School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

Backdoor Set Detection for 3CNF Formulas

by

Andrew Kaploun

Thesis submitted as a requirement for the degree of

Bachelor of Science in Computer Science and Engineering (Honours)

Submitted:	November 2019	Student ID:	z5025092
Supervisor:	A/Prof. Serge Gaspers	Topic ID:	

Abstract

In this thesis, we attempt to improve on the current lower bounds for constructing weak backdoors from 3CNF boolean formulas. In doing so, we investigate the Parameterized Measure and Conquer technique, as well as local search and its associated derandomizations. In preliminary research, we improve the algorithm for finding a weak backdoor from 3CNF to the Null formula to $O^*(2.38298^k)$, and research randomized approaches to improve the bound for a backdoor to $0 - Val$. This has immediate applications in SAT solving, and is novel in that it aims to introduce new approaches to backdoor algorithms.

Acknowledgements

I want to thank my supervisor, Serge Gaspers, for his expertise and erudition, fast feedback and responses, and continuous vital advice, feedback, and ideas, without whom the thesis would not exist. Also, I would like to give thanks to Aleks Ignjatovic, my assessor, for his invaluable feedback and advice.

This work has been inspired immeasurably by the combined body of literature of academics I cite in this thesis, who I unfortunately cannot thank individually despite their crucial contribution to this work.

Lastly, I would like to thank my parents, Nina and Ian, for providing me with food, support, and the education and background to be where I am.

Abbreviations

BE Bachelor of Engineering

LaTeX A document preparation computer program

PhD Doctor of Philosophy

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Parameterized computation was first formulated by Downey and Fellows [Dow12], and was novel in that it gave a formal framework for NP-Hard problems to be tractable. Since then, it has flourished into a rich field of literature, including multiple textbooks, [DF99] [FG06] [Nie06] [CFK$^+$15] with an innumerable variety of problems to solve.

Backdoors to Satisfaction by Serge Gaspers [GS12] defined the notion of Parameterized algorithms for backdoors to SAT classes. Misra et al. in 2013 [MORS13] summarized a number of upper and lower bounds for backdoor algorithms which guided our investigation.

In this thesis, we aim to attack new and existing problems in Parameterized SAT backdoors. Concretely, we wish to improve on the existing runtime bounds on WB(3CNF, 0-Val) and WB(3CNF, NULL).

In terms of positioning our attack, or in other words, preparation, we will need to develop a rigorous understanding of algorithms used to improve bounds on similar problems. Throughout the thesis, we argue why we believe the techniques we investigate translate fruitfully into solutions for our problems.

The first such technique we study is Parameterized Measure and Conquer, which we track through its evolution in terms of its applications to the 3-Hitting-Set problem.

We look at approaches by Niedermeider and Rossmanith [NR03] , Fernau [Fer10], and finally Wahlström, [Wah07] where the technique of Parameterized measure and conquer was iterated upon and the bounds for the 3-Hitting-Set problem improved. Additionally, we explain how WB(3CNF, NULL) is a generalisation of the 3-Hitting-Set problem, and why these techniques are relevant. Additionally, we look at some applications of k-Leaf-Spanning-Tree, and explain why this application of Parameterized Measure and Conquer may not be applicable to WB(3CNF, 0-Val).

The other technique that we will investigate is randomized local search. First proposed by Schöning [Sch99], we will look at the original derandomizations of this algorithm by Dantsin, Schöning et al. [DGH$^+$02], an improved derandomization by Moser and Scheder [MS11], and finally an improvement to the algorithm by Kutzkov and Scheder [KS10]. We will then explain how this may apply to WB(3CNF, 0-Val).

Chapter 1 gives a high level summary of the topics that we will discuss. Chapter 2 explains the concepts that underlie our investigation. Chapter 3 gives a review and summary of the relevant literature. Chapter 4 gives some preliminary results we have proved so far. Chapter 5 proposes work to perform in Thesis B and C. Finally, chapter 6 draws up conclusions on our work, evaluates our progress so far, and gives recommendations for the remaining course of our research.

Chapter 2

Background

2.1 Parameterized Algorithms

For the purpose of this thesis, it is sufficient for simplicitly to define Parameterized algorithms as algorithms with an additional parameter for runtime analysis, giving us [CFK+15] **Definition 2.1.1.** An algorithm is described by the below:

Input of size $n \in \mathbb{N}$
Runtime described by $f(n)$, for some function $f : \mathbb{N} \to \mathbb{N}$

Whereas, Parameterized algorithms are described with **Definition 2.1.2.**:

Input of size $n \in \mathbb{N}$
Parameter $k \in \mathbb{N}$
Can describe the runtime by $f(k, n)$, for some function $f : (\mathbb{N}, \mathbb{N}) \to \mathbb{N}$

We use O^* notation heavily to allow us to ignore polynomial factors in the input size.

Definition 2.1.3. $O^*(f(k)) \equiv O(\text{poly}(n) \cdot f(k))$

2.2 Vertex Cover

The size of the solution is often a parameter, for which Vertex Cover is a classic example. We give this simple example to eventually act as a familiar bridge into more complex algorithms we describe later.

Input: A graph $G = (V, E)$ and natural number k
Parameter: k
Question: Is there a set of not more than k vertices such that $\forall e \in E, \exists v \in G$ such that v is adjacent to e?

Example: Branching Algorithm [CFK+15]

Algorithm 1: Vertex Cover

VC (V, E), k

```
if |E| = 0 then
    return True
end
if k = 0 then
    return False
end
Pick an arbitrary edge uv ∈ E
return VC(G − u, k − 1)
```

Example: Branching Algorithm

Analysis We bound the number of leaves in the search tree. Since each tree node takes polynomial time to process, the runtime is O^*(number of leaves in the search tree). Let $T(k)$ be the number of leaves in the search tree if we have budget k.

$$T(k) \leq T(k-1) + T(k-1).$$

Hence since we branch into two at each point in the search tree, and there are at most k layers, we can say that there are at most $O(2^k)$ nodes, thus we have a run time bound

by $O^*(2^k)$.

2.3 SAT

Literal A literal is a boolean variable with or without a negation. (eg. $x, \neg y$)

SAT

Input: A logical formula ϕ consisting of conjunctions ($\wedge$), disjunctions ($\vee$), and literals.
Question: Is there an assignment of true and false values such that ϕ is true?

Disjunctive Clause A disjunctive clause, or simply a **clause**, is a set of literals connected by a disjunction ($\vee$).

Conjunctive Normal Form A formula ϕ in CNF or Conjunctive Normal Form consists of clauses separated by conjunctions.

Example $(x \vee a \vee b) \wedge (\neg x \vee c \vee d) \wedge (\neg x \vee e \vee f)$

Assignment If we have a formula ϕ and a set of assignments τ, we denote ϕ with the assignments in τ substituted in by $\phi[\tau]$. (And clean up stray true clauses)

Example $(x \vee a \vee b) \wedge (\neg x \vee c \vee d) \wedge (\neg x \vee e \vee f)[x \leftarrow false] = (a \vee b)$

2.4 SAT Backdoors

SAT Class A class of SAT is a set of SAT formulae that satisfy some property.

SAT Class Examples A class of SAT is a set of SAT formulae that satisfy some property.

- 3-CNF: Formulae in CNF with clauses containing at most 3 variables.
- 2-CNF: Formulae in CNF with clauses containing at most 2 variables.

- 0-Val: Each clause has at least one negative literal (one variable of the form $\neg x$.) Note that if a formula $\phi \in 0 - val$, an entirely negative assignment satisfies ϕ.
- The Null Formula: True

Weak Backdoors A weak backdoor from class $\mathcal{C}_1$ to $\mathcal{C}_2$ for a formula ϕ is a truth assignment $\tau : Var(X) \to \{0, 1\}$ such that $\phi[\tau] \in \mathcal{C}_2$, and $\phi[\tau]$ is satisfiable.

Backdoors We also call weak backdoors ***backdoors*** or simply WB.

Backdoor Algorithms

Input: A formula $\phi \in \mathcal{C}_1$ and natural number k.
Parameter: k
Question:
Does there exist a backdoor from ϕ to a formula in $\mathcal{C}_2$ that consists of an assignment of no more than k variables? [GS12]

Example $WB(3CNF, NULL)$ with parameter k asks if we can make an assignment of no more than k variables that satisfies some satisfiable formula input into the algorithm.

For our purposes, assume no clause contains a variable both negatively and positively (eg. $(x \vee \neg x)$)

d-Hitting-Set

Input: A collection C of subsets of a finite set S, where $\|C\| \leq d$, and an integer k
Parameter: k
Question: Is there a subset $S' \subseteq S$ with $\|S'\| \leq k$ that requires S' to contain at least one element from each subset in C?

Example $\{(a, b, c), (c, d, e)\}, k = 1$

Chapter 3

Literature Survey

3.1 Summary

In the literature review, we look at, in particular, Parameterized Measure and Conquer, and local search. Evolutions of Parameterized Measure and Conquer have often manifested in improvements in the runtime d-Hitting-Set. Local Search has been applied to many problems, with SAT as a significant example.

3.2 Parameterized Measure and Conquer

Niedermeier and Rossmanith [NR03] gave an early form of parameterized measure and conquer that gave an $O^*(2.27^k)$ runtime. They encoded the state in equations resembling

$$T(k) = 1 + T(k-1) + T(k-2) + B(k-1)$$

$$B(k) = 1 + B(k-1) + T(k-1)$$

The 'state', i.e. whether it has a 2-set, is encoded in whether the equation is $B(k)$ or $T(k)$.

We give an exposition of their branching that could inspire our future work. For the algorithm, Niedermeier talked about simple cases as follows:

- If we have a singleton element $\{x\}$, we clearly must include x in our hitting set.
- If an element only occurs in one set, delete the element.
- We say element x is dominated by an element y if y occurs in every set x occurs in. We can clearly say that if y dominates x, then we can remove x from the instance since we may as well include y instead of x.

Next, to bound B_k, that is, when we have at least one clause with a set of size 2, Niedermeider describes the following cases

- If there are sets $\{x, y\}$ and $\{x, a, b\}$, we branch on either taking x or y to be in our hitting set, but not both. If we take y, then we know we do not take x, thus we can delete x from $\{x, a, b\}$, giving us a bound of B_{k-1}. Since no element occurs only once, there must be some set $\{y, a', b'\}$ and thus a similar case must follow for y, giving a bound of $2B_{k-1}$
- if we have $\{x, y_1\}, \{x, y_2\}, \{x, a, b\}$ we must either take x, or we must take both y_1 and y_2. Thus, our branching is bounded at $T_{k-1} + T_{k-2}$.
- in the case with $\{x, y\}, \{x, a, b\}, \{x, a, c\}$, we either take x for a T_{k-1} branch, or if we take y, we have the following cases for $\{a, b\}, \{a, c\}$:
  - We take a for a total bound of T_{k-2}
  - We take b, c for a total bound of T_{k-3}

  In total, we have $T_{k-1} + T_{k-2} + T_{k-3}$
- Otherwise, we have $\{x, y\}, \{x, a, b\}, \{x, c, d\}$, in which case we either take x, or if we take y, we are left with $\{a, b\}, \{c, d\}$. We thus branch additionally on either picking a or b, giving us two separate cases where we delete an extra variable and are left with a set of size 2, thus in total we have $T_{k-1} + 2B_{k-2}$.

Otherwise, we observe the case where an element occurs thrice: $\{x, a_1, a_2\}, \{x, b_1, b_2\}, \{x, c_1, c_2\}$.

- We branch on picking x or not picking x. If we pick x, we trivially have the branch T_{k-1}. Otherwise, we branch on picking a_1 or a_2, then branch on picking b_1 or b_2, thus giving us 4 cases in total where there is a set of size 3, which overall gives $T_{k-1} + 4B_{k-2}$.

For the case where an element occurs 4 times, $\{x, a_1, a_2\}, \{x, b_1, b_2\}, \{x, c_1, c_2\}, \{x, d_1, d_2\}$, we can similarly branch over either picking x, or branch over the sets

$$\{a_1, a_2\}, \{b_1, b_2\}, \{c_1, c_2\},$$

resulting with the same reasoning in a bound of $T_{k-1} + 8B_{k-3}$.

Finally, for the case where every variable occurs twice, consider a set $\{a, b, c\}$. We branch over picking a, b, or c. Without loss of generality, if we pick a, then we have $\{b, y, z\}$ elsewhere, and since b is now dominated, we must change this to $\{y, z\}$. Thus, we introduce a set of size 2 and since this case occurs in all 3 branches, we have $3B_{k-1}$.

These rules allowed Niedermeier to create a system of two equations representing the number of nodes in the search tree, and thus prove their result.

Fernau [Fer10] did a more detailed case analysis where for $d-$Hitting-Set, if there are at least n 2-sets,

$$T_d^n(k) = \text{The number of leaves in the search tree with a budget of } k,$$

and hence achieved a bound of $O^*(2.1788^k)$. We found this system more expedient to work with, and used it in our work.

Note that the methods described so far still only have the capability to take into account one type of 'measure'. If we want to encode a number of characteristics in our equation, and solve this to obtain a bound in terms of the number of nodes of the tree, we quickly run into complications.

3.3 Wahlström's Method

Walhström [Wah07] refined an approach for exact exponential algorithms by Eppstein [Epp06]. He gave an approach for assigning many weights, for both parameterized and exact exponential problems. He defines states of a problem F as

$$S(F) = k \iff F \text{ is in state } S_k.$$

Then we can define a measure to take into account the parameter and the state for the weight

$$f(F) = n(F) - \psi(S(F)).$$

With this technique, it is easy to consider all of our states, define all of the transitions, and then analyse the resulting exponential equation with a convex solver to find an upper bound for the branching. Using this technique, Walhström created an alorithm that ran in $O^*(2.0755^k)$, the best known current bound.

3.4 k-Leaf-Spanning-Tree

Input: A graph G, a natural number k.
Parameter: k
Question: Does G contain a subgraph that is a spanning tree with k leaves?

This problem was extensively studied since 1988 when it was proven to be FPT. [FL92] Curiously, one of the earliest bounds for the problem was $(O^*(17k^4!))$. [Bod89]. After a lot of work in the area in the intermediate years, Kneis et al. [KLR11] proved an $O^*(4^k)$ bound, and Daligault et al. [DGKY10] improved this to $O^*(3.72^k)$ soon after.

Afterwards, Kneis et al. used the same idea as above for an exact exponential algorithm, however with terminology more that is more conventional these days. The recursive call included the following sets, and the measure incorporated the size of each:

- Leaf nodes: Nodes that we know are leaves of the spanning tree

- Internal nodes: Nodes that we know are internal to the spanning tree
- Branching nodes: Nodes that lie at our 'border': They may or may not become leaf nodes or end nodes, and will be branched on first
- Floating vertices: Vertices that are leaves, but not yet 'attached' to the tree
- Free vertices: Vertices that do not belong to any of the above category

The current lowest bound, using similar terminology but an intricate observation into the cases, is by Zehavi [Zeh18].

3.5 Local Search for k-SAT

Schöning [Sch99] gave a very simple but powerful algorithm for SAT.

Algorithm 2: *SAT*

Randomized SAT *(ϕ, τ)*

Randomly pick an assignment for phi

while *ϕ is unsatisfied and counter* $\leq 3n$ **do**

Pick an unsatisfied clause C uniformly at random

Pick a literal x from C uniformly at random

flip the underlying variable for x to be true

end

Schöning proved that if we repeat this algorithm ad infinitum, the expected value of the runtime is

$$O^*\left(\left(\frac{2(k-1)}{k}\right)^n\right),$$

and thus $O^*(1.334^n)$ for $k = 3$.

To prove this, we give some definitions.

Definition 3.5.1 The hamming distance $\mathcal{H}$ between two equal length bitstrings is the number of positions in which they differ.

Definition 3.5.2 $B_{\mathcal{H}}(s, n)$ Denotes the set of all bitstrings no more than n hamming distance from string s.

To illustrate the simplicity of this algorithm, we give a proof jointly inspired by the original paper, a course taught by Spielman, [Spi07] and Fomin Kratsch [FK10].

Proposition 3.5.3 Schöning's Algorithm runs in an expected time of $O^*(1.334^n)$.

Proof.

The idea of the proof is that if each run of the procedure has probability p of finding an assignment, then we need $1/p$ runs on average to find an assignment.

First, we will show that the procedure has a $O\frac{1}{1.5^n}$) chance of succeeding. We show this by considering the states 0 through to n, where state i denotes a hamming distance of i from the closest assignment. We think of each flip as having some probability of increasing the state by 1, and some probability of decreasing the state by 1. The subroutine can be thought of as a random walk over these states.

When chosen uniformly at random, we know that each 'flip' has at least a $\frac{1}{3}$ chance of decreasing the hamming distance, and thus we can say the chance of increasing the distance is $\frac{2}{3}$.

Thus, if l is the hamming distance from a satisfying assignment of our initial randomly chosen assignment, we have that the probability of reaching the satisfying assignment given our random walk of $3n$ steps is at least the probability that $2l$ of the first $3l$ steps decrease the distance, which is represented by the expression

$$\binom{3l}{2l}\left(\frac{1}{3}\right)^{2l}\left(\frac{2}{3}\right)^{l}.$$

For a brief aside, we note that using Stirling's formula and the corresponding binary entropy function, we have the approximation

$$\binom{n}{\alpha n} \approx \left(\frac{1}{\alpha}\right)^{\alpha n}\left(\frac{1}{1-\alpha}\right)^{(1-\alpha)n}$$

to within a polynomial factor, which allows us to say

$$\binom{3l}{l} \geq \frac{1}{5l}\frac{3^{3l}}{2^{2l}},$$

and hence

$$\binom{3l}{2l}\left(\frac{1}{3}\right)^{2l}\left(\frac{2}{3}\right)^{l} \geq \frac{1}{\sqrt{5n}}\frac{3^{3l}}{2^{2l}}\left(\frac{1}{3}\right)^{2l}\left(\frac{2}{3}\right)^{l}.$$

Then, to calculate the expected value over the probability space of all 2^n starting assignments, note that the probability of picking an assignment with starting distance l is

$$\binom{n}{l},$$

and hence the probability is at least

$$\begin{aligned}\sum_{l=0}^{n}\frac{1}{2^n}\binom{n}{l}\frac{1}{\sqrt{5n}}\frac{3^{3l}}{2^{2l}}\left(\frac{1}{3}\right)^{2l}\left(\frac{2}{3}\right)^{l} &\geq \frac{1}{\sqrt{5n}}\frac{1}{2^n}\sum_{l=0}^{n}\frac{1}{2^n}\binom{n}{l}\frac{1}{2^l}\\ &= \frac{1}{\sqrt{5n}}\frac{1}{2^n}\left(1+\frac{1}{2}\right)^{n}\\ &= \frac{1}{\sqrt{5n}}\left(\frac{3}{4}\right)^{n}.\end{aligned}$$

Thus, since we know that if an event occurs with probability p, then the expected number of trials after which the event will occur is $\frac{1}{p}$, we conclude that the expected number of polynomial-time subroutines we run before we find our assignment is $\sqrt{5n}\left(\frac{4}{3}\right)^{n}$, which is

$$O^{*}(\left(\frac{4}{3}\right)^{n}),$$

as required. □

3.6 Derandomization Results

The first derandomisation was obtained by Dantsin, Goerdt, Hirsch, Kannan, Kleinberg, Papadimitriou, Raghavan and Schöning [DGH^{+}02] who gave a derandomization

that gives a deterministic runtime of

$$O^*\left(\left(\frac{2k}{k+1}\right)^n\right),$$

which for $k = 3$ is $O^*(1.5^n)$.

A faster derandomization was given by Moser & Scheder [MS11], who Derandomized this algorithm to prove a bound of

$$O^*\left(\left(\frac{2(k-1)}{k}\right)^{n+o(n)}\right),$$

and thus $O^*(1.334^{n+o(n)})$ for $k = 3$.

Since we have not yet evaluated which derandomization is best for our purposes, and which optimizations we wish to make, it is in our best interest to describe the general idea for how the algorithms are derandomized. First, they define a problem Ball-3-SAT as follows:

Input: A 3-CNF formula F over n variables, a truth assignment α to the variables, and a natural number r. Question: Is there a satisfying assignment to F within hamming distance r of α?

Then, given some runtime, we then come up with some covering code of the Hamming Space over n variables.

Definition. A Covering Code is a set $S = \{s_1, ..., s_{2|\mathcal{H}|}\}$ of balls whose union $\bigcup_{s_i \in S} s_i = \mathcal{H}$ is the entire Hamming space.

Thus, we give a covering code of the Hamming space, and run the algorithm for each ball in the code, thus if there is a satisfying assignment in the Hamming Space, we find it in some ball.

Of particular interest to us is the derandomization of Kutzkov and Scheder [KS10], who published a derandomization running in $O^*(1.439^n)$, which is of interest to us since they did this entirely by improving Ball-3-SAT instead of the covering code, which as we will describe in our future plans, may have relevance to WB(3NCF, 0-Val).

3.7 WB(3CNF, 0-Val)

For WB(3CNF, 0-Val), Raman Shankar [RS13] used a branching analysis, not based in measure and conquer, to improve on the trivial $O^*(3^k)$ trivial bound, giving an algorithm that runs in $O^*(2.85^k)$. Note that of course the trivial bound can be obtained with a simple algorithm. Note that any formula in 0-Val must be able to be satisfied by an assignment of only False. Thus, we will only need to set variables to True in our backdoor.

Algorithm 3: WB(3CNF, 0-Val)

WB *(ϕ, k, τ)*

```
if k < 0 then
    return False
end
if φ[τ] ∈ 0-Val then
    return True
end
Pick an arbitrary unsatisied clause {x, y, z}
return Best(WB(φ, k − 1, τ ∪ (x ← True)), WB(φ, k − 1, τ ∪ (y ← True)),
 WB(φ, k − 1, τ ∪ (z ← True))
```

Trivially, this algorithm runs in $O^*(3^k)$ since the branching depth is no more than k, and we branch by a factor of 3. We omit the $O^*(2.85^k)$ algorithm, since the branch analysis is orthogonal to the techniques we wish to pursue.

Raman Shankar recommended in the conclusion that perhaps it may be a potential research problem to find a parameterized bound for WB(3CNF, Null). This will prove to be of some influence for the problem we chose to research.

Note that it can be easily intuitively observed that WB(3CNF, 2CNF) can be reduced to 3-Hitting-Set, which is indeed what Misra, Ordyniak, Raman, and Szeider [MORS13] proved in a summary of upper and lower bounds on backdoors. This lets us observe the

relationship between 3-Hitting-Set and WB(3NCF, NULL), by seeing that WB(3CNF, 2CNF) is a special case of WB(3CNF, NULL) where all variables only occur positively. We will relegate this to Chapter 4 to prove, but this fact is useful in visualising the relationship between the two problems.

Chapter 4

Preliminary Results and Preparations

Now that we have reviewed the relevant literature, we give some preliminary results that form a portion of our work.

4.1 Preliminary Definitions

In order to describe our algorithm concisely, we give some definitions.

Definition 4.1.1 A literal x in a clause is referred to as an (a, b) literal if x occurs a times in ϕ, and $\neg x$ occurs b times in ϕ.

Definition 4.1.2 (a, b) variables are made up of (a, b) and (b, a) literals.

Example 4.1.3 $(x \vee a \vee b) \wedge (\neg x \vee c \vee d) \wedge (\neg x \vee e \vee f)$

Definition 4.1.4 We say that a literal of the form (a', b') is of the form

- $(a+, b')$ if $a \leq a'$
- $(a', b+)$ if $b \leq b'$

- $(a+, b+)$ if both of the above conditions hold

Definition 4.1.5 In our algorithm, τ' will be a set containing literals that we guarantee we will not set to true.

Definition 4.1.6 A semi-2-clause is a 3-clause where 1 literal is in τ'.

4.2 Lemmas

Now, as a prerequisite to proving the runtime of our algorithm, we must prove some lemmas about solving particular instances of our problem.

Lemma 4.2.1 If every variable is of the form $(1+, 0)$, then we can solve this branch of WB(3CNF, NULL) in $O^*(2.0755^k)$.

Proof.

We construct a reduction to 3-Hitting-Set. First, replace every negative occurrence $(\neg x)$ of a variable with a positive occurrence (x). By symmetry, this is a valid reduction. For a 3CNF SAT formula ϕ consisting of clauses, simple map each clause with some variables x, y, z to a set with elements denoted by x, y, z in our instance of 3-Hitting-Set. It is then immediately clear that there exists a hitting set of no more than k variables if and only if there exists an satisfying assignment of only $True$ and size no more than k, since we can prove that for any variable x, the clauses satisfied when x is True map precisely to the sets hit when x is in our hitting set. □

Lemma 4.2.2 If every variable is of the form $(1, 1)$, then we can solve $WB(3CNF, NULL)$ in polynomial time.

Proof. First, we show that there exists a satisfying assignment of size $|\mathcal{C}|$, where $\mathcal{C}$ is the set of clauses of ϕ, if and only if the formula is satisfiable. For one side of the inequality, note $|\tau| \geq |\mathcal{C}|$ since each variable assignment can only satisfy one clause.

Then, to obtain an assignment τ such that $|\tau| \leq |\mathcal{C}|$, take any formula τ which has size greater than $|\mathcal{C}|$. While $|\tau| \leq |\mathcal{C}|$, pick an arbitrary variable from a clause that has more than 1 satisfied literal, and remove it from τ.

Since Tovey [Tov84] proved there exists such an assignment for any such ϕ, and that we can find in polynomial time, we complete the proof. □

4.3 Reduction Rules

Here, we describe the rules that all run in polynomial time that we use to reduce our problem to something more solvable, which all run in polynomial time. We also give a proof that these rules reduce true instances to true instances.

To give some intuition, the reduction rules will reduce the problem to something where we can branch better than the trivial 3-direction branching. i.e. $((1, 2+) \vee (1+, 1+) \vee (1+, 1+))$ or $((2+, 2+) \vee y \vee z)$

Rule 1. If there exists a clause with only one literal, add the variable to τ so as to make the literal true.

Proof. Trivial. □

Rule 2. If the same literal occurs more than once in any clause, remove the duplicate occurrences.

Proof. Trivial. □

Rule 3. If every variable is of the form $(1, 1)$, apply lemma 2.

Proof. Trivial. □

Rule 4. If ϕ has only variables of the form $(1, 1)$ and $(1+, 0)$, and we have a clause that contains a $(1, 1)$ literal and a $(1+, 0)$ literal, delete the $(1, 1)$ literal. **Proof.** Suppose that there exists an assignment of no more than k variables where for a $(1, 1)$ variable

x and $(1+,0)$ variable y in the same clause, we assign x but not y. But then we can construct an assignment of no more than k variables with y true and x unassigned. Thus, we can simply delete x from the clause since it will not be used to satisfy the clause. □

Rule 5. If ϕ has only variables of the form $(1,1)$ and $(1+,0)$, and no clauses have literals of both forms:

- If we have l clauses of $(1,1)$ variables, by Lemma 2 we can call our algorithm with parameters
$$G \leftarrow \phi - \{\text{clauses with } k \text{ variables}\}, k \leftarrow k - l$$

Clearly, if a formula consists of two disjoint sets of clauses $\mathcal{C}_1$ and $\mathcal{C}_2$, that share no variables, then if $\mathcal{C}_1$ can be satisfied in no less than k' assignments, then since a formula is a conjunction of clauses, ϕ is satisfiable if and only if $\mathcal{C}_2$ is satisfiable with $k - k'$ assignments.

Rule 6. If a clause contains a $(1,2+)$ literal and a $(1+,0)$ literal, delete the $(1,2+)$ literal.

Suppose there is an assignment of no more than k variables where for a $(1,2+)$ variable x and $(1+,0)$ variable y in the same clause, we assign x but not y. But then we can construct an assignment of no more than k variables with y true and x unassigned. Thus, we can simply delete x from the clause since it will not be used to satisfy the clause.

4.4 Branching Rules & Analysis

Definition. Let $T_n(k)$ denote the runtime of the algorithm for an instance where

- The parameter is k.
- (# of 2-clauses) + (# of semi-2-clauses) $\geq n$

Rule 1. If there is a 2-clause with literals x and y, branch on

- Adding a truth assignment that makes x true to τ
- Adding a truth assignment that makes y true to τ

Analysis.

Clearly, in either case, we have one fewer 2-clauses, but also one fewer variable, thus giving us the bound $T_n(k) \leq T_{n-1}(k-1) + T_{n-1}(k-1)$.

Rule 2.

If ϕ contains a clause that contains a literal x of form $(2+, 2+)$ branch on the following:

- Add an assignment to τ that makes x true.
- Add x to τ'

Analysis.

We know that either we will assign x True, or mark that it will never be True. Thus, we have the following cases:

- If we assign $x \leftarrow$ True, then since x is $(2+, 2+)$, then the two clauses containing $\neg x$ cannot be satisfied by $\neg x$, thus we may as well remove $\neg x$ from them, creating two 2-clauses. Thus, since we also assigned x, our branching is bound by $\mathrm{T}_{n+2}(k-1) If we add x to \tau'$, then similarly, the two clauses containing x cannot be satisfied by x, so we may as well delete it, creating two clauses and making the bound $T_{n+2}(k)$.

This gives a total bound of $T_n(k) \leq T_{n+2}(k) + T_{n+2}(k-1)$ $\square$

Rule 3. Note that after exhaustively applying the reduction rules, if we have a $(1, 2+)$ literal x, x shares a clause with only literals of the form $(1+, 1+)$.

Thus, denote our clause by $(x \vee y \vee z)$ branch on

- Add an assignment to τ that makes x true.
- Add an assignment to τ that makes y true.
- Add an assignment to τ that makes z true.

Analysis.

Let the literals in the clause be $\{x, y, z\}$. We branch on adding x, y, or z to the assignment. If we set x to true, then the two clauses containing $\neg x$ cannot be satisfied by $\neg x$, thus we may as well remove $\neg x$ from them, creating two 2-clauses. Using the note for y and z, we apply the same analysis except there is only one negative clause, giving a bound of $T_{n+1}(k-1)$ for y and z. Thus, we have a total of $T_n(k) \leq T_{n+2}(k-1) + 2T_{n+1}(k-1)$.

Our main subroutine for the algorithm is thus:

Algorithm 4: $WB(3CNF, NULL)$

$WB(3CNF, NULL)$ *(ϕ, k, τ')*

Apply the branching rules exhaustively.

if *Branching Rule 1 applies* **then**
    Apply Branching Rule 1
else if *Branching Rule 2 applies* **then**
    Apply Branching Rule 2
else if *Branching Rule 3 applies* **then**
    Apply Branching Rule 3

Theorem. The runtime of our algorihtm is $O^*(2.38298^k)$.

Proof.

$$T_n(k) \leq max \begin{cases} T_{n+2}(k) + T_{n+2}(k-1) \\ T_{n+2}(k-1) + 2T_{n+1}(k-1) \end{cases}$$

Applying Rule 1,

$$T_n(k) \leq max \begin{cases} 4T_n(k-2) + 4T_n(k-3) \\ 4T_n(k-3) + 4T_n(k-2) \end{cases}$$

Thus a suitable function is an exponential function with base c such that

$$c^k \leq 4c^{k-2} + 4c^{k-3} \iff c^3 \leq 4c + 4.$$

So we can pick $c = 2.38298$ giving us the bound

$$O^*(2.38298^k).$$

□

Chapter 5

Project Plan

5.1 Exploring Parameterized Measure and Conquer for WB(3CNF, NULL)

With regard to WB(3CNF, NULL), we obviously have an advantage when we have 2-clauses and semi-2-clauses. We can attempt to find other 'measures' that gives us an advantage, although this may potentially have a high effort cost and low return.

In addition, upon closer examination of the Neidermeider paper, [NR03] we notice that there is a case distinction made for the case where a set of size 2 exists, instead of simply branching like we do. Thus, it may be possible that we can obtain better bounds for $T_1(k)$ and $T_2(k)$ in our proof, and hence a better bound overall. If we were to use something like Wahlström's method [Wah07] to aid us in doing so, this would allow us to perform a case analysis in a more strategic way and avoid explicitly bounding so many cases.

5.2 WB(3CNF, 0-Val)

One possible parameter we can explore is having a set of 'unassigned' variables, a set of 'definitely not true', and a set of 'definitely not false' variables to aid in the analysis. This has worked well for problems where you have to 'pick' some number of variables, like k-leaf-spanning tree.

For local search, it quickly became apparent that the techniques applicable to k-Leaf Spanning Tree problem will not be particularly useful. Although it seemed tempting at first to increase the number of 'states' that a variable can be in if a simple measure yields unfruitful results, we found that for the following reason, our problem was disanalogous: In k-Leaf Spanning Tree, we decrease the measure when we know that a variable is a 'Floating Leaf ', that is, we know it is a leaf, but do not know which internal node it will be attached to. Whereas, in any reasonable reduction step that we would perform, if we make a variable True, we immediately delete the clause since it it satisfied, and if we fix a variable to be false, we immediately delete the variable, since it cannot satisfy any clauses. Thus, there is not much information to gain about neighbours if we fix the 'state ' of a variable, unlike in k-Leaf Spanning Tree.

However, we believe we will be able to improve on the bound using local search. Similar to the local search for SAT, we can start with an all '0' assignment, randomly pick an unsatisfied clause and randomly satisfy unsatisfied clauses with a '1'. Then, to give a sketch of the algorithm, we can apply a strategy similar to the randomized SAT algorithm, and start at an assignment chosen uniformly at random, calling the subroutine to 'flip ' variables to satisfy our formula. We are very eager to think about and describe this algorithm in detail, and provide a proof, throughout Thesis B and Thesis C.

5.3 Additional Work

Concretely, the work we have described entails describing some additional cases for our current algorithm, and creating a local search algorithm for WB(3CNF, 0-Val). If we complete these during Thesis B or Thesis C, we have no doubt that throughout the year we will encounter a topic to pursue that is amenable to our interests.

Chapter 6

Conclusion

Throughout this work, we have considered the history of the Parameterized Measure and Conquer technique, the local search technique, and given an overview of their applicability to weak SAT backdoor algorithms.

We believe that we have made inroads into improving algorithms for weak SAT backdoors, firstly by improving the bound for WB(3NF, NULL) to $O^*(2.38298^k)$, and also by investigating techniques for improving the current bound of $O^*(2.85^k)$ for WB(3CNF, 0-Val).

In moving forward, we will attempt to improve on the bound for WB(3CNF, 0-Val), and potentially attempt to parlay our efforts into extending the scope of our investigation to other problems.

Bibliography

[Bod89] Hans L. Bodlaender. On linear time minor tests and depth first search. In *Algorithms and Data Structures, Workshop WADS '89, Ottawa, Canada, August 17-19, 1989, Proceedings*, pages 577–590, 1989.

[CFK+15] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.

[DGH+02] Evgeny Dantsin, Andreas Goerdt, Edward A. Hirsch, Ravi Kannan, Jon M. Kleinberg, Christos H. Papadimitriou, Prabhakar Raghavan, and Uwe Schöning. A deterministic $(2\text{-}2/(k+1))^n$ algorithm for k-sat based on local search. *Theor. Comput. Sci.*, 289(1):69–83, 2002.

[DGKY10] Jean Daligault, Gregory Z. Gutin, Eun Jung Kim, and Anders Yeo. FPT algorithms and kernels for the directed k-leaf problem. *J. Comput. Syst. Sci.*, 76(2):144–152, 2010.

[Dow12] Rod Downey. The birth and early years of parameterized complexity. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, pages 17–38, 2012.

[Epp06] David Eppstein. Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Trans. Algorithms*, 2(4):492–509, 2006.

[Fer10] Henning Fernau. A top-down approach to search-trees: Improved algorithmics for 3-hitting set. *Algorithmica*, 57(1):97–118, 2010.

[FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.

[FK10] Fedor V. Fomin and Dieter Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2010.

[FL92] Michael R. Fellows and Michael A. Langston. On well-partial-order theory and its application to combinatorial problems of VLSI design. *SIAM J. Discrete Math.*, 5(1):117–126, 1992.

[GS12] Serge Gaspers and Stefan Szeider. Backdoors to satisfaction. In *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*, pages 287–317, 2012.

[KLR11] Joachim Kneis, Alexander Langer, and Peter Rossmanith. A new algorithm for finding trees with many leaves. *Algorithmica*, 61(4):882–897, 2011.

[KS10] Konstantin Kutzkov and Dominik Scheder. Using CSP to improve deterministic 3-sat. *CoRR*, abs/1007.1166, 2010.

[MORS13] Neeldhara Misra, Sebastian Ordyniak, Venkatesh Raman, and Stefan Szeider. Upper and lower bounds for weak backdoor set detection. In *Theory and Applications of Satisfiability Testing - SAT 2013 - 16th International Conference, Helsinki, Finland, July 8-12, 2013. Proceedings*, pages 394–402, 2013.

[MS11] Robin A. Moser and Dominik Scheder. A full derandomization of schöning's k-sat algorithm. In *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 245–252, 2011.

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[NR03] Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *J. Discrete Algorithms*, 1(1):89–102, 2003.

[RS13] Venkatesh Raman and Bal Sri Shankar. Improved fixed-parameter algorithm for the minimum weight 3-sat problem. In *WALCOM: Algorithms and Computation, 7th International Workshop, WALCOM 2013, Kharagpur, India, February 14-16, 2013. Proceedings*, pages 265–273, 2013.

[Sch99] Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 410–414, 1999.

[Spi07] Daniel A Spielman. Note's on schöning's algorithm for 3-sat. University Lecture. Available at http://www.cs.yale.edu/homes/spielman/366/schoening.pdf (23/4/2007), 2007.

[Tov84] Craig A. Tovey. A simplified np-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

[Wah07] Magnus Wahlström. *Algorithms, measures and upper bounds for satisfiability and related problems.* PhD thesis, Linköping University, Sweden, 2007.

[Zeh18] Meirav Zehavi. The k-leaf spanning tree problem admits a klam value of 39. *Eur. J. Comb.*, 68:175–203, 2018.