = 1$. When the marking is odd (i.e. $\sum_{v \in V} m(v) = 1 (mod 2)$), no such subgraph exists and so the formula is unsatisfiable.

Tseitin [74] defined these formulas and used them to prove the first superpolynomial resolution complexity lower bound (for regular resolution). He showed that the refutation length of some Tseitin formula is superpolynomial, when regular resolution is used. Urquhart [75] improved Tseitin’s result by considering Tseitin formulas on expander graphs. He showed an exponential resolution complexity lower bound for the general resolution proof system. This natural formula family was also studied recently on the complexity measure of refutation space (e.g. [48, 12]).

Pigeonhole Principles. The pigeonhole principle (PHP$^{n+1}_n$) is an unsatisfiable CNF formula capturing a famous principle: there is no one to one mapping from $n+1$ pigeons to $n$ holes. The weak pigeonhole principle (PHP$^m_n$) is a generalized version of the pigeonhole principle that allows a larger number of $m$ pigeons. It is a weaker statement than the pigeonhole principle when $m > n + 1$. Hence, it may have shorter resolution proof when $m$ is sufficiently large.

The variables of PHP$^m_n$ are $x_{i,j}$ for $1 \leq i \leq m, 1 \leq j \leq n$. The boolean variable $x_{i,j}$ indicates the condition of the $i$-th pigeon sitting in the $j$-th hole. The clauses of PHP$^m_n$ are (i) $\bigvee_{j=1}^n x_{i,j}$ for $1 \leq i \leq m$ and (ii) $\overline{x_{i,j}} \lor \overline{x_{k,j}}$ for $1 \leq i < k \leq m, 1 \leq j \leq n$. The former set of clauses restrict each pigeon to sit in at least one hole, while the latter set of clauses restrict each hole to contain at most one pigeon.

The (weak) pigeonhole principle is well studied in the area of propositional proof complexity. The first superpolynomial lower bound on resolution complexity was proved by Haken by considering PHP$^{n+1}_n$ [55]. He showed that the length of any resolution refutation of PHP$^{n+1}_n$ is at least exponential in $n$.

Buss and Turán [32] generalized Haken’s result to the weak pigeonhole principle. They proved a superpolynomial lower bound on resolution complexity when $m = o(n^2 / \log n)$. On the other hand, Buss and Pitassi [31] showed that there exists a polynomial resolution proof for large $m = 2^\sqrt{n \log n}$ (polynomial in $m$; exponential in $n$). Finally, Raz [69] and Razborov [70] proved that any resolution refutation of PHP$^m_n$ has length at least $exp(\Omega(n^\epsilon))$, for some constant $\epsilon > 0$.

---

1 We defined the proof complexity of satisfiable instances is defined to be $\infty$. 
2.2.2 Refutation Width

The *refutation width* also draws a lot of attention. The *width of a clause* \( C \), \( w(C) \) is defined to be the number of literals appearing in \( C \). The *width of a set of clauses* \( F \), \( w(F) \) is defined to be the maximal width among the clauses in the set. The *width of deriving a clause* \( C \) from a formula \( F \) is defined to be the minimum width among all derivations of \( C \) from \( F \). Similarly, the *refutation width* of \( F \), denoted by \( w(F \vdash 0) \), is defined to be the minimum width among all the refutations of \( F \).

A famous result of Ben-Sasson and Wigderson [24] identified an interesting relation between refutation width and refutation length. Roughly speaking, the following theorems show that there is a short resolution refutation only if a narrow refutation exists. This provides a general strategy for proving resolution complexity lower bounds. By using this unified approach, Ben-Sasson and Wigderson re-proved nearly all lower bound results on refutation length known at that time. Details of this approach will be discussed in Section 2.5.

**Theorem 2.1** (Theorem 3.3 in [24]). \( w(F \vdash 0) \leq w(F) + \log \text{RES}_T(F) \) for any unsatisfiable CNF formula \( F \).

**Theorem 2.2** (Theorem 3.5 in [24]). \( w(F \vdash 0) \leq w(F) + O(\sqrt{n \ln \text{RES}(F)}) \) for any unsatisfiable CNF formula \( F \).

Theorem 2.2 was later extended by Urquhart [77] so that it can be applied directly to prove a resolution complexity lower bound for the pigeonhole principle.

Although algorithms for solving SAT are not the focus of this thesis, it is worth noting that there is a direct connection between the refutation width of a formula and the hardness of computing a resolution proof - there are simple algorithms that construct a resolution proof in time \( n^{O(w(F \vdash 0))} \) (e.g. [24, 10]).

2.2.3 Refutation Space

The *refutation space* is another widely studied complexity measure [48, 20, 9, 51, 27]. The *space* of a refutation \( \pi \), denoted by \( \text{space}(\pi) \), is the maximal number of clauses one needs to store during the construction of the refutation. The *refutation space* of a formula \( F \), denoted by \( \text{space}(F \vdash 0) \), is the minimal space among all the refutations of \( F \). As a remark, there is a line of related research concerning the *total space*, which is defined to be the minimal number of symbols one needs to store during the construction of a refutation [5, 28, 25].

Esteban and Torán [48] started the line of research on refutation space. Their results included a number of relations between refutation length and refutation space. In particular, the following theorem was used in a later study to prove a tree resolution complexity lower bound for random \( k \)-SAT [20].

**Theorem 2.3** (Theorem 2.1 in [48]). \( \text{space}(F \vdash 0) \leq \lceil \log \text{RES}_T(F) \rceil + 1 \), for any unsatisfiable CNF formula \( F \).

Esteban and Torán [48] also identified some relations between refutation width and tree refutation space. The result was later extended by Asterias and Dalmasu [9]. They generalized the relation to the general...
refutation space.

**Theorem 2.4** (Corollary 5.2 in [48]). \( \text{treespace}(F \vdash 0) - 1 \geq w(F \vdash 0) - w(F) \), for any unsatisfiable CNF formula \( F \).

**Theorem 2.5** (Theorem 3 in [9]). \( \text{space}(F \vdash 0) \geq w(F \vdash 0) - k + 1 \), for any unsatisfiable \( k \)-CNF formula \( F \).

Another interesting result of Esteban and Torán [48] states that every unsatisfiable formula with \( n \) variables has refutation space at most \( n + 1 \).

**Theorem 2.6** (Theorem 12 in [48]). Every unsatisfiable formula with \( n \) variables can be resolved using resolution in space at most \( n + 1 \).

There are studies concerning the relations between the refutation space and the practical hardness of formulas for SAT solvers [8, 59]. Also, there are results for refutation space lower bounds on different formula families [48, 5, 20, 25]. These studies are less related to this thesis, and will not be discussed here.

### 2.2.4 Separation Results for Length, Space and Width

The width-length relations by Ben-Sasson and Wigderson (Theorem 2.1 and Theorem 2.2) are useful tools in proving lower bounds on the refutation length. It is a natural question whether these width-length relations can be improved. Bonet and Galesi [30] answered this question negatively. They constructed a family of unsatisfiable CNF formulas such that each formula has polynomial refutation length and \( \Omega(\sqrt{n}) \) refutation width. This formula family implies that the bound in Theorem 2.2 is asymptotically tight, up to \( \log \) factors.

**Theorem 2.7** (Theorem 3.8 in [30]). There is a family of unsatisfiable CNF formulas, with constant width and \( O(n) \) variables such that the refutation length is at most polynomial and the refutation width is \( \Omega(\sqrt{n}) \).

On the other hand, Atserias et al. showed that the simple upper bound \( \text{RES}(F) \leq n^{O(w(F \vdash 0))} \) is asymptotically tight as well [11].

**Theorem 2.8** (Theorem 1 in [11]). Let \( w = w(n) = O(n^c) \) for some positive constant \( c < 1/4 \). There is a family of unsatisfiable 3-CNF formulas with \( O(n) \) variables such that the refutation length is \( n^{\Theta(w)} \) and the refutation width is \( O(w) \).

Similar questions concerning space-width, and length-space relations were also studied. Theorem 2.5 states that \( \text{space}(F \vdash 0) = w(F \vdash 0) + \Theta(1) \). A natural follow-up question is whether the refutation space and the refutation width coincide. Nordstrom [67] answered this question negatively by separating space and width. He constructed a family of unsatisfiable formulas with \( O(1) \) refutation width but \( \Theta(\log n) \) refutation space.

**Theorem 2.9** (Corollary 1.2 in [67]). For every \( k \geq 4 \), there exists a family of unsatisfiable \( k \)-CNF formulas with \( O(n) \) variables such that the refutation width is \( O(1) \) and the refutation space is \( \Theta(\log n) \).
Finally, for the length-space relation, it is known that any resolution refutation of length \( O(n) \) can be transformed into a resolution refutation with space \( O(n/ \log n) \) \([57, 48]\). This \( O(n/ \log n) \) upper bound is not much stronger than the general \( n + 1 \) upper bound for every unsatisfiable formula (Theorem 2.6). A natural question is whether the refutation length is capable of giving a much stronger upper bound on refutation space. This question was answered negatively by Ben-Sasson and Nordstrom \([22]\). They presented a family of unsatisfiable CNF formulas of size \( O(n) \) such that the refutation length is \( \Theta(n) \) and the refutation space is \( \Omega(n/ \log n) \).

**Theorem 2.10** (Corollary 1.2 in \([22]\)). There is a family of unsatisfiable 6-CNF formulas of size \( O(n) \) such that the refutation length is \( O(n) \), the refutation width is \( O(1) \) and the refutation space is \( \Omega(n/ \log n) \).

Note that Theorem 2.10 also gives a better separation between refutation space and refutation width than the one stated in Theorem 2.9.

There is a line of research concerning the trade-offs between different complexity measures \([30, 56, 19, 68, 12, 23, 26, 17, 73, 72]\). Some unsatisfiable formulas may have different optimal refutations corresponding to each of these three complexity measures such that the decrease in one measure may cause an increase in another one. For example, there may be a (relatively) short refutation and a (relatively) narrow refutation, but not a both short and narrow refutation. These studies are less related to this thesis, and will not be discussed here.

### 2.2.5 Separation between Resolution and Tree Resolution

Tree resolution is a restricted version of general resolution. Each derived clause is used at most once as an assumption clause in a tree refutation - if a derived clause is used more than once, it has to be re-derived each time it is used. It is not surprising that general resolution outperforms the tree-like variant in some settings \([76, 38, 29, 30, 21]\). The best known result separating resolution and tree resolution is by Ben-Sasson et al. \([21]\).

**Theorem 2.11** (Theorem 1 in \([21]\)). There exists an infinite family of unsatisfiable CNF formulas, of size \( O(n) \), such that there exist \( O(n) \)-size resolution refutations, but every tree refutation has size at least \( \exp(\Omega(n/ \log n)) \).

Furthermore, they proved an upper bound on the magnitude of the separation gap. This shows that the separation result is nearly optimal.

**Theorem 2.12** (Theorem 2 in \([21]\)). \( \text{RES}_T(F) = \exp(O(\frac{\text{RES}(F) \log \log \text{RES}(F)}{\log \text{RES}(F)})) \), for every unsatisfiable CNF formula \( F \).

### 2.3 Random Constraint Satisfaction Problems

Given an event \( E = E_n \) depending on parameter \( n \), it is said to occur with high probability (w.h.p.) if \( \lim_{n \to \infty} \Pr[E] = 1 \); and occur with uniformly positive probability (w.u.p.p.) if \( \lim \inf \Pr[E] > 0 \). In the
following, the number of variables $n$ and the number of clauses (or constraints) $m$ are arbitrarily large growing to infinity; other parameters are all constants, unless specified.

### 2.3.1 Random $k$-SAT

Random $k$-SAT is the most well-studied random constraint satisfaction problem. There are numerous works on this topic over the past decades. Let $D_k(n)$ denote the set of all $M = \binom{n}{k}2^k$ clauses of size $k$ on $n$ variables. A random $k$-SAT instance is a random $k$-CNF formula, which can be generated by choosing an instance from the set of all the $k$-CNF formulas with $m$ clauses of $D_k(n)$ with uniformly probability. Denote this distribution by $F_k(n, m)$. We consider constant $k$ and arbitrarily large $n$ and $m$ growing to infinity. The clause-variable ratio $\Delta = m/n$ is referred to as the clause density.

Another natural model for generating a random $k$-CNF formula is to choose each of the $M$ possible clauses from $D_k(n)$ independently with probability $p = m/M$. Denote this distribution by $F_k(n, p)$. The probability $p$ is referred to as the clause probability.

There is a well-known asymptotic equivalence between these two models (e.g. see Propositions 1.12 and 1.15 in [58]). A property is said to be convex with respect to the set of clauses if for any sets of clauses $A \subseteq B \subseteq C$, the property holding in both $A$ and $C$ implies the property holds in $B$ as well. For any convex property $Q$ and any $p = p(n) \in [0, 1]$, if $Q$ holds w.h.p. in a random instance drawn from $F_k(n, m)$ for every $m = (1 \pm o(1))M \cdot p$, then it must also hold w.h.p. in a random instance drawn from $F_k(n, p)$. Similarly, if a convex property holds w.h.p. in a random instance drawn from $F_k(n, p)$, then it must hold w.h.p. in a random instance drawn from $F_k(n, m)$ with $m = p \cdot M$.

Well known examples of convex properties include subproblem containment, the property of containing a certain small $k$-SAT instance as a subproblem; satisfiability, the property of having a satisfying assignment; and the property of having resolution refutation of size at most $L$.

It is worth noting that these properties are also monotone / anti-monotone with respect to the set of clauses - a property is said to be monotone with respect to the set of clauses if for any sets of clauses $A \subseteq B$, the property holding in $A$ implies the property holds in $B$ as well; it is said to be anti-monotone if the property holding in $B$ implies the property holds in $A$ as well. We can see from the definitions that every monotone / anti-monotone property is a convex property.

### 2.3.2 Satisfiability Threshold

The satisfiability threshold is the most studied property of random $k$-SAT. One of the first results on random $k$-SAT showed that when the clause density is at least $2^{k-1} \ln 2$, w.h.p. a random $k$-CNF formula is unsatisfiable [52]. On the other hand, results showed that when the clause density is lower than $2^k/k$, a satisfying assignment can be found by simple algorithms w.u.p.p. [34, 35]. A major question in the study of random $k$-SAT is the existence of a sharp threshold of satisfiability around a constant critical clause density $\Delta_k$. It is known that $\Delta_2 = 1$ but the value of $\Delta_k$ is unknown for any $k \geq 3$.

**Conjecture 2.13** (Satisfiability Threshold Conjecture). *For every $k \geq 3$, there exists a constant $\Delta_k$*
such that for any $\epsilon > 0$,

$$\lim_{n \to \infty} Pr[F_k(n, m) \text{ is satisfiable}] = \begin{cases} 1 & \text{if } m = (\Delta_k - \epsilon)n, \\ 0 & \text{if } m = (\Delta_k + \epsilon)n. \end{cases}$$

A big step in proving this conjecture was made by Friedgut [53]. Instead of a constant critical clause density $\Delta_k$, he showed the existence of a sharp threshold around a critical sequence of clause density $\Delta_k(n)$.

**Theorem 2.14** ([53]). For every $k \geq 3$, there exists a sequence $\Delta_k(n)$ such that for any $\epsilon > 0$,

$$\lim_{n \to \infty} Pr[F_k(n, m) \text{ is satisfiable}] = \begin{cases} 1 & \text{if } m = (\Delta_k(n) - \epsilon)n, \\ 0 & \text{if } m = (\Delta_k(n) + \epsilon)n. \end{cases}$$

A series of results improved the bounds on the location of the satisfiability threshold to $2^k \ln 2 - \Theta(1) \leq \Delta_k(n) \leq 2^k \ln 2 - \Theta(1)$ [47, 60, 3, 40], i.e. the gap between the bounds is at most an absolute constant. Recently, a celebrated result by Ding et al. [46] proved the satisfiability threshold conjecture and determined the value of $\Delta_k$ for all $k \geq k_0$, where $k_0$ is a positive constant.

### 2.3.3 Algorithmic Barrier and Resolution Complexity

For clause density below the satisfiability threshold, w.h.p. a random $k$-CNF formula is satisfiable. A primary goal is to design efficient algorithms for finding satisfying assignments. It is well-known that a satisfying assignment can be found by simple algorithms w.u.p.p., when the clause density is lower than $c(2^k / k)$, for some small constant $c > 0$ [34, 35]. However, this problem becomes much more difficult when the clause density grows farther. The best known algorithm is only capable of solving the problem w.h.p., when the clause density is lower than $2^k \ln k/k$ [39]; even though the proven location of satisfiability threshold is between $2^k \ln 2 - \Theta(1) \leq \Delta_k \leq 2^k \ln 2 - \Theta(1)$. This phenomenon is called the **algorithmic barrier**.

For clause density above the satisfiability threshold, w.h.p. a random $k$-CNF formula is unsatisfiable. A phenomenon similar to the algorithmic barrier also happens here. The proven threshold location is at $2^k \ln 2 - \Theta(1)$, which is $\Theta(1)$ when $k$ is a constant. However, there is no known polynomial size proof of unsatisfiability for an unsatisfiable random $k$-CNF formula with constant clause density. The best known result was proved by Allen et al. [6]. They showed that w.h.p. a polynomial size proof of unsatisfiability exists when the clause density is $\Omega(n^{k^2 / 2})$. Coja-Oghlan et.al [41] and Feige et al. [50] proved slightly better results when $k$ is even. They showed that w.h.p. a polynomial size proof of unsatisfiability exists when the clause density is $\Omega(n^{k^2 / 2})$ and $k$ is even. For random $3$-CNF formulas, Feige et al. [49] and Müller and Tzameret [66] showed that w.h.p. a polynomial size proof of unsatisfiability exists when the clause density is $\Omega(n^{0.4})$.

In this thesis, we focus on the resolution proof system, which is used in most practical SAT solvers. It is known that w.h.p. an unsatisfiable random $k$-CNF formula requires an exponential size resolution
refutation, when the clause density is a constant \[37\]. The best known upper bound was proved by Beame et al. \[14\]. They showed that w.h.p. the resolution complexity of an unsatisfiable random \(k\)-CNF formula is at most polynomial when the clause density is \(\Omega(n^{k-2}/\log^{k-2}n)\). Details of these resolution complexity results will be discussed in Section 2.4.

### 2.3.4 Random \((d, k, t)\)-CSP Model

The random \((d, k, t)\)-CSP model is a natural generalization of the random \(k\)-SAT model. In a \((d, k, t)\)-CSP instance, every variable can take a value from the same domain of values \(D = \{1, \ldots, d\}\). A restriction is a \(k\)-tuple of values \((\delta_1, \ldots, \delta_k)\) on a \(k\)-tuple of variables \((x_1, \ldots, x_k)\), with all \(\delta_i \in D\). It forbids assigning the \(k\)-tuple of values to the \(k\)-tuple of variables, i.e. it forces at least one \(x_i\) to be assigned a value \(\delta \neq \delta_i\).

A constraint consists of \(t\) restrictions on the same \(k\) variables. In the following, consider any integers \(d, k \geq 2\) and \(t \geq 1\).

An assignment of values to the variables satisfies a restriction if the restriction does not match the corresponding \(k\)-tuple of values; it satisfies a constraint if all the \(t\) restrictions are satisfied; and satisfies a CSP instance if all the constraints are satisfied. A value assignment satisfies a CSP instance is called a satisfying assignment of the CSP instance. A CSP instance is said to be satisfiable if it has at least one satisfying assignment; otherwise, it is said to be unsatisfiable.

We define the distribution \(\text{CSP}^{d,k,t}_{n,m}\) as follows: generate a random \((d, k, t)\)-CSP instance with \(m\) constraints and \(n\) variables by selecting independently, uniformly and without replacement \(m\) \(k\)-tuples on \(n\) variables; then for each \(k\)-tuple of variables, choose one of the \((d^k)\) possible constraints independently and uniformly. The constraint-variable ratio \(\Delta = m/n\) is referred to as the constraint density.

We define another distribution \(\text{CSP}^{d,k,t}_{n,p}\) as follows: first choose each of the \(M = (\binom{n}{k})\) \(k\)-tuples on \(n\) variables independently with probability \(p\); then for each chosen \(k\)-tuple of variables, choose one of the \((d^k)\) possible constraints independently and uniformly. The probability \(p\) is referred to as the constraint probability.

Just like the \(k\)-SAT models, a convex property must hold w.h.p. in a random instance drawn from \(\text{CSP}^{d,k,t}_{n,p}\) if it holds w.h.p. in a random instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\) for every \(m = (1 \pm o(1))M \cdot p\); similarly, a convex property must hold w.h.p. in a random instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\) if it holds w.h.p. in a random instance drawn from \(\text{CSP}^{d,k,t}_{n,p}\) with \(p = m/M\). (e.g. see Propositions 1.12 and 1.15 in [58]).

For completeness, we will present the main results of this thesis for both random models \(\text{CSP}^{d,k,t}_{n,m}\) and \(\text{CSP}^{d,k,t}_{n,p}\).

The random \((d, k, t)\)-CSP model is a generalization of the random \(k\)-SAT model. It allows a more general domain of \(d\) variable values instead of \(\{\text{True, False}\}\), and allows \(t\) restrictions on variable values in each constraint instead of one. We can see that a random \(k\)-SAT model is equivalent to a special case of the random \((d, k, t)\)-CSP model with \(d = 2\) and \(t = 1\).
Chapter 2. Background

2.3.5 Resolution Proof System for General CSP Models

The resolution proof system is designed for boolean formulas, but not for the more general CSPs. Mitchell [62, 63] considered two natural ways to extend the resolution proof system to the more general setting of CSPs. One approach, known as \textit{NG-RES}, extends resolution rules to the setting of CSPs by using \(d\) assumption clauses instead of two, where \(d\) is the domain size. Another approach, known as \textit{C-RES}, converts a CSP instance to a boolean CNF-formula and then applies the basic resolution rules. Mitchell showed that for every CSP instance, the size of a C-RES refutation is polynomially bounded from above by the size of a NG-RES refutation. The C-RES approach was hence chosen in later studies [64, 33]. We will also use the C-RES refutation in this thesis.

The C-RES procedure for transforming a \((d, k, t)\)-CSP instance into a CNF formula is quite straightforward. Given any \((d, k, t)\)-CSP instance \(P\), the corresponding CNF formula \(CNF(P)\) can be constructed as follows. First, for each variable \(v\) in \(P\), assign \(d\) boolean variables \(x_{v,1}, \ldots, x_{v,d}\), one for each value in the domain \(D\). Then, the following two sets of clauses are added to ensure that \(P\) is satisfiable iff \(CNF(P)\) is. (I) For each variable \(v\) in \(P\), \(CNF(P)\) has a \textit{domain clause} \((x_{v,1} \lor \ldots \lor x_{v,d})\) to force at least one of the \(x_{v,i}\)'s to be TRUE. (II) For each restriction \((v_1 : \delta_1, \ldots, v_k : \delta_k)\) in the constraints of \(P\), \(CNF(P)\) has a \textit{conflict clause} \((\overline{x_{v_1,\delta_1}} \lor \ldots \lor \overline{x_{v_k,\delta_k}})\) to express the conflict of that restriction. For any satisfying assignment of \(CNF(P)\), if multiple \(x_{v,i}\)'s are TRUE for any \(v\), then we can set all but one of them to FALSE while keeping the assignment satisfying - this does not violate the domain clause since one of them remains TRUE; and this does not violate any conflict clause since a conflict clause just forbids all of its variables to be TRUE. It is now very clear that \(P\) is satisfiable iff \(CNF(P)\) is.

We can see that for any \((d, k, t)\)-CSP instance \(P\), the size of \(CNF(P)\) is polynomially bounded from above by the size of \(P\) (when \(d\) and \(t\) are constants). In this thesis, when we consider the resolution complexity of a CSP instance \(P\), we are actually referring to the resolution complexity of \(CNF(P)\).

2.3.6 Constraint Hypergraph

Random constraint satisfaction problems are closely related to random hypergraphs. Define the \textit{constraint hypergraph} of a CSP instance \(P\) to be a hypergraph which takes the variable set of \(P\) as vertex set, and takes the constraint set of \(P\) as edge set. Then random hypergraph properties, such as the edge density for existence of small subgraphs, can be used in the proof of resolution complexity (e.g. [64, 33]).

2.4 Some Results in Resolution Complexity of Random CSPs

Recall that we defined the resolution complexity and the tree resolution complexity of satisfiable instances to be \(\infty\).
2.4.1 Random $k$-SAT

Random $k$-SAT is the most well-studied random constraint satisfaction problem. There are numerous works on this topic over the past decades. The study of resolution complexity on random CSPs started with a seminal paper by Chvátal and Szemerédi [37]. They proved that w.h.p. the resolution complexity is exponential for any random $k$-SAT instance with constant clause density.

**Theorem 2.15** (Theorem in [37]). *For all constants $\Delta > 0$ and $k \geq 3$, there exists a constant $\epsilon > 0$ such that w.h.p. a random $k$-SAT instance drawn from $F_k(n, m)$ with $m = \Delta n$ has resolution complexity at least $(1 + \epsilon)^n$.***

The proof technique was extended to prove superpolynomial resolution complexity lower bounds for settings with super-constant clause density [15, 54]. The result was further improved, and the proof was simplified by Beame et al. [14] and by Ben-Sasson and Wigderson [24, 18]. Ben-Sasson and Wigderson proved the following result by using the relation between refutation length and refutation width (Theorem 2.1 and Theorem 2.2).

**Theorem 2.16** (Theorem 2.24 in [18]). *For any clause density $\Delta > 0$ and constant $k \geq 3$, there exists a constant $\epsilon > 0$ such that for any random $k$-SAT instance drawn from $F_k(n, m)$ with $m = \Delta n$, (i) w.h.p. the tree resolution complexity is $\exp(\Omega\left(\frac{n}{\Delta^{2/(k-2)+\epsilon}}\right))$; (ii) w.h.p. the resolution complexity is $\exp(\Omega\left(\frac{n}{\Delta^{2/(k-2)+\epsilon}}\right))$.***

On the other hand, Fu [54] showed that if the clause density is $\Omega(n^{k-2})$, then w.h.p. the resolution complexity of a random $k$-SAT instance is at most polynomial in $n$. Beame et al. [14] improved the bound by providing a simple DPLL algorithm to construct a resolution proof in polynomial time when the clause density is $\Omega(n^{k-2}/\log^k n)$. Since the DPLL procedure corresponds to the tree resolution proof system, this result implies an upper bound on both resolution complexity and tree resolution complexity. As a remark, Monasson [65] studied some DPLL algorithms on random $k$-SAT and showed that the DPLL tree has a polynomial average size when the constraint density is $\Omega(n^{k-2}/\log^k n)$. This agrees with the above result by Beame et al [14].

**Theorem 2.17** (Theorem 6.3 in [14]). *Let clause density $\Delta$ be greater than the satisfiability threshold. If $F$ is a random $k$-SAT instance drawn from $F_k(n, m)$ with $m = \Delta n$, then there is a simple DPLL algorithm which w.h.p. produces a resolution refutation in time $2^{O(n/\Delta^{1/(k-2)})} n^{O(1)}$.***

Consider a random $k$-SAT instance for any $k \geq 3$. Theorem 2.16 and Theorem 2.17 imply that the tree resolution complexity is w.h.p. superpolynomial when the clause density is $n^{k-2-\epsilon}$, and is w.h.p. at most polynomial when the clause density is $\Omega(n^{k-2}/\log^k n)$. These bounds are nearly tight as there is only a small gap with order of magnitude $n^{o(1)}$. For resolution complexity, these theorems imply that the resolution complexity is w.h.p. superpolynomial when the clause density is $n^{k-2-\epsilon}$, and is w.h.p. at most polynomial when the clause density is $\Omega(n^{k-2}/\log^k n)$. There is a larger gap with order of magnitude $n^{k-2+o(1)}$. 
2.4.2 Random \((d, k, t)\)-CSP

Mitchell [62, 63] studied extensions of the resolution complexity notions to the more general setting of CSPs. He applied the width-length approach in the random \((d, k, t)\)-CSP model and proved that w.h.p. a random \((d, k, t)\)-CSP instance has exponential resolution complexity for a particular range of \((d, k, t)\).

**Theorem 2.18** ([63]). When \(t \leq (d - 1)/2, k = 2\) or \(t \leq d - 1, k \geq 3\), for any constant \(\Delta > 0\) w.h.p. a random instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\) with \(m = \Delta n\) has exponential resolution complexity.

On the other hand, Achlioptas et al. [2] showed that w.h.p. the resolution complexity is constant for large values of \(t\).

**Theorem 2.19** ([2]). For any constants \(d, k \geq 2\) and \(t \geq d^{k-1}\), and for any constant \(\Delta > 0\), w.h.p. a random instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\) with \(m = \Delta n\) has constant resolution complexity.

Molloy and Salavatipour [64] extended Mitchell’s results. They determined the resolution complexity for each constant triple \((d, k, t)\) in the range of \(t < d^{k-1}\).

**Theorem 2.20** (Theorem 2 and Theorem 3 in [64]). For any constants \(d, k \geq 2\) and \((d - 1)d^{k-2} \leq t < d^{k-1}\), there exists a constant constraint density \(\Delta^* = \Delta^*(d, k, t)\) such that w.h.p. a random instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\) with \(m < \Delta^* n\) has exponential resolution complexity; and w.h.p. a random instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\) with \(m > \Delta^* n\) has polynomial resolution complexity.

**Theorem 2.21** (Theorem 1 in [64]). For any constants \(d, k \geq 2\) and \(1 \leq t < (d - 1)d^{k-2}\), and for every constant constraint density \(\Delta > 0\), w.h.p. a random instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\) with \(m = \Delta n\) has exponential resolution complexity.

Together, these theorems give the whole picture of the resolution complexity of random \((d, k, t)\)-CSP instances for all constants \(d, k, t\) and \(\Delta\), except for \((d - 1)d^{k-2} \leq t < d^{k-1}\) and \(\Delta = \Delta^*(d, k, t)\), the threshold constraint density.

2.4.3 Other Random Models

Achlioptas et al. [1] studied the resolution complexity of \((2 + p)\)-SAT. They proved that if the density of 2-clauses is smaller than one, then w.h.p. a random \((2 + p)\)-SAT instance has no subexponential size resolution refutation. It is well-known that if the density of 2-clauses is larger than one, then w.h.p. a random \((2 + p)\)-SAT instance has a linear-size resolution refutation. Thus, the following theorem implies a sharp threshold for exponential resolution complexity of \((2 + p)\)-SAT.

**Theorem 2.22** (Theorem 1.1 in [1]). For every constant \(\epsilon > 0\) and constant clause density \(\Delta\), w.h.p. a random \((2 + p)\)-SAT instance with \((1 - \epsilon)n\) 2-clauses and \(\Delta n\) 3-clauses has resolution complexity at least \(2^{\Omega(n)}\).

Chan and Molloy [33] gave a dichotomy theorem for the resolution complexity of random CSPs. They proved that for a very broad family of CSPs, either w.h.p. the resolution complexity is polylogarithmic for sufficiently large constraint density \(\Delta\), or w.h.p. it is exponential for every \(\Delta > 0\).
2.5 Techniques in Resolution Complexity of Random CSPs

2.5.1 Lower Bound by Width-Length Relation

According to the width-length relations by Ben-Sasson and Wigderson (Theorem 2.1 and Theorem 2.2), a short resolution refutation exists only if a narrow resolution refutation exists. Hence, we can prove a resolution complexity lower bound by showing the existence of a wide clause (i.e. a clause with a large number of literals) in every resolution refutation of the given formula. This width-length approach provides an unified and powerful framework for proving resolution complexity lower bounds. The best known resolution complexity lower bounds of random CSPs can all be proved by using this approach.

To show the presence of a wide clause, the now standard approach is to (i) show that there is a wide clause in every resolution refutation if a CSP instance satisfies certain sparsity conditions; and (ii) show that w.h.p. the sparsity conditions hold in a CSP instance drawn from the given random model. An example of the sparsity conditions for random 3-SAT are stated as (A) and (B) below.

(A) Every subformula with at most $\alpha = \alpha(n)$ variables is satisfiable.

(B) Every subformula with $n'$ variables has at least $\zeta = \zeta(n)$ degree one variables, where $\frac{1}{2}\alpha \leq n' \leq \alpha$.

Roughly speaking, Condition (A) states that we need a subformula of size greater than $\alpha$ to derive unsatisfiability. With this property, we can show that in any resolution refutation, there must be a clause that is minimally derived from a subformula $F'$ with size $\frac{1}{2}\alpha \leq n' \leq \alpha$ - we say a clause is minimally derived from a formula if it cannot be derived from any of its subformulas. According to Condition (B), the subformula $F'$ must contain at least $\zeta$ degree one variables. Notice that we cannot resolve on a degree one variable because one of its literals does not appear in the formula. Thus, all the $\zeta$ degree one variables must remain in the clause minimally derived from $F'$. Therefore, for any formula satisfying both Conditions (A) and (B), every resolution refutation of this formula must contain a clause with size at least $\zeta$.

In general, the magnitude of $\alpha$ may vary from $\Theta(n)$ to $\Theta(n^\gamma)$ for some small constant $\gamma > 0$, depending on the random models and the results we want to prove. The low degree variables in Condition (B) are called the boundary variables. Informally, these variables always appear in a clause minimally derived from the given subformula. The definition of boundary variables also depends the random models and the results we want to prove. For example, a more complicated definition of boundary variables was used in the study of the random $(d,k,t)$-CSP model [64].

On the other hand, we may also refine Condition (B) in order to prove a better complexity lower bound. Instead of using boundary variables, Ben-Sasson [18] showed that a clause must be wide if it is minimally derived from a sparse CNF formula.

**Theorem 2.23** (Theorem 2.18 in [18]). For a clause $C$ minimally derived from a CNF formula $F$, the number of literals in $C$ is at least the number of literals in $F$ minus the number of clauses in $F$.

With the above theorem, Ben-Sasson used a weaker Condition (B') to prove a lower bound on the resolution complexity for random $k$-SAT for any $k \geq 3$. 
(B') There exists some constant \( \sigma > 0 \) such that every subformula with \( n' \) variables has clause density at most \( 1 - \sigma \), where \( \frac{1}{2} \alpha \leq n' \leq \alpha \).

For any \( k \geq 3 \), this weaker condition can be satisfied more easily. Actually, Condition (B) implies Condition (B') - if a \( k \)-SAT instance has clause density higher than \( 1 - \sigma \), then it must contain a subformula with only a few degree one variables (less than \( \zeta n \) for any constant \( \zeta > 0 \)). However, Condition (B') does not imply Condition (B) - the (generalized) handshaking lemma implies that the sum of degrees is \( (1 - \sigma) \cdot k \) when the clause density is \( 1 - \sigma \); so, when \( k \geq 3 \), there are many \( k \)-CNF formulas with clause density \( (1 - \sigma) \) that does not contain any degree one variable.

2.5.2 Another Lower Bound Technique

**Bottleneck Counting.** The first resolution complexity lower bound was proved by Haken [55], by using the technique known as *bottleneck counting*. This technique was developed and used in later works on resolution complexity [75, 37, 54, 14].

Given a CNF formula \( F \) and one of its subformula \( F' \subset F \), a truth assignment \( A \) is said to be a *partial solution* for \( F' \) if it satisfies all the clauses in \( F' \). A clause *falsifies* a partial solution \( A \) if it is not satisfied by \( A \).

Roughly speaking, the Bottleneck Counting technique consists of two main steps. An important intuition is that a CNF formula needs an exponentially long resolution refutation only if it has a large number of partial solutions. The first step of Bottleneck Counting is to prove a formal restatement of the above intuition - the given CNF formula has exponentially many *special* large-size partial solutions. The second step is to prove that for any resolution refutation \( \pi \), each of the special partial solutions is falsified by a wide clause in \( \pi \). A wide clause is easy to satisfy and cannot falsify too many large partial solutions. Hence, there must be (exponentially) many wide clauses in \( \pi \), in order to falsify all the special partial solutions. This proves a resolution complexity lower bound.

The Bottleneck Counting technique uses a width-length relation of resolution refutation implicitly. This implicit relation was later formalized by Ben-Sasson and Wigderson as Theorem 2.1 and Theorem 2.2. With these width-length relations, they re-derived all the resolution complexity results previously proved by the Bottleneck Counting technique.

2.5.3 Upper Bound Techniques

The width-length relations provide a standard approach for proving resolution complexity lower bounds of random CSPs. To prove resolution complexity upper bounds, there are two common approaches. One approach is to construct an algorithm which takes a random CSP instance as input and returns a resolution proof. Beame et al. [14] analyzed the performance of a DPLL algorithm on random \( k \)-SAT instances. They showed that w.h.p. a simple DPLL algorithm produces a resolution refutation in polynomial time, when the constraint density is sufficiently high (Theorem 2.17). Their result gave the best known upper bound on both the resolution complexity and the tree resolution complexity of
random \( k \)-SAT.

However the analysis may get more complicated when we consider more general random CSP models. Instead of using an algorithmic approach, upper bounds are proved by showing that w.h.p. a random CSP instance contains a small unsatisfiable subproblem \([36, 64, 33]\). These small subproblems (subformulas), usually of size \( O(\log n) \), can be refuted in polynomial time by brute force. This implies a polynomial tree resolution complexity upper bound.

Molloy and Salavatipour \([64]\) extended the “snake” structure in random 2-SAT by Chvátal and Reed \([36]\) to the setting of random \((d, k, t)\)-CSP. They used a special type of constraint, called forcer edges, to constraint a small unsatisfiable subproblem, and proved a resolution complexity upper bound for the random \((d, k, t)\)-CSP model (Theorem 2.20). A constraint is an \((x_1: \delta_1) \rightarrow (x_2: \delta_2)\) forcer edge if fixing the value at \(x_1\) to \(\delta_1\) will force us to assign \(\delta_2\) to \(x_2\). In other words, the constraint contains all the \((d - 1)d^{k-2}\) restrictions with \(x_1: \delta_1\) and \(x_2: \gamma\) for every \(\gamma \neq \delta_2\).

By connecting appropriate forcer edges, we can get an \((x_1: \delta_1) \rightarrow (x_\ell: \delta_\ell)\) forcing path such that fixing the value at \(x_1\) to \(\delta_1\) will force us to assign \(\delta_\ell\) to \(x_\ell\). By making the forcing path into a cycle, we can get a forbidding cycle such that fixing the value at \(x_1\) to \(\delta_1\) will force us to assign \(\delta_\ell\) to \(x_1\). Suppose \(\delta_\ell \neq \delta_1\), this cycle forbids us from assigning \(\delta_1\) to \(x_1\). By placing \(d\) appropriate forbidding cycles at a variable \(x\), we can forbid all the \(d\) values in the domain from being assigned to \(x\). This \((d, k, t)\)-CSP instance is unsatisfiable, and is called a forbidding flower, which is a generalization of the “snake” structure by Chvátal and Reed \([36]\).

For any constants \(d, k \geq 2\) and \((d - 1)d^{k-2} \leq t < d^{k-1}\), by using the second moment method, we can prove that w.h.p. a forbidding flower appears in a random \((d, k, t)\)-CSP instance when the constraint density is sufficiently high. With appropriate size (usually \(\Theta(\log n)\)), a forbidding flower can be refuted in polynomial time by brute force. This gives an upper bound on both resolution complexity and tree resolution complexity (Theorem 2.20).

**Remark 2.24.** Each forcer edge contains at least \((d - 1)d^{k-2}\) restrictions. Thus, there does not exist any forcer edge when \(t < (d - 1)d^{k-2}\). In other words, this approach does not work when \(1 \leq t < (d - 1)d^{k-2}\). In fact, we know from Theorem 2.21 that the resolution complexity of a random \((d, k, t)\)-CSP instance is w.h.p. exponential when the constraint density is any constant; however, not much is known when the constraint density grows beyond constant. In this thesis, we focus on the random \((d, k, t)\)-CSP model with \(1 \leq t < (d - 1)d^{k-2}\), and study the resolution and tree resolution complexity in the range of super-constant constraint density.

### 2.6 Formal Definitions and Notations

In the following, we will formally restate some of the ideas we just discussed, in order to avoid ambiguity. These definitions and notations are required in later chapters.
2.6.1 Notations regarding the Resolution Proof System

Definition 2.25 (Resolution Derivation). The resolution derivation \( \pi \) of a clause \( C \) from a boolean CNF formula \( F \) is a sequence of clauses \( C_1, \ldots, C_r = C \) such that each \( C_i \) is either an initial clause from \( F \), or is derived from two earlier clauses \( C_j, C_k \) for \( j, k < i \) by one of the two rules in the resolution proof system.

Definition 2.26 (Resolution Refutation). A resolution refutation is a resolution derivation of the empty clause \( \emptyset \) (i.e. FALSE).

Definition 2.27 (Clause Derived and Minimally Derived). We say that a clause \( C \) is derived from a boolean CNF formula \( F \) if there exists a resolution derivation of \( C \) from \( F \). The clause \( C \) is minimally derived from \( F \) if it is derived from \( F \) and it is not derived from any subformula of \( F \).

Definition 2.28 (Length of a Derivation). The length of a derivation is the number of clauses in the derivation.

Definition 2.29 (Resolution Complexity). The resolution complexity of a boolean CNF formula \( F \), denoted by \( \text{RES}(F) \), is defined to be the length of the shortest resolution refutation of \( F \). If \( F \) is satisfiable, no such refutation exists and so \( \text{RES}(F) = \infty \).

Definition 2.30 (Width). The width of a clause \( C \), \( w(C) \) is defined to be the number of literals appearing in \( C \). The width of a set of clauses \( F \), \( w(F) \) is defined to be the maximal width among the clauses in the set. The width of deriving a clause \( C \) from a boolean CNF formula \( F \) is defined to be the minimum width among all derivations of \( C \) from \( F \).

Definition 2.31 (Refutation Width). The refutation width of a boolean CNF formula \( F \), denoted by \( w(F \vdash 0) \), is defined to be the minimum width among all the refutations of \( F \). If \( F \) is satisfiable, no such refutation exists and so \( w(F \vdash 0) = \infty \).

Definition 2.32 (Derivation Graph). For any resolution derivation \( \pi \), we define its derivation graph \( G_{\pi} \) to be a directed acyclic graph where (i) vertices in \( G_{\pi} \) are the clauses of \( \pi \); (ii) directed edges in \( G_{\pi} \) are added from the assumption clauses to the consequence clause, for each derivation step.

Definition 2.33 (Tree Resolution Derivation and Refutation). A tree resolution derivation is a resolution derivation with derivation graph being a tree. A tree resolution refutation is a tree resolution derivation of the empty clause \( \emptyset \) (i.e. FALSE).

Definition 2.34 (Tree Resolution Complexity). The tree resolution complexity of a boolean CNF formula \( F \), denoted by \( \text{RES}_T(F) \), is defined to be the length of the shortest tree resolution refutation of \( F \). If \( F \) is satisfiable, no such refutation exists and so \( \text{RES}(F) = \infty \).

2.6.2 Notations regarding Constraint Satisfaction Problems

We will define the following notations regarding to the \((d, k, t)\)-CSP model, because we only consider \((d, k, t)\)-CSP instances in this thesis. Note that these notations can be extended easily for more general CSPs.
Definition 2.35 (Restriction). Given a set of \( k \) variables \( \{x_1, \ldots, x_k\} \), a restriction \( R \) on these \( k \) variables is a string of the form \( R = (x_1 : \delta_1, \ldots, x_k : \delta_k) \), where \( \delta_i \in \{1, \ldots, d\} \) for \( i = 1, 2, \ldots, k \). Denote by \( V(R) \) the set of variables appearing in \( R \). We say that \( R \) contains the variable set \( V(R) \).

Definition 2.36 (Constraint). Given a set of \( k \) variables \( \{x_1, \ldots, x_k\} \), a constraint \( e \) on these \( k \) variables is a set of \( t \) restrictions on these \( k \) variables. Denote by \( V(e) \) the set of variables appearing in \( e \). We say that \( e \) contains the variable set \( V(e) \).

Definition 2.37 ((\( d, k, t \))-CSP Model). A \((d, k, t)\)-CSP instance is defined as a pair \( \{V, E\} \) with parameters \( d, k, t \), where \( V = \{v_1, \ldots, v_n\} \) is a set of variables, \( E = \{e_1, \ldots, e_m\} \) is a set of constraints. Each variable \( v_i \in V \) can take a value from the same domain of values \( D = \{1, \ldots, d\} \). Each constraint \( e_j \in E \) contains \( k \) variables from \( V \), and consists of \( t \) restrictions containing the same set of \( k \) variables. The variable set and constraint set of a \((d, k, t)\)-CSP instance \( P \) are referred to as \( V(P) \) and \( E(P) \) respectively.

Definition 2.38 (Subproblem). A \((d, k, t)\)-CSP instance \( P' \) is a subproblem of a \((d, k, t)\)-CSP instance \( P \) if (i) \( E(P') \subseteq E(P) \), (ii) \( V(P') \subseteq V(P) \), (iii) \( V(e') \subseteq V(P') \) for every \( e' \in E(P') \).

Definition 2.39 ((Partial) Value Assignment). A value assignment on a variable set \( V \) is a function mapping each variable \( v \in V \) to a value in \( D = \{1, \ldots, d\} \). A value assignment of a \((d, k, t)\)-CSP instance \( P \) is a value assignment on \( V(P) \); and a partial value assignment is a value assignment on a variable subset \( V' \subseteq V(P) \).

Definition 2.40 (Satisfy). For any value assignment, it satisfies a restriction \( R = (x_1 : \delta_1, \ldots, x_k : \delta_k) \) if \( A(x_i) \neq \delta_i \) for some \( i = 1, 2, \ldots, k \); it satisfies a constraint \( e_j \) if it satisfies every restriction in \( e_j \); it satisfies a \((d, k, t)\)-CSP instance \( P \) if it satisfies every constraint in \( E(P) \).

Definition 2.41 (Satisfying Assignment and Satisfiability). A value assignment \( A \) is called a satisfying assignment of a \((d, k, t)\)-CSP instance \( P \) if \( A \) satisfies \( P \). A \((d, k, t)\)-CSP instance \( P \) is called satisfiable if there exists a satisfying assignment of \( P \); otherwise, it is called unsatisfiable.

Definition 2.42 (Constraint Hypergraph). Define the constraint hypergraph of a \((d, k, t)\)-CSP instance \( P \) to be a hypergraph \( H \) which takes the variable set of \( P \) as the set of vertices, and takes the constraint set of \( P \) as the set of hyperedges.

Definition 2.43 (Constraint Density or Edge Density). The constraint density (or edge density) of a \((d, k, t)\)-CSP instance \( P \) is defined to be \( |E(P)|/|V(P)| \).

Definition 2.44 (\( \Delta \)-sparse). A \((d, k, t)\)-CSP instance is called \( \Delta \)-sparse if it contains no subproblem with constraint density higher than \( \Delta \).

Definition 2.45 ((Tree) Resolution on \((d, k, t)\)-CSP Instances). For any \((d, k, t)\)-CSP instance \( P \), a (tree) resolution derivation of \( P \) is a (tree) resolution derivation of \( CNF(P) \), which is the CNF formula constructed from \( P \) by the C-RES procedure.
Definition 2.46 (Clause Derived and Minimally Derived). We say that a clause \( C \) is derived from a \((d,k,t)\)-CSP instance \( P \) if there exists a resolution derivation of \( C \) from \( CNF(P) \). The clause \( C \) is minimally derived from \( P \) if it is derived from \( P \) and it is not derived from any subproblem of \( P \).

2.6.3 Random \((d,k,t)\)-CSP Models

Definition 2.47 (With High Probability (w.h.p.)). Given an event \( E = E_n \) depending on parameter \( n \), it is said to occur with high probability (w.h.p.) if \( \lim_{n \to \infty} Pr[E] = 1 \).

Definition 2.48 (With Uniformly Positive Probability (w.u.p.p.)). Given an event \( E = E_n \) depending on parameter \( n \), it is said to occur with uniformly positive probability (w.u.p.p.) if \( \lim \inf Pr[E] > 0 \).

Definition 2.49 (Random \((d,k,t)\)-CSP Model, \( CSP_{n,p}^{d,k,t} \)). Given parameters \((d,k,t,n,p)\), a random \((d,k,t)\)-CSP instance with \( n \) variables is generated as follows: (i) choose a random \( k \)-uniform hypergraph with \( n \) vertices to be the constraint hypergraph, by picking each of the possible hyperedges independently with edge probability \( p \); (ii) for each hyperedge \( e \) in the hypergraph, choose with uniform probability a constraint from the set of all \((d^k/t)\) possible constraints. Denote this random model by \( CSP_{n,p}^{d,k,t} \).

Definition 2.50 (Constraint Probability or Edge Probability). The parameter \( p \) in the \( CSP_{n,p}^{d,k,t} \) model is called the constraint probability or the edge probability.

Definition 2.51 (Another Random \((d,k,t)\)-CSP Model \( CSP_{n,m}^{d,k,t} \)). Given parameters \((d,k,t,n,m)\), a random \((d,k,t)\)-CSP instance is generated by choosing with uniform probability a CSP instance from the set of all \((d,k,t)\)-CSP instances with \( n \) variables and \( m \) constraints. Denote this random model by \( CSP_{n,m}^{d,k,t} \).
Chapter 3

Our Results

In this thesis, we study the resolution complexity and the tree resolution complexity of the random $(d, k, t)$-CSP model $\text{CSP}_{d,k,t}^{n,m}$ with super-constant constraint density. In particular, we are interested in the ranges of constraint density where the resolution and tree resolution complexity drop from superpolynomial to polynomial.

Our Setting

We consider the random $(d, k, t)$-CSP model $\text{CSP}_{d,k,t}^{n,m}$ with constants $d, k \geq 2$ since the model is trivial when either $d = 1$ or $k = 1$. By Theorem 2.19 and Theorem 2.20 [2, 64], when $t \geq (d - 1)d^{k-2}$ and the constraint density is a sufficiently large constant, w.h.p. a random $(d, k, t)$-CSP instance has at most polynomial resolution complexity. These results also imply that w.h.p. a random $(d, k, t)$-CSP instance has at most polynomial tree resolution complexity as well. Hence, we will focus on the settings with $1 \leq t < (d - 1)d^{k-2}$ (see also Remark 2.24).

Note that our random model generalizes random $k$-SAT for $k \geq 3$, but not random 2-SAT - the parameters are $d, k = 2$ and $t = 1$ for 2-SAT, which implies $t = (d - 1)d^{k-2}$, so random 2-SAT is generalized by the random $(d, k, t)$-CSP model with constants $d, k \geq 2$ and $t \geq (d - 1)d^{k-2}$.

Finally, recall that we define the resolution complexity and the tree resolution complexity of satisfiable instances to be $\infty$.

New Results

We introduce a general approach for studying the resolution complexity and the tree resolution complexity of random $(d, k, t)$-CSP, for any constant triple $(d, k, t)$ with $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$. We conjecture that there exists a constant $\Lambda(d, k, t)$ such that

Conjecture 3.1 (Upper Bound). For any positive constants $d, k, t$ and $\epsilon$ with $1 \leq t < (d - 1)d^{k-2}$
Chapter 3. Our Results

and \(d, k \geq 2\), when the constraint density \(\Delta = n^{k-1 - \frac{1}{\Lambda(d,k,t)}} + \epsilon\), w.h.p. a random instance drawn from \(\text{CSP}_{n,m=\Delta n}^{d,k,t}\) has constant resolution complexity and constant tree resolution complexity.

**Conjecture 3.2** (Tree Resolution Complexity Lower Bound). For any positive constants \(d, k, t\) and \(\epsilon\) with \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\), when the constraint density \(\Delta = n^{k-1 - \frac{1}{\Lambda(d,k,t)}} - \epsilon\), there exists a constant \(\gamma > 0\) such that w.h.p. a random instance drawn from \(\text{CSP}_{n,m=\Delta n}^{d,k,t}\) has tree resolution complexity \(\exp(\Omega(n^\gamma))\).

**Conjecture 3.3** (Resolution Complexity Lower Bound). For any positive constants \(d, k, t\) and \(\epsilon\) with \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\), when the constraint density \(\Delta = n^{k-1 - \frac{1}{\Lambda(d,k,t)}} - \epsilon\), there exists a constant \(\gamma > 0\) such that w.h.p. a random instance drawn from \(\text{CSP}_{n,m=\Delta n}^{d,k,t}\) has resolution complexity \(\exp(\Omega(n^\gamma))\).

We successfully prove these conjectures in the setting with \(k = 2\). In other words, we determine the value of a constant \(\Lambda(d,k,t)\) such that

**Theorem 3.4** (Upper Bound). For any positive constants \(d, k, t\) and \(\epsilon\) with \(1 \leq t < d - 1\) and \(k = 2\), when the constraint density \(\Delta = n^{k-1 - \frac{1}{\Lambda(d,k,t)}} + \epsilon\), w.h.p. a random instance drawn from \(\text{CSP}_{n,m=\Delta n}^{d,k,t}\) has constant resolution complexity and constant tree resolution complexity.

**Theorem 3.5** (Tree Resolution Complexity Lower Bound). For any positive constants \(d, k, t\) and \(\epsilon\) with \(1 \leq t < d - 1\) and \(k = 2\), when the constraint density \(\Delta = n^{k-1 - \frac{1}{\Lambda(d,k,t)}} - \epsilon\), there exists a constant \(\gamma > 0\) such that w.h.p. a random instance drawn from \(\text{CSP}_{n,m=\Delta n}^{d,k,t}\) has tree resolution complexity \(\exp(\Omega(n^\gamma))\).

Together, the **tree resolution complexity** is w.h.p. \(\exp(\Omega(n^\gamma))\) when the constraint density is \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}} - \epsilon\), and is w.h.p. constant when the constraint density is \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}} + \epsilon\). In other words, the tree resolution complexity drops from superpolynomial to polynomial in a narrow range between \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}} - \epsilon\) and \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}} + \epsilon\). These bounds are **tight up to a \(o(1)\) term in the exponent**, and they nearly match the best known bounds for the random \(k\)-SAT model - the best known bounds for random \(k\)-SAT are also tight up to a \(o(1)\) term in the exponent (Theorem 2.16 and 2.17).

**Theorem 3.6** (Resolution Complexity Lower Bound). For any positive constants \(d, k, t\) and \(\epsilon\) with \(1 \leq t < d - 1\) and \(k = 2\), when the constraint density \(\Delta = n^{k-1 - \frac{1}{\Lambda(d,k,t)}} - \epsilon\), there exists a constant \(\gamma > 0\) such that w.h.p. a random instance drawn from \(\text{CSP}_{n,m=\Delta n}^{d,k,t}\) has resolution complexity \(\exp(\Omega(n^\gamma))\).

Together, the **resolution complexity** is w.h.p. \(\exp(\Omega(n^\gamma))\) when the constraint density is \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}} - \epsilon\), and is w.h.p. constant when the constraint density is \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}} + \epsilon\). In other words, the tree resolution complexity drops from superpolynomial to polynomial when the constraint density grows from \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}} - \epsilon\) to \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}} + \epsilon\). There is a gap of \(k-2 + o(1)\) in the exponent, and these bounds are similar to the best known bounds for random \(k\)-SAT - there is a gap of \(k-2 + o(1)\) in the exponent for random \(k\)-SAT (Theorem 2.16 and 2.17).
We get looser bounds for general resolution because the width-length relation we apply in our proof gives a weaker lower bound for resolution complexity. Recall that the width-length relations by Ben-Sasson and Wigderson [24] are \( w(F \vdash 0) \leq w(F) + \log \text{RES}_T(F) \) (Theorem 2.1) and \( w(F \vdash 0) \leq w(F) + O(\sqrt{n\ln \text{RES}(F)}) \) (Theorem 2.2). The lower bound on resolution complexity \( \text{RES}(F) \) is significantly weaker than the lower bound on tree resolution complexity \( \text{RES}_T(F) \).

Apart from the setting (I) with \( k = 2 \), we also applied the same approach to prove the conjectures in two other settings:

(II) With \( t = 1 \), the random \((d, k, t)\)-CSP model is a generalization of the random \( k \)-SAT model by allowing a more general domain of \( d \geq 2 \) variable values instead of exactly 2. In this setting, we determined \( \Lambda(d, k, t) = d - 1 \). Note that when \( d = 2 \), it is the setting of random \( k \)-SAT, and we have \( \Lambda(d, k, t) = 1 \). With \( \Lambda(d, k, t) = 1 \), the gaps in the exponent become \( o(1) \) and \( \frac{k-2}{2} + o(1) \) respectively, which are the same as the best known results for the random \( k \)-SAT model.

(III) With \( d = 2 \), the random \((d, k, t)\)-CSP model is a generalization of the random \( k \)-SAT model by allowing \( t \geq 1 \) restrictions in each constraint instead of exactly one. In this setting, we determined \( \Lambda(d, k, t) = \frac{1}{\log_{d}(t)+1} \). Similarly, when \( t = 1 \), it is the setting of random \( k \)-SAT, and we have \( \Lambda(d, k, t) = 1 \). Again, the gaps in the exponent become \( o(1) \) and \( \frac{k-2}{2} + o(1) \), matching those for the random \( k \)-SAT model. Therefore, these new results can be regarded as generalizations of the resolution complexity results for random \( k \)-SAT with \( k \geq 3 \).

In the following chapters, we will first present our approach for studying the resolution complexity of random \((d, k, t)\)-CSP in Chapter 4. Then, we will see how to apply this approach to the setting with \( 1 \leq t < (d-1)d^{k-2} \) and \( k = 2 \) in Chapter 5 and Chapter 6. Finally, we will briefly discuss the applications of this approach to other models of random CSPs in Chapter 7. The proofs of the conjectures in settings (II) and (III), i.e. with either \( t = 1 \) or \( d = 2 \), are included in Appendix C and D.

It is well known that any property holding w.h.p. in a random instance drawn from \( \text{CSP}_{d,k,t}^{d,k,t} \) will also hold w.h.p. in a random instance drawn from \( \text{CSP}_{n,p}^{d,k,t} \) with some \( p = m/M \) (e.g. see the proofs of Propositions 1.12 and 1.15 in [58]). For completeness, we will include the equivalent statements of Conjecture 3.1, 3.2 and 3.3 and Theorem 3.4, 3.5 and 3.6 for the model \( \text{CSP}_{n,p}^{d,k,t} \) in Appendix E.

As a remark, we also proved the conjectures for (IV) the general case with \( 1 \leq t < (d-1)d^{k-2} \) and \( d, t \geq 2 \). However, the proofs are significantly more complicated and are very messy. We are still trying to simplify the proofs, so these results and proofs are not included in this thesis.
Chapter 4

A General Approach

We introduce an approach for studying the resolution complexity and the tree resolution complexity of the random \((d, k, t)\)-CSP model \(\text{CSP}^{d,k,t}_{n,m}\) with constants \(d, k, t\) such that \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\).

4.1 A Reduction to a Non-Random Problem

We will prove lower and upper bounds stated in Theorem 3.4, 3.5 and 3.6 by solving the following non-random problem.

- Key Problem: How sparse can an unsatisfiable \((d, k, t)\)-CSP instance be?

This non-random problem is of independent interest, as we discussed in Chapter 1. Formally, the problem asks what is the infimum of the range of \(\lambda\) such that there exists a \(\lambda\)-sparse unsatisfiable \((d, k, t)\)-CSP instance.

Let \(\Lambda(d, k, t)\) be the solution of this problem. The necessary and sufficient conditions for \(\Lambda(d, k, t)\) to be the infimum are (i) for every \(\lambda > \Lambda(d, k, t)\), there exists a \(\lambda\)-sparse unsatisfiable instance; and (ii) for every \(\lambda < \Lambda(d, k, t)\), every \(\lambda\)-sparse instance is satisfiable.

In the following, we will see how to apply some standard techniques to derive the lower and upper bounds stated in our theorems. Let \(P\) be a random instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\).

A Tree Resolution Complexity Lower Bound. The width-length relations by Ben-Sasson and Wigderson [24] provide a unified framework for proving lower bounds on resolution and tree resolution resolution complexity from a refutation width lower bound. Roughly speaking, the tree resolution complexity is w.h.p. \(\exp(\Omega(n^{\gamma}))\) if w.h.p. every \(O(n^{\gamma})\)-size subproblem of \(P\) is satisfiable, which can be proved by showing that (I) w.h.p. every \(O(n^{\gamma})\)-size subproblem of \(P\) is \((\Lambda(d, k, t) - \sigma)\)-sparse, and
Chapter 4. A General Approach

(II) every \((\Lambda(d,k,t) - \sigma)\)-sparse \((d,k,t)\)-CSP instance is satisfiable (not just random instances). The former can be proved by some standard arguments using the first moment method, when the constraint density of the given random model is \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}}\). The latter follows directly from the definition of \(\Lambda(d,k,t)\). By the width-length relation by Ben-Sasson and Wigderson, these prove that the tree resolution complexity is w.h.p. \(\exp(\Omega(n^{\gamma}))\) when the constraint density is at most \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}}\). The lower bound for resolution complexity can also be proved in the same manner.

A Tree Resolution Complexity Upper Bound. Recall that \(\Lambda(d,k,t)\) is the infimum of the range of \(\Lambda\) such that there exists a \(\Lambda\)-sparse unsatisfiable \((d,k,t)\)-CSP instance. For every constant \(\mu > 0\), there exists a fixed \((\Lambda(d,k,t) + \mu)\)-sparse unsatisfiable \((d,k,t)\)-CSP instance \(G\) (i.e. one with size independent of \(n\) and \(m\)). By some standard arguments using the second moment method, we can show that when the constraint density is \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}}\), there exists some constant \(\mu\), such that w.h.p. the \((\Lambda(d,k,t) + \mu)\)-sparse \((d,k,t)\)-CSP instance \(G\) appears in \(P\) as a subproblem. Since \(G\) has constant size, we can obtain a short tree resolution refutation easily by brute force. Therefore, the tree resolution complexity is w.h.p. constant when the constraint density is at least \(n^{k-1 - \frac{1}{\Lambda(d,k,t)}}\), and so is the resolution complexity.

Therefore, we are proving resolution complexity results for random CSPs by studying some basic properties of non-random CSP instances. Specifically, we are solving our Key Problem of determining the infimum \(\Lambda(d,k,t)\) of the range of \(\Lambda\) such that there exists a \(\Lambda\)-sparse unsatisfiable \((d,k,t)\)-CSP instance.

Our Approach. We solve our Key Problem as follows. First, we construct a special \(\Lambda(d,k,t)\)-sparse \((d,k,t)\)-CSP instance \(H^*\), which is a generalization of the forcer edge by Molloy and Salavatipour [64]. For any constant \(\mu > 0\), by using \(H^*\) as the building blocks, we are able to construct a \((\Lambda(d,k,t) + \mu)\)-sparse unsatisfiable \((d,k,t)\)-CSP instance \(G_\ell\). Then, we prove that every \(\Lambda(d,k,t)\)-sparse \((d,k,t)\)-CSP instance is satisfiable. Together, these imply that \(\Lambda(d,k,t)\) is the infimum specified in our Key Problem. Then, the lower and upper bounds on the resolution and tree resolution complexity follow directly.

Thus the first step of our approach is to identify the special \((d,k,t)\)-CSP instance \(H^*\).

After identifying the structure of \(H^*\), another important technical work in this thesis is to prove that every \(\Lambda(d,k,t)\)-sparse \((d,k,t)\)-CSP instance is satisfiable. Note that for \(k\)-SAT, which is a special case of \((d,k,t)\)-CSP, a similar statement can be proved by some arguments using matching (e.g. see [24]). However, with a more general domain of variable values, and with multiple restrictions per constraint, those arguments do not work for \((d,k,t)\)-CSP.

In this thesis, we will prove that every \(\Lambda(d,k,t)\)-sparse \((d,k,t)\)-CSP instance is satisfiable by transforming a \((d,k,t)\)-CSP instance into a Token Allocation Problem, which is a combinatorial problem we create for the purpose of this proof (see Section 4.4). We will show that there exists a solution of the corresponding token allocation problem if the given CSP instance is \(\Lambda(d,k,t)\)-sparse. Then, we will present an algorithm that produces a satisfying assignment on the CSP instance from a solution of the corresponding token allocation problem. Together, these prove that every \(\Lambda(d,k,t)\)-sparse \((d,k,t)\)-CSP
instance has a satisfying assignment.

In the following, we will first define the special \((d, k, t)\)-CSP instance \(H^*\) in the next section. Then, we will outline the proofs of our main results in Section 4.3. The details regarding the Token Allocation Problem will be presented in Section 4.4.

### 4.2 The Special \((d, k, t)\)-CSP instance \(H^*\)

Generalizing the “snakes” by Chvátal and Reed [36], Molloy and Salavatipour [64] used a special type of constraints, called forcer edges, to construct a small unsatisfiable forbidding flower. Recall that a forcer edge must contain at least \((d - 1)d^{k-2}\) restrictions, which is impossible in our setting where \(1 \leq t < (d - 1)d^{k-2}\). Thus, we have to extend the idea of forcer edges significantly to more general forcer CSP instances - instead of using a single constraint, we will use a set of constraints and variables (i.e. a CSP instance) to play the role of forcer edge.

The key property of forcers is that: given a set of variables with fixed values, we can use forcers to fix the values on another set of variables.

**Definition 4.1** \((I : A_I) \rightarrow (u : \delta)\) Forcer). Define an \((I : A_I) \rightarrow (u : \delta)\) forcer to be a \((d, k, t)\)-CSP instance \(H\) containing (i) an input variable set \(I\), (ii) an output variable \(u\), and (iii) possibly some other variables and constraints such that there is no satisfying assignment in which \(I\) is assigned values \(A_I\) and \(u\) is assigned any value other than \(\delta\). Like any CSP instance, denote the variable set and constraint set by \(V(H)\) and \(E(H)\). Furthermore, denote the input variable set and output variable by \(I(H)\) and \(u(H)\). (For clarification, \(I(H) \cup \{u(H)\} \subseteq V(H)\).)

In other words, after fixing \(A_I\) at \(I\), an \((I : A_I) \rightarrow (u : \delta)\) forcer forces us to assign \(\delta\) to \(u\).

We require a different measure of density on forcers. Since the primary usage of forcers is to act as building blocks in the construction of a forbidding flower, we want a measure to reflect the density of the forbidding flower they appear in, instead of the forcers themselves.

**Definition 4.2** (Modified Constraint Density). Define the modified constraint density of a forcer \(H\) to be \(|E(H)|/(|V(H)| - |I(H)|)\).

The CSP instance \(H^*\) is a forcer with a special structure. In particular, we want the modified constraint density of \(H^*\) to be \(\Lambda(d, k, t)\). Then, by using \(H^*\) as the building blocks, we can construct unsatisfiable instances with constraint densities arbitrarily close to \(\Lambda(d, k, t)\). We will see the constructions of those unsatisfiable instances in the next section. For the setting with \(k = 2\), the detailed structure of \(H^*\) will be presented in Chapter 5.

**Definition 4.3** \((\Lambda(d, k, t))\). Define \(\Lambda(d, k, t) = |E(H^*)|/(|V(H^*)| - |I(H^*)|)\).
4.3 Proofs of the Upper and Lower Bounds

We consider the random \((d, k, t)\)-CSP model \(\text{CSP}^{d,k,t}_{n,m}\) with any constants \(d, k, t\) such that \(1 \leq t < (d - 1)d^{k-2}\) and \(k \geq 2\). In this section, we will describe the proofs for Conjecture 3.1, 3.2 and 3.3. Conjecture 3.1 states an upper bound for both resolution complexity and tree resolution complexity; while Conjecture 3.2 and 3.3 state lower bounds for tree resolution complexity and resolution complexity respectively.

The following are just outlines of the proofs since we need the detailed structure of \(H^*\) to complete the proofs. These outlines hold for all constants \(d, k, t\) with \(1 \leq t < (d - 1)d^{k-2}\) and \(k \geq 2\). The structure of \(H^*\) and the exact value of \(\Lambda(d, k, t)\) for the setting with \(k = 2\) will be presented in Chapter 5.

4.3.1 A Tree Resolution Complexity Upper Bound

In this section, we will describe our approach for proving an upper bound on the tree resolution complexity of random \((d, k, t)\)-CSP instances. For the setting with \(k = 2\), the result is stated in Theorem 3.4.

For any positive constant \(\mu\), we will define an unsatisfiable \((d, k, t)\)-CSP instance \(G_\ell\), which is \((\Lambda(d, k, t) + \mu)\)-sparse and has constant size. Then, we will prove that for any constant \(\epsilon > 0\), there exists a pair of positive constants \(\mu\) and \(\ell\) such that when the constraint density is \(n \frac{k - 1}{\Lambda(d, k, t) + \epsilon}\), w.h.p. a random \((d, k, t)\)-CSP instance contains \(G_\ell\) as a subproblem. We can refute this constant size subproblem in constant time by brute force. Thus, both the resolution complexity and the tree resolution complexity are constants.

We will start by describing how to use the special forcer \(H^*\) to construct the unsatisfiable instance \(G_\ell\). Our forcer is a generalization of the forcer edge by Molloy and Salavatipour [64]. It is not surprising that we can construct a small unsatisfiable CSP instance by generalizing their definitions of forcing path, forbidding cycle and forbidding flower.

First, given a set of variables \(I\) with fixed values, we can fix the values on another set of \(|I|\) variables by using \(|I|\) forcers.

**Definition 4.4 (Forcing Block).** Define a forcing block to be a \((d, k, t)\)-CSP instance \(B\) consisting of \(|I|\) forcers: an \((I : A_I) \rightarrow (u_i : \delta_i)\) forcer \(H_i\), for \(i = 0, 1, ..., |I| - 1\). The forcers do not share any variable and constraint, except for the input variable set \(I\). Denote by \(U = \{u_1, ..., u_{|I|}\}\) the output variable set, and by \(A_U = \{u_1 : \delta_1, ..., u_{|I|} : \delta_{|I|}\}\) the forced value assignment on \(U\). This CSP instance is called an \((I : A_I) \rightarrow (U : A_U)\) forcing block. The input and output variable sets are referred to as \(I(B)\) and \(U(B)\) respectively.

In other words, after fixing \(A_I\) at \(I\), an \((I : A_I) \rightarrow (U : A_U)\) forcing block will forces us to assign \(A_U\) to \(U\).

Then, given a set of variables \(I\) with fixed values, we can use \(\ell\) different forcing blocks to construct a sparse CSP instance with size \(\Theta(\ell)\) to fix the values on \(\ell\) other sets of variables with size \(|I|\).

**Definition 4.5 (Forcing Path).** Define a forcing path to be a \((d, k, t)\)-CSP instance consisting of \(\ell\) forcing
blocks: an \((X_i : A_i) \rightarrow (X_{i+1} : A_{i+1})\) forcing block for \(i = 1, 2, \ldots, \ell\). The forcing blocks do not share any variable and constraint, except that \(X_{i+1}\) is shared by the \((X_i : A_i) \rightarrow (X_{i+1} : A_{i+1})\) forcing block and the \((X_{i+1} : A_{i+1}) \rightarrow (X_{i+2} : A_{i+2})\) forcing block, for \(i = 1, \ldots, \ell - 1\). This CSP instance is called an \((X_1 : A_1) \rightarrow (X_{\ell+1} : A_{\ell+1})\) forcing path.

If we make a forcing path into a cycle, then we get a cycle forcing \((X_1 : A_1) \rightarrow (X_{\ell+1} : A_{\ell+1})\). If \(A_1 \neq A_{\ell+1}\), then this cycle forbids us from assigning \(A_1\) to \(X_1\).

**Definition 4.6** (Forbidding Cycle). Define a forbidding cycle to be a \((d, k, t)\)-CSP instance consisting of an \((X_1 : A_1) \rightarrow (X_{\ell} : A_{\ell})\) forcing path and an \((X_{\ell} : A_{\ell}) \rightarrow (X_1 : A_1)\) forcing block for some \(A_{\ell+1} \neq A_1\). The forcing path and forcing block do not share any variable and constraint, except for variable sets \(X_1\) and \(X_\ell\). This CSP instance is called an \(X_1 \sim A_1\) forbidding cycle.

By placing \(d^{|I|}\) appropriate forbidding cycles at a variable set \(I\), we can forbid all the \(d^{|I|}\) possible value assignments from being assigned to \(I\).

**Definition 4.7** (Forbidding Flower). Define a forbidding flower to be a \((d, k, t)\)-CSP instance consisting of \(|I|\) forbidding cycles: an \(I \sim A_I\) forbidding cycle for each of the \(d^{|I|}\) possible value assignments \(A_I\) on a variable set \(I\). The forbidding cycles do not share any variable and constraint, except for variable set \(I\), which is referred to as \(I(G)\).

By the definition of \((d, k, t)\)-CSPs, we are allowed to use all the \(\binom{d^k}{t}\) possible types of constraints to construct a \((d, k, t)\)-CSP instance. We will see in Claim 4.9 stated below that given any \((I : A_I) \rightarrow (u : \delta)\) forcer, we can change some restrictions in the constraints to obtain an \((I : A_I') \rightarrow (u : \delta')\) forcer for any \(|I|\)-tuples of values \(A_I'\) and any value \(\delta'\). Thus, we can construct a forbidding flower by using only forcers structurally similar to \(H^*\).

**Definition 4.8.** Two CSP instances are said to be structurally similar if their constraint hypergraphs are isomorphic to each other.

**Claim 4.9.** For any tuple of values \(A_I'\) and any value \(\delta'\), there exists an \((I : A_I') \rightarrow (u : \delta')\) forcer \(H\) having the same constraint hypergraph and modified constraint density as \(H^*\).

**Proof.** Start with the \((I : A_I) \rightarrow (u : \delta)\) forcer \(H^*\). We can modify the restrictions in the constraints to obtain an \((I : A_I') \rightarrow (u : \delta')\) forcer for any \(A_I'\) and \(\delta'\) by the following procedure. Since the following procedure only changes the restrictions in the constraints, the constraint hypergraph and the modified constraint density remain the same. Let \(I = \{x_1, \ldots, x_{|I|}\}\), and \(A_I = \{x_1 : \gamma_1, \ldots, x_{|I|} : \gamma_{|I|}\}\) and \(A_I' = \{x_1 : \gamma_1', \ldots, x_{|I|} : \gamma_{|I|}'\}\).

(1) For any restriction \(R\) in \(H^*\), if \(x_i : \gamma_i\) appears in \(R\), then change the restriction by replacing \(x_i : \gamma_i\) with \(x_i : \gamma_i'\); on the other hand, if \(x_i : \gamma_i'\) appears in \(R\), then change the restriction by replacing \(x_i : \gamma_i\) with \(x_i : \gamma_i'\). This transforms an \((I : A_I) \rightarrow (u : \delta)\) forcer into an \((I : A_I') \rightarrow (u : \delta)\) forcer.

(2) For any restriction \(R\) in \(H^*\), if \(u : \delta\) appears in \(R\), then change the restriction by replacing \(u : \delta\) with \(u : \delta'\); on the other hand, if \(u : \delta'\) appears in \(R\), then change the restriction by replacing \(u : \delta\) with \(u : \delta'\). This transforms an \((I : A_I') \rightarrow (u : \delta)\) forcer into an \((I : A_I') \rightarrow (u : \delta')\) forcer. □
Definition 4.10 (A Family of Special Forbidding Flowers \( \mathcal{G}^* \)). For any integer \( \ell \geq 2 \), define \( G_\ell \) to be a forbidding flower such that (i) every forcer in \( G_\ell \) is structurally similar to \( H^* \); (ii) each forbidding cycle has length \( \ell \); (iii) each forcing block satisfies Property 4.11 stated below. Denote the family of all these forbidding flowers by \( \mathcal{G}^* = \{ G_\ell \} \).

Property 4.11. For each forcing block \( B \) in \( G_\ell \), denote the constraint hypergraphs of the forcers by \( \mathcal{H} = \{ H_0, H_1, \ldots, H_{|I(B)|-1} \} \), and the vertices corresponding to the input variable set by \( \{ v_0, v_1, \ldots, v_{|I(B)|-1} \} \). Recall that the hypergraphs in \( \mathcal{H} \) are isomorphic to each other. Let \( f_i \) be the bijective function between \( H_0 \) and \( H_i \) for \( i = 1, \ldots, |I(B)| - 1 \). The forcers in \( B \) are arranged in a symmetric way such that 

\[
 v_j = f_i(v_{(i+j) \mod |I|})
\]

This property ensures that no dense subproblem arises due to bad arrangement of forcers in a forcing block.

Consider a random instance \( P \) drawn from CSP\(_{d,k,t}^{n,m}\) with any constants \( d, k, t \) such that \( 1 \leq t < (d - 1)d^{k-2} \) and \( k \geq 2 \), and with constraint density \( n^{k-1 - \frac{1}{\Lambda(d,k,t)^{+}\epsilon}} \). By using the second moment method, we can show that for any sufficiently sparse constant size \( (d, k, t) \)-CSP instance, w.h.p. it appears in \( P \) as a subproblem. The proof is included in Section A.3.

Lemma 4.12. There exists a constant \( \mu > 0 \) such that for any constant size \( (\Lambda(d,k,t) + \mu) \)-sparse \( (d, k, t) \)-CSP instance, w.h.p. it appears in \( P \) as a subproblem.

Now, we can describe our proof for the upper bound on tree resolution complexity. Suppose we can show that (I) for any constant integer \( \ell \geq 2 \), \( G_\ell \in \mathcal{G}^* \) has constant numbers of variables and constraints; and (II) for every positive constant \( \mu \), there exists a constant \( \ell \) such that \( G_\ell \in \mathcal{G}^* \) is \( (\Lambda(d,k,t) + \mu) \)-sparse.

For any constant \( \mu > 0 \), properties (I) and (II) imply that there exists an unsatisfiable \( (d, k, t) \)-CSP instance \( G_\ell \) that is \( (\Lambda(d,k,t) + \mu) \)-sparse and has constant size. By Lemma 4.12, w.h.p. such a \( G_\ell \) appears in a random instance drawn from CSP\(_{d,k,t}^{n,m}\). Since \( G_\ell \) has constant size, it can be refuted easily in constant time. Therefore, when the constraint density is \( n^{k-1 - \frac{1}{\Lambda(d,k,t)^{+}\epsilon}} \), w.h.p. both the resolution complexity and tree resolution complexity are constant. This proves the tree resolution complexity upper bound stated in Conjecture 3.1, contingent on providing proofs of properties (I) and (II), and Lemma 4.12.

For the setting with \( k = 2 \), we will see the detailed structure of \( H^* \) and prove properties (I) and (II) as Claim 5.2 and Claim 5.3 in Chapter 5. With the proof of Lemma 4.12 shown in Appendix A.3, this gives the upper bound result stated in Theorem 3.4.

### 4.3.2 A Tree Resolution Complexity Lower Bound

In this section, we will describe our approach for proving a lower bound on the tree resolution complexity of random \( (d, k, t) \)-CSP instances. Consider the random \( (d, k, t) \)-CSP model CSP\(_{d,k,t}^{n,m}\) with any constant triple \( (d, k, t) \) such that \( 1 \leq t < (d - 1)d^{k-2} \) and \( k \geq 2 \), and with constraint density \( n^{k-1 - \frac{1}{\Lambda(d,k,t)^{+}\epsilon}} \) for any constant \( \epsilon > 0 \). We want to show that w.h.p. a random \( (d, k, t) \)-CSP instance drawn from this model has tree resolution complexity \( \exp(\Omega(n^{\gamma})) \) for some constant \( \gamma > 0 \). For the setting with \( k = 2 \), the result is stated in Theorem 3.5.
We will prove the lower bound by using the width-length relation stated in Theorem 2.1. Let $P$ be a CSP instance drawn from the random model. All we need to do is to prove that w.h.p. for every resolution refutation of $P$, there is a clause with $\Theta(n^\gamma)$ variables for some positive constant $\gamma$. The proof is based on the following three lemmas:

**Lemma 4.13.** There exist positive constants $\gamma$ and $\sigma$ such that w.h.p. every subproblem of $P$ with $O(n^\gamma)$ variables is $(\Lambda(d, k, t) - \sigma)$-sparse.

**Lemma 4.14.** Every $(d, k, t)$-sparse $(d, k, t)$-CSP instance is satisfiable.

**Lemma 4.15.** For any positive constant $\sigma$, if a $(d, k, t)$-CSP instance with $n'$ variables is $(\Lambda(d, k, t) - \sigma)$-sparse, then every clause minimally derived from this CSP instance must contain $\Theta(n')$ variables.

We will prove Lemma 4.13 by applying some standard arguments using the first moment method, which is included in Appendix A.1. As mentioned in Section 4.1, proving that every $(d, k, t)$-sparse $(d, k, t)$-CSP instance is satisfiable, i.e. Lemma 4.14, is one of the main technical works in this thesis. We will prove Lemma 4.14 and 4.15 by translating the $(d, k, t)$-CSP instance into a Token Allocation Problem. We will see the details in Section 4.4.

Now, we will prove a tree resolution complexity lower bound by using these three lemmas and a result by Mitchell [63]:

**Lemma 4.16** (Lemma 1 in [63]). Given a $(d, k, t)$-CSP instance $P$, if every subproblem with $O(n')$ variables is satisfiable, then there is a clause $C$ in every resolution refutation of $P$ such that $C$ is minimally derived from a subproblem with $\Theta(n')$ variables.

Consider a $(d, k, t)$-CSP instance $P$ drawn from the random model, and a resolution refutation of $P$. Lemma 4.13 and Lemma 4.14 imply that w.h.p. every subproblem with $O(n^\gamma)$ variables is satisfiable. By Lemma 4.16, this implies that w.h.p. there is a clause $C$ in the refutation such that $C$ is minimally derived from a subproblem with $\Theta(n')$ variables. By Lemma 4.13, such a small subproblem is $(\Lambda(d, k, t) - \sigma)$-sparse. Then by Lemma 4.15, the clause $C$ has $\Theta(n')$ variables. By the width-length relation stated in Theorem 2.1, w.h.p. the tree resolution complexity is $\exp(O(n'))$ when the constraint density is $n^{k-1} \frac{1}{\Lambda(d, k, t)} - \epsilon$. This proves the tree resolution complexity lower bound stated in Conjecture 3.2, contingent on providing proofs of Lemma 4.13, 4.14 and 4.15. For the setting with $k = 2$, this result is stated in Theorem 3.5.

### 4.3.3 A Resolution Complexity Lower Bound

In this section, we will describe our approach for proving a lower bound on the resolution complexity of random $(d, k, t)$-CSP instances. Consider the random $(d, k, t)$-CSP model $\text{CSP}_{n,m}^{d,k,t}$ with any constant triple $(d, k, t)$ such that $1 \leq t < (d - 1)d^{k-2}$ and $k \geq 2$, and with constraint density $n^{k-1} \frac{1}{\Lambda(d, k, t)} - \epsilon$ for any constant $\epsilon > 0$. We want to show that w.h.p. a random $(d, k, t)$-CSP instance drawn from this model has resolution complexity $\exp(O(n^\gamma))$ for some constant $\gamma > 0$. For the setting with $k = 2$, the result is stated in Theorem 3.6.
Chapter 4. A General Approach

The proof is very similar to the one presented in the previous section. The only differences are (i) the constraint density in the random model is $n^{k-1-\frac{1}{2}\Lambda(d,k,t) - \epsilon}$ instead of $n^{k-1-\frac{1}{2}\Lambda(d,k,t) - \epsilon}$, (ii) we use Lemma 4.17 stated below instead of Lemma 4.13, and (iii) we use the width-length relation stated in Theorem 2.2, instead of Theorem 2.1.

Lemma 4.17. There exist positive constants $\gamma$ and $\sigma$ such that w.h.p. every subproblem of $P$ with $O(n^{0.5+\gamma})$ variables is $(\Lambda(d,k,t) - \sigma)$-sparse.

We will prove Lemma 4.17 by applying some standard arguments using the first moment method, which is included in Appendix A.2.

Consider a $(d,k,t)$-CSP instance $P$ drawn from the random model, and a resolution refutation of $P$. Lemma 4.17 and Lemma 4.14 imply that w.h.p. every subproblem with $O(n^{0.5+\gamma})$ variables is satisfiable. By Lemma 4.16, this implies that w.h.p. there is a clause $C$ in the refutation such that $C$ is minimally derived from a subproblem with $\Theta(n^{0.5+\gamma})$ variables. By Lemma 4.13, such a small subproblem is $(\Lambda(d,k,t) - \sigma)$-sparse. Then by Lemma 4.15, the clause $C$ has $\Theta(n^{0.5+\gamma})$ variables. By the width-length relation stated in Theorem 2.2, w.h.p. the tree resolution complexity is $\exp(\Omega(n^\gamma))$ when the constraint density is $n^{k-1-\frac{1}{2}\Lambda(d,k,t) - \epsilon}$. This proves the resolution complexity lower bound stated in Conjecture 3.3, contingent on providing proofs of Lemma 4.17, 4.14 and 4.15. For the setting with $k = 2$, this result is stated in Theorem 3.6.

4.4 The Token Allocation Problem

As mentioned in the previous section, we will prove Lemma 4.14 and Lemma 4.15 by translating the $(d,k,t)$-CSP instance into a Token Allocation Problem. First, we will try to explain the intuition behind the Token Allocation Problem. Consider the following trivial algorithm (shown in Figure 4.1) for finding a satisfying assignment on a given $k$-SAT instance. Denote the set of variables and clauses by $V$ and $E$.

$k$-SAT($I = (V, E)$)

1. Initialize $U = V$.
2. For each clause $e \in E$,
   3. Pick a variable $v \in U$ appearing in $e$. If no such variable exists, abort the algorithm.
   4. Assign a value to $v$ to satisfy the clause $e$. Set $U = U - \{v\}$.
5. Return the value assignment the algorithm chose.

Figure 4.1: A trivial algorithm for solving $k$-SAT.

If the algorithm returns a value assignment at Step (5), this value assignment must satisfy all the clauses. Now, we generalize the algorithm for $(d,k,t)$-CSP. Consider a $(d,k,t)$-CSP instance with variable set $V$ and constraint set $E$. The algorithm is shown in Figure 4.2.

If we can pick a variable $v$ at Step (4), then we make the restriction inactive by removing the correspond-
DKT($P = (V, E)$)

1. For each $v \in V$, let $D_v = \{1, 2, ..., d\}$ be the set of values the algorithm may assign to $v$.

2. For each $e \in E$,

3. For each restriction $R = \{u_1 : \delta_1, ..., u_k : \delta_k\}$ in $e$,

4. Pick a variable $u_i$ with $|D_{u_i}| \geq 2$. If no such variable exists, abort the algorithm.

5. Set $D_{u_i} = D_{u_i} - \{\delta_i\}$.

6. For each variable $v \in V$,

7. Pick a value $\delta \in D_v$. Set $D_v = \{\delta\}$, and assign $\delta$ to $v$.

8. Return the value assignment.

Figure 4.2: A trivial algorithm for solving $(d, k, t)$-CSP.

Note that for each restriction, the algorithm may remove one value at $D_{u_i}$. This is not very "efficient" - by removing $c$ values at $D_{u_i}$, it is possible to make more than $c$ restrictions inactive. Still, it illustrates an interesting idea - variables sacrifice their values for the constraints they appear in, in order to make the restrictions inactive. This idea can be formally expressed as a Token Allocation Problem - we try to allocate tokens (which represent values at the variables) from variables to the constraints they appear in, and then we remove values at variables according the token allocation, in order to make the restrictions inactive.

Definition 4.18 ($(a, b)$-Token Allocation Problem). Consider a variable set $U$, a constraint set $F$, and parameters $a, b \in \mathbb{N}$. Each variable $v \in U$ has $a$ tokens and each constraint $e \in F$ demands $b$ tokens from the variables it contains. The goal is to find a valid $(a, b)$-token allocation (defined as follows).

Definition 4.19 (Token Allocation). A token allocation is a distribution of tokens from each variable to the constraints it appears in. The numbers of tokens going from each variable to each constraint must be non-negative integers.

Definition 4.20 (Valid $(a, b)$-Token Allocation). A token allocation is a valid $(a, b)$-token allocation if (i) every variable $v \in U$ gives out at most $a$ tokens, and (ii) every constraint $e \in F$ receives exactly $b$ tokens.

Definition 4.21 ($\alpha$ and $\beta$). $\alpha = |E(H^*)|$ and $\beta = |V(H^*)| - |I(H^*)|$.

By Definition 4.3, we have

$$\alpha/\beta = \Lambda(d, k, t). \quad (4.4.1)$$

Since we are using the token allocation problem to prove every $\Lambda(d, k, t)$-CSP instance is satisfiable (Lemma 4.14), we will focus on $(\alpha, \beta)$-token allocation problem. For the setting with $k = 2$, we will
provide the exact values of $\alpha, \beta$ and $\Lambda(d, k, t)$ in Chapter 5, after we specify the structure of $H^*$. With a valid $(\alpha, \beta)$-token allocation, we can improve the “inefficient” trivial algorithm DKT to a better algorithm TA-SA, which is capable of finding satisfying assignments on input CSP instances with higher constraint density. This is essential in our proofs of satisfiability in Lemma 4.14 (and Lemma 4.15), and thus essential in the lower bound proofs presented in Section 4.3.2 and 4.3.3 - we would get much worse lower bounds if we used the trivial algorithm DKT instead. The algorithm TA-SA will be shown in Section 4.5, where we will prove the following lemma.

**Lemma 4.22.** Given any $(d, k, t)$-CSP instance $H$, if there exists a valid $(\alpha, \beta)$-token allocation on $(U \subseteq V(H), F \subseteq E(H))$, then there exists a value assignment on $U$ that satisfies every constraint in $F$.

This allows us to prove the satisfiability of a $(d, k, t)$-CSP instance by showing the existence of a valid $(\alpha, \beta)$-token allocation. To show the existence of a valid $(\alpha, \beta)$-token allocation, we will consider the following max-flow formulation of the token allocation problem.

### 4.4.1 Max-flow Formulation

Consider a variable set $U$ and a constraint set $F$. We can formulate an $(a, b)$-Token Allocation Problem as a Max-Flow problem: the source vertex gives each vertex representing $v \in U$ at most $a$ tokens, and the sink vertex wants $b$ tokens from each vertex representing $e \in F$. Formally, we construct an auxiliary directed graph $H' = (V_U + V_F, E')$ as follows:

- Start with an empty graph.
- Add a source vertex $s$ and a sink vertex $t$.
- For each $v \in U$, add a vertex $v_v$ in $V_U$ to represent the variable.
- For each $e \in F$, add a vertex $v_e$ in $V_F$ to represent the constraint.
- For each $v \in U, e \in F$, if $v$ appears in $e$, then add a directed edge going from $v_v$ to $v_e$, with capacity $b$.
- Add directed edges from $s$ to every $v_v \in V_U$, with capacity $a$.
- Add directed edges from every $v_e \in V_F$ to $t$, with capacity $b$.

Since each directed edge in $H'$ has integral capacity, there exists an integral maximum flow. We can see that each integral $(s, t)$-flow of value $b \cdot |V_F|$ in $H'$ corresponds to a valid $(a, b)$-token allocation on $(U, F)$, and vice versa. Now, we can prove Lemma 4.14 and Lemma 4.15, which are required for the complexity lower bound proofs in Section 4.3.2 and 4.3.3.

### 4.4.2 Proof of Lemma 4.14

Lemma 4.14 states that every $\Lambda(d, k, t)$-sparse $(d, k, t)$-CSP instance $H$ is satisfiable. By applying Lemma 4.22, we can prove Lemma 4.14 by showing the existence of a valid $(\alpha, \beta)$-token allocation
Claim 4.23. For any $\Lambda(d, k, t)$-sparse $(d, k, t)$-CSP instance $H$, there exists a valid $(\alpha, \beta)$-token allocation on $(U = V(H), F = E(H))$.

Proof. Consider the max-flow formulation and the auxiliary graph $H'$. If there is an $(s, t)$-flow of value $\beta \cdot |V_F|$ in $H'$, then exists a corresponding valid $(\alpha, \beta)$-token allocation on $(U, F)$. In the following, we will prove this claim by showing the existence of such an $(s, t)$-flow.

Suppose to the contrary that there exists a minimum $(s, t)$-cut $(S, T)$ of size strictly smaller than $\beta \cdot |V_F|$ in $H'$. Consider any directed edge $(v_u, v_e) \in E'$ going from $S$ to $T$. Since the capacity of $(v_u, v_e)$ equals the capacity of $(v_e, t)$, we can move $v_e$ to $S$ while keeping the cut minimum. Hence, we can assume that no directed edge going from $S$ to $T$ is either $(s, v_u)$ for some $v_u \in V_U$ or $(v_e, t)$ for some $v_e \in V_F$. Therefore, the flow value is $\alpha \cdot |V_U \cap T| + \beta \cdot |V_F \cap S|$. The cut size is strictly smaller than $\beta \cdot |V_F|$, so we have $\alpha \cdot |V_U \cap T| < \beta \cdot |V_F \cap T|$. Since $\alpha/\beta = \Lambda(d, k, t)$ (by (4.4.1)), this contradicts the fact that $H$ is $\Lambda(d, k, t)$-sparse, and proves the claim.

4.4.3 Proof of Lemma 4.15

Consider a $(\Lambda(d, k, t) - \sigma)$-sparse $(d, k, t)$-CSP instance $H$ with variable set $V$ and constraint set $E$ where $|V| = \Theta(n')$. In the following, we will prove Lemma 4.15, which states that any clause minimally derived from $H$ must contain $\Theta(n')$ variables.

We can remove a variable if it does not appear in any constraint - the variable is irrelevant to the satisfiability of the CSP instance and it cannot give any token to any constraint. Thus, without loss of generality, we assume (i) every variable appears in at least one constraint.

Suppose to the contrary that there is a clause $C$ minimally derived from $H$ containing $o(n')$ variables. Denote by $V_C \subseteq V$ the set of variables corresponding to the variables appearing in $C$. Consider a valid $(\alpha, \beta)$-token allocation problem on $(U = V - V_C, F = E)$, and its max-flow formulation with auxiliary directed graph $H' = (V_U + V_F, E')$.

Suppose there is an $(s, t)$-flow with value $\beta \cdot |V_F|$ in the auxiliary graph. By Lemma 4.22, there is a value assignment $A$ on $V - V_C$ that satisfies every constraint in $E$. Assign values on $V_C$ to make $C$ evaluate to False. Together with $A$, we obtain a value assignment satisfying $H$ but not $C$, which contradicts the soundness of the resolution proof system.

Therefore, there exists a minimum $(s, t)$-cut $(S, T)$ of size strictly smaller than $\beta \cdot |V_F|$. Consider any directed edge $(v_u, v_e) \in E'$ going from $S$ to $T$. Since the capacity of $(v_u, v_e)$ equals the capacity of $(v_e, t)$, we can move $v_e$ to $S$ while keeping the cut minimum. Hence, we can assume that (ii) no directed edge $(v_u, v_e) \in E'$ goes from $S$ to $T$.

The size of the cut separating $\{s\}$ from all the other vertices equals $\alpha \cdot |V_U|$. It is given that $|V_C| = o(n') = o(|U|)$, $H$ is $(\Lambda(d, k, t) - \sigma)$-sparse and $\alpha/\beta = \Lambda(d, k, t)$ (by (4.4.1)), so we have $|F|/|U| < \Lambda(d, k, t)$.
Hence, $\alpha \cdot |V_U| > \beta \cdot |V_F|$. Since the min-cut has size strictly smaller than $\beta \cdot |V_F|$, this cut separating $\{s\}$ from all others cannot be the min-cut, i.e. $|S| > 1$. Furthermore, it is impossible that $|S \cap V_U| = 0$ and $|S \cap V_F| > 0$ since all the directed edges going from $S$ to $T$ are saturated and all the directed edges going from $T$ to $S$ have zero flow. Therefore, we have (iii) $|S \cap V_U| > 0$

Let $U_T \subseteq U$ be the set of variables represented by vertices in $V_U \cap T$, let $F_T \subseteq F$ be the set of constraints represented by vertices in $V_F \cap T$. By (ii), every constraint $e \in F_T$ only contains variables in $U_T \cup V_C$. Let $H_T$ be the subproblem of $H$ with variable set $U_T \cup V_C$ and constraint set $F_T$. By (iii), $V(H_T) \neq V(H)$. There exists a value assignment on $V(H_T)$ that makes $C$ evaluate to $FALSE$ but satisfies every constraint in $E(H_T)$; otherwise, we can derive $C$ from $H_T$, which contradicts the fact that $C$ is minimally derived from $H$. Denote such a value assignment by $A_T$.

Let $U_S \subseteq U$ be the set of variables represented by vertices in $V_U \cap S$, let $F_S \subseteq F$ be the set of constraints represented by vertices in $V_F \cap S$. Note that all the directed edges going from $S \cap V_F$ to $T$ are saturated and all the directed edges going from $T$ to $S$ have zero flow. Furthermore, by (ii), no directed edge goes from $S \cap V_U$ to $T \cap V_F$. Therefore, after removing all $v \in T - t$ in the auxiliary graph, there is still an $(s, t)$-flow with size $\beta \cdot |S \cap V_F|$. This implies a valid $(\alpha, \beta)$-token allocation on $(U_S, F_S)$. By Lemma 4.22, there exists a value assignment on $U_S$ that satisfies all constraints in $F_S$. Denote such a value assignment by $A_S$.

Together, we have a value assignment $A_T \cup A_S$ that satisfies all the constraints in $H$ but not $C$. This contradicts the soundness of the resolution proof system, and proves the lemma.

### 4.5 Algorithm TA-SA and Proof of Lemma 4.22

Consider any $(d, k, t)$-CSP instance $H$ and a valid $(\alpha, \beta)$-token allocation on $(U \subseteq V(H), F \subseteq E(H))$. We will present an algorithm TA-SA, which constructs a value assignment on $U$ that satisfies every constraint in $F$. This proves Lemma 4.22.

Let $C = \{c_{(e,v)}\}$ be any valid $(\alpha, \beta)$-token allocation on $(U, F)$ where a variable $v$ gives $c_{(e,v)}$ tokens to a constraint $e$, for every $v \in U$ and $e \in F$. $D_v$ will be the set of values the algorithm may assign to $v$, so initially $D_v = \{1, \ldots, d\}$. At any step of the algorithm, we call a restriction $R = (x_1 : \delta_1, \ldots, x_k : \delta_k)$ inactive if $\delta_i \notin D_{x_i}$, for some $i = 1, 2, \ldots k$; otherwise, it is called an active restriction. All restrictions are initially active, and they are made inactive as the algorithm proceeds.

We will define a function $g(c) : \mathbb{N} \rightarrow \mathbb{N}$ to satisfy two key properties stated below. These two properties will ensure that with a valid $(\alpha, \beta)$-token allocation, the algorithm TA-SA (shown in Figure 4.3) will return a value assignment on $U$ that satisfies every constraint in $F$.

This algorithm is a modification of the trivial algorithm DKT (shown in Figure 4.2). Instead of removing one value at $D_v$ to make one restriction in a constraint $e$ inactive at each step, this algorithm removes the $g(c)$ most frequently forbidden values at $D_v$, in order to make a large portion of the restrictions in $e$ inactive. This will be explained in Claim 4.26 later this section.

Therefore, the correctness of this algorithm relies on the token function $g(c)$. We must choose a specific
TA-SA($U, F, \mathcal{C} = \{c_{(e,v)}\}$)

1. For each $v \in U$:
   
   (2) Let $D_v = \{1, \ldots, d\}$ be the set of values the algorithm may assign to $v$.

   (3) For each $e \in F$ that $v$ appears in:
      
      (4) Sort $D_v = \{d_i\}$ in decreasing order by the frequency of $(v : d_i)$ appearing in the active restrictions of $e$, then remove the first $g(c_{(e,v)})$ values in $D_v$. In other words, remove the $g(c_{(e,v)})$ values at $D_v$ that are most frequently forbidden by the active restrictions of $e$.

      (5) Remove all but one value from $D_v$, and assign the value remaining to $v$.

   (6) Return the value assignment.

Figure 4.3: An algorithm constructing a satisfying assignment from a valid $(\alpha, \beta)$-token allocation.

$g(c)$ satisfying the following two properties for every valid $(\alpha, \beta)$-token allocation on any subsets of variables and constraints of any $(d, k, t)$-CSP instance. For the setting with $k = 2$, we will see the exact definition of $g(c)$ in Chapter 6.

Claim 4.24. Let $\mathcal{C} = \{c_{(e,v)}\}$ be any valid $(\alpha, \beta)$-token allocation on $(U, F)$, then $\sum_e g(c_{(e,v)}) \leq d - 1$ for every $v \in U$.

Claim 4.25. Let $\mathcal{C} = \{c_{(e,v)}\}$ be any valid $(\alpha, \beta)$-token allocation on $(U, F)$, then $\sum_v [g(c_{(e,v)}) + 1] > t$ for every $e \in F$.

Claim 4.24 ensures that for every variable $v$, a total of at most $d - 1$ values are removed at $D_v$ during all the executions of Step (4). Hence, the algorithm can always assign a value to $v$ at Step (5). On the other hand, Claim 4.25 ensures that all the restrictions are inactive in the end, as shown in the following claim. Thus, given a valid $(\alpha, \beta)$-token allocation and a token function $g(c)$ satisfying these two claims, the algorithm always returns a satisfying assignment.

Claim 4.26. All restrictions are inactive when the algorithm terminates.

Proof. As the algorithm proceeds, we remove values at $D_v$ for some variables and make some restrictions inactive. Whenever we reach Step (4), for any positive integer $c$, there are at most $c$ values in $D_v$ that each appears in more than a fraction of $1/(c + 1)$ active restrictions in $e$. So, by removing $g(c_{(e,v)})$ values at Step (4), any value remaining at $D_v$ appears in at most a fraction of $1/g(c_{(e,v)}) + 1$ of the active restrictions in $e$. Therefore, no matter which value the algorithm later assigns to $v$ at Step (5), at most $\sum_v [g(c_{(e,v)}) + 1] < 1$, i.e. all restrictions are inactive.

This proves Lemma 4.22, contingent on providing parameters $\alpha, \beta$ and a token function $g(c)$ satisfying Claim 4.24 and Claim 4.25. As a remark, the parameters $\alpha, \beta$ are important in the design of the token function $g(c)$. This is not very surprising because in a valid $(\alpha, \beta)$-token allocation, each variable gives out at most $\alpha$ tokens and each constraint receives exactly $\beta$ tokens.
4.6 Chapter Summary

We presented a general approach for studying the resolution and tree resolution complexity of random $(d, k, t)$-CSP. Given a random $(d, k, t)$-CSP model, if we can identify the special forcer $H^*$ and design a suitable token function $g(c)$, then we can prove the lower and upper bounds on resolution and tree resolution complexity stated in Conjecture 3.1, 3.2 and 3.3.

This approach works for every constant triple $(d, k, t)$ with $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$. In the following two chapters, we will use this approach to prove resolution complexity results for the random $(d, k, t)$-CSP model $\text{CSP}_{n,m}^{d,k,t}$ with $k = 2$ and constants $d, t$ where $1 \leq t < d - 1$. These results are stated in Theorem 3.4, 3.5 and 3.6. Applications of this approach to other models of random CSPs will be briefly discussed in Chapter 7.

To complete the proof of the upper bound stated in Theorem 3.4, we will provide the detailed structure of the special forcer $H^*$, and use $H^*$ to construct the family of forbidding flowers $G^*$ such that (i) for any constant integer $\ell \geq 2$, $G_\ell \in G^*$ has constant numbers of variables and constraints; and (ii) for any positive constant $\mu$, there is a sufficiently large constant $\ell$ such that $G_\ell \in G^*$ is $(\Lambda(d, k, t) + \mu)$-sparse. We will see the detailed structure of $H^*$ in Chapter 5, and prove these two properties in Section 5.3.

To complete the proof of the lower bound stated in Theorem 3.5 and 3.6, we will define a token function $g(c)$ satisfying Claim 4.24 and Claim 4.25. Recall that the parameters $\alpha, \beta$ of the token allocation problem depend on the structure of $H^*$ (Definition 4.21). We will design the function $g(c)$ by studying the structure of $H^*$ in Chapter 6. Then we will prove these two claims in Section 6.4 and Section 6.5.
Chapter 5

The Special Forcer $H^*$

Consider the random $(d, k, t)$-CSP model $\text{CSP}^{d,k,t}_{n,m}$ with any constant triple $(d, k, t)$ such that $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$.

In this chapter, we focus on the setting with $k = 2$. We will first construct the special forcer $H^*$. With the detailed structure of $H^*$, we can prove that the forbidding flower $G_\ell \in \mathcal{G}^*$ is small and sparse (Claim 5.2 and Claim 5.3). Then, we will determine the value of $\Lambda(d, k, t)$ by studying the structure of $H^*$. This proves the upper bound on both resolution complexity and tree resolution complexity stated in Theorem 3.4. When $k = 2$, each constraint contains two variables, so the constraint hypergraph is actually a graph. Also, we have $t < (d - 1)d^{k-2} = d - 1$.

5.1 An Outline on the Construction of $H^*$

By Definition 4.1, a forcer is a $(d, k, t)$-CSP instance with a special property that: there exists a value $\delta$ and a tuple of values $A_I$ such that after fixing $A_I$ at the input variable set $I$, we are forced to assign $\delta$ to the output variable $u$, i.e. all the other $d - 1$ values are forbidden at $u$.

Start with a set of variables $I$ and a value assignment $A_I$ that we fix at $I$. We are going to construct the special forcer $H^*$ to forbid $d - 1$ values at the output variable $u$. We require $H^*$ to be as sparse as possible because we use $H^*$ as building blocks to construct unsatisfiable instances with constraint densities arbitrarily close to the lowest possible value. Our intuition is that: we should forbid exactly $d - 1$ values at $u$ instead of forbidding all the $d$ values; otherwise, it is possible to forbid only $d - 1$ values by a sparser structure, which implies that there is a forcer sparser than $H^*$. Similarly, we should not forbid all the $d$ values at any variable in $H^*$.

Consider a constraint $e$ containing a variable $w$ and a variable $v \in I$. By choosing appropriate restrictions in $e$, the restrictions can forbid $t$ values at $w$ since the value at $v$ is fixed. Such a constraint is called a $t$-forbidding edge. Consider $x_1$ different $t$-forbidding edges containing $w$ and different $v_i \in I$. Each $t$-forbidding edge forbids $t$ values, so there are $r_1 = d - x_1 \cdot t$ values remaining that $w$ can take.
Similarly, consider a constraint \( e' \) containing a variable \( w' \) and a variable with \( x_1 \cdot t \) values forbidden (i.e. a variable with \( r_1 \) values remaining). By choosing appropriate restrictions in \( e' \), this constraint can forbid \( t_2 = \lfloor t/r_1 \rfloor \) values at \( w' \). Such a constraint is called a \( t_2 \)-forbidding edge. Consider \( x_1 \) different \( t \)-forbidding edges containing \( w' \) and different \( v_i \in I \), and \( x_2 \) different \( t_2 \)-forbidding edges containing \( w' \) and different \( v_j \in I \). Each \( t \)-forbidding edge forbids \( t \) values and each \( t_2 \)-forbidding edge forbids \( t_2 \) values, so there are \( r_2 = d - x_1 \cdot t - x_2 \cdot t_2 = r_1 - x_2 \cdot t_2 \) values remaining that \( w' \) can take.

If \( r_2 = 1 \), then we get the forcer we want; otherwise, we will use some other structures to forbid the remaining \( r_2 - 1 \) values at the target variable. In the following, we will present the above idea formally.

### 5.2 Forbidden and Forcers

We will enforce that each input variable \( v \in I \) appears in exactly one constraint in \( E(H^*) \), which is feasible because we are free to choose the size of the input variable set \( I \). The following variables are used to define the structure of \( H^* \).

\[
\begin{align*}
x_1 &= \lfloor (d - 1)/t \rfloor & (5.2.1) \\
r_1 &= d - x_1 \cdot t & (5.2.2) \\
t_2 &= \lfloor t/r_1 \rfloor & (5.2.3) \\
x_2 &= \lfloor (r_1 - 1)/t_2 \rfloor & (5.2.4) \\
r_2 &= r_1 - x_2 \cdot t_2 & (5.2.5)
\end{align*}
\]

The following are cases of different values of \( d, k, t \) since the values of \( r_1, t_2 \) are determined by the values of \( d, k, t \).

#### 5.2.1 CASE 1: \( r_1 = 1 \)

**\( t \)-forbidders.** Given a set of input variables \( I \) with fixed values \( A_I \), create a constraint \( e \) between a variable \( v \in I \) and a variable \( u \). By choosing appropriate restrictions in \( e \), it can forbid \( t \) values at \( u \). Recall that \( t < d - 1 \) in our setting, so there are still some values remaining that \( u \) can take. Such a constraint is called a \( t \)-forbidding edge. We call the whole structure a \( t \)-forbidder, which has an input variable \( v \in I \), an output variable \( u \), and a \( t \)-forbidding edge. (Note that the output variable of a \( t \)-forbidder is not necessary the output variable of \( H^* \).)

**\( r_1 \)-forcers and \( r_1 \)-free variables.** By (5.2.1) and (5.2.2), we have \( x_1 = \lfloor (d - 1)/t \rfloor \) and \( r_1 = d - x_1 \cdot t \). Start with a new variable \( u \). Make \( u \) to be the output variable of \( x_1 \) different \( t \)-forbidders (i.e. a CSP instance consisting of \( x_1 \) different \( t \)-forbidders sharing only the output variable), then we can forbid \( x_1 \cdot t \) values at \( u \). In other words, we force \( u \) to take one of the \( r_1 \) remaining values; and we call the variable an \( r_1 \)-free variable. We call this structure an \( r_1 \)-forcer, which has an input variable set \( I' \subseteq I \) (which
are used in the t-forbidders), an output variable $u$, and $x_1$ different t-forbidders that all have $u$ as the common output variable. The structure is shown in Figure 5.1.

**Forcer $H^*$ in CASE 1.** For $r_1 = 1$, $H^*$ is an $r_1$-forcer. It contains (i) an input variable set $I$ of $x_1$ variables, (ii) an output variable $u$, (iii) $x_1$ different t-forbidders that all have $u$ as the common output variable. By assigning a tuple of values $A_t$ to $I$, we are forced to assign a value $\delta_u$ to $u$. The structure is shown as an $r_1$-forcer in Figure 5.1.

5.2.2 CASE 2: $1 < r_1 \leq t_2$

**t-forbidders.** Same as those defined in CASE 1.

**$r_1$-forcers and $r_1$-free variables.** Same as those defined in CASE 1.

Since $r_1 > 1$, there are more than one remaining values that an $r_1$-free variable can take. We will continue and forbid $r_1 - 1$ values at an $r_1$-free variable.

By (5.2.3), $t \neq t_2$ when $r_1 > 1$. So the following definition of $t_2$-forbidders does not override the definition of t-forbidders. Similarly, other definitions of other forcers and forbidders also do not override each other since the corresponding values can never coincide (we may have $t = r_1$ or $t_2 = r_1$, but we only define $t_2$-forbidder, t-forbidder and $r_1$-forer, so no ambiguity in the definitions).

**$t_2$-forbidders.** Create a constraint $e$ between an $r_1$-free variable $v$ and a target variable $u$. By (5.2.3), we have $t_2 = \lfloor t/r_1 \rfloor$. By choosing appropriate restrictions in $e$, it can forbid $t_2$ values at $u$. Such a constraint is called a $t_2$-forbidding edge. We call this structure a $t_2$-forbidder, which has an input variable set $I' \subseteq I$ (used in the $r_1$-forcer), an output variable $u$, an $r_1$-free variable with its $r_1$-forcer, and a $t_2$-forbidding edge.

Since $r_1 \leq t_2$, we cannot use a $t_2$-forbidder to forbid $t_2$ values at an $r_1$-free variable; otherwise, all the values will be forbidden. Instead, we will forbid $r_1 - 1$ values at an $r_1$-free variable by using the
1-forbidders we define as follows.

Consider a constraint \( e \) between a variable \( v \) with \( t' \leq t \) values remaining and a target variable \( u \). By choosing appropriate restrictions in \( e \), it can forbid one value at \( u \). However, we do not want \( t' \leq t/2 \), because this implies that \( e \) can forbid more than one value at \( u \). Forbidding only one value when the same number of constraints could have forbidden two indicates that this construction is probably not the sparsest possible.

To obtain a suitable variable with \( t' \in (t/2, t] \) values remaining, we will use a structure defined as follows.

\((r_1 + t - t_2)\)-forcers and \((r_1 + t - t_2)\)-free variables.\) Start with an \( r_1 \)-free variable \( u \) with its \( r_1 \)-forcer. Remove a \( t \)-forbidder in the \( r_1 \)-forcer, and add an extra \( t_2 \)-forbidder having \( u \) as the output variable. Then, there are \((r_1 + t - t_2)\) values remaining at \( u \); and we call it an \((r_1 + t - t_2)\)-free variable. We call this structure an \((r_1 + t - t_2)\)-forcer, which has an input variable set \( I' \subseteq I \), an output variable \( u \), and \( x_1 - 1 \) different \( t \)-forbidders and one \( t_2 \)-forbidders that all have \( u \) as the common output variable (there are originally \( x_1 \) different \( t \)-forbidders in an \( r_1 \)-forcer).

By (5.2.3), \( t_2 = \lfloor t/r_1 \rfloor \). Also, we have \( r_1 > 1 \) in CASE 2. These imply \( t_2 \leq t/2 \). On the other hand, we have \( r_1 \leq t_2 \) in CASE 2. Together, we have \( t' = r_1 + t - t_2 \in (t/2, t] \), as desired. Now, we can construct a 1-forbidder as follows. The structure is shown in Figure 5.2.

\[\text{Figure 5.2: An } (r_1 + t - t_2)\text{-forcer, with } x_1 = 3.\]

1-forbidders. Create a constraint \( e \) between an \((r_1 + t - t_2)\)-free variable \( v \) and a target variable \( u \). By choosing appropriate restrictions in \( e \), it can forbid one value at \( u \) since \( r_1 + t - t_2 \leq t \). Such a constraint is called a 1-forbidding edge. We call this structure a 1-forbidder, which has an input variable set \( I' \subseteq I \), an output variable \( u \), an \((r_1 + t - t_2)\)-free variable with its \((r_1 + t - t_2)\)-forcer, and a 1-forbidding edge.

With \( x_1 \) different \( t \)-forbidders and \( r_1 - 1 \) different 1-forbidders, we can forbid \( d - 1 \) values at a variable.
**Chapter 5. The Special Forcer \(H^*\)**

**Forcer \(H^*\) in CASE 2.** \(H^*\) contains (i) an input variable set \(I\) with size \((2r_1 - 1)x_1 + (r_1 - 1)\), (ii) an output variable \(u\), (iii) \(x_1\) different \(t\)-forbidders and \(r_1 - 1\) different \(1\)-forbidders that all these forbidders have \(u\) as the common output variable. By assigning a tuple of values \(A_I\) to \(I\), we are forced to assign a value \(\delta_u\) to \(u\). The structure is shown in Figure 5.3.

The value of \(|I|\) is not used in any proof since the crucial parameter \(\Lambda(d, k, t)\) is defined to be \(|E|/(|V| - |I|)\). Thus, the simple calculation to determine \(|I|\) will be skipped.

![Diagram](image)

**Figure 5.3:** The special forcer \(H^*\) for CASE 2, with \(x_1 = 3\) and \(r_1 = 4\).

**5.2.3 CASE 3: \(r_1 > t_2\)**

**\(t\)-forbidders.** Same as those defined in CASE 1.

**\(r_1\)-forcers and \(r_1\)-free variables.** Same as those defined in CASE 1.

**\(t_2\)-forbidders.** Same as those defined in CASE 2.

**\(r_2\)-forcers and \(r_2\)-free variables.** By (5.2.4) and (5.2.5), we have \(x_2 = \lfloor (r_1 - 1)/t_2 \rfloor\) and \(r_2 = r_1 - x_2 \cdot t_2\). Start with an \(r_1\)-free variable \(u\). If we make \(u\) to be the output variable of \(x_2\) different \(t_2\)-forbidders, then we can forbid \(x_2 \cdot t_2\) additional values at \(u\). There are \(r_2\) values remaining that \(u\) can take. This structure is called an \(r_2\)-forcer, which has an input variable set \(I' \subseteq I\), an output variable \(u\), an \(r_1\)-forcer and \(x_2\) different \(t_2\)-forbidders such that the \(r_1\)-forcer and all the \(t_2\)-forbidders have \(u\) as their common output variable. We call the output variable \(u\) an \(r_2\)-free variable. Note that we override the
notation here: an \( r_2 \)-free variable will not be called \( r_1 \)-free any more, although it is the output variable of an \( r_1 \)-forcer. The structure is shown in Figure 5.4.

\[
\text{Input variables} \quad r_1\text{-free variables} \quad r_2\text{-free variable}
\]

\[
\begin{align*}
&\text{x}_1 \text{ different t}-\text{forbidding edges} \\
&\text{x}_1 \text{ different t}-\text{forbidding edges for each } r_1\text{-free variable} \\
&\text{x}_2 \text{ different t}_2\text{-forbidding edges, connecting } x_2 \text{ different } r_1\text{-free variables to the } r_2\text{-free variable}
\end{align*}
\]

Figure 5.4: An \( r_2 \)-forcer, with \( x_1 = 3 \) and \( x_2 = 4 \).

If \( r_2 = 1 \), then \( H^* \) is an \( r_2 \)-forcer we just defined. Otherwise, we will continue and forbid \( r_2 - 1 \) values at an \( r_2 \)-free variable. Note that \( r_1 > t_2 \) in CASE 3, i.e. \( r_1 + t - t_2 > t \). Thus, the \( (r_1 + t - t_2) \)-free variables and the 1-forbidders in CASE 2 do not work in CASE 3. In the following, we will define \( (r_2 + t - t_2) \)-free variables and use them to construct 1-forbidders.

**\( (r_2 + t - t_2) \)-forcers and \( (r_2 + t - t_2) \)-free variables.** Start with an \( r_2 \)-free variable \( u \) with its \( r_2 \)-forcer. Remove a \( t \)-forbidder in the \( r_2 \)-forcer, and add an extra \( t_2 \)-forbidder having \( u \) as the output variable. Thus, there are \( (r_2 + t - t_2) \) values remaining at \( u \); and we call the variable an \( (r_2 + t - t_2) \)-free variable. We call this structure an \( (r_2 + t - t_2) \)-forcer, which has an input variable set \( I' \subseteq I \), an output variable \( u \), and \( x_1 - 1 \) different \( t \)-forbidders and \( x_2 + 1 \) different \( t_2 \)-forbidder that all have \( u \) as the common output variable (there are originally \( x_1 \) different \( t \)-forbidders and \( x_2 \) different \( t_2 \)-forbidder in an \( r_2 \)-forcer). The structure is shown in Figure 5.5.

As in CASE 2, we require that \( t' = r_2 + t - t_2 \in (t/2, t] \), which is proved as follows. By the definitions, we have (i) \( r_2 = r_1 - x_2 \cdot t_2 < r_1 \), (ii) \( t_2 = \lceil t/r_1 \rceil \), and (iii) \( t_2 \geq r_2 > 1 \). By (i) and (iii), we have (iv) \( r_1 \geq 3 \). By (ii) and (iv), we have (v) \( t_2 < t/2 \). Finally, by (iii) and (v), we have \( r_2 + t - t_2 \in (t/2, t] \).

**1-forbidders.** Create a constraint \( e \) between an \( (r_2 + t - t_2) \)-free variable \( v \) and a target variable \( u \). By choosing appropriate restrictions in \( e \), it can forbid one value at \( u \). Such a constraint is called a 1-forbidding edge. We call this structure a 1-forbidder, which has an input variable set \( I' \subseteq I \), an output variable \( u \), an \( (r_2 + t - t_2) \)-free variable with its \( (r_2 + t - t_2) \)-forcer, and a 1-forbidding edge.
Figure 5.5: An \((r_2 + t - t_2)\)-forcer, with \(x_1 = 3\) and \(x_2 = 4\).

With \(x_1\) different \(t\)-forbidders, \(x_2\) different \(t_2\)-forbidders and \(r_2 - 1\) different 1-forbidders, we can forbid \(d - 1\) values at a variable. Note that when \(r_2 = 1\), the number of 1-forbidders required is \(r_2 - 1 = 0\).

**Forcer \(H^*\) in CASE 3.** \(H^*\) contains (i) an input variable set \(I\) with size \(r_2(x_2 + 1) \cdot x_1 + (r_2 - 1)(x_1 - 1)\). (ii) an output variable \(u\), (iii) \(x_1\) different \(t\)-forbidders, \(x_2\) different \(t_2\)-forbidders and \(r_2 - 1\) different 1-forbidders that all have \(u\) as the common output variable. By assigning a tuple of values \(A_I\) to \(I\), we are forced to assign a value \(\delta_u\) to \(u\). The structure is shown in Figure 5.6.

The value of \(|I|\) is not used in any proof since the crucial parameter \(\Lambda(d, k, t)\) is defined to be \(|E|/(|V| - |I|)\). Thus, the simple calculation to determine \(|I|\) will be skipped.

## 5.3 Some Properties of \(H^*\) and \(G^*\)

**Claim 5.1.** There are constant numbers of variables and constraints in \(H^*\).

**Proof.** Since the parameters \(d, k, t\) are constants, the variables \(x_1, r_1, x_2, r_2\) are also constants. We define \(H^*\) by these parameters, so the numbers of variables and constraints in \(H^*\) must be constants. \(\square\)

In the following, we will prove two important properties of \(G^*\), the family of forbidding flowers we defined in Section 4.3.1. First, let’s go through the definition of \(G^*\) again (Definitions 4.1, 4.4, 4.5, 4.6, 4.7 and 4.10).

By using \(|I|\) forcers, we can construct an \((I : A_I) \rightarrow (U : A_U)\) forcing block such that after fixing values \(A_I\) at a variable set \(I\), we are forced to assign values \(A_U\) at a variable set \(U\), were \(|I| = |U|\). With \((X_i : A_i) \rightarrow (X_{i+1} : A_{i+1})\) forcing blocks for \(i = 1, 2, \ldots, \ell\), we obtain an \((X_1 : A_1) \rightarrow (X_{\ell+1} : A_{\ell+1})\).
forcing path such that after fixing values $A_1$ at variable set $X_1$, we are forced to assign values $A_{\ell+1}$ at $X_{\ell+1}$. If $X_1 = X_{\ell+1}$ and $A_1 \neq A_{\ell+1}$, we obtain an $X_1 \not\approx A_1$ forbidding cycle which forbids us from assigning $A_1$ to $X_1$. Finally, with $d[|I|]$ different $I \not\approx A_I$ forbidding cycles for each of the $d[|I|]$ possible value assignments $A_I$ on a variable set $I$, all the possible value assignments are forbidden from being assigned to $I$. This unsatisfiable $(d, k, t)$-CSP instance is called a forbidding flower.

By the definition of $(d, k, t)$-CSP, we are allowed to use all the $(d^k)$ possible types of constraints to construct a $(d, k, t)$-CSP instance. By Claim 4.9, given any $(I : A_I) \rightarrow (u : \delta)$ forcer, we can change some restrictions in the constraints to obtain an $(I : A'_I) \rightarrow (u : \delta')$ forcer for any $|I|$-tuple of values $A'_I$ and any value $\delta'$. Thus, we can construct a forbidding flower by using only forcers structurally similar to $H^*$ - two CSP instances are said to be structurally similar if their constraint hypergraphs are isomorphic to each other (Definition 4.8).

For any positive integer $\ell \geq 2$, define $G_\ell$ to be a forbidding flower such that (i) every forcer in $G_\ell$ is structurally similar to $H^*$, and (ii) each cycle has length $\ell$. Denote the family of all these forbidding
flowers by $G^* = \{G_\ell\}$. In the following, we will prove the two properties of $G_\ell \in G^*$ required in the upper bound proof presented in Section 4.3.1.

**Claim 5.2.** For any constant integer $\ell \geq 2$, $G_\ell \in G^*$ has constant numbers of variables and constraints.

**Proof.** With the length $\ell$ being a constant, a forbidding flower consists of a constant number of forcers, each has constraint hypergraph isomorphic to the constraint hypergraph of $H^*$. By Claim 5.1, there are constant numbers of variables and constraints in $H^*$, and so does $G_\ell$. □

**Claim 5.3.** For any positive constant $\mu$, there is a sufficiently large constant $\ell$ such that $G_\ell \in G^*$ is $(\Lambda(d, k, t) + \mu)$-sparse.

**Proof.** Let $I = I(G_\ell)$ be the variable set shared by the forbidding cycles in $G_\ell$. Let $P$ be a subproblem of $G_\ell$ obtained by removing constraint set $E(H)$ and variable set $V(H) - I(H) - u(H)$, for every forcer $H$ in the $\lfloor \ell/2 \rfloor$-th forcing block at each forbidding cycle. We can show that no subproblem of $P$ has constraint density higher than $\Lambda(d, k, t)$. This property will be proved as Claim 5.6 later this section.

Let $G'$ be any subproblem of $G_\ell$. If $G'$ has fewer than $\ell/3$ variables, its constraint hypergraph must be isomorphic to the constraint hypergraph of some subproblem of $P$, which implies that $G'$ has constraint density at most $\Lambda(d, k, t)$.

Suppose $G'$ has at least $\ell/3$ variables. Note that $P$ is obtained by removing at most $d |I| \times |I| \times |V(H^*)|$ variables and at most $d |I| \times |I| \times |E(H^*)|$ constraints. Thus, we can obtain a subproblem $G''$ of $G'$ such that $G''$ is also a subproblem of $P$, by removing at most $d |I| \times |I| \times |V(H^*)|$ variables and at most $d |I| \times |I| \times |E(H^*)|$ constraints.

Since $G''$ is a subproblem of $P$, it has constraint density at most $\Lambda(d, k, t)$. Thus, the constraint density of $G'$ is at most $(\Lambda(d, k, t) + \frac{d |I| \times |E(H^*)|}{\ell/3})$, which is $(\Lambda(d, k, t) + \mu)$ for some sufficiently large constant $\ell$. □

We will use the following two claims to prove Claim 5.6. The proofs of these two claims are shown in Appendix B.

**Claim 5.4.** For every $(I : A_I) \rightarrow (U : A_U)$ forcing block $B$ in $G_\ell$, every subset of constraints $E' \subseteq E(B)$ contains at least $\Lambda(d, k, t)$ variables in $V(B) - I$.

**Claim 5.5.** For every $(I : A_I) \rightarrow (U : A_U)$ forcing block $B$ in $G_\ell$, every subset of constraints $E' \subseteq E(B)$ contains at least $\Lambda(d, k, t)$ variables in $V(B) - U$.

**Claim 5.6.** No subproblem of $P$ has constraint density higher than $\Lambda(d, k, t)$.

**Proof.** By the definition of $P$, it is a subproblem of $G_\ell$ and consists of a set of forcing blocks. Let $E_1 = \{B_i\}$ be the set of the first $\lfloor \frac{\ell - 1}{2} \rfloor$ forcing blocks in each forbidding cycle of $G_\ell$, and let $E_2 = \{B_j\}$ be the set of the last $\ell - 1 - \lfloor \frac{\ell - 1}{2} \rfloor$ forcing blocks in each forbidding cycle of $G_\ell$. Let $P_F = \cup_{B_i \in E_1} B_i$ and $P_R = \cup_{B_j \in E_2} B_j$. By the definition of $G_\ell$, every forcer of $G_\ell$ is structurally similar to $H^*$.

By the definition of the $P$ and $G_\ell$, we can partition $P$ into: (i) the variable set $I(G_\ell)$, (ii) a pair $(V(B_i) - I, E(B_i))$ for every $B_i \in E_1$, and (iii) a pair $(V(B_j) - U, E(B_j))$ for every $B_j \in E_2$. 


Any subproblem $P'$ of $P$ can be partitioned into $\{P'_1, \ldots, P'_z\}$ accordingly. By Claim 5.4 and Claim 5.5, the constraint-variable ratio for each $P'_i$ is at most $\Lambda(d, k, t)$. Thus, the constraint density of $P'$ is at most $\Lambda(d, k, t)$, which completes the proof. 

5.4 The Value of $\Lambda(d, k, t)$

By Definition 4.3, $\Lambda(d, k, t) = |E(H^*)|/(|V(H^*)| - |I(H^*)|)$. In this section, we will determine the value of $\Lambda(d, k, t)$ by counting the number of variables in $V(H^*) - I(H^*)$ and the number of constraints in $E(H^*)$. Since the input variable set $I(H^*)$ is excluded in the definition of $\Lambda(d, k, t)$, we will not count the input variables in the following calculations.

The following are cases of different values of $d, k, t$ since the values of $r_1, t_2$ are determined by the values of $d, k, t$. Also, recall that we defined $\alpha = |E(H^*)|$ and $\beta = |V(H^*)| - |I(H^*)|$, i.e. $\alpha/\beta = \Lambda(d, k, t)$ in Chapter 4.

5.4.1 CASE 1: $r_1 = 1$

By the construction presented in Section 5.2.1, $H^*$ is an $r_1$-forcer, which consists of (i) one variable and (ii) $x_1$ constraints. So $\alpha = x_1$, $\beta = 1$ and $\Delta = x_1/1 = x_1$.

5.4.2 CASE 2: $1 < r_1 \leq t_2$

By the construction presented in Section 5.2.2, $H^*$ consists of (i) an $r_1$-forcer, plus (ii) $r_1 - 1$ different 1-forbidders. Note that a 1-forbidder has the same number of variables and constraints as a CSP instance consisting of an $r_1$-forcer, plus an extra $t_2$-forbidder. Hence, $H^*$ is equivalent to (i) $r_1$ different $r_1$-forcers, plus (ii) $r_1 - 1$ different $t_2$-forbidders.

Each $t_2$-forbidder is a subproblem consisting of a constraint, and an $r_1$-forcer. There are $r_1 - 1$ different $t_2$-forbidders in $H^*$. Hence, $H^*$ is equivalent to (i) $n_1 = r_1 + (r_1 - 1)$ different $r_1$-forcers, plus (ii) $r_1 - 1$ constraints.

Each $r_1$-forcer is an output variable plus $x_1$ constraints. There are $n_1$ different $r_1$-forcers in $H$. Hence,
(i) $H$ has $n_1$ variables and (ii) $x_1 \cdot n_1 + m_1$ constraints. Therefore,

\[
\beta = n_1 \\
= 2r_1 - 1
\]

\[
\alpha = x_1 \cdot n_1 + (r_1 - 1) \\
= x_1 \cdot (2r_1 - 1) + r_1 - 1 \\
= (2x_1 + 1) \cdot r_1 - (x_1 - 1)
\]

\[
\Lambda(d, k, t) = \frac{\alpha}{\beta} \\
= x_1 + 1 - \frac{r_1}{2r_1 - 1}
\]

### 5.4.3 CASE 3: $r_1 > t_2$

By the construction presented in Section 5.2.3, $H^*$ consists of (i) an $r_2$-forcer, plus (ii) $r_2 - 1$ different 1-forbidders. Note that a 1-forbidder has the same number of variables and constraints as a CSP instance consisting of an $r_2$-forcer, plus an extra $t_2$-forbidder. Hence, $H^*$ is equivalent to (i) $r_2$ different $r_2$-forcers, plus (ii) $r_2 - 1$ different $t_2$-forbidders.

Each $r_2$-forcer is an $r_1$-forcer with extra $x_2$ different $t_2$-forbidders. There are $r_2$ different $r_2$-forcers. Hence, $H^*$ is equivalent to (i) $r_2$ different $r_1$-forcers, plus (ii) $m_1 = x_2 \cdot r_2 + (r_2 - 1)$ different $t_2$-forbidders.

Each $t_2$-forbidder is a subproblem consisting of a constraint, and an $r_1$-forcer. There are $m_1$ different $t_2$-forbidders in $H^*$. Hence, $H^*$ is equivalent to (i) $n_1 = m_1 + r_2$ different $r_1$-forcers, plus (ii) $m_1$ constraints.

Each $r_1$-forcer is an output variable plus $x_1$ constraints. There are $n_1$ different $r_1$-forcers in $H$. Hence, $H$ has (i) $n_1$ variables and (ii) $x_1 \cdot n_1 + m_1$ constraints. Therefore,

\[
\beta = n_1 = m_1 + r_2 \\
= (x_2 \cdot r_2 + r_2 - 1) + r_2 \\
= (x_2 + 1) \cdot r_2 + r_2 - 1 \\
= (x_2 + 2) \cdot r_2 - 1
\]

\[
\alpha = x_1 \cdot n_1 + m_1 \\
= x_1 \cdot \beta + (x_2 \cdot r_2 + r_2 - 1) \\
= x_1 \cdot [(x_2 + 2) \cdot r_2 - 1] + [(x_2 + 1) \cdot r_2 - 1]
\]

\[
\Lambda(d, k, t) = \frac{\alpha}{\beta} \\
= (x_1 + 1) - \frac{r_2}{(x_2 + 2) \cdot r_2 - 1}
\]
5.4.4 The Variables Defining $\Lambda(d, k, t)$

As a summary, we use the following variables to define $\Lambda(d, k, t)$.

In all cases, we have:

$$\Lambda(d, k, t) = \alpha/\beta \quad (5.4.1)$$

In CASE 1, we define:

$$\alpha = x_1, \beta = 1 \quad (5.4.2)$$

In CASE 2, we define:

$$\beta = 2r_1 - 1 \quad (5.4.3)$$
$$\alpha = x_1 \cdot \beta + (r_1 - 1) \quad (5.4.4)$$

In CASE 3, we define:

$$m_1 = x_2 \cdot r_2 + (r_2 - 1) \quad (5.4.5)$$
$$\beta = m_1 + r_2 \quad (5.4.6)$$
$$\alpha = x_1 \cdot \beta + m_1 \quad (5.4.7)$$

Therefore, the values of variables $\alpha, \beta$ and $\Lambda(d, k, t)$ are all determined by the values of $d, k, t$. Furthermore, they are all constants when $d, k, t$ are constants.

5.5 Chapter Summary

In this chapter, we defined the special forcer $H^*$ used in the approach presented in Chapter 4, for the random $(d, k, t)$-CSP model with $1 \leq t < (d - 1)d^{k-2}$ and $k = 2$. With the detailed structure of $H^*$, we determined the value of the constant $\Lambda(d, k, t)$.

We proved the two properties of $G_\ell \in G^*$ required in the application of the second moment method (Claim 5.2 and Claim 5.3), contingent on providing the proofs of Claim 5.4 and 5.5, which are included in Appendix B. With the standard arguments using the second moment method (Lemma 4.12) shown in Appendix A.3, this proves the upper bound for both resolution complexity and tree resolution complexity stated in Theorem 3.4.

The following proof just repeats what we described in Chapter 4. We include it here for the purpose of completeness.

**Proof of Theorem 3.4:** Consider the random model $\text{CSP}_{n,m}^{d,k,t}$ with any constant triple $(d, k, t)$ such that $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$, and with constraint density $n^{k-1 - \frac{1}{k(d,k,d)^{\epsilon}}}$ for any constant $\epsilon > 0$.

For any constant $\mu > 0$, Claim 5.2 and Claim 5.3 imply that there exists an unsatisfiable $(d, k, t)$-CSP
instance $G_\ell \in G^*$ that is $(\Lambda(d, k, t)+\mu)$-sparse and has constant size. By Lemma 4.12, when the constraint density is $n^{k-1-\frac{1}{\Lambda(d,k,t)+\epsilon}}$, w.h.p. such a $G_\ell$ appears in a random instance drawn from $\text{CSP}_{n,m}^{d,k,t}$. Since $G_\ell$ has constant size, it can be refuted easily in constant time. Therefore, when the constraint density is $n^{k-1-\frac{1}{\Lambda(d,k,t)+\epsilon}}$, w.h.p. both the resolution complexity and tree resolution complexity are constant. This proves Theorem 3.4. \hfill \Box
Chapter 6

The Token Function $g(c)$

Consider the random $(d, k, t)$-CSP model $\text{CSP}_{n,m}^{d,k,t}$ with any constant triple $(d, k, t)$ such that $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$.

In this chapter, we will design the token function $g(c)$ required in the algorithm TA-SA, for the setting with $k = 2$. This will complete the proof of Lemma 4.22, and thus the proof of the lower bounds on resolution and tree resolution complexity stated in Theorem 3.5 and 3.6. When $k = 2$, each constraint contains two variables, so the constraint hypergraph is actually a graph. Also, we have $t < (d-1)d^{k-2} = d - 1$.

6.1 An Outline on the Design of $g(c)$

In Section 4.5, we presented the algorithm TA-SA, which constructs a satisfying assignment from a valid $(\alpha, \beta)$-token allocation. Here, we will design the token function $g(c) : \mathbb{N} \rightarrow \mathbb{N}$ used in the algorithm. Let $\mathcal{C} = \{c_{(e,v)}\}$ be a valid $(\alpha, \beta)$-token allocation, i.e. $\sum_v c_{(e,v)} \leq \alpha$ for every variable $v$, and $\sum_e c_{(e,v)} = \beta$ for every constraint $e$. To ensure the correctness of the algorithm, we need the following two properties of $g(c)$ to hold for every valid $(\alpha, \beta)$-token allocation $\mathcal{C}$ on any variable set $U$ and constraint set $F$ of any $(d, k, t)$-CSP instance:

$$\sum_e g(c_{(e,v)}) \leq d - 1 \text{ for every variable } v \in U \text{ (Claim 4.24)}$$

$$\prod_e (g(c_{(e,v)}) + 1) > t \text{ for every constraint } e \in F \text{ (Claim 4.25)}$$

As mentioned in Section 4.1, we want $\Lambda(d, k, t)$ to be the highest possible value such that every $\Lambda(d, k, t)$-sparse $(d, k, t)$-CSP instance is satisfiable. Recall that $\Lambda(d, k, t) = |E(H^*)|/(|V(H^*)| - |I(H^*)|)$. Our intuition is that: the pair $(V(H^*) - I(H^*), E(H^*))$ should be an extreme case with some characteristic structure such that if a token function $g(c)$ satisfies the two claims in a valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$, then it will satisfy the two claims in every valid $(\alpha, \beta)$-token allocation on any
subsets of variables and constraints of any \((d, k, t)\)-CSP instance as well.

We will first construct the unique valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\) in Section 6.2. Then, we will design the token function in Section 6.3, so that Claim 4.24 and Claim 4.25 hold for this valid \((\alpha, \beta)\)-token allocation. Finally, we will prove in Section 6.4 and Section 6.5 that this token function satisfies these two claims in every valid \((\alpha, \beta)\)-token allocation.

6.2 A Valid \((\alpha, \beta)\)-Token Allocation on \((V(H^*) - I(H^*), E(H^*))\)

In the following, we will design a valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\). Recall that in a valid \((\alpha, \beta)\)-token allocation, each variable gives out \textbf{at most} \(\alpha\) tokens, and each constraint receives exactly \(\beta\) tokens.

\textbf{Remark 6.1.} The following valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\) is unique, which is a consequence of the pair \((V(H^*) - I(H^*), E(H^*))\) being an extreme case. This property is not hard to verify. However, we consider a valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\) in this chapter only because we believe it will give us some clues in the design of the token function \(g(c)\). The token allocation itself is not used explicitly when we prove the token function \(g(c)\) satisfies Claim 4.24 and 4.25. Thus, this uniqueness property will not be proved formally.

6.2.1 CASE 1: \(r_1 = 1\)

Consider the construction of \(H^*\) presented in Section 5.2.1. There is only one variable in \(V(H^*) - I(H^*)\), the output variable \(u\); and there are \(x_1\) different \(t\)-forbidding edges in \(E(H^*)\). Trivially, in every valid \((\alpha, \beta)\)-token allocation, each \(t\)-forbidding edge receives \(\beta\) tokens from \(u\); and \(u\) gives out \(x_1 \cdot \beta\) tokens in total. By (5.4.2), \(\alpha = x_1, \beta = 1\). Thus, the only variable \(u\) gives out \(x_1 \cdot 1 = \alpha\) tokens in total.

6.2.2 CASE 2: \(1 < r_1 \leq t_2\)

Consider the construction of \(H^*\) presented in Section 5.2.2.

\textbf{t-Forbidding Edges.} Each \(t\)-forbidding edge \(e\) has only one variable in \(V(H^*) - I(H^*)\) (the other variable is in \(I(H^*)\)). Thus, in the valid \((\alpha, \beta)\)-token allocation, \(e\) must receive \(\beta\) tokens from its only variable in \(V(H^*) - I(H^*)\).

\textbf{r_1-Free Variables.} Each \(r_1\)-free variable \(v\) is the output variable of \(x_1\) different \(t\)-forbidders. Thus, in the valid \((\alpha, \beta)\)-token allocation, \(v\) must give \(\beta\) tokens to each of the \(x_1\) \(t\)-forbidding edges (in the \(t\)-forbidders). By (5.4.4), \(\alpha = x_1 \cdot \beta + (r_1 - 1)\). Hence, there are \(r_1 - 1\) tokens remaining at \(v\).
Chapter 6. The Token Function $g(c)$

$t_2$-Forbidding Edges. Each $t_2$-forbidding edge $e$ has two variables: (i) the output variable $u$ of the $t_2$-forbidder containing $e$, and (ii) the $r_1$-free variable $w$ in the $t_2$-forbidder containing $e$. Since there are $r_1 - 1$ tokens remaining at the $r_1$-free variable, $e$ can get $r_1 - 1$ tokens from $w$. By (5.4.3), $\beta = 2r_1 - 1$. Hence, $e$ demands $r_1$ tokens from the output variable $u$.

$(r_1 + t - t_2)$-Free Variables. Each $(r_1 + t - t_2)$-free variable $v$ is the output variable of $x_1 - 1$ different $t$-forbidders and one $t_2$-forbidder. Thus, in the valid $(\alpha, \beta)$-token allocation, $v$ must give $\beta$ tokens to each of the $x_1 - 1$ $t$-forbidding edges (in the $t$-forbidders), and $r_1$ tokens to the only $t_2$-forbidding edge (in the $t_2$-forbidder). By (5.4.4), $\alpha = x_1 \cdot \beta + (r_1 - 1)$. Hence, there are $\beta + (r_1 - 1) - r_1 = \beta - 1$ tokens remaining at $v$.

1-Forbidding Edges. Each 1-forbidding edge $e$ has two variables: (i) the output variable $u$ of the 1-forbidder containing $e$, and (ii) the $(r_1 + t - t_2)$-free variable $w$ in the 1-forbidder containing $e$. Since there are $\beta - 1$ tokens remaining at an $(r_1 + t - t_2)$-free variable, $e$ can get $\beta - 1$ tokens from $w$. Thus, $e$ demands one token from the output variable $u$.

The Output Variable of $H^*$. The output variable $v$ of $H^*$ is the output variable of $x_1$ different $t$-forbidders and $r_1 - 1$ different 1-forbidders. Thus, in the valid $(\alpha, \beta)$-token allocation, $v$ must give $\beta$ tokens to each of the $x_1$ $t$-forbidding edges (in the $t$-forbidders), and one token to each of the $r_1 - 1$ 1-forbidding edges (in the 1-forbidders). By (5.4.4), $\alpha = x_1 \cdot \beta + (r_1 - 1)$. Thus, $v$ gives out exactly $\alpha$ tokens in total.

6.2.3 CASE 3: $r_1 > t_2$

Consider the construction of $H^*$ presented in Section 5.2.3. The valid $(\alpha, \beta)$-token allocation is similar to the one in CASE 2.

$t$-Forbidding Edges. Each $t$-forbidding edge $e$ has only one variable in $V(H^*) - I(H^*)$ (the other variable is in $I(H^*)$). Thus, in the valid $(\alpha, \beta)$-token allocation, $e$ must receive $\beta$ tokens from its only variable in $V(H^*) - I(H^*)$.

$r_1$-Free Variables. Each $r_1$-free variable $v$ is the output variable of $x_1$ different $t$-forbidders. Thus, in the valid $(\alpha, \beta)$-token allocation, $v$ must give exactly $\beta$ tokens to each of the $x_1$ $t$-forbidding edges (in the $t$-forbidders). By (5.4.7), $\alpha = x_1 \cdot \beta + m_1$. Hence, there are $m_1$ tokens remaining at $v$.

$t_2$-Forbidding Edges. Each $t_2$-forbidding edge $e$ has two variables: (i) the output variable $u$ of the $t_2$-forbidder containing $e$, and (ii) the $r_1$-free variable $w$ in the $t_2$-forbidder containing $e$. Since there are $m_1$ tokens remaining at an $r_1$-free variable, $e$ can get $m_1$ tokens from $w$. By (5.4.6), $\beta = m_1 + r_2$. Hence, $e$ demands $r_2$ tokens from the output variable $u$. 
**Chapter 6. The Token Function $g(c)$**

$r_2$-Free Variable. Each $r_2$-free variable $v$ is the output variable of $x_1$ different $t$-forbidders and $x_2$ different $t_2$-forbidders. Thus, in the valid $(\alpha, \beta)$-token allocation, $v$ must give $\beta$ tokens to each of the $x_1$ $t$-forbidding edges (in the $t$-forbidders), and $r_2$ tokens to each of the $x_2$ $t_2$-forbidding edges (in the $t_2$-forbidders). By (5.4.7) and (5.4.5), $\alpha = x_1 \cdot \beta + m_1$ and $m_1 = x_2 \cdot r_2 + (r_2 - 1)$. Hence, there are $(r_2 - 1)$ tokens remaining at $v$.

$(r_2 + t - t_2)$-Free Variables. Each $(r_2 + t - t_2)$-free variable $v$ is the output variable of $x_1 - 1$ different $t$-forbidders and $x_2 + 1$ different $t_2$-forbidders. Thus, in the valid $(\alpha, \beta)$-token allocation, $v$ must give $\beta$ tokens to each of the $x_1 - 1$ $t$-forbidding edges (in the $t$-forbidders), and $r_2$ tokens to each of the $x_2 + 1$ $t_2$-forbidding edges (in the $t_2$-forbidders). By (5.4.7) and (5.4.5), $\alpha = x_1 \cdot \beta + m_1$ and $m_1 = x_2 \cdot r_2 + (r_2 - 1)$. Hence, there are $\beta + (r_2 - 1) - r_2 = \beta - 1$ tokens remaining at $v$.

1-Forbidding Edges. Each 1-forbidding edge $e$ has two variables: (i) the output variable $u$ of the 1-forbidder containing $e$, and (ii) the $(r_2 + t - t_2)$-free variable $w$ in the 1-forbidder containing $e$. Since there are $\beta - 1$ tokens remaining at an $(r_2 + t - t_2)$-free variable, $e$ can get $\beta - 1$ tokens from $w$. Thus, $e$ demands one token from the output variable $u$.

**The Output Variable of $H^*$**. The output variable $v$ of $H^*$ is the output variable of $x_1$ different $t$-forbidders, $x_2$ different $t_2$-forbidders and $r_2 - 1$ different 1-forbidders. Thus, in the valid $(\alpha, \beta)$-token allocation, $v$ must give $\beta$ tokens to each of the $x_1$ $t$-forbidding edges (in the $t$-forbidders), $r_2$ tokens to each of the $x_2$ $t_2$-forbidding edges (in the $t_2$-forbidders), and one token to each of the $r_2 - 1$ 1-forbidding edges (in the 1-forbidders). By (5.4.7) and (5.4.5), $\alpha = x_1 \cdot \beta + m_1$ and $m_1 = x_2 \cdot r_2 + (r_2 - 1)$. Thus, $v$ gives out exactly $\alpha$ tokens in total.

### 6.3 The Design of the Token Function $g(c)$

In this section, we will design the token function $g(c)$ used in the algorithm TA-SA. For reference, we repeat the algorithm here in Figure 6.1.

Note that in the algorithm TA-SA, $g(c(e,v))$ is the number of values we remove in $D_v$ when a variable $v$ gives $c(e,v)$ tokens to a constraint $e$. Our goal is to design a token function $g(c)$ which satisfies the two required properties in every valid $(\alpha, \beta)$-token allocation on any variable set $U$ and any constraint set $F$ of any $(d, k, t)$-CSP instance: i.e.

$$\sum_e g(c(e,v)) \leq d - 1 \text{ for every variable } v \in U \quad \text{(Claim 4.24)}$$

$$\prod_v (g(c(e,v)) + 1) > t \text{ for every constraint } e \in F \quad \text{(Claim 4.25)}$$

We believe the structure $(V(H^*) - I(H^*), E(H^*))$ is an extreme case, such that a token function satisfying the two claims in a valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$ will also satisfy the two
claims in every valid \((\alpha,\beta)\)-token allocation on any subsets of variables and constraints of any \((d,k,t)\)-CSP instance. Thus, we are going to design our token function \(g(c)\) to satisfy the two claims in the valid \((\alpha,\beta)\)-token allocation on \((V(H^*), I(H^*), E(H^*))\) presented in the previous section. This valid \((\alpha,\beta)\)-token allocation is unique, as mentioned in Remark 6.1.

Note that each constraint receives at most \(\beta\) tokens from a variable. So, we only have to define \(g(c)\) for \(c \in [0, \beta]\).

Recall that \(g(c_{(e,v)})\) is the number of values the algorithm removes in \(D_v\) when a variable \(v\) gives \(c_{(e,v)}\) tokens to a constraint \(e\). If a variable gives no token to a constraint, then the algorithm does not remove any values at the variable. Hence, we set \(g(0) = 0\).

**Remark 6.2.** Intuitively, the more tokens a variable gives to a constraint, the more values the algorithm will remove at the variable for that constraint. Thus, we will ensure that \(g(b) \geq b\) for every \(b \in [0, \beta]\).

**Remark 6.3.** The algorithm DKT (shown in Figure 4.2) is similar to the algorithm TA-SA (shown in Figure 6.1) with a simple token function \(g(b) = b\) for every \(b \in [0, \beta]\). Thus, we can regard TA-SA as an modification of DKT that uses a much better token function.

\[
\begin{align*}
\text{TA-SA}(U, F, C = \{c_{(e,v)}\}) \\
(1) & \text{ For each } v \in U: \\
(2) & \text{ Let } D_v = \{1, \ldots, d\} \text{ be the set of values the algorithm may assign to } v. \\
(3) & \text{ For each } e \in F \text{ that } v \text{ appears in:} \\
(4) & \text{ Sort } D_v = \{d_i\} \text{ in decreasing order by the frequency of } (v : d_i) \text{ appearing in the active restrictions of } e, \text{ then remove the first } g(c_{(e,v)}) \text{ values in } D_v. \text{ In other words, remove the } g(c_{(e,v)}) \text{ values at } D_v \text{ that are most frequently forbidden by the active restrictions of } e. \\
(5) & \text{ Remove all but one value from } D_v, \text{ and assign the value remaining to } v. \\
(6) & \text{ Return the value assignment.}
\end{align*}
\]

Figure 6.1: An algorithm constructing a satisfying assignment from a valid \((\alpha,\beta)\)-token allocation.

### 6.3.1 CASE 1: \(r_1 = 1\)

Consider the construction of \(H^*\) presented in Section 5.2.1, and the valid \((\alpha,\beta)\)-token allocation presented in Section 6.2.1. Each \(t\)-forbidding edge forbids \(t\) values (i.e. removes \(t\) values) at the output variable \(u\); and receives \(\beta = 1\) token from \(u\) in the valid \((\alpha,\beta)\)-token allocation. Thus, we set \(g(1) = t\). Together, we have:

\[
\begin{align*}
g(0) &= 0 \\
g(1) &= t
\end{align*}
\]

This token function satisfies Claim 4.24 and 4.25 in every valid \((\alpha,\beta)\)-token allocation on any subsets of variables and constraints of any \((d,k,t)\)-CSP instance, as we will see in the proofs in Section 6.4 and Section 6.5.
6.3.2 CASE 2: $1 < r_1 \leq t_2$

Consider the construction of $H^*$ presented in Section 5.2.2, and the valid $(\alpha, \beta)$-token allocation presented in Section 6.2.2. First, we will figure out the values of $g(c)$ for $c \in \{\beta, r_1, r_1 - 1\}$, which is the set of values used in the valid $(\alpha, \beta)$-token allocation.

Each $t$-forbidding edge forbids $t$ values (i.e. removes $t$ values) at the output variable $u$; and receives $\beta$ tokens from $u$ in the valid $(\alpha, \beta)$-token allocation. Thus, we set $g(\beta) = t$.

For each $r_1$-free variable $v$, $t$ values are forbidden (i.e. $t$ values are removed) by each of the $x_1$ different $t$-forbidding edges. There are $r_1$ values remaining. On the other hand, in the valid $(\alpha, \beta)$-token allocation, $v$ has $r_1 - 1$ tokens remaining after giving tokens to the $t$-forbidding edges.

Each $t_2$-forbidding edge $e$ forbids $t_2$ values (i.e. removes $t_2$ values) at the output variable $u$; and receives $r_1$ tokens from $u$ in the valid $(\alpha, \beta)$-token allocation. Thus, we set $g(r_1) = t_2$. The other variable in the constraint is an $r_1$-free variable $v$. There are $r_1$ values remaining at $v$, so the algorithm TA-SA can remove $r_1 - 1$ of them for $e$; and $v$ gives $r_1 - 1$ tokens to $e$ in the valid $(\alpha, \beta)$-token allocation. Thus, we set $g(r_1 - 1) = r_1 - 1$.

Together, we have:

\begin{align*}
g(r_1 - 1) &= r_1 - 1 \\
g(r_1) &= t_2 \\
g(\beta) &= t
\end{align*}

(6.3.3)  
(6.3.4)  
(6.3.5)

This gives a token function satisfying Claim 4.24 and 4.25 for the valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$.

**The general token function.** Now, we will decide the value of $g(c)$ for other values of $c$. These values are less sensitive, since they do not arise in the extreme case of $(V(H^*) - I(H^*), E(H^*))$. We will simply set those values of $g(c)$ such that $g(c)$ grows as linearly as possible between the special points of $c \in \{r_1 - 1, r_1, \beta\}$. In fact, these values are obtained though trial and error, when we were working on the proofs of Claim 4.24 and 4.25.

\begin{align*}
g(b) &= b & \text{for } b \in [0, r_1 - 1) \\
g(r_1 + b) &= t_2 + b \cdot \left\lfloor \frac{t_2}{t - r_1} \right\rfloor & \text{for } r_1 + b \in (r_1, \beta) \text{ and } b > 0
\end{align*}

(6.3.6)  
(6.3.7)

Together, we have the following token function $g(c)$ for every $c \in [0, \beta]$. We will prove in Section 6.4 and Section 6.5 that this token function satisfies both Claim 4.24 and 4.25 for every valid $(\alpha, \beta)$-token...
allocation on any subsets of variables and constraints of any \((d, k, t)\)-CSP instance.

\[
g(b) = b \\
g(r_1 + b) = t_2 + b \cdot \left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor \\
g(\beta) = t
\]

\text{for } b \in [0, r_1) \quad \text{(6.3.8)}
\text{for } r_1 + b \in [r_1, \beta) \text{ and } b \geq 0 \quad \text{(6.3.9)}
\text{(6.3.10)}

6.3.3 \text{ CASE 3: } r_1 > t_2

Consider the construction of \(H^*\) presented in Section 5.2.3, and the valid \((\alpha, \beta)\)-token allocation presented in Section 6.2.3. Similarly to CASE 2, we will first figure out the values of \(g(c)\) for \(c \in \{\beta, m_1, r_2, r_2 - 1\}\), which are the set of values used in the valid \((\alpha, \beta)\)-token allocation.

Each \(t\)-forbidding edge forbids \(t\) values (i.e. removes \(t\) values) at the output variable \(u\); and receives \(\beta\) tokens from \(u\) in the valid \((\alpha, \beta)\)-token allocation. Thus, we set \(g(\beta) = t\) (as in CASE 2).

For each \(r_1\)-free variable \(v\), \(t\) values are forbidden (i.e. \(t\) values are removed) by each of the \(x_1\) different \(t\)-forbidding edges. There are \(r_1\) values remaining. On the other hand, in the valid \((\alpha, \beta)\)-token allocation, \(v\) has \(m_1\) tokens remaining after giving tokens to the \(t\)-forbidding edges.

Each \(t_2\)-forbidding edge \(e\) forbids \(t_2\) values (i.e. removes \(t_2\) values) at the output variable \(u\); and receives \(r_2\) tokens from \(u\) in the valid \((\alpha, \beta)\)-token allocation. Thus, we set \(g(r_2) = t_2\) (as in CASE 2). The other variable \(e\) containing is an \(r_1\)-free variable \(v\). There are \(r_1\) values remaining at \(v\), so the algorithm \text{TA-SA} can remove \(r_1 - 1\) of them for \(e\); and \(v\) gives \(m_1\) tokens to \(e\) in the valid \((\alpha, \beta)\)-token allocation. Thus, we set \(g(m_1) = r_1 - 1\) (as in CASE 2).

For each \(r_2\)-free variable \(v\), \(t\) values are forbidden (i.e. \(t\) values are removed) by each of the \(x_1\) different \(t\)-forbidding edges, and \(t_2\) values are forbidden (i.e. \(t_2\) values are removed) by each of the \(x_2\) different \(t_2\)-forbidding edges. There are \(r_2\) values remaining, i.e. the algorithm can later remove \(r_2 - 1\) values at \(v\). On the other hand, in the valid \((\alpha, \beta)\)-token allocation, after giving tokens to the \(t\)-forbidding edges and \(t_2\)-forbidding edges, \(v\) has \(r_2 - 1\) tokens remaining. As mentioned in Remark 6.2, we enforce \(g(b) \geq b\) for every \(b \in [0, \beta]\). Hence, we set \(g(r_2 - 1) = r_2 - 1\).

Together, we have:

\[
g(r_2 - 1) = r_2 - 1 \\
g(r_2) = t_2 \\
g(m_1) = r_1 - 1 \\
g(\beta) = t
\]

\text{(6.3.11)} \quad \text{(6.3.12)} \quad \text{(6.3.13)} \quad \text{(6.3.14)}

This gives a token function satisfying Claim 4.24 and 4.25 for the valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\).
**Chapter 6. The Token Function \( g(c) \)**

The **general token function.** As in CASE 2, we will simply set the other values of \( g(c) \) in a way such that \( g(c) \) grows as linearly as possible between the special points of \( c \in \{r_2-1, r_2, m_1, \beta \} \). Again, these values are obtained though trial and error, when we were working on the proofs of Claim 4.24 and 4.25.

\[
\begin{align*}
g(b) &= b & \text{for } b \in [0, r_2 - 1) \\
g(ar_2 + b) &= a \cdot g(r_2) + b & \text{for } ar_2 + b \in (r_2, m_1) \text{ and } b \in (0, r_2) \\
g(m_1 + b) &= g(m_1) + b \cdot \left\lfloor \frac{t - r_1 + 1}{\beta - m_1} \right\rfloor & \text{for } m_1 + b \in (m_1, \beta) \text{ and } b \geq 0
\end{align*}
\]

Together, we have the following token function \( g(c) \) for every \( c \in [0, \beta] \). We will prove in Section 6.4 and Section 6.5 that this token function satisfies both Claim 4.24 and 4.25 for every valid \( (\alpha, \beta) \)-token allocation on any subsets of variables and constraints of any \( (d, k, t) \)-CSP instance.

\[
\begin{align*}
g(ar_2 + b) &= a \cdot t_2 + b & \text{for } an_2 + b \in [0, m_1] \text{ and } b \in [0, r_2) \\
g(m_1 + b) &= (r_1 - 1) + b \cdot \left\lfloor \frac{t - r_1 + 1}{\beta - m_1} \right\rfloor & \text{for } m_1 + b \in [m_1, \beta) \text{ and } b \geq 0 \\
g(\beta) &= t
\end{align*}
\]

For convenience, \( g(m_1) \) is defined twice in (6.3.18) and (6.3.19). This does not raise any issue since the values of \( g(m_1) \) in these two definitions coincide: (i) by (6.3.19), \( g(m_1) = r_1 - 1 \), which is \( x_2 \cdot t_2 + r_2 - 1 \) (by (5.2.5)); (ii) by (5.4.5), \( m_1 = x_2 \cdot r_2 + (r_2 - 1) \); by (6.3.18), \( g(x_2 \cdot r_2 + (r_2 - 1)) = x_2 \cdot t_2 + r_2 - 1 \) as well.

**6.4 Proof of Claim 4.24**

Consider a \( (d, k, t) \)-CSP instance \( P \), and a valid \( (\alpha, \beta) \)-token allocation \( C = \{c_{(e,v)}\} \) on \( (U \subseteq V(P), F \subseteq E(P)) \) such that \( \sum_v c_{(e,v)} \leq \alpha \) for every \( v \in U \) and \( \sum_e c_{(e,v)} = \beta \) for every \( e \in F \). In the following, we will prove that the token function we defined in Section 6.3 satisfies Claim 4.24, which states that

\[
\sum_v g(c_{(e,v)}) \leq d - 1 \text{ for every variable } v \in U.
\]

Note that each constraint receives at most \( \beta \) tokens from a variable. So, every \( c_{(e,v)} \leq \beta \).

The following are cases of different values of \( d, k, t \) since the values of \( r_1, t_2 \) are determined by the values of \( d, k, t \).
6.4.1 CASE 1: \( r_1 = 1 \)

This is an easy case. By (5.4.2), we have \( \beta = 1 \) and \( \alpha = x_1 \), so every \( c_{e,v} \) is either 1 or 0 in a valid \((\alpha, \beta)\)-token allocation. By (6.3.1) and (6.3.2) \( g(1) = t \) and \( g(0) = 0 \). Thus the sum \( \sum_e g(c_{e,v}) \) is always upper-bounded by \( \alpha \cdot t = x_1 \cdot t \), which is \( d - 1 \) by (5.2.2).

6.4.2 CASE 2: \( 1 < r_1 \leq t_2 \)

For any positive integer \( c \), define a token distribution of \( c \) tokens \( T_c = \{c_i\} \) to be a list of positive integers such that \( \sum_i c_i = c \). A token distribution is called sum-extreme if \( \sum_i g(c_i) \) is maximized. Recall that a variable gives out at most \( \alpha \) tokens in a valid token allocation. Given any sum-extreme token distribution \( T_c \) for any \( c < \alpha \), we can obtain a token distribution \( T_\alpha \) with a larger sum \( \sum_i g(c_i) \) by simply adding \( (\alpha - c) \) copies of 1. Therefore, the sum over a sum-extreme \( T_\alpha \) is larger than the sum over any \( T_c \) with \( c < \alpha \). Thus, it suffices to prove the claim by showing that the sum \( \sum_i g(c_i) \) over a sum-extreme \( T_\alpha \) is at most \( d - 1 \).

We will use the following property, which will be proved later in this section. Intuitively, this property follows from the fact that \( g(b)/b \) is maximized at \( b = \beta \).

Claim 6.4. For any integer \( c \in [\beta, \alpha] \), there exists a sum-extreme token distribution \( T_c \) such that \( \beta \in T_c \).

With \( c = \alpha \), Claim 6.4 implies that there exists a sum-extreme \( T_\alpha \) containing \( \beta \). Therefore, given any sum-extreme \( T_{\alpha-\beta} \), we can get a sum-extreme \( T_\alpha = \{\beta\} \cup T_{\alpha-\beta} \). If \( \alpha - \beta \geq \beta \), we can apply Claim 6.4 with \( c = \alpha - \beta \), to conclude that there exists a sum-extreme \( T_{\alpha-\beta} \) containing \( \beta \). This implies the existence of a sum-extreme \( T_\alpha \) where \( \beta \) appears twice. By (5.4.4), \( \alpha = x_1 \cdot \beta + (r_1 - 1) \). Hence, we can repeat this \( x_1 \) times and conclude that there exists a sum-extreme \( T_\alpha \) where \( \beta \) appears \( x_1 \) times, i.e. a sum-extreme \( T_\alpha = \{\beta, ..., \beta\} \cup T_{r_1-1} \).

Let \( T' \) be a sum-extreme \( T_\alpha = \{\beta, ..., \beta\} \cup T_{r_1-1} \), where \( \beta \) appears \( x_1 \) times. By (6.3.8), \( g(b) = b \) for \( b \in [0, r_1) \). So the \( r_1 - 1 \) tokens in \( T_{r_1-1} \) must contribute a total of \( r_1 - 1 \) to the sum \( \sum_i g(c_i) \). Therefore, the sum over \( T' \) is:

\[
\sum_i g(c_i) = x_1 \cdot g(\beta) + r_1 - 1 \\
= x_1 \cdot t + r_1 - 1 \quad \text{by (6.3.10)} \\
= d - 1 \quad \text{by (5.2.2)}
\]

This proves Claim 4.24. As we can see, the token allocation \( T' \) is basically the same as the way we distribute the tokens at \( u(H^*) \). This is consistent with our intuition that \((V(H^*) - I(H^*), E(H^*)) \) is an extreme case.

In the following, we will complete the proof of Claim 4.24 by proving Claim 6.4. To prove Claim 6.4, we need the following property.

Claim 6.5. \( \left| \frac{t - t_2}{\beta - r_1} \right| \geq t_2 \).
Proof. Since $t_2$ is an integer, it suffices to prove $\frac{t - t_2}{\beta - r_1} \geq t_2$. By (5.4.3), we have $\beta - r_1 = r_1 - 1$. Thus, $\frac{t - t_2}{\beta - r_1} = \frac{t - t_2}{r_1 - 1}$. By (5.2.3), we have $t \geq r_1 \cdot t_2$. Thus,

$$\frac{t - t_2}{\beta - r_1} = \frac{t - t_2}{r_1 - 1} \geq t_2 \cdot \frac{r_1 - 1}{r_1 - 1} = t_2.$$

$\Box$

It will be helpful to remember $1 < r_1 \leq t_2 < t$ and $r_1 < \beta$ - the inequality $r_1 < \beta$ follows from (5.4.3); $1 < r_1 \leq t_2$ follows from the setting of CASE 2; and $t_2 < t$ follows from (5.2.3).

**Proof of Claim 6.4:** For any $c \in [\beta, \alpha]$, we are going to prove that there exists a sum-extreme token distribution $T_c$ that contains $\beta$.

Consider a sum-extreme token distribution $T_c = \{c_1, c_2, ..., c_\ell\}$. Without loss of generality, assume $c_i \geq c_{i+1}$ for $i = 1, 2, ..., \ell - 1$. Since each constraint receives at most $\beta$ tokens from a variable, we can assume that $c_1 \leq \beta$. Furthermore, assume that (A) $T_c$ is a sum-extreme token distribution with the largest $c_1$ among all the sum-extreme token distributions.

Case (I), $c_1 = \beta$. Then Claim 6.4 holds.

Case (II), $c_1 \in [r_1, \beta)$. Since $c \geq \beta$ and $c_1 < \beta$, there exists at least one additional value $c_2 \in T_c - \{c_1\}$.

Case (IIa), $c_2 \in (r_1, \beta)$. By (6.3.9), $g(r_1 + b) = t_2 + b \cdot \left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor$ for $r_1 + b \in [r_1, \beta)$ and $b \geq 0$. By (6.3.10), $g(\beta) = t \geq t_2 + (\beta - r_1) \cdot \left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor$. Therefore, we can replace $c_1, c_2$ by $c_1 + 1$ and $c_2 - 1$ to obtain another sum-extreme token distribution with a member larger than $c_1$, which contradicts assumption (A).

Case (IIb) $c_2 = r_1$. By (6.3.9), $g(r_1 + b) = t_2 + b \cdot \left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor$ for $r_1 + b \in [r_1, \beta)$ and $b \geq 0$. By (6.3.10), $g(\beta) = t \geq t_2 + (\beta - r_1) \cdot \left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor$. By Claim 6.5, $\left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor \geq t_2 = g(r_1)$ Thus, $g(c_1 + 1) + g(c_2 - 1) \geq g(c_1) + g(c_2)$. This implies that we can replace $c_1, c_2$ by $c_1 + 1$ and $c_2 - 1$ to obtain another sum-extreme token distribution with a member larger than $c_1$, which contradicts assumption (A).

Case (IIc), $c_2 \in [1, r_1)$. By (6.3.8), $g(b) = b$ for $b \in [0, r_1)$. By (6.3.9), $g(r_1 + b) = t_2 + b \cdot \left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor$ for $r_1 + b \in [r_1, \beta)$ and $b \geq 0$. By (6.3.10), $g(\beta) = t \geq t_2 + (\beta - r_1) \cdot \left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor$. By Claim 6.5, $\left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor \geq t_2$, which implies $\left\lfloor \frac{t - t_2}{\beta - r_1} \right\rfloor \geq 1$. Hence, we can replace $c_1, c_2$ by $c_1 + 1$ and $c_2 - 1$ to obtain another sum-extreme token distribution with a member larger than $c_1$, which contradicts assumption (A).

Case (III), $c_1 \in (0, r_1)$. Since $c \geq \beta > r_1$ and $c_1 < r_1$, there is at least one additional value $c_2 \in T_c - \{c_1\}$. By (6.3.8), $g(b) = b$ for $b \in [0, r_1)$. By (6.3.9), we have $g(r_1) = t_2$. We have $r_1 \leq t_2$ in the setting of CASE 2. Therefore, we can replace $c_1, c_2$ by $c_1 + 1$ and $c_2 - 1$ to obtain another sum-extreme token distribution with a member larger than $c_1$, which contradicts assumption (A).

Combining Case (I), (II) and (III), we can conclude that $c_1 = \beta$, i.e. there exists a sum-extreme token distribution $T_c$ that contains $\beta$. $\Box$
6.4.3 CASE 3: $r_1 > t_2$

The proof is similar to the one in CASE 2. For any positive integer $c$, define a token distribution of $c$ tokens $T_c = \{c_i\}$ to be a list of positive integers such that $\sum_i c_i = c$. A token distribution is called sum-extreme if $\sum_i g(c_i)$ is maximized. Recall that a variable gives out at most $\alpha$ tokens in a valid $(\alpha, \beta)$-token allocation. Given any sum-extreme token distribution $T_c$ for any $c < \alpha$, we can obtain a token distribution $T_\alpha$ with a larger sum $\sum_i g(c_i)$ by simply adding $(\alpha - c)$ copies of 1. Therefore, the sum over a sum-extreme $T_\alpha$ is larger than the sum over any $T_c$ with $c < \alpha$. Thus, it suffices to prove the claim by showing that the sum $\sum_i g(c_i)$ over a sum-extreme token distribution $T_\alpha$ is at most $d - 1$.

We will use the following two properties, which will be proved later in this section. Intuitively, these properties follow from the fact that $g(b)/b$ is maximized at $b = \beta$ for $b \in [1, \beta]$.

**Claim 6.6.** For any integer $c \in [r_2, m_1]$, there exists a sum-extreme token distribution $T_c$ such that $r_2 \in T_c$.

**Claim 6.7.** For any integer $c \in [\beta, \alpha]$, there exists a sum-extreme token distribution $T_c$ such that $\beta \in T_c$.

With $c = \alpha$, we can apply Claim 6.7 to conclude that there exists a sum-extreme $T_\alpha$ containing $\beta$. Therefore, given any sum-extreme $T_{\alpha-\beta}$, we can get a sum-extreme $T_\alpha = \{\beta\} \cup T_{\alpha-\beta}$. If $\alpha - \beta \geq \beta$, we can apply Claim 6.7 with $c = \alpha - \beta$, to conclude that there exists a sum-extreme $T_{\alpha-\beta}$ containing $\beta$. This implies the existence of a sum-extreme $T_\alpha$ where $\beta$ appears twice. By (5.4.7), $\alpha = x_1 \cdot \beta + m_1$.

Hence, we can repeat this for $x_1$ times and conclude that there exists a sum-extreme $T_\alpha$ where $\beta$ appears $x_1$ times, i.e. a sum-extreme $T_\alpha = \{\beta, \ldots, \beta\} \cup T_{m_1}$.

Similarly, we can apply Claim 6.6 with $c = m_1$ to conclude that there exists a sum-extreme $T_{m_1}$ containing $r_2$. By (5.4.5), $m_1 = x_2 \cdot r_2 + (r_2 - 1)$. Hence, we can repeat this for $x_2$ times to conclude that there exists a sum-extreme $T_{m_1}$ where $r_2$ appears $x_2$ times, i.e. a sum-extreme $T_{m_1} = \{r_2, \ldots, r_2\} \cup T_{r_2-1}$. Thus, there exists a sum-extreme $T_\alpha$ where $\beta$ appears $x_1$ times and $r_2$ appears $x_2$ times, i.e. a sum-extreme $T_\alpha = \{\beta, \ldots, \beta\} \cup \{r_2, \ldots, r_2\} \cup T_{r_2-1}$.

Let $T'$ be a sum-extreme $T_\alpha$ where $\beta$ appears $x_1$ times and $r_2$ appears $x_2$ times. There are $r_2 - 1$ tokens remaining. By (6.3.18), $g(b) = b$ for $b \in [0, r_2)$.

Hence, these $r_2 - 1$ tokens in $T_{r_2-1}$ must contribute a total of $r_2 - 1$ to the sum $\sum_i g(c_i)$. Therefore, the sum over $T'$ is:

$$\sum_i g(c_i) = x_1 \cdot g(\beta) + x_2 \cdot g(r_2) + r_2 - 1$$

$$= x_1 \cdot t + x_2 \cdot t_2 + r_2 - 1$$

$$= x_1 \cdot t + r_1 - 1$$

$$= d - 1$$

by (6.3.20) and (6.3.18)

by (5.2.5)

by (5.2.2)

This proves Claim 4.24. As in CASE 2, the token allocation $T'$ is basically the same as the way we distribute the tokens at $u(H^*)$. This is consistent with our intuition that $(V(H^*) - I(H^*), E(H^*))$ is an extreme case.
In the following, we will complete the proof of Claim 4.24 by proving Claim 6.6 and Claim 6.7. To prove Claim 6.7, we need the following property.

**Claim 6.8.** \[ \left\lfloor \frac{t-r_1+1}{\beta-m_1} \right\rfloor \geq t_2. \]

**Proof.** Since \( t_2 \) is an integer, it suffices to prove \( \frac{t-r_1+1}{\beta-m_1} \geq t_2 \). By (5.4.6), \( \beta = m_1 + n_2 \), so we have \( \frac{t-r_1+1}{\beta-m_1} = \frac{t-r_1+1}{r_2} \).

When \( t_2 = 1 \), we have \( r_2 = 1 \) (by (5.2.4) and (5.2.5)). Thus, \( \frac{t-r_1+1}{r_2} \geq 1 = t_2 \).

On the other hand, when \( t_2 > 1 \), we have

\[
\frac{t-r_1+1}{r_2} \geq \frac{t_2 \cdot r_1 - r_1 + 1}{r_2} \quad \text{(by (5.2.3))}
\]
\[
> \frac{r_1(t_2-1)}{n_2}
\]
\[
= \frac{(x_2 \cdot t_2 + r_2)(t_2-1)}{r_2} \quad \text{(by (5.2.5))}
\]
\[
\geq (x_2 + 1)(t_2-1) \quad (t_2 \geq r_2 \text{ by (5.2.4) and (5.2.5)})
\]
\[
\geq t_2 \quad (\text{since } x_2 \geq 1 \text{ and } t_2 > 1)
\]

It will be helpful to remember that \( 1 \leq r_2 \leq t_2 < r_1 \leq \beta \) and \( r_2 < m_1 < \beta \) - the inequality \( 1 \leq r_2 \leq t_2 \) follows from (5.2.2), (5.2.4) and (5.2.5); \( t_2 < r_1 \) follows from the setting of CASE 3; \( r_1 \leq \beta \) follows from (5.2.1) and (5.2.2); and \( r_2 < m_1 < \beta \) (5.4.5) and (5.4.6).

**Proof of Claim 6.6:** For any \( c \in [r_2, m_1] \), we are going to prove that there exists a sum-extreme token distribution \( T_c \) that contains \( r_2 \).

Consider a sum-extreme token distribution \( T_c = \{c_1, c_2, \ldots, c_l\} \). Without loss of generality, assume \( c_i \geq c_{i+1} \) for \( i = 1, 2, \ldots, l-1 \). Since each constraint receives at most \( \beta \) tokens from a variable, we can assume that \( c_1 \leq \beta \). Furthermore, assume that (A) \( T_c \) is a sum-extreme token distribution with the largest \( c_1 \) among all the sum-extreme token distributions.

Case (I), \( c_1 = r_2 \). Then there exists a sum-extreme token distribution \( T_c \) that contains \( r_2 \).

Case (II), \( c_1 \in (r_2, m_1] \). Let \( c_1 = an_2 + b \). By (6.3.18), \( g(ar_2 + b) = a \cdot t_2 + b \) for \( an_2 + b \in [0, m_1] \) and \( b \in [0, n_2) \). In other words, we can decompose \( c_1 \) to \( a \) copies of \( r_2 \) and one copy of \( b \) while keeping the token distribution sum-extreme. This implies that there exists a sum-extreme token distribution \( T_c \) that contains \( r_2 \).

Case (III), \( c_1 \in (0, r_2) \). Since \( c \in [r_2, m_1] \) and \( c_1 < r_2 \), there is at least one additional value \( c_2 \in T_c - \{c_1\} \).

By (5.2.4) and (5.2.5), \( t_2 \geq r_2 \), which implies \( t_2 \geq r_2 \). By (6.3.18), \( g(r_2) = t_2 \), and \( g(b) = b \) for every \( b < r_2 \). Therefore, we can replace \( c_1, c_2 \) by \( c_1 + 1 \) and \( c_2 - 1 \) to obtain another sum-extreme token distribution with a member larger than \( c_1 \), which contradicts assumption (A).

Combining Case (I), (II) and (III), we can conclude that there exists a sum-extreme token distribution
Chapter 6. The Token Function \( g(c) \)

\( T_c \) that contains \( r_2 \).

**Proof of Claim 6.7:** For any \( c \in [\beta, \alpha] \), we are going to prove that there exists a sum-extreme token distribution \( T_c \) that contains \( \beta \).

Consider a sum-extreme token distribution \( T_c = \{c_1, c_2, \ldots, c_\ell\} \). Without loss of generality, assume \( c_i \geq c_{i+1} \) for \( i = 1, 2, \ldots, \ell - 1 \). Since each constraint receives at most \( \beta \) tokens from a variable, we can assume that \( c_1 \leq \beta \). Furthermore, assume that (A) \( T_c \) is a sum-extreme token distribution with the largest \( c_1 \) among all the sum-extreme token distributions.

Case (I), \( c_1 = \beta \). Then there exists a sum-extreme token distribution \( T_c \) that contains \( \beta \).

Case (II), \( c_1 \in (m_1, \beta) \). Since \( c_1 < \beta \) and \( c \geq \beta \), there is at least one additional value \( c_2 \in T_c - \{c_1\} \).

Case (IIa), \( c_2 \in (m_1, \beta) \). By (6.3.19), \( g(m_1 + b) = (r_1 - 1) + b \cdot \left\lfloor \frac{t - r_1 + 1}{m_1 - x_2} \right\rfloor \) for \( m_1 + b \in [m_1, \beta] \) and \( b \geq 0 \). By (6.3.20), \( g(\beta) = t \). Therefore, we can replace \( c_1, c_2 \) by \( c_1 + 1 \) and \( c_2 - 1 \) to obtain another sum-extreme token distribution with a member larger than \( c_1 \), which contradicts assumption (A).

Case (IIb), \( c_2 \in (0, r_2) \). By (6.3.19), \( g(r_2 + b) = (x_2 \cdot r_2 + r_2 - 1) + r_2 \cdot \left\lfloor \frac{t - r_1 + 1}{m_1 - x_2} \right\rfloor \) for \( r_2 + b \in [0, r_2] \) and \( b \in [0, r_2) \). In other words, we can decompose \( c_2 \) to \( a \) copies of \( r_2 \) and one copy of \( b \) while keeping the token distribution sum-extreme. Therefore, we can assume that \( c_2 \notin (n_2, m_1] \). This reduce Case (IIb) to Case (IIc).

Case (IIc), \( c_2 \in (0, r_2) \). Let \( c_2 = a \cdot r_2 + b \) for some integer \( a > 0 \). By (6.3.18), \( g(ar_2 + b) = a \cdot t_2 + b \cdot \left\lfloor \frac{t - r_1 + 1}{m_1 - x_2} \right\rfloor \) for \( ar_2 + b \in [0, m_1] \) and \( b \in [0, r_2) \). In other words, we can decompose \( c_2 \) to \( a \) copies of \( r_2 \) and one copy of \( b \) while keeping the token distribution sum-extreme. Therefore, we can assume that \( c_2 \notin (n_2, m_1] \). This reduce Case (IIb) to Case (IIc).

Case (III), \( c_1 \in (0, m_1] \). Then \( c_1' \in (0, m_1] \) for every \( c_1' \in T_c' \) since \( c_1 \) is the maximum value in \( T_c \). By Claim 6.6, there exists some sum-extreme \( T_c' \) containing \( r_2 \); and similarly, there exists some sum-extreme \( T_{c-r_2} \) containing \( r_2 \). Note that

\[
\beta = m_1 + r_2 \quad \text{(by (5.4.6))} \\
= (x_2 \cdot r_2 + r_2 - 1) + r_2 \quad \text{(by (5.4.5))} \\
= (x_2 + 1) \cdot r_2 + (r_2 - 1)
\]

By applying Claim 6.6 \((x_2 + 1)\) times, we can conclude that there exists some sum-extreme \( T'_c \) where \( r_2 \).
appears at least $x_2 + 1$ times. Now, note that

$$(x_2 + 1) \cdot r_2 = x_2 \cdot r_2 + (r_2 - 1) + 1$$

$$= m_1 + 1$$  \hspace{1cm} \text{(by (5.4.5))}

$$g(m_1 + 1) = r_1 - 1 + \left\lceil \frac{t - r_1 + 1}{\beta - m_1} \right\rceil$$  \hspace{1cm} \text{(by (6.3.19))}

$$\geq r_1 - 1 + t_2$$  \hspace{1cm} \text{(by Claim 6.8)}

$$= x_2 \cdot t_2 + r_2 - 1 + t_2$$  \hspace{1cm} \text{(by (5.2.5))}

$$\geq (x_2 + 1) \cdot t_2$$

$$= (x_2 + 1) \cdot g(n_2)$$  \hspace{1cm} \text{(by (6.3.19))}

Hence, we can replace $x_2 + 1$ copies of $n_2$ by $m_1 + 1$ while keeping the token distribution sum-extreme. In other words, there exists a sum-extreme token distribution with a member larger than $c_1$ which contradicts assumption (A).

Combining Case (I), (II) and (III), we can conclude that there exists a sum-extreme token distribution $T_c$ that contains $\beta$. 

\[\square\]

### 6.5 Proof of Claim 4.25

Consider a $(d, k, t)$-CSP instance $P$, and a valid $(\alpha, \beta)$-token allocation $C = \{c_{(e,v)}\}$ on $(U \subseteq V(P), F \subseteq E(P))$ such that $\sum_{v} c_{(e,v)} \leq \alpha$ for every $v \in U$ and $\sum_{e} c_{(e,v)} = \beta$ for every $e \in F$. In the following, we will prove that the token function we defined in Section 6.3 satisfies Claim 4.25 which states that

$$\prod_{e} (g(c_{(e,v)}) + 1) > t$$ for every constraint $e \in F$.

When $k = 2$, each constraint contains exactly two variables. It suffices to prove

$$(g(c_1) + 1) \cdot (g(c_2) + 1) > t,$$ for any positive integers $c_1 \geq c_2$ such that $c_1 + c_2 = \beta$.

The following are cases of different values of $d, k, t$ since the values of $r_1, t_2$ are determined by the values of $d, k, t$.

#### 6.5.1 Case 1: $r_1 = 1$

We have $\beta = 1$, so each constraint receives a token from one of the two variables. By (6.3.2) and (6.3.1),

$$(g(1) + 1) \cdot (g(0) + 1) = (t + 1) \cdot 1 > t.$$
6.5.2 **CASE 2**: $1 < r_1 \leq t_2$

Recall that our goal is to prove $(g(c_1) + 1) \cdot (g(c_2) + 1) > t$, for any positive integers $c_1 \geq c_2$ such that $c_1 + c_2 = \beta$.

It will be helpful to remember $1 < r_1 \leq t_2 < t$ and $r_1 < \beta$ - the inequality $r_1 < \beta$ follows from (5.4.3); $1 < r_1 \leq t_2$ follows from the setting of CASE 2; and $t_2 < t$ follows from (5.2.3).

**Case A**, $c_1 = \beta$. Then $c_2 = 0$. By (6.3.10) and (6.3.8),

$$ (g(c_1) + 1) \cdot (g(c_2) + 1) = (t + 1) \cdot 1 > t. $$

**Case B**, $c_1 = r_1$. By (5.4.3), $\beta = 2r_1 - 1$, so we have $c_2 = r_1 - 1$. Then by (6.3.8) and (6.3.9),

$$ (g(c_1) + 1) \cdot (g(c_2) + 1) = (t_2 + 1) \cdot r_1 > t. $$

**Case C**, $c_1 \in (n_2, \beta)$. Let $c_1 = r_1 + b$ for some integer $b \geq 1$. By (5.4.3), $\beta = 2r_1 - 1$, so we have $c_2 = r_1 - 1 - b \geq 1$, which implies $r_1 \geq 3$. Then

$$ (g(c_1) + 1) \cdot (g(c_2) + 1) = (t_2 + b \cdot \lfloor \frac{t - t_2}{\beta - r_1} \rfloor + 1) \cdot (r_1 - b) \quad \text{(by (6.3.9) and (6.3.8))} $$

$$ \geq (t_2 + b \cdot t_2 + 1) \cdot (r_1 - b) \quad \text{(by Claim 6.5)} $$

$$ = ((b + 1)t_2 + 1) \cdot (r_1 - b) $$

$$ > (b + 1)(r_1 - b)t_2 $$

which is minimized when $b = r_1 - 2$ or $b = 1$ - a simple analysis show that the function $f(b) = (b+1)(r_1-b)$ is a convex function with the maximum at $b = (r_1 - 1)/2$, i.e. the minimum is at either endpoint of the range. At both endpoints, we have $(b + 1)(r_1 - b) = 2 \cdot (r_1 - 1)$. As shown earlier, $r_1 \geq 3$, so we have $(b + 1)(r_1 - b) \geq r_1 + 1$. Thus,

$$ (g(c_1) + 1) \cdot (g(c_2) + 1) > (b + 1)(r_1 - b)t_2 $$

$$ \geq (r_1 + 1) \cdot t_2 $$

$$ > t \quad \text{(by (5.2.3))} $$

**Case D**, $c_1 < r_1$. By (5.4.3), $\beta = 2r_1 - 1$, so we have $c_2 \geq r_1 > c_1$, which violates the assumption that $c_1 \geq c_2$.

6.5.3 **CASE 3**: $r_1 > t_2$

The proof is similar to the one in CASE 2. Recall that our goal is to prove $(g(c_1) + 1) \cdot (g(c_2) + 1) > t$, for any positive integers $c_1, c_2$ such that $c_1 + c_2 = \beta$ and $c_1 \geq c_2$. 
It will be helpful to remember that \(1 \leq r_2 \leq t_2 < r_1 \leq \beta\) - the inequality \(1 \leq r_2 \leq t_2\) follows from (5.2.2), (5.2.4) and (5.2.5); \(t_2 < r_1\) follows from the setting of CASE 3; \(r_1 \leq \beta\) follows from (5.2.2) and (5.2.5); and \(r_2 < m_1 < \beta\) (5.4.5) and (5.4.6).

**Case A**, \(c_1 = \beta\). Then, \(c_2 = 0\). By (6.3.20) and (6.3.18),

\[
(g(c_1) + 1) \cdot (g(c_2) + 1) = (t + 1) \cdot 1 > t.
\]

**Case B**, \(c_1 = m_1\). By (5.4.6), \(\beta = m_1 + r_2\), so we have \(c_2 = r_2\). Then by (6.3.19) and (6.3.18),

\[
(g(c_1) + 1) \cdot (g(c_2) + 1) = r_1 \cdot (t_2 + 1)
\]

> \(t\). (by (5.2.3))

**Case C**, \(c_1 \in (m_1, \beta)\). By (5.4.6), \(\beta = m_1 + r_2\). Let \(c_1 = m_1 + b\) for some \(b \in [1, r_2]\). Then we have \(c_2 = r_2 - b \geq 1\), which implies \(r_2 \geq 2\). Then by (6.3.19) and (6.3.18),

\[
(g(c_1) + 1) \cdot (g(c_2) + 1) = (r_1 + b \cdot \left\lfloor \frac{t - r_1 + 1}{\beta - m_1} \right\rfloor) \cdot (r_2 - b + 1)
\]

= \(r_1 + b \cdot \left\lfloor \frac{t - r_1 + 1}{r_2} \right\rfloor) \cdot (r_2 - b + 1)
\]

\(\geq 2r_1 + (b \cdot \left\lfloor \frac{t - r_1 + 1}{r_2} \right\rfloor) \cdot (r_2 - b + 1)\) (since \(r_2 - b \geq 1\))

\(> 2r_1 + (b \cdot \frac{t - r_1 + 1 - r_2}{r_2}) \cdot (r_2 - b + 1)\)

Note that the function \(f(b) = \frac{b}{r_2} \cdot (r_2 - b + 1)\) is a convex function with maximum at \(b = (r_2 + 1)/2\), i.e. with minimum at either endpoint of the range. At both endpoints, we have \(\frac{b}{r_2} \cdot (r_2 - b + 1) \geq 1\), which implies

\[
(b \cdot \frac{t - r_1 + 1 - r_2}{r_2}) \cdot (r_2 - b + 1) \geq t - r_1 + 1 - r_2.
\]

Thus, \((g(c_1) + 1) \cdot (g(c_2) + 1) > 2r_1 + t - r_1 + 1 - r_2 = t + (r_1 - r_2 + 1)\). By (5.2.2), \(r_1 > r_2\). Thus \(r_1 - r_2 + 1 > 1\), which implies \((g(c_1) + 1) \cdot (g(c_2) + 1) > t\).

**Case D**, \(c_1 \in (r_2, m_1)\). By (5.4.6), \(\beta = m_1 + r_2\), so we have \(c_2 \in (r_2, m_1)\) as well. Let \(c_1 = a_1 \cdot r_2 + b_1\) and \(c_2 = a_2 \cdot r_2 + b_2\) for some positive integers \(a_1, a_2\) such that \(b_1, b_2 \in [0, r_2]\). By (5.4.6) and (5.4.5), \(\beta = m_1 + r_2\) and \(m_1 = x_2 \cdot r_2 + (r_2 - 1)\), so \(\beta = (x_2 + 1) \cdot r_2 + (r_2 - 1)\). Thus,

\[
c_1 + c_2 = (a_1 + a_2) \cdot n_2 + (b_1 + b_2) = \beta = (x_2 + 1) \cdot r_2 + (r_2 - 1).
\]

This implies \((b_1 + b_2) \mod r_2 = r_2 - 1\). Since \(b_1, b_2 \in [0, r_2]\), this implies \(b_1 + b_2 < r_2 + (r_2 - 1)\), i.e. \(b_1 + b_2 = r_2 - 1\). Thus, \(a_1 + a_2 = x_2 + 1\).

By (6.3.18), \(g(ar_2 + b) = a \cdot t_2 + b\) for \(ar_2 + b \in [0, m_1]\) and \(b \in [0, r_2]\). Therefore, \(g(c_1) + g(c_2) = \)
71

\( (x_2 + 1) \cdot t_2 + (r_2 - 1) \). By (5.4.5), \( m_1 = x_2 \cdot r_2 + (r_2 - 1) \). Thus, \( g(m_1) = x_2 \cdot t_2 + (r_2 - 1) \) and 
\( g(m_1) + g(r_2) = (x_2 + 1) \cdot t_2 + (r_2 - 1) \). Therefore, we can replace \( c_1, c_2 \) with \( m_1 = x_1 \cdot n_2 + (r_2 - 1) \) and \( n_2 \) without increasing \( g(c_1) + g(c_2) \). By Claim 6.9 below, this would reduce the product \( (g(c_1) + 1) \cdot (g(c_2) + 1) \) at the same time. Hence,

\[
(g(c_1) + 1) \cdot (g(c_2) + 1) \geq (g(m_1) + 1) \cdot (g(r_2) + 1)
\]

\[
= r_1 \cdot (t_2 + 1)
\]

\[
> t. \quad \text{(by (6.3.19) and (6.3.18))}
\]

\[
\text{(by (5.2.3))}
\]

**Case E, \( c_1 \leq r_2 \).** By (5.4.6), \( \beta = m_1 + r_2 \), so we have \( c_2 \geq m_1 > c_1 \), which violates the assumption that \( c_1 \geq c_2 \).

**Claim 6.9.** For any positive integers \( a \geq b \geq c \), \( a \cdot b > (a + c) \cdot (b - c) \).

**Proof.** \( (a + c) \cdot (b - c) = ab - c^2 - (a - b)c < a \cdot b \).

### 6.6 Chapter Summary

In this chapter, we designed the token function \( g(c) \) required in the approach presented in Section 4, for the random \((d, k, t)\)-CSP model with \( 1 \leq t < (d - 1) d^{k-2} \) and \( k = 2 \). The design of \( g(c) \) is inspired by the unique valid \((\alpha, \beta)\)-token allocation on \((V(H^*), I(H^*), E(H^*))\). This token function satisfies Claim 4.24 and Claim 4.25 for every valid \((\alpha, \beta)\)-token allocation, which ensures the correctness of the algorithm **TA-SA** presented in Section 4.5. This completes the proof of Lemma 4.22, and thus completes the proofs of Lemma 4.14 and 4.15. Therefore, this proves the lower bounds on resolution and tree resolution complexity stated in Theorem 3.5 and 3.6, contingent on providing the proofs of Lemma 4.13 and 4.17, which are included in Appendix A.

The following proofs just repeat what we described in Chapter 4. We include them here for the purpose of completeness.

**Proof of Theorem 3.5:** Consider the random \((d, k, t)\)-CSP model \( \text{CSP}^{d,k,t}_{n,m} \) with any constant triple \((d, k, t)\) such that \( 1 \leq t < (d - 1) d^{k-2} \) and \( k \geq 2 \), and with constraint density \( n^{k-1 - \frac{1}{\Lambda(d,k,t) - \epsilon}} \).

Consider a \((d, k, t)\)-CSP instance \( P \) drawn from the random model, and any resolution refutation of \( P \). Lemma 4.13 and Lemma 4.14 imply that w.h.p. every subproblem with \( O(n^\gamma) \) variables is satisfiable. By Lemma 4.16, this implies that w.h.p. there is a clause \( C \) in the refutation such that \( C \) is minimally derived from a subproblem with \( \Theta(n^\gamma) \) variables. By Lemma 4.13, such a small subproblem is \((\Lambda(d,k,t) - \sigma)\)-sparse. Then by Lemma 4.15, the clause \( C \) has \( \Theta(n^\gamma) \) variables. By the width-length relation stated in Theorem 2.1, when the constraint density is \( n^{k-1 - \frac{1}{\Lambda(d,k,t) - \epsilon}} \), w.h.p. the tree resolution complexity is \( \exp(\Omega(n^\gamma)) \). This proves Theorem 3.5.

**Proof of Theorem 3.6:** Consider the random \((d, k, t)\)-CSP model \( \text{CSP}^{d,k,t}_{n,m} \) with any constant triple
Consider a \((d, k, t)\)-CSP instance \(P\) drawn from the random model, and a resolution refutation of \(P\). Lemma 4.17 and Lemma 4.14 imply that w.h.p. every subproblem with \(O(n^{0.5+\gamma})\) variables is satisfiable. By Lemma 4.16, this implies that w.h.p. there is a clause \(C\) in the refutation such that \(C\) is minimally derived from a subproblem with \(\Theta(n^{0.5+\gamma})\) variables. By Lemma 4.13, such a small subproblem is \((\Lambda(d, k, t) - \sigma)\)-sparse. Then by Lemma 4.15, the clause \(C\) has \(\Theta(n^{0.5+\gamma})\) variables. By the width-length relation stated in Theorem 2.2, when the constraint density is \(n^{\frac{k-1}{2} - \frac{1}{2\Lambda(d, k, t)^{-\varepsilon}}}\), w.h.p. the tree resolution complexity is \(exp(\Omega(n^\gamma))\). This proves Theorem 3.6. \(\square\)
Chapter 7

Concluding Remarks

We introduced an approach to study the resolution complexity and the tree resolution complexity of the random \((d, k, t)\)-CSP model, and successfully applied this approach to the setting with constants \(d, k, t\) where \(1 \leq t < (d - 1)d^{k-2}\) and \(k = 2\). New results include lower and upper bounds on the resolution complexity and the tree resolution complexity. In particular, we showed that the tree resolution complexity is w.h.p. \(exp(\Omega(n^\gamma))\) when the constraint density is \(n^{k-1-\frac{1}{\Lambda(d, k, t)}}\); and is w.h.p. constant when the constraint density is \(n^{k-1-\frac{1}{\Lambda(d, k, t)}}\); these bounds are tight up to a \(o(1)\) term in the exponent.

We also applied the same approach to the setting with \(t = 1\) and \(d, k \geq 2\), which is a generalization of the random \(k\)-SAT model by allowing a more general domain of variable values instead of \{TRUE, FALSE\}; and to the setting with \(d = 2, 1 \leq t < (d - 1)d^{k-2}\) and \(k \geq 2\), which is a generalization of the random \(k\)-SAT model by allowing more restrictions in each constraint. Thus, our results can be regarded as generalizations of the resolution complexity results for random \(k\)-SAT with \(k \geq 3\). The details are included in Appendix C and Appendix D.

As a direction for future research, we are interested in applying this approach to other models of random CSPs, for instances:

**Random \((d, k, t)\)-CSP model with constants** \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\). We successfully applied this approach to the setting with \(k = 2\) (and \(d \geq 2, 1 \leq t < (d - 1)d^{k-2}\)). A natural follow-up work is to extend the result to the general setting with \(k \geq 2\). Actually, we have already proved Conjectures 3.1, 3.2 and 3.3 for this general case. However, the proofs are significantly more complicated and are very messy. We are still trying to simplify the proofs, so these results and proofs are not included in this thesis.

**Random \((d, k, t)\)-CSP model with super-constants** \(d, k, t\). Note that the asymptotic orders of \(d, k, t\) only affect the arguments using the first and the second moment methods (these arguments are shown in Appendix A). It is straightforward to see that if the asymptotic orders of \(d, k, t\) are small enough, then the arguments using the first and the second moment methods will still work. For example, when
Chapter 7. Concluding Remarks

$d, k, t$ are of asymptotic order $\log \log \log n$, we will consider a forbidding flower with $\log n$ length cycles, instead of constant length cycles; then the arguments using the first and the second moment methods will still work. However, when $d, k, t$ grow further (e.g. with $d, k, t = \Omega(\log n)$), we may need to change those arguments significantly. For example, instead of forbidding flowers, we may have to consider some more general families of unsatisfiable CSP instances. It is of interest to determine how far these parameters can grow.

Random CSP models with constraint set symmetric over the domain of variable values.

In the random $(d, k, t)$-CSP model, the canonical constraint set contains all the $(d^k)$ possible constraints, thus the set of constraints is highly symmetric over the domain of variable values. With this property, given an $(I : A_I) \rightarrow (u : \delta)$ forcer, we can change some restrictions in the constraints to obtain an $(I : A'_I) \rightarrow (u : \delta')$ forcer for any $|I|$-tuple of values $A'_I$ and any value $\delta'$. This allows us to construct the forbidding flower $G_\ell$ by using only forcers “similar” to $H^*$. For example, consider the formula encoding the $r$-colorability problem on a random $k$-uniform hypergraph. There is also some symmetry on the constraint set, and we can apply a similar approach to prove resolution complexity results (when $r, k$ are positive constants). Now, a natural question is: can we always apply a similar approach on other random CSP models with symmetric constraint set?

Random CSP models without symmetric constraint set.

As mentioned above, our approach requires the set of constraints to be symmetric over the domain of variable values - without this property, we cannot construct the forbidding flower $G_\ell$ by using only forcers “similar” to $H^*$. For example, Chan and Molloy [33] studied a very broad family of random CSPs, where symmetry on the constraint set cannot be guaranteed. Thus, our approach cannot be applied directly to their setting. Assume that there exists an unsatisfiable CSP instance in the given random CSP model. A natural guess is that we can construct a generalized version of the forbidding flower, by using a collection of very “different” forcers $F$ such that each $(I : A_I) \rightarrow (u : \delta)$ forcer in $F$ is as sparse as possible. We determined the key variable $\Lambda(d, k, t)$ for random $(d, k, t)$-CSP from the structure of the sparsest forcer $H^*$. A hunch is that in general, a similar variable can be determined from the structure of the densest forcer in $F$.

Another direction of future research is to prove the existence of a threshold-like phenomenon for the resolution complexity of random CSPs. For the random $k$-SAT model with any $k \geq 3$, we know that the tree resolution complexity of a random $k$-SAT instance is w.h.p. superpolynomial when the clause density is $n^{k-2-\epsilon}$; and is w.h.p. at most polynomial when the clause density is $\Omega(n^{k-2}/\log^{k-2} n)$ [14, 24, 18]. These bounds are tight up to a $o(1)$ term in the exponent. However, the best known bounds for the resolution complexity are $n^{k-2-\epsilon}$ and $\Omega(n^{k-2}/\log^{k-2} n)$; there is still a gap of $\frac{k-2}{2} + o(1)$ in the exponent [14, 24, 18]. The situation is the similar for the random $(d, k, t)$-CSP model, as we discussed in Chapter 3.

One possible approach to prove a threshold-like phenomenon for resolution complexity is to study the width-length relation for the resolution refutation of random CSPs. In our approach, we apply the width-length result by Ben-Sasson and Wigderson (Theorem 2.2) in proving the resolution complexity lower bound stated in Theorem 3.6. Having a stronger width-length relation will directly imply a better resolution complexity lower bound, which will give a better lower bound on the range of constraint density.
for resolution complexity to drop from superpolynomial to polynomial. As mentioned in Section 2.2.4, there exists a family of unsatisfiable CNF formulas such that the bound in Theorem 2.2 is asymptotically tight up to \( \log n \) factors \([30]\). Therefore, it is impossible to obtain a stronger width-length relation for the general setting of any CNF formula. However, those unsatisfiable CNF formulas contain some very dense subformulas such that w.h.p. a random \( k \)-SAT instance does not contain any such subformula when the clause density is \( n^{k-2-\epsilon} \). Therefore, it is still plausible to prove a better width-length relation for the settings of random CSPs.

Another possible approach to prove a threshold-like phenomenon for resolution complexity is to identify some good necessary and sufficient conditions such that general resolution outperforms the tree-like variant. As mentioned in Section 2.2.5, there are families of unsatisfiable CNF formulas such that general resolution outperforms tree resolution. Roughly speaking, those formulas contain some dense but satisfiable subformulas, which can be ignored in the setting of random \( k \)-SAT (and random \((d, k, t)\)-CSP) - if a random \( k \)-SAT instance w.h.p. contains a dense subformula, then it w.h.p. contains a dense unsatisfiable subformula. Therefore, if those dense satisfiable subformulas are necessary for general resolution to outperform tree resolution, then general resolution is not stronger than tree resolution in the setting of random \( k \)-SAT (and random \((d, k, t)\)-CSP). This will prove that the resolution and tree resolution complexity actually have the same threshold-like phenomenon.
Bibliography


[36] Vasek Chvátal and Bruce A. Reed. Mick gets some (the odds are on his side). In FOCS, pages 620–627. IEEE, 1992.


Appendix A

Proofs using the First and the Second Moment Methods

A.1 Proof of Lemma 4.13 by the First Moment Method

Consider the random \((d, k, t)\)-CSP model \(\text{CSP}_{n,m}^{d,k,t}\) with any constant triple \((d, k, t)\) such that \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\), and with constraint density \(\Delta = n^{k-1} - \frac{1}{\Lambda(d, k, t)}\). Let \(P\) be a CSP instance drawn from \(\text{CSP}_{n,m}^{d,k,t}\).

**Lemma 4.13**: there exist positive constants \(\gamma\) and \(\sigma\) such that w.h.p. every subproblem of \(P\) with \(O(n^{\gamma})\) variables is \((\Lambda(d, k, t) - \sigma)\)-sparse.

We will use the following property in the proof. This property holds for any constant triple \((d, k, t)\) with \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\) (see Remark A.2). In particular, this property can be verified easily for the settings we consider in this thesis, as we determined the values of \(\Lambda(d, k, t)\) for those settings.

**Claim A.1.** \(\Lambda(d, k, t) > \frac{1}{k-1}\).

**Remark A.2.** If \(\Lambda(d, k, t) \leq \frac{1}{k-1}\), then there exists an \((\frac{1}{k-1} + \mu)\)-sparse forbidding flower for every constant \(\mu > 0\). By standard arguments using the second moment method, we can show that when the constraint density is a sufficiently large constant, w.h.p. a random \((d, k, t)\)-CSP instance contains a forbidding flower as a subproblem. This contradicts the Theorem 2.21, which states that the resolution complexity is \(\exp(\Theta(n))\) when \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\) and the constraint density is a constant. Therefore, Claim A.1 holds for any constant triple \((d, k, t)\) such that \(1 \leq t < (d - 1)d^{k-2}\) and \(d, k \geq 2\).

Let \(\phi = (\Lambda(d, k, t) - \sigma)\). Let \(X_u\) be the number of subproblems of \(P\) with at most \(u = O(n^{\gamma})\) variables and exactly \([\phi u]\) constraints. Let \(X = \sum_u X_u\). We can prove Lemma 4.13 by showing that: there exist positive constants \(\gamma\) and \(\sigma\) such that such that \(\mathbb{E}(X) = o(1)\).

There are \(\binom{n}{u}\) ways to choose a set of \(u\) variables and \(\binom{\Delta n}{[\phi u]}\) ways to choose a set of \([\phi u]\) constraints. For each chosen constraint, the probability for every variable to lie within the chosen set of \(u\) variables
is \((u/n)^k\). Therefore, we have

\[
\mathbb{E}(X) = \sum_u X_u \\
\leq \sum_u \left(\begin{array}{c} n \\ u \end{array} \right) \left( \frac{\Delta n}{\phi u} \right) \left( \frac{u}{n} \right)^{k[\phi u]} \\
\leq \sum_u \left( \frac{en}{u} \right)^u \left( \frac{\Delta n}{\phi u} \right)^{\phi u} \left( \frac{u}{n} \right)^{k[\phi u]} \\
\leq \sum_u \left( \Theta \left[ \left( \frac{u}{n} \right)^{(k-1)\phi-1} \times \Delta^\phi \right] \right)^u
\]

By Claim A.1, we choose sufficiently small \(\sigma\) such that \(\phi = (\Lambda(d,k,t) - \sigma) > \frac{1}{k-1}\). Then, \((\frac{u}{n})^{(k-1)\phi-1}\) is maximized when \(u\) is maximized, i.e. when \(u = \theta(n^\gamma)\). Since \(\Delta = n^{k-1-\frac{1}{d(k-1)}}\) and \(\phi = \Lambda(d,k,t) - \sigma\), we have \(\Delta^\phi = n^{(k-1)\phi-1-\epsilon'}\) for some sufficiently small constants \(\sigma, \epsilon' > 0\). Thus,

\[
\mathbb{E}(X) \leq \sum_u \left( \Theta \left[ \left( \frac{u}{n} \right)^{(k-1)\phi-1} \times \Delta^\phi \right] \right)^u \\
\leq \sum_u \left[ \Theta(n^\gamma)^{(k-1)\phi-1} \times n^{-(k-1)\phi+1} \times n^{(k-1)\phi-1-\epsilon'} \right]^u \quad \text{(for some constants } \sigma, \epsilon' > 0) \\
= \sum_u \left[ \Theta(n^{-\epsilon''}) \right]^u \quad \text{(for some constants } \gamma, \epsilon'' > 0) \\
\leq \Theta(n^{-\epsilon''}) \quad \text{(sum of an infinite geometric sequence)} \\
\leq o(1)
\]

### A.2 Proof of Lemma 4.17 by the First Moment Method

Consider the random \((d,k,t)\)-CSP model \(\text{CSP}^{d,k,t}_{n,m}\) with any constant triple \((d,k,t)\) such that \(1 \leq t < (d-1)d^{k-2}\) and \(d,k \geq 2\), and with constraint density \(\Delta = n^{k-1-\frac{1}{d(k-1)}}\). Let \(P\) be a CSP instance drawn from \(\text{CSP}^{d,k,t}_{n,m}\).

**Lemma 4.17:** there exist positive constants \(\gamma\) and \(\sigma\) such that w.h.p. every subproblem of \(P\) with \(O(n^{0.5+\gamma})\) variables is \((\Lambda(d,k,t) - \sigma)\)-sparse.

The proof is basically the same is the proof of Lemma 4.13 presented in Appendix A.1.

Let \(X_u\) be the number of subproblems of \(P\) with at most \(u = O(n^{0.5+\gamma})\) variables and exactly \([\phi u]\) constraints. Let \(X = \sum_u X_u\). We can prove Lemma 4.17 by showing that: there exist positive constants \(\gamma\) and \(\sigma\) such that such that \(\mathbb{E}(X) = o(1)\).

There are \(\binom{n}{u}\) ways to choose a set of \(u\) variables and \(\binom{\Delta n}{[\phi u]}\) ways to choose a set of \([\phi u]\) constraints. For each chosen constraint, the probability for every variable to lie within the chosen set of \(u\) variables
is $(u/n)^k$. Therefore, we have

$$\mathbb{E}(X) = \sum_u X_u$$

$$\leq \sum_u \left( \binom{n}{u} \left( \frac{\Delta n}{\phi u} \right) \left( \frac{u}{n} \right)^k \right)$$

$$\leq \sum_u \left( \binom{en}{u} \left( e \Delta n \right) \left( \frac{u}{n} \right)^k \right)$$

$$\leq \sum_u \left( \left( \frac{u}{n} \right)^{(k-1)\phi - 1} \times \Delta^\phi \right)^u$$

By Claim A.1, we choose sufficiently small $\sigma$ such that $\phi = (\Lambda(d,k,t) - \sigma) > \frac{1}{k-1}$, then $(\frac{u}{n})^{(k-1)\phi - 1}$ is maximized when $u$ is maximized, i.e. when $u = \theta(n^{0.5+\gamma})$. Since $\Delta = n^{\frac{k-1}{\phi} - \frac{1}{2} - \epsilon}$ and $\phi = \Lambda(d,k,t) - \sigma$, we have $\Delta^\phi = n^{k-1/\phi - \frac{1}{2} - \epsilon'}$ for some sufficiently small constants $\sigma, \epsilon > 0$. Thus,

$$\mathbb{E}(X) \leq \sum_u \left( \left( \frac{u}{n} \right)^{(k-1)\phi - 1} \times \Delta^\phi \right)^u$$

$$\leq \sum_u \left[ \Theta(n^{\gamma \cdot (k-1)\phi - 1} \times n^{\frac{k-1}{\phi} - \frac{1}{2} - \epsilon'}) \right]^u \quad \text{(for some constants } \sigma, \epsilon > 0)$$

$$= \sum_u \left[ \Theta(n^{-\epsilon''}) \right]^u \quad \text{(for some constants } \gamma, \epsilon'' > 0)$$

$$\leq \Theta(n^{-\epsilon''}) \quad \text{(sum of an infinite geometric sequence)}$$

$$\leq o(1)$$

### A.3 Proof of Lemma 4.12 by the Second Moment Method

Consider the random $(d,k,t)$-CSP model $\text{CSP}_{n,m}^{d,k,t}$ with any constant triple $(d,k,t)$ such that $1 \leq t < (d-1)d^{k-2}$ and $d,k \geq 2$, and with constraint density $\Delta = n^{k-1/\Lambda(d,k,t) - 1/\epsilon}$ for any constant $\epsilon > 0$. Let $I$ be a random instance drawn from $\text{CSP}_{n,m}^{d,k,t}$.

**Lemma 4.12.** There exists a constant $\mu > 0$ such that for any constant size $(\Lambda(d,k,t) + \mu)$-sparse $(d,k,t)$-CSP instance, w.h.p. it appears in $I$ as a subproblem.

We are going to prove this statement by using the second moment method. In the following, we will consider the $\text{CSP}_{n,p}^{d,k,t}$ instead, because the calculations for the second moment are usually simpler with $\text{CSP}_{n,p}^{d,k,t}$. Specifically, we are going to prove Lemma A.3 stated below. Consider the random $(d,k,t)$-CSP model $\text{CSP}_{n,p}^{d,k,t}$ with any constant triple $(d,k,t)$ such that $1 \leq t < (d-1)d^{k-2}$ and $d,k \geq 2$, and with constraint probability $p = n^{\frac{1}{\Lambda(d,k,t) + 1/\epsilon}}$ for any constant $\epsilon > 0$. Let $P$ be a random instance drawn from $\text{CSP}_{n,p}^{d,k,t}$.

**Lemma A.3.** There exists a constant $\mu > 0$ such that for any constant size $(\Lambda(d,k,t) + \mu)$-sparse $(d,k,t)$-CSP instance, w.h.p. it appears in $P$ as a subproblem.
As mentioned in Section 2.3.4, a convex property must hold w.h.p. in a random instance drawn from \( \text{CSP}^{d,k,t}_{n,m} \) if it holds w.h.p. in a random instance drawn from \( \text{CSP}^{d,k,t}_{n,p} \) with \( p = m/M \) (where \( M = \Theta(n^k) \) is the total number of possible constraints). Also, the property for a CSP instance to contain another CSP instance as a subproblem is a convex property with respect to the set of constraints - adding an extra constraint does not damage any subproblem a CSP instance containing. Therefore, Lemma A.3 above implies Lemma 4.12.

### A.3.1 Definitions

In the random \((d, k, t)\)-CSP model \( \text{CSP}^{d,k,t}_{n,p} \) there are \( \binom{d^k}{t} = \Theta(1) \) different types of constraints, and each appears with equal probability. Thus, w.h.p. the probability \( p' \) for any particular type of constraint to appear on a \( k \)-tuple of variables is of the same asymptotic order as the constraint probability \( p \) i.e. \( p' = \Theta(p) \).

Let \( G \) be any constant size \((\Lambda(d, k, t) + \mu)\)-sparse instance with \( u \) variables and \( h \) constraints, where \( \mu \) is a positive constant we will specify later.

**Definition A.4** (Potential Copy). A potential copy of \( G \), denoted by \( A \), is a CSP instance on the variables of the given CSP instance \( P \) such that (i) the constraint hypergraphs of \( A \) and \( G \) are isomorphic to each other, and (ii) for each constraint in \( A \) and the corresponding constraint in \( G \), they are the same type of constraint. The potential copy \( A \) is realized if \( A \) is a subproblem of \( P \), i.e. (a) all the constraints of \( A \) are selected as constraints for \( P \), and (ii) all the selected constraints are of the right types.

**Remark A.5.** For any set of \( |V(G)| \) variables, there are more than one potential copy on it. Also, if the constraint hypergraph of \( G \) has non-trivial automorphisms, then a potential copy may be counted multiple times in the calculations of the first and the second moments. However, \( G \) has constant size, which implies that there are only \( \Theta(1) \) ways to arrange the variables. Therefore, we will see that the above factors do not affect the asymptotic orders in the following calculations of the first and the second moments.

For any potential copy of \( G \), denoted by \( A \), let \( X_A \) be the indicator variable for the event that \( A \) is realized. Let \( X = \sum_A X_A \). By applying the second moment method, we can prove Lemma A.3 by showing that \( \mathbb{E}(X^2) \leq (1 + o(1))\mathbb{E}(X)^2 \). The following arguments are similar to those in [64] and [33].

### A.3.2 First Moment

Recall that \( G \) is a constant size instance with \( u \) variables and \( h \) constraints; the constraint probability \( p = n^{-\frac{1}{\Lambda(d,k,t)+\mu}} \); the probability for any particular type of constraint to appear on a \( k \)-tuple of variables \( p' = \Theta(p) \).

There are \( \binom{u}{k} \) ways to choose a set of \( u \) variables and the probability for each correct constraint to appear
Appendix A. Proofs using the First and the Second Moment Methods

85

is \( p' \). Since \( u, h \) are constants, there are only \( \Theta(1) \) ways to arrange the variables and constraints. Thus,

\[
\mathbb{E}[X] = \binom{n}{u} \cdot (p')^h \cdot \Theta(1) \\
= [\Theta(n)]^u \times [\Theta(p)]^h \\
= [\Theta(n)]^u \times [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^h
\]

A.3.3 Second Moment

Recall that \( G \) is a constant size instance with \( u \) variables and \( h \) constraints; the constraint probability \( p = n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}} \); the probability for any particular type of constraint to appear on a \( k \)-tuple of variables \( p' = \Theta(p) \).

We want to show that \( \mathbb{E}(X^2) \leq (1 + o(1))\mathbb{E}(X)^2 \). Let \( A \) be a potential copy of \( G \). It is well known (e.g. Corollary 4.3.5 in [7]) that we only have to show that

\[
\sum_{B: E(B) \cap E(A) \neq \emptyset} \mathbb{P}[X_B = 1 | X_A = 1] = o(E[X]). \tag{A.3.1}
\]

For any set of constant number of constraints \( F \) on the variables of \( V(P) \), let \( Y_F \) be the event that all constraints of \( F \) are selected as constraints for the given CSP instance \( P \) (we allow the types of constraints to be different). We have

\[
\mathbb{P}[Y_F = 1] = \mathbb{P}^{|F|} = [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^{|F|}. \tag{A.3.2}
\]

For any potential copy \( B \), let \( Y_B \) be the event that all constraints of \( E(B) \) are selected as constraints for the CSP instance. We have

\[
\mathbb{P}[X_B = 1 | X_A = 1] \leq \mathbb{P}[Y_B = 1 | X_A = 1] \leq \mathbb{P}[Y_B = 1] / [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^{E(B) \cap E(A)}. \tag{A.3.3}
\]

Let \( B_0 \) be any potential copy of \( G \), we have

\[
\sum_{B: E(B) \cap E(A) \neq \emptyset} \mathbb{P}[X_B = 1 | X_A = 1] \leq \mathbb{P}[Y_{B_0} = 1] \sum_{B: E(B) \cap E(A) \neq \emptyset} [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^{-|E(B) \cap E(A)|}. \tag{A.3.4}
\]

As shown in the previous section, \( \mathbb{E}[X] = [\Theta(n)]^u \times [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^h \). By (A.3.2), \( \mathbb{P}[Y_{B_0} = 1] = [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^h \). By equation (A.3.4), we can prove (A.3.1) by showing that

\[
\sum_{B: E(B) \cap E(A) \neq \emptyset} [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^{-|E(B) \cap E(A)|} = [o(n)]^u. \tag{A.3.5}
\]
When $|E(B) \cap E(A)| = h'$ for any positive constant $h'$, $A$ and $B$ share $u' \geq \frac{h'}{\Lambda(d,k,t)+\mu}$ variables because $A$ and $B$ are potential copies of $G$ and $G$ is $(\Lambda(d,k,t)+\mu)$-sparse.

There are $\Theta(n^{u-u'})$ ways to select the other variables in $B$. Since the size of $G$ is constant, we have $u, h = \Theta(1)$. Thus, there are only $\Theta(1)$ ways to arrange the constraints and variables in $B$.

Hence, for any specific $h'$,

$$
\sum_{B:|E(B) \cap E(A)|=h'} [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})^{-|E(B) \cap E(A)|}] = \Theta(n^{u-u'}) \times [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^{h'}.
$$

(A.3.6)

Recall that $u' \geq \frac{h'}{\Lambda(d,k,t)+\mu}$. There exists some constants $\mu, \epsilon_2 > 0$ such that

$$
[\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^{h'} \leq [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})]^{\Lambda(d,k,t)+\mu} \leq [\Theta(n^{1-\epsilon_2})]^{u'}.
$$

(A.3.7)

By (A.3.6) and (A.3.7), for any specific $h'$,

$$
\sum_{B:|E(B) \cap E(A)|=h'} [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})^{-|E(B) \cap E(A)|}] \leq \Theta(n^{u-u'}) \times [\Theta(n^{1-\epsilon_2})]^{u'} = [o(n)]^{u'}.
$$

Finally, since $h = \Theta(1)$, summing over all $h' \in [1, h]$ gives the desired equation:

$$
\sum_{B:|E(B) \cap E(A)| \neq 0} [\Theta(n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}})^{-|E(B) \cap E(A)|}] = \Theta(1) \times [o(n)]^{u} = [o(n)]^{u}.
$$

This proves (A.3.5), and thus proves (A.3.1). Hence, by the second moment method, for any constant $\epsilon > 0$, there exists a constant $\mu > 0$ such that for any constant size $(\Lambda(d,k,t)+\mu)$-sparse instance $G$, when the constraint probability is $n^{-\frac{1}{\Lambda(d,k,t)+\epsilon}}$, w.h.p. $G$ appears in a random $(d,k,t)$-CSP instance. This completes the proof.
Appendix B

Sparsity of Forcing Blocks

Recall that for any \((d, k, t)\)-CSP instance \(H\), in a valid \((\alpha, \beta)\)-token allocation on \((U \subseteq V(H), F \subseteq E(H))\), every variable \(v \in U\) gives out at most \(\alpha\) tokens and every constraint \(e \in F\) receives exactly \(\beta\) tokens form its variables. Note that a constraint \(e \in F\) may have variables in \(V(H) - U\).

As defined in Section 4.2, \(H^*\) is a special forcer with modified constraint density \(\Lambda(d, k, t)\), i.e. \(\Lambda(d, k, t) = |E(H^*)|/|V(H^*)| - |I(H^*)|\).

The following property links a valid \((\alpha, \beta)\)-token allocation to the sparsity of the corresponding pair of variable set and constraint set.

Claim B.1. Given any variable set \(U\) and constraint set \(F\) of any \((d, k, t)\)-CSP instance, if there exists a valid \((\alpha, \beta)\)-token allocation on \((U, F)\), then every subset of constraints \(F' \subseteq F\) contains at least \(\Lambda(d, k, t)\) variables in \(U\).

Proof. By (4.4.1), we have \(\Lambda(d, k, t) = \alpha/\beta\). If there is any subset of constraints \(F'\) containing less than \(\Lambda(d, k, t)\) variables in \(U\), then the number of tokens at those variables is insufficient for the constraints in \(F'\). This contradicts the fact that there exists a valid \((\alpha, \beta)\)-token allocation on \((U, F)\). Thus, no such subset of constraints exists, which proves the claim.

B.1 Proof of Claim 5.4

Claim 5.4: for every \((I : A_I) \rightarrow (U : A_U)\) forcing block \(B\) in \(G\), every subset of constraints \(F' \subseteq E(B)\) contains at least \(\Lambda(d, k, t)\) variables in \(V(B) - I\).

We will use the following property in the proof.

Claim B.2. There exists a valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\).

Proof. A valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\) is shown in Section 6.2.
Proof of Claim 5.4: By Claim B.1, this claim can be proved by showing the existence of a valid \((\alpha, \beta)\)-token allocation on \((V(B) - I, E(B))\).

Let \(H = \{H_i\}\) be the set of forcers in \(B\). Consider any \(H_i, H_j \in H\) with \(H_i \neq H_j\). They do not share any variable except for the input variables: if \(v \in V(H_i) \cap V(H_j)\), then \(v \in I\). Hence, if there exists a valid \((\alpha, \beta)\)-token allocation on \((V(H_i) - I(H_i), E(H_i))\) for every \(H_i \in H\), then the union of these token allocations will be a valid \((\alpha, \beta)\)-token allocation on \((V(B) - I, E(B))\).

Recall that every forcer in \(G_\ell\) is structurally similar to \(H^*\) - two CSP instances are said to be structurally similar if their constraint hypergraphs are isomorphic to each other (Definition 4.8). Claim B.2 implies that there is a valid \((\alpha, \beta)\)-token allocation on \((V(H_i) - I(H_i), E(H_i))\) for every \(H_i \in H\). This completes the proof.

\[\square\]

B.2 Proof of Claim 5.5

Claim 5.5: for every \((I : A_I) \rightarrow (U : A_U)\) forcing block \(B\) in \(G_\ell\), every subset of constraints \(E' \subseteq E(B)\) contains at least \(\Lambda(d, k, t)\) variables in \(V(B) - U\).

The proof is similar to the one for Claim 5.4. We will use the following property, which will be proved in Appendix B.3.

Claim B.3. There exists a valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - u(H^*), E(H^*))\) such that the total number of tokens going from \(I(H^*)\) to \(E(H^*)\) is \(\alpha\).

Proof of Claim 5.4: By Claim B.1, this claim can be proved by showing the existence of a valid token allocation on \((V(B) - U, E(B))\).

Let \(H = \{H_i\}\) be the set of forcers in \(B\). Since every forcer in \(G_\ell\) is structurally similar to \(H^*\) - two CSP instances are said to be structurally similar if their constraint hypergraphs are isomorphic to each other (Definition 4.8). Claim B.3 implies that for every forcer \(H_i \in H\), there exists a valid \((\alpha, \beta)\)-token allocation on \((V(H_i) - I(H_i), E(H_i))\) such that variables in \(I(H_i)\) give a total of \(\alpha\) tokens to constraints in \(E(H_i)\). Now, consider the union of those valid \((\alpha, \beta)\)-token allocations.

Note that for any different \(H_i, H_j \in H\), they do not share any variable except for the input variables: if \(v \in V(H_i) \cap V(H_j)\), then \(v \in I\). Therefore, the union of those valid \((\alpha, \beta)\)-token allocations is a valid \((\alpha, \beta)\)-token allocation on \((V(B) - U, E(B))\) if each \(v \in I\) gives out a total of at most \(\alpha\) tokens, which will be proved as follows.

Let \(\{c_0, c_1, \ldots, c_{|I| - 1}\}\) be the number of tokens the input variables \(I\) give to the \(|I|\) constraints in the valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - u(H^*), E(H^*))\) specified at Claim B.3. Thus, \(\sum_{i=0}^{|I|-1} c_i = \alpha\).

Denote the input variable set by \(\{x_0, x_1, \ldots, x_{|I| - 1}\}\) and the set of forcers of \(B\) by \(\{H_0, H_1, \ldots, H_{|I| - 1}\}\). By Property 4.11, the forcers in \(B\) are arranged in a symmetric way such that we can ensure that \(x_i\) gives \(c_i \mod |I|\) tokens to a constraint in the \((\alpha, \beta)\)-token allocation on \((V(H_j) - u(H_j), E(H_j))\). Therefore, every \(v \in I\) gives out \(\sum_{i=0}^{|I|-1} c_i = \alpha\) tokens in total. This completes the proof.
B.3 Proof of Claim B.3

In this section, we will prove Claim B.3 by constructing a valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - u(H^*), E(H^*))\) such that the total number of tokens going from \(I(H^*)\) to \(E(H^*)\) is \(\alpha\).

The idea is simple. Given a valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\), we can construct a valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - u(H^*), E(H^*))\) by changing the suppliers of tokens from \(u(H^*)\) to \(I(H^*)\). As shown in Section 6.2, the constraints in \(E(H^*)\) receive exactly \(\alpha\) tokens from \(u(H^*)\) in the valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\). Therefore, after changing the suppliers, that the total number of tokens going from \(I(H^*)\) to \(E(H^*)\) is exactly \(\alpha\).

The following cases correspond to the cases we analysed when constructing the forcer \(H^*\) in Section 5.2, and a valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\) in Section 6.2.

B.3.1 CASE 1: \(r_1 = 1\)

Consider the construction of \(H^*\) presented in Section 5.2.1, and the valid \((\alpha, \beta)\)-token allocation presented in Section 6.2.1. For reference, the structure of \(H^*\) is shown in Figure B.1.

![Figure B.1: The special forcer \(H^*\) for CASE 1, with \(x_1 = 3\).](image)

In the valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\) presented in Section 6.2.1, the output variable \(u(H^*)\) of \(H^*\) gives \(n_1\) tokens to each of the \(x_1\) different \(t_1\)-forbidding edges.

By the definition of \(H^*\), each \(t_1\)-forbidding edge \(e\) containing \(u(H^*)\) also contains an input variable \(v_i \in I(H^*)\). We change the suppliers of tokens such that instead of receiving \(n_1\) tokens from \(u(H^*)\), \(e\) will receive \(n_1\) tokens from \(v_i\).

B.3.2 CASE 2: \(1 < r_1 \leq t_2\)

Consider the construction of \(H^*\) presented in Section 5.2.2 and the valid \((\alpha, \beta)\)-token allocation presented in Section 6.2.2. For reference, the structure of \(H^*\) is shown in Figure B.2.

In the valid \((\alpha, \beta)\)-token allocation on \((V(H^*) - I(H^*), E(H^*))\) presented in Section 6.2.2, the output variable \(u(H^*)\) of \(H^*\) gives \(n_1\) tokens to each of the \(x_1\) different \(t_1\)-forbidding edges, and one token to each of the \(r_1 - 1\) different 1-forbidding edges.
Figure B.2: The special forcer $H^*$ for CASE 2, with $x_1 = 3$ and $r_1 = 4$.

By the definition of $H^*$, each $t_1$-forbidding edge $e$ containing $u(H^*)$ also contains an input variable $v_i \in I(H^*)$. We change the suppliers of tokens such that instead of receiving $n_1$ tokens from $u(H^*)$, $e$ will receive $n_1$ tokens from $v_i$.

By the definition of $H^*$, (I) each 1-forbidding edge $e_1$ containing $u(H^*)$ is inside a 1-forbidder $F$; (II) the other variable appearing in $e_1$ is the $(r_1 + t_1 - t_2)$-free variable $v$ in $F$; (III) there is a $t_2$-forbidding edge $e_2$ containing $v$ and an $r_1$-free variable $w$ inside a $t_2$-forbidder of $F$; (IV) there is a $t_1$-forbidding edge $e_3$ containing $w$ and $v_i \in I(H^*)$. In the valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$ presented in Section 6.2.2, (i) $e_1$ receives one token from $u(H^*)$ and $n_1 - 1$ tokens from $v$; (ii) $e_2$ receives $n_2$ tokens from $v$ and $n_1 - n_2$ tokens from $w$; (iii) $e_3$ receives $n_1$ tokens from $w$. We change the suppliers of tokens such that (a) $e_1$ receives 0 token from $u(H^*)$ and $n_1$ tokens from $v$; (b) $e_2$ receives $n_2 - 1$ tokens from $v$ and $n_1 - n_2 + 1$ tokens from $w$; (c) $e_3$ receives $n_1 - 1$ tokens from $w$ and one token from $v_i$.

### B.3.3 CASE 3: $r_1 > t_2$

Consider the construction of $H^*$ presented in Section 5.2.3, and the valid $(\alpha, \beta)$-token allocation presented in Section 6.2.3. The arguments are basically the same as those in CASE 2. For reference, the structure of $H^*$ is shown in Figure B.3.

In the valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$ presented in Section 6.2.3, the output variable $u(H^*)$ of $H^*$ gives $n_1$ tokens to each of the $x_1$ different $t_1$-forbidding edges, $n_2$ tokens to each of the $x_2$ different $t_2$-forbidding edges, and one token to each of the $r_2 - 1$ different 1-forbidding edges.

By the definition of $H^*$, each $t_1$-forbidding edge $e$ containing $u(H^*)$ also contains an input variable
Appendix B. Sparsity of Forcing Blocks

Figure B.3: The special forcer $H^*$ for CASE 3, with $x_1 = 3$, $x_2 = 4$ and $r_2 = 5$.

$v_i \in I(H^*)$. We change the suppliers of tokens such that instead of receiving $n_1$ tokens from $u(H^*)$, $e$ will receive $n_1$ tokens from $v_i$.

By the definition of $H^*$, (I) each $t_2$-forbidding edge $e_1$ containing $u(H^*)$ is inside a $t_2$-forbidder $F$; (II) the other variable appearing in $e_1$ is the $r_1$-free variable $v$ in $F$; (III) there is a $t_1$-forbidding edge $e_2$ containing $v$ and $v_i \in I(H^*)$. In the valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$ presented in Section 6.2.3, (i) $e_1$ receives $n_2$ tokens from $u(H^*)$ and $n_1 - n_2$ tokens from $v$; (ii) $e_2$ receives $n_1$ tokens from $v$. We change the suppliers of tokens such that (a) $e_1$ receives 0 token from $u(H^*)$ and $n_1$ tokens from $v$; (b) $e_2$ receives $n_1 - n_2$ tokens from $v$ and $n_2$ tokens from $v_i$.

By the definition of $H^*$, (I) each 1-forbidding edge $e_1$ containing $u(H^*)$ is inside a 1-forbidder $F$; (II) the other variable appearing in $e_1$ is the $(r_2 + t_1 - t_2)$-free variable $v$ in $F$; (III) there is a $t_2$-forbidding edge $e_2$ containing $v$ and an $r_1$-free variable $w$ inside a $t_2$-forbidder of $F$; (IV) there is a $t_1$-forbidding edge $e_3$ containing $w$ and $v_i \in I(H^*)$. In the valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$ presented in Section 6.2.3, (i) $e_1$ receives one token from $u(H^*)$ and $n_1 - 1$ tokens from $v$; (ii) $e_2$ receives $n_2$ tokens
from $v$ and $n_1 - n_2$ tokens from $w$; (ii) $e_3$ receives $n_1$ tokens from $w$. We change the suppliers of tokens such that (a) $e_1$ receives 0 token from $u(H^*)$ and $n_1$ tokens from $v$; (b) $e_2$ receives $n_2 - 1$ tokens from $v$ and $n_1 - n_2 + 1$ tokens from $w$; (c) $e_3$ receives $n_1 - 1$ tokens from $w$ and one token from $v_i$. 
Appendix C

The Setting with $t = 1$

Consider the random $(d, k, t)$-CSP model $\text{CSP}_{n,m}^{d,k,t}$ with any constant triple $(d, k, t)$ such that $t = 1$ and $d, k \geq 2$. This random model is a generalization of the random $k$-SAT model by allowing a more general domain of $d$ variable values instead of $\{\text{TRUE, FALSE}\}$.

We will prove Conjecture 3.1, 3.2 and 3.3 with $\Lambda(d, k, t) = d - 1$. The proof follows directly from the approach we presented in Section 4. In this setting with $t = 1$, the forcer $H^*$ and the token function $g(c)$ are basically the same as those in CASE 1 of the setting with $k = 2$. The reader can check that the assumption of $k = 2$ is not used in (i) Chapter 4; (ii) the proofs of the two properties of $G^*$, i.e. Claim 5.2 and 5.3; and (iii) the arguments using the first and the second moment methods in Appendix A. Therefore, we can directly apply these arguments to the setting with $t = 1$ and $d, k \geq 2$.

What remains to show are (1) the special forcer subgraph $H^*$; (2) the values of $\Lambda(d, k, t), \alpha$ and $\beta$; (3) the valid $(\alpha, \beta)$-token allocations related to $H^*$; and (4) the token function $g(c)$. We will present all these in the next sections.

C.1 The Special Forcer Subgraph $H^*$

1-forciders. Given a set of input variables $I$ with fixed values $A_I$, create a constraint $e$ containing a variable $u$ and $k - 1$ variables $v_i \in I$. Recall that $t = 1$. By choosing an appropriate restriction in $e$, it can forbid one value at $u$. Such a constraint is called a 1-forbidding edge. We call the whole structure a 1-forbider, which has a set of $k - 1$ input variables with fixed values, an output variable $u$, and a 1-forbidding edge.

Forcer Subgraph $H^*$. $H^*$ contains (i) a set of $(d - 1)(k - 1)$ input variables $I$ with fixed values, (ii) an output variable $u$, (iii) $d - 1$ different 1-forbidding edges.
C.2 Values of $\Lambda(d, k, t)$, $\alpha$ and $\beta$

By Definition 4.21,

$$\alpha = |E(H^*)| = d - 1, \beta = |V(H^*)| - |I(H^*)| = 1.$$ 

By Definition 4.3,

$$\Lambda(d, k, t) = \frac{|E(H^*)|}{|V(H^*)| - |I(H^*)|} = d - 1.$$ 

C.3 Valid $(\alpha, \beta)$-Token Allocations

In a valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$, each constraint receives $\beta = 1$ tokens from $u(H^*)$, which gives out $d - 1 = \alpha$ tokens in total.

In a valid $(\alpha, \beta)$-token allocation on $(V(H^*) - u(H^*), E(H^*))$ specified in Claim B.3, each constraint receives $\beta = 1$ tokens from a variable $v_i \in I(H^*)$; and variables in $I(H^*)$ gives out $d - 1 = \alpha$ tokens in total.

With these two valid $(\alpha, \beta)$-token allocations, it is straightforward to prove Claim 5.2 and Claim 5.3, by repeating the arguments we used for the setting with $k = 2$.

C.4 Token Function $g(c)$

Define $g(0) = 0$ and $g(1) = t = 1$. It is easy to verify that the two required properties of $g(c)$ holds for every valid $(\alpha, \beta)$-token allocation $\mathcal{C}$ on any variable set $U$ and constraint set $F$ of any $(d, k, t)$-CSP instance:

$$\sum_\varepsilon g(c_{(\varepsilon, v)}) \leq d - 1 \text{ for every variable } v \in U \text{ (Claim 4.24)}$$

$$\prod_\varepsilon (g(c_{(\varepsilon, v)}) + 1) > t \text{ for every constraint } \varepsilon \in F \text{ (Claim 4.25)}$$
Appendix D

The Setting with $d = 2$

Consider the random $(d, k, t)$-CSP model $\text{CSP}_{n,m}^{d,k,t}$ with any constant triple $(d, k, t)$ such that $d = 2$, $k \geq 2$ and $1 \leq t < (d-1)d^{k-2} = 2^{k-2}$. This random model is a generalization of the random $k$-SAT model by allowing more restrictions in each constraint.

We will prove Conjecture 3.1, 3.2 and 3.3 with $\Lambda(d, k, t) = \lfloor \log_2 t \rfloor + 1$. The proof follows directly from the approach we presented in Section 4. In this setting with $d = 2$, the forcer $H^*$ and the token function $g(c)$ are basically the same as those in CASE 1 of the setting with $k = 2$. The reader can check that the assumption of $k = 2$ is not used in (i) Chapter 4; (ii) the proofs of the two properties of $G^*$, i.e. Claim 5.2 and 5.3; and (iii) the arguments using the first and the second moment methods in Appendix A. Therefore, we can directly apply these arguments to the setting with $d = 2$, $k \geq 2$ and $1 \leq t < (d-1)d^{k-2} = 2^{k-2}$.

What remains to show are (1) the special forcer subgraph $H^*$; (2) the values of $\Lambda(d, k, t)$, $\alpha$ and $\beta$; (3) the valid $(\alpha, \beta)$-token allocations related to $H^*$; and (4) the token function $g(c)$. We will present all these in the next sections.

For convenience, let $y = \lfloor \log_2 t \rfloor$. Since $d = 2$, this implies $|t/2^y| = 1$.

D.1 The Special Forcer Subgraph $H^*$

1-forbidders. Given a set of input variables $I$ with fixed values $A_I$, create a constraint $e$ containing a variable $u$, a set of $k-1-y$ variables $v_i \in I$ and $y$ other variables. Recall that $|t/2^y| = 1$. By choosing appropriate restrictions in $e$, it can forbid one value at $u$. Such a constraint is called a 1-forbidding edge. We call the whole structure a 1-forbidder, which has a set of $k-1-y$ input variables with fixed values, an output variable $u$, a set of $y$ other variables, and a 1-forbidding edge.

Forcer Subgraph $H^*$. $H^*$ contains (i) a set of $k-1-y$ input variables $I$ with fixed values, (ii) an output variable $u$, (iii) $y$ other variables, (iv) a 1-forbidding edge.
D.2 Values of $\Lambda(d, k, t), \alpha$ and $\beta$

By Definition 4.21,
\[ \alpha = |E(H^*)| = 1, \beta = |V(H^*)| - |I(H^*)| = y + 1. \]

By Definition 4.3,
\[ \Lambda(d, k, t) = \frac{|E(H^*)|}{|V(H^*)| - |I(H^*)|} = \frac{1}{y + 1}. \]

D.3 Valid $(\alpha, \beta)$-Token Allocations

In a valid $(\alpha, \beta)$-token allocation on $(V(H^*) - I(H^*), E(H^*))$, each variable in $V(H^*) - I(H^*)$ gives $\alpha = 1$ tokens to the constraint, which receives $y + 1 = \beta$ tokens in total.

In a valid $(\alpha, \beta)$-token allocation on $(V(H^*) - u(H^*), E(H^*))$ specified in Claim B.3, each of the $y$ variables in $V(H^*) - u(H^*) - I(H^*)$ gives $\alpha = 1$ tokens to the constraint, and there is a variable $v_i \in I(H^*)$ that gives one token to the constraint as well. The constraint receives $y + 1 = \beta$ tokens in total.

With these two valid $(\alpha, \beta)$-token allocations, it is straightforward to prove Claim 5.2 and Claim 5.3, by repeat the arguments we used for the setting with $k = 2$.

D.4 Token Function $g(c)$

Define $g(0) = 0$ and $g(1) = 1$. It is easy to verify that the two required properties of $g(c)$ holds for every valid $(\alpha, \beta)$-token allocation $\mathcal{C}$ on any variable set $U$ and constraint set $F$ of any $(d, k, t)$-CSP instance:
\[
\sum_{e} g(c(e, v)) \leq d - 1 \text{ for every variable } v \in U \text{ (Claim 4.24)}
\]
\[
\prod_{e} (g(c(e, v)) + 1) > t \text{ for every constraint } e \in F \text{ (Claim 4.25)}
\]
Appendix E

Results for the Random Model

Consider the random $(d, k, t)$-CSP model $\text{CSP}_{d,k,t}^{n,p}$ with any constant triple $(d, k, t)$ such that $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$.

We presented our results for the random $(d, k, t)$-CSP model $\text{CSP}_{n,m}^{d,k,t}$ in Conjecture 3.1, 3.2 and 3.3 and Theorem 3.4, 3.5 and 3.6. Here, we provide the equivalent statements for the random $(d, k, t)$-CSP model $\text{CSP}_{n,p}^{d,k,t}$.

**Conjecture E.1** (Equivalent to Conjecture 3.1). For any positive constants $d, k, t$ and $\epsilon$ with $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$, when the constraint probability $p$ is $n^{-\frac{1}{\Lambda(d,k,t)}} + \epsilon$, w.h.p. a random instance drawn from $\text{CSP}_{n,p}^{d,k,t}$ has constant resolution complexity and constant tree resolution complexity.

**Conjecture E.2** (Equivalent to Conjecture 3.2). For any positive constants $d, k, t$ and $\epsilon$ with $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$, when the constraint probability $p$ is $n^{-\frac{1}{\Lambda(d,k,t)}} - \epsilon$, there exists a constant $\gamma > 0$ such that w.h.p. a random instance drawn from $\text{CSP}_{n,p}^{d,k,t}$ has tree resolution complexity $\exp(\Omega(n^{\gamma}))$.

**Conjecture E.3** (Equivalent to Conjecture 3.3). For any positive constants $d, k, t$ and $\epsilon$ with $1 \leq t < (d - 1)d^{k-2}$ and $d, k \geq 2$, when the constraint probability $p$ is $n^{-\frac{1}{\Lambda(d,k,t)}} - \epsilon$, there exists a constant $\gamma > 0$ such that w.h.p. a random instance drawn from $\text{CSP}_{n,p}^{d,k,t}$ has tree resolution complexity $\exp(\Omega(n^{\gamma}))$.

**Theorem E.4** (Equivalent to Theorem 3.4). For any positive constants $d, k, t$ and $\epsilon$ with $1 \leq t < d - 1$ and $k = 2$, when the constraint probability $p$ is $n^{-\frac{1}{\Lambda(d,k,t)}} + \epsilon$, w.h.p. a random instance drawn from $\text{CSP}_{n,p}^{d,k,t}$ has constant resolution complexity and constant tree resolution complexity.

**Theorem E.5** (Equivalent to Theorem 3.5). For any positive constants $d, k, t$ and $\epsilon$ with $1 \leq t < d - 1$ and $k = 2$, when the constraint probability $p$ is $n^{-\frac{1}{\Lambda(d,k,t)}} - \epsilon$, there exists a constant $\gamma > 0$ such that w.h.p. a random instance drawn from $\text{CSP}_{n,p}^{d,k,t}$ has tree resolution complexity $\exp(\Omega(n^{\gamma}))$.

**Theorem E.6** (Equivalent to Theorem 3.6). For any positive constants $d, k, t$ and $\epsilon$ with $1 \leq t < d - 1$ and $k = 2$, when the constraint probability $p$ is $n^{-\frac{1}{\Lambda(d,k,t)}} - \epsilon$, there exists a constant $\gamma > 0$ such that
w.h.p. a random instance drawn from $\text{CSP}_{n,p}^{d,k,t}$ has tree resolution complexity $\exp(\Omega(n^\gamma))$.

Note that the property of having a (tree) resolution refutation with length at most $\ell$ is a convex property with respect to the set of constraints - adding an extra constraint does not damage any (tree) resolution refutation. Similarly, the property of not having any (tree) resolution refutation with length at most $\ell$ is also a convex property with respect to the set of constraints - removing a constraint does not create any shorter (tree) resolution refutation. As mentioned in Section 2.3.4, a convex property must hold w.h.p. in a random instance drawn from $\text{CSP}_{n,m}^{d,k,t}$ if it holds w.h.p. in a random instance drawn from $\text{CSP}_{n,m}^{d,k,t}$ for every $m = (1 \pm o(1))M \cdot p$. Therefore the statements provided in Chapter 3 for $\text{CSP}_{n,m}^{d,k,t}$ implies the statements above for $\text{CSP}_{n,p}^{d,k,t}$ - the $o(1)$ term in $(1 \pm o(1))M \cdot p$ is absorbed by the $n^\gamma$ term in the constraint density.