On Exact Algorithms for the Maximum Independent Set Problem

WONG, Wing Chun

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of Master of Philosophy in Mathematics

©The Chinese University of Hong Kong August 2008
Thesis/Assessment Committee

Professor Ng Kung Fu (Chair)
Professor Feng Dejun (Thesis Supervisor)
Professor Cai Leizhen (Committee Member)
Professor Ting Hing-fung (External Examiner)
Abstract of thesis entitled:
On Exact Algorithms for the Maximum Independent Set Problem
Submitted by WONG, Wing Chun
for the degree of Master of Philosophy
at The Chinese University of Hong Kong in June 2008

The maximum independent set problem is one of the classical NP-complete problem. So far, no efficient exact algorithms are found. The design of exact algorithms for the maximum independent set problem has a long history. Some algorithms use complicated rules in order to get better performance. Recently, a novel approach called “Measure and Conquer” has been invented with improved asymptotic running time over previous methods. However, the published analysis is somewhat complicated, and the idea becomes less transparent, due to the necessity of including many branching rules in order to obtain good running time. In this thesis, we demonstrate the underlying principle of “Measure and Conquer” by applying this approach to several simple algorithms. It will be shown that considerably better running time can be obtained in some cases over the traditional way of analyzing such algorithms.
摘要

最大獨立集合問題是其中一個經典的NP-complete之問題。至今仍沒有人找到有效率（時間複雜性為多項式）的演算法予以解決。設計解決最大獨立集合問題的演算法已有很悠久的歷史。有些演算法運用複雜的規則達到更高的效率。我們會在這篇論文中研討優化演算法的原理，並運用較簡單的演算法作例子，解釋如何在時間複雜性的上限取得改進。
Acknowledgement

I would like to express my deepest gratitude to my supervisors Professor Andrew Yao and Professor De-jun Feng for their guidance and kindly support in my graduate studies, especially for their suggestions during thesis preparation. I would also like to thank my friends, Kwok-kun Kwong, Ka-shing Ng, Fu-man Lau, Siu-cheong Lau, Kwok-wai Chan, Kai-Leung Chan and Chun-kit Suen who are very supportive when I work on this thesis. I particularly would like to thank Professor Xiaoming Sun, who is always willing to explain my questions patiently when I encounter difficulties. Finally, I wish to thank my parents and Miss Enid Chang, for their endless love, support and encouragement.
This work is dedicated to my parents.
# Contents

Abstract

Acknowledgement

1 Introduction

2 Background Study
   2.1 Basic Definitions and Notations
   2.2 Tarjan and Trojanowski's algorithm
      2.2.1 Techniques
      2.2.2 Algorithm
   2.3 Fomin, Grandoni and Kratsch's Algorithm
      2.3.1 Techniques
      2.3.2 Algorithm

3 Improvements
   3.1 Tarjan and Trojanowski's Algorithm
      3.1.1 Correctness and Running Time Analysis
      3.1.2 Improvement
      3.1.3 Using more weights
   3.2 The First Algorithm
      3.2.1 Standard Analysis
      3.2.2 Measure and Conquer
      3.2.3 Using more weights
   3.3 The Second Algorithm
3.3.1 Running Time Analysis 44
3.3.2 Using More Weights 45
3.4 The Third Algorithm 46

4 Lower Bounds 52
4.1 Tarjan and Trojanowski's Algorithm 52
4.2 The First Algorithm 55
4.3 The Second Algorithm 58
4.4 The Third Algorithm 60

5 Conclusion 63

Bibliography 66
List of Figures

2.1 (a): A graph with a maximum independent set of size 3; (b): Folding of vertex 1 in (a) ............ 11
2.2 (a): A graph with a maximum independent set of size 4; (b): Folding of vertex 1 in (a) ............ 12
2.3 $u$ is a mirror of $v$ ........................................ 13
2.4 $u$ is not a mirror of $v$ ................................. 13
2.5 A simple algorithm given by Fomin, Grandoni and Kratsch [3] ........................................ 15

3.1 The Original Algorithm (Part I) [8] ............... 20
3.2 The Original Algorithm (Part II) [8] .............. 21
3.3 The Original Algorithm (Part III) [8] .......... 22
3.4 The Original Algorithm (Part IV) [8] ........... 23
3.5 The Original Algorithm (Part V) [8] .......... 24
3.6 The Original Algorithm (Part VI) [8] .......... 25
3.7 The Modified Algorithm (Part I) ................... 26
3.8 The Modified Algorithm (Part II) ................. 27
3.9 A simple algorithm that solves the maximum independent set problem, adopted from [9] ....... 38
3.10 The second algorithm, modified from [9] ........ 43
3.11 The third algorithm, modified from [3, 1] ....... 47
3.12 $v$ is not foldable in the first graph while rule (2) applies in the third and fourth graphs ........ 48

4.1 Input graph $G_l$ for $l = 3$, for the modified Tarjan and Trojanowski’s Algorithm ............... 53
4.2 The graph after removing $u_1$. ........................... 55
4.3 The graph after removing $u_1$ and $N^*(u_2)$........... 56
4.4 Input graph $G_l$ for $l = 3$, for the first algorithm . 57
4.5 Input graph $G_l$ for $l = 3$, for the third algorithm . 61
List of Tables

5.1 Results on the four algorithms ............... 64
Chapter 1

Introduction

The problem of whether $P = NP$ is one of the greatest open problems in theoretical computer science. So far, no one has been able to prove the existence of a problem in $NP$ which is not in $P$. It is now commonly believed that $P \neq NP$ since nobody succeeded in finding a polynomial time exact algorithm for an NP-complete problem.

The maximum independent set problem (MIS) is a well-known NP-complete problem [6]. Finding a polynomial time algorithm for the problem seems to be remote. One trivial algorithm is to search through all subsets of vertices. This will take $O^*(2^n)$ time. Researchers tried to design faster algorithms with time complexity $O^*(2^{cn})$ by reducing the exponent $c$, where $0 < c < 1$. Many of the existing exact algorithms for the MIS problem are search-tree based which involve a large number of reduction and branching rules. The number of vertices $n$ of a graph $G$ is often taken as a measure of the size of the problem in the running time analysis. The case analysis is often very complicated and tedious
CHAPTER 1. INTRODUCTION

due to a large number of reduction and branching rules. Standard techniques are often used to solve the linear recurrences derived from the long list of branching rules.

Recently, a new approach called "Measure and Conquer" is introduced [2, 3]. The idea is to use a non-standard measure for the size of a problem in order to capture certain properties of an algorithm. Fomin, Grandoni and Kratsch applied the above method on the minimum dominating set problem [2] and the maximum independent set problem [3]. Their algorithm for the maximum independent set problem is very simple and can be described in a few lines. It has running time $O^*(2^{0.288n})$.

In this thesis, we will investigate on the idea and underlying principles of speeding up of existing exact algorithms for the maximum independent set problem. In chapter 2, background and the history of designing exact algorithms for the problem will be introduced. We will also give some details on the techniques used in the algorithms. In chapter 3, we will discuss principles of speeding up algorithms by "Measure and Conquer". We will also illustrate this approach by applying the technique on four different simple algorithms. Although the algorithms that we study are slower than that in [3], we give a less complicated running time analysis and we can illustrate the idea of "Measure and Conquer" more clearly. The first one is modified from Tarjan and Trojanowski's algorithm [8]. The modified algorithm consists of two pages and the improvement is not good. It has running time $O^*(2^{0.39606n})$. The other three algorithms are mainly modified from Woeginger's [9], Fomin, Grandoni and
Kratsch's [3] and Beigel's [1]. The best algorithm among the three has running time $O^*(2^{0.29470n})$. The improvement by applying "Measure and Conquer" is significant. In chapter 4, lower bounds for time complexity of the four algorithms will be investigated to see if there are still rooms for improvement. Chapter 5 summarizes this thesis.

□ End of chapter.
Chapter 2

Background Study

Woeginger published a survey on exact algorithms for NP-hard problems [9], which includes the history of the design of exact algorithms for the MIS problem.

There is a long history of the design of exact algorithms to solve the MIS problem. Tarjan and Trojanowski (1977) were the first to break the $O(2^n)$ trivial bound. Their polynomial space algorithm has running time $O(2^{3.3})$ [8]. Their algorithm is based on the idea of dominance which will be introduced in section 2.2.1. Jian (1986) followed the approach of Tarjan and Trojanowski and performed a smarter case analysis to obtain a polynomial space algorithm with time complexity $O(2^{0.304n})$ [5]. In 1986, Robson published an $O(2^{0.296n})$-time polynomial space algorithm [7].

All the above-mentioned algorithms are search-tree based. The case analysis is difficult and tedious. They use a long list of branching and reduction rules. Each branching rule will lead to a linear recurrence which can be solved by standard techniques.
In the case analysis, they use the number of vertices $n$ of the given graph $G$ as the size of the problem. We will introduce and modify Tarjan and Trojanowski's algorithm [8] in chapter 3.

There emerged a new approach on the design of algorithms for NP-hard problems in recent years. In 2006, Fomin, Grandoni and Kratsch applied the “Measure and Conquer” technique to design a very simple algorithm which runs with time complexity $O(2^{0.288n})$ [3]. The algorithm can be described in a few lines. It is based on the ideas of dominance, folding and mirroring.

In the following, we will first introduce basic definitions and notations we use in this thesis. We will then introduce different techniques used in the mentioned algorithms. The lemmas and theorems in this chapter are known results. They are mainly adopted from [8, 3].

2.1 Basic Definitions and Notations

An undirected simple graph $G = (V, E)$ consists of a finite set $V$ of vertices and an edge set $E$ of unordered pairs of vertices. Let $v \in V$, note that $(v, v) \notin E$ in an undirected simple graph. Two vertices $v, w \in V$ are adjacent if $(v, w) \in E$. An unordered pair of vertices $(v, w)$ is called an anti-edge if $(v, w) \notin E$.

Let $S \subseteq V$. $S$ is an independent set if for all distinct $v, w \in S$, $(v, w) \notin E$. Let $\alpha(G)$ denote the maximum of the cardinalities of the independent sets of $G$. An independent set $S$ of $G$ with $|S| = \alpha(G)$ is called a maximum independent set. $S = \{3, 4, 5\}$ is a maximum independent set in Figure 2.1 (a)
while $S = \{1, 4, 5, 6\}$ is a maximum independent set in Figure 2.2 (a). In general, $\alpha(G) \geq |T|$ if $T$ is a maximal independent set. (maximal with respect to subset inclusion.)

We denote by $G[S]$ the graph induced by $S$, i.e. $G[S] = (S, E[S])$ where $E[S] = \{(v, w) \in E \mid v, w \in S\}$. We also let $G - S = G[V - S]$.

We define the degree of a vertex $v$ to be $d(v) = |\{(v, u) \in E \mid u \in V\}|$, i.e. the number of edges incident to $v$. A sequence of vertices $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n$ is a path of length $n$ if $(v_i, v_{i+1}) \in E$ for all $i = 1, 2, \ldots, n - 1$. The distance between two vertices $v$ and $w$ is the minimum length of the paths connecting them. Let $N^d(v)$ be the set of vertices at distance $d$ from $v$. In particular, $N^1(v) = N(v) = \{w \in V \mid (v, w) \in E\}$ is the neighbourhood of $v$. We also let $N^*(v) = N(v) \cup \{v\}$.

2.2 Tarjan and Trojanowski's algorithm

Tarjan and Trojanowski's algorithm has running time $O(2^{|S|})$ [8]. Their algorithm is recursive and based on a complicated case analysis. The algorithm, which consists of 5 pages, is primarily based on the concept of dominance.

2.2.1 Techniques

The following result is standard. For completeness, we give a self-contained proof here.

**Lemma 2.1.** Let $G = (V, E)$ and $v \in V$. Then $\alpha(G) = \max\{1+$
\[ \alpha(G - N^*(v)), \alpha(G - \{v\}) \].

**Proof.** Let \( S \) be a maximum independent set in \( G \). If \( v \in S \), then \( S - \{v\} \) is an independent set of \( G - N^*(v) \). In this case, we have \( \alpha(G) - 1 \leq \alpha(G - N^*(v)) \). If \( v \notin S \), \( S \) is an independent set of \( G - \{v\} \). In this case, we have \( \alpha(G) \leq \alpha(G - \{v\}) \). Thus, \( \alpha(G) \leq \max\{1 + \alpha(G - N^*(v)), \alpha(G - \{v\})\} \).

Conversely, let \( S_1 \) be a maximum independent set of \( G - N^*(v) \) and \( S_2 \) be a maximum independent set of \( G - \{v\} \). Then \( S_1 \cup \{v\} \) is an independent set of \( G \) and \( S_2 \) is also an independent set of \( G \). Both \( \alpha(G) \geq 1 + \alpha(G - N^*(v)) \) and \( \alpha(G) \geq \alpha(G - \{v\}) \) hold. Thus, \( \alpha(G) \geq \max\{1 + \alpha(G - N^*(v)), \alpha(G - \{v\})\} \) holds. \( \square \)

We extend the above idea to the following, which is adopted from [8].

**Lemma 2.2** (Adopted from [8]). Let \( G = (V, E) \). Assume that \( S \subseteq V \) and \( N(S) = \bigcup_{v \in S} N(v) \). Then \( \alpha(G) = \max\{|I| + \alpha(G - S - N(I)) \mid I \text{ is an independent set of } G[S]\} \).

**Proof.** Suppose \( S \neq \emptyset \) and \( S \neq V \).

Let \( J \) be a maximum independent set in \( G \). Take \( I = J \cap S \). Then \( I \) is an independent set of \( G[S] \). Note that \( J - I \) is an independent set of \( G - S - N(I) \). Thus, \( \alpha(G) \leq |I| + \alpha(G - S - N(I)) \leq \max\{|K| + \alpha(G - S - N(K)) \mid K \text{ is an independent set of } G[S]\} \).

Conversely, let \( I \) be an independent set of \( G[S] \) and \( J \) be a maximum independent set of \( G - S - N(I) \). Then \( I \cup J \) is
of branching rules. The algorithm solves fewer subproblems by applying dominance. The algorithm finally branches at $v$ with $d(v) \geq 6$ by Lemma 2.1.

The list of branching rules leads to a list of linear recurrences which can be solved by standard techniques. The running time was found to be $O(2^{3})$ [8].

2.3  Fomin, Grandoni and Kratsch’s Algorithm

Fomin, Grandoni and Kratsch proved that their algorithm has running time $O(2^{0.288n})$ [3] by applying the technique of “Measure and Conquer” in the running time analysis. Their algorithm is very simple and only consists of five rules. It is based on the concepts of dominance, folding and mirroring. The definitions, lemmas and theorems in this section are mainly adopted from [3].

2.3.1 Techniques

**Lemma 2.3** (Dominance, adopted from [3]). Let $G = (V, E)$. If there exist $v, w \in V$ such that $N^*(w) \subseteq N^*(v)$ ($w$ dominates $v$), then $\alpha(G) = \alpha(G - \{v\})$.

**Proof.** The inequality $\alpha(G) \geq \alpha(G - \{v\})$ is trivial since a maximum independent set $S$ of $G - \{v\}$ is an independent set of $G$. It remains to prove $\alpha(G) \leq \alpha(G - \{v\})$.

Let $S$ be a maximum independent set of $G$. Suppose $v \in S$. 
Since $N^*(w) \subseteq N^*(v)$ by assumption, we have for all $z \in N(w)$, $z \in N^*(v)$. As a result, we have $z \notin S$ for all $z \in N(w) - \{v\}$ since $v \in S$ by assumption. Therefore, it follows that $S \cup \{w\} - \{v\}$ is an independent set of $G - \{v\}$. On the other hand, suppose $v \notin S$, then $S$ is an independent set of $G - \{v\}$. In both cases, we have $\alpha(G) \leq \alpha(G - \{v\})$. Thus, the result follows.

In the following, the idea of folding is introduced, which is a key technique in the algorithm.

**Definition 2.2.** Let $G = (V, E)$. Three vertices $v_1$, $v_2$ and $v_3$ form an anti-triangle if $(v_i, v_j) \notin E$ for all $i, j = 1, 2, 3, i \neq j$.

**Definition 2.3** (Adopted from [3]). Let $G = (V, E)$ and $v \in V$. $v$ is foldable if $N(v) = \{u_1, u_2, ..., u_{d(v)}\}$ contains no anti-triangles, i.e. there do not exist three vertices in $N(v)$ which form an anti-triangle.

For example, vertex 1 in Figure 2.1 (a) is foldable since vertices 2, 3 and 4 do not form an anti-triangle. (Vertices 2 and 3 are adjacent.) Note that a vertex of degree 2 (where dominance cannot apply) is always foldable. Vertex 1 in Figure 2.2 is an example.

**Definition 2.4** (Adopted from [3]). Let $G = (V, E)$. Assume that $v \in V$ and $v$ is foldable. Folding of $v$ transforms $G$ into a new graph $\bar{G}(v)$ by the following steps:

1. add a new vertex $u_{ij}$ for each anti-edge $(u_i, u_j)$ in $N(v)$;
2. add edges between each $u_{ij}$ and the vertices in $N(u_i) \cup N(u_j)$;
Figure 2.1: (a): A graph with a maximum independent set of size 3; (b): Folding of vertex 1 in (a)

3. add one edge between each pair of new vertices;
4. remove $N^*(v)$.

Figures 2.1 and 2.2 are examples of folding.

A property of folding concerning the independence number $\alpha(G)$ is given below.

**Lemma 2.4** (Adopted from [3]). Let $G = (V, E)$ and $v \in V$. If $v$ is foldable, then $\alpha(G) = 1 + \alpha(\tilde{G}(v))$.

**Proof.** Let $S$ be a maximum independent set of $G$. If $v \in S$, then $S - \{v\}$ is an independent set of $\tilde{G}(v)$. If $v \notin S$, $N(v) \cap S \neq \emptyset$ since $S$ is of maximum cardinality. Note that $|N(v) \cap S| \leq 2$, otherwise $N(v) \cap S$ contains an anti-triangle contradicting $v$ is foldable. If $N(v) \cap S = \{w\}$, then $S - \{w\}$ is an independent set of $\tilde{G}(v)$. Suppose $N(v) \cap S = \{w_1, w_2\}$. Note that $S \cap (N(w_1) \cup N(w_2)) = \emptyset$. Furthermore, $N(w_1) \cup N(w_2) - N^*(v) \subseteq N(w_{12})$
CHAPTER 2. BACKGROUND STUDY

an independent set of G. Since \( I \cap J = \phi \), we have \( \alpha(G) \geq \max\{|K| + \alpha(G - S - N(K)) \mid K \text{ is an independent set of } G[S]\} \).

By Lemma 2.2, the algorithm selects \( S \subseteq V \), finds an independent set \( I \) of \( G[S] \) and for each such \( I \), finds a maximum independent set of \( G - S - N(I) \). This method is further improved by Tarjan and Trojanowski by introducing the concept of dominance \([8]\).

**Definition 2.1 (Dominance \([8]\)). Let \( G = (V, E) \). Suppose that \( S \subseteq V \) and \( I, J \) are independent sets in \( G[S] \). \( I \) dominates \( J \) in \( S \) if for any \( J' \subseteq V - S \) such that \( J' \cup J \) is independent, there is a set \( I' \subseteq V - S \) such that \( I' \cup I \) is independent and \( |I' \cup I| \geq |J' \cup J| \).

Suppose \( I \) dominates \( J \) in \( S \), then \(|I| + \alpha(G - S - N(I)) \geq |J| + \alpha(G - S - N(J))\). Thus, the algorithm needs not solve the subproblem of \( J \). For example, let \( v \in V \) and \( S = N^*(v) \). If \( w \in N(v) \), then \( \{v\} \) dominates \( \{w\} \) in \( S \) since if \( J \subseteq V - N^*(v) \) and \( J \cup \{w\} \) is independent, then \( J \cup \{v\} \) is also independent.

**2.2.2 Algorithm**

Let \( G = (V, E) \). The algorithm selects a vertex \( v \) with minimum degree. For a degree 1 vertex \( v \), suppose \( (v, w) \in E \). By above, \( \{v\} \) dominates \( \{w\} \) in \( N^*(v) \). Thus, the algorithm only solves the subproblem of \( G - N^*(v) \). For vertices of degree 2 to 5, the algorithm solves different subproblems based on a long list...
and $S$ does not contain any new vertex in $\tilde{G}(v)$. Therefore, we have $S \cap N(\ell_{12}) = \emptyset$. Thus, $S \cup \{\ell_{12}\} - \{\ell_1, \ell_2\}$ is an independent set of $\tilde{G}(v)$ and $\alpha(G) \leq 1 + \alpha(\tilde{G}(v))$ is proved.

It remains to show $\alpha(G) \geq 1 + \alpha(\tilde{G}(v))$. Let $S$ be a maximum independent set of $\tilde{G}(v)$. If $S$ does not contain any new vertices $\ell_{ij}$, then $S \cup \{v\}$ is an independent set of $G$ since $S \cap N^*(v) = \emptyset$. Otherwise, $S$ must contain only one such $\ell_{ij}$ since all newly added vertices are adjacent to each others by the construction of $\tilde{G}(v)$. Note that $S \cap N(\ell_{ij}) = \emptyset$ and $S \cap N^*(v) = \emptyset$. Furthermore, we have $N(\ell_i) \cup N(\ell_j) - N^*(v) \subseteq N(\ell_{ij})$. Therefore, we have $S \cap (N(\ell_i) \cup N(\ell_j)) = \emptyset$. In this case, $S \cup \{\ell_i, \ell_j\} - \{\ell_{ij}\}$ is an independent set of $G$. Thus, the desired inequality is proved. 

We will introduce mirroring, which is a little trick to help speed up the algorithm.

**Definition 2.5.** Let $G = (V, E)$ and $S \subseteq V$. $S$ is a **clique** if
CHAPTER 2. BACKGROUND STUDY

Figure 2.3: \( u \) is a mirror of \( v \).

Figure 2.4: \( u \) is not a mirror of \( v \).

for all distinct \( v, w \in S \), \( (v, w) \in E \).

Definition 2.6 (Adopted from [3]). Let \( G = (V, E) \) and \( v \in V \). A vertex \( u \in N^2(v) \) is a mirror of \( v \) if \( N(v) - N(u) \) is a clique (possibly empty).

Vertex \( u \) is a mirror of \( v \) in Figure 2.3 while vertex \( u \) is not a mirror of \( v \) in Figure 2.4. We denote the set of mirrors of \( v \) by \( M(v) \). In Figure 2.1 (a), \( M(1) = \{6\} \). If \( v \) is not contained in any maximum independent set, then the same holds for its mirrors \( M(v) \). A precise statement is given below.

Lemma 2.5 (Mirroring, adopted from [3]). Let \( G = (V, E) \) and
\( v \in V \). Then \( \alpha(G) = \max\{\alpha(G-\{v\}-M(v)), 1+\alpha(G-N^*(v))\} \).

**Proof.** By the proof of Lemma 2.1, it suffices to show \( \alpha(G) \leq \max\{\alpha(G-\{v\}-M(v)), 1+\alpha(G-N^*(v))\} \).

Let \( S \) be a maximum independent set of \( G \). If \( v \in S \), then \( S-\{v\} \) is an independent set of \( G - N^*(v) \). Thus, \( \alpha(G) \leq 1+\alpha(G-N^*(v)) \leq \max\{\alpha(G-\{v\}-M(v)), 1+\alpha(G-N^*(v))\} \). If \( v \notin S \), we have \( |S \cap N(v)| \geq 2 \). (If \( S \cap N(v) = \emptyset \), then \( S \) is not of maximum cardinality. If \( |S \cap N(v)| = 1 \), let say \( S \cap N(v) = \{w\} \), then choose \( S' = S \cup \{v\} - \{w\}. \) \( S' \) is a maximum independent set containing \( \{v\} \) and the first case applies.) Let \( u \in M(v) \).

Note that \( |S \cap (N(v) - N(u))| \leq 1 \) (\( N(v) - N(u) \) is a clique by assumption). Furthermore, we have \( (S \cap N(v) - N(u)) \cup (S \cap N(v) \cap N(u)) = S \cap N(v) \) and \( (S \cap (N(v) - N(u)) \cap (S \cap N(v) \cap N(u)) = \emptyset \). It follows that \( |S \cap N(v) \cap N(u)| \geq 1 \). This implies \( S \cap N(u) \neq \emptyset \). As a result, we have \( u \notin S \) which implies \( \alpha(G) \leq \alpha(G-\{v\}-M(v)) \leq \max\{\alpha(G-\{v\}-M(v)), 1+\alpha(G-N^*(v))\} \). \( \square \)

### 2.3.2 Algorithm

Fomin, Grandoni and Kratsch’s algorithm [3], called \( mis \) is given in Figure 2.5.

Let \( L(n) \) be the maximum number of leaves in the search tree to solve a problem of size \( n \), where \(|V| = n\). Note that \( L(0) = 1 \). If rule (1) is satisfied, we have \( L(n) \leq L(n_1) + L(n-n_1) \) where
int mis(G)
{
    (0) if (|V| = 1), return 1;
    (1) if (G contains a connected component C, where C ≠ G),
      return mis(C) + mis(G - C);
    (2) if (there exist vertices v and w such that w dominates v [Lemma 2.3]),
      return mis(G - {v});
    (3) if (there exists a foldable vertex v with d(v) ≤ 4, and N(v) contains
          at most 3 anti-edges),
      return 1+mis(\overline{G}(v));
    (4) select a vertex v of maximum degree;
      return max{mis(G - {v} - M(v)), 1+mis(G - N*(v))};
}

Figure 2.5: A simple algorithm given by Fomin, Grandoni and Kratsch [3]

1 ≤ n₁ ≤ n - 1. If any of rule (2) or (3) is satisfied, then
L(n) ≤ L(n - 1). If rule (4) is satisfied, suppose we branch
at v with d(v) = 3. Consider the first case, the worst case is
M(v) = φ, we can only remove {v}. After removing {v} there
must exist a degree two vertex, thus we can apply rule (2) or
(3). In the first case, at least 2 vertices are removed. In the
second case, N*(v) is at least of size 4 since d(v) = 3. Combin-
ing the two cases, we have L(n) ≤ L(n - 2) + L(n - 4). Now,
suppose d(v) ≥ 4. In the first case, at least {v} is removed. In
the second case, N*(v) is at least of size 5 since d(v) ≥ 4. Thus,
we have L(n) ≤ L(n - 1) + L(n - 5).

To solve the above recurrences, we first prove a general the-
orem. The result is standard. For completeness, we give a self-
contained proof here.

Theorem 2.1. Let S ⊆ R be a well-ordered set. Suppose we
have \( L(n) \leq \max_{1 \leq i \leq r}\{ L(n-a_i)+L(n-b_i) \} \), where \( r \in \mathbb{N}, n, a_i, b_i, n-a_i, n-b_i \in S \) and \( L(k) = 1 \) for all \( k \leq n_0 = \min\{a_i, b_i\} \). Let \( f_i(x) = x^{-a_i} + x^{-b_i} - 1 \), \( \alpha_i \) be the largest real root of \( f_i(x) = 0 \) and \( \alpha = \max_{1 \leq i \leq r}\alpha_i \). Then \( L(n) \leq \alpha^n \) for all \( n \in S \).

**Proof.** Fix \( i \) with \( 1 \leq i \leq r \). Note that \( f_i(1) = 1 \). Furthermore, we have \( \lim_{x \to +\infty} f_i(x) = -1 \) and \( f_i(x) \) is decreasing and continuous for all \( x > 0 \). Therefore, we have \( \alpha_i > 1 \). Thus, \( \alpha > 1 \).

Note that we have \( L(n_0) = 1 \leq \alpha^0 \). Assume \( L(k) \leq \alpha^k \) for all \( k < n \). Since \( f_i(x) \) is decreasing for all \( x > 0 \), we have \( f_i(x) \leq 0 \) for all \( x \geq \alpha \). In particular, for all \( 1 \leq i \leq r \) and \( n \in S \), we have \( \alpha^n-a_i + \alpha^n-b_i \leq \alpha^n \). By induction assumption, we have \( L(n) \leq \max_{1 \leq i \leq r}\{ L(n-a_i)+L(n-b_i) \} \leq \max_{1 \leq i \leq r}\{ \alpha^n-a_i + \alpha^n-b_i \} \leq \alpha^n \). \( \square \)

By Theorem 2.1, the above problem is reduced to solving the roots of \( x^{-4} + x^{-2} - 1 = 0 \) and \( x^{-5} + x^{-1} - 1 = 0 \). By numerical methods, we find that \( \alpha = 1.3247... < 2^{0.406} \). Note that at each step, the size of the graph of subproblem generated decreases at least by one. As a result, the depth of the search tree is at most \( n \). Moreover, the algorithm only takes polynomial time at each step to solve a subproblem (without considering the recursive calls to the algorithm). Thus, the running time of the polynomial space algorithm is \( O^*(2^{0.406n}) \). This is not good when compared with algorithms of Tarjan and Trojanowski [8] and Jian [5]. However, in the above running time analysis, we do not take into account the fact that decreasing the degree of a vertex (though not removing from graph) helps speed up the
algorithm.

Fomin, Grandoni and Kratsch designed a new measure to measure the size of the graph $G$ given. They assign different weights to vertices of different degrees. By this technique of “Measure and Conquer”, they proved that their algorithm has running time $O^*(2^{0.288n})$ [3].

□ End of chapter.
Chapter 3

Improvements

In chapter 2, we showed that Fomin, Grandoni and Kratsch's algorithm has running time $O^*(2^{0.406n})$ if we analyze the algorithm with standard techniques. Fomin, Grandoni and Kratsch found a much tighter bound using "Measure and Conquer". In this chapter, we will illustrate this technique with four examples. We hope to obtain a considerably better running time in the four examples.

We will first start at the algorithm of Tarjan and Trojanowski [8], which was published in 1977. Their algorithm consists of a total of five pages. In order to obtain a simpler case analysis, we will modify their algorithm and apply "Measure and Conquer" hoping we can achieve some improvements on the time complexity.

3.1 Tarjan and Trojanowski’s Algorithm

We modify the original algorithm of Tarjan and Trojanowski [8] by considering the rule of branching (Lemma 2.1) at a vertex of
degree 4 or more. Since the original algorithm is complicated, we work on a simpler modified algorithm to illustrate the idea of "Measure and Conquer". The original algorithm of Tarjan and Trojanowski [8] is given in Figures 3.1, 3.2, 3.3, 3.4, 3.5 and 3.6. The modified algorithm is in Figures 3.7 and 3.8.

For the correctness of the modified algorithm, we need the following lemma.

**Lemma 3.1.** Let $G = (V, E)$ be a connected graph, where $|V| = n$. Suppose for all $v \in V$, $d(v) = 2$. Then $\alpha(G) = \lceil \frac{n}{2} \rceil$, where $[m]$ is the largest integer less than or equal to $m$.

**Proof.** Note that the graph consists of a cycle since $G$ is connected and $d(v) = 2$ for all $v \in V$. Suppose $v_i \in V$ such that $v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_n \rightarrow v_1$ forms a cycle. Now, let $S$ be a maximum independent set and without loss of generality that $v_1 \in S$. Then $v_{2k-1} \in S$ for all $k = 1, \ldots, \lceil \frac{n}{2} \rceil$. It follows that $\alpha(G) = \lceil \frac{n}{2} \rceil$. \qed
int mis(G) {

(0.1) if (|V| = 0), return 0;

(0.2) if (G consists of distinct connected components G_1, G_2, ..., G_k), return \sum_{i=1}^{k} \text{mis}(G_i);

(1) if (there exists v such that d(v) = 1), (Suppose (v, w) \in E) return \text{mis}(G - \{w\});

(2) if (there exists v such that d(v) = 2),

(2.1) if (d(v) = 2 for all v \in V)

return \left\lceil \frac{n}{2} \right\rceil;

(2.2) if (there exist v, w_1 \in V such that d(v) = 2, d(w_1) \geq 3, (v, w_1) \in E)

(Suppose N(v) = \{w_1, w_2\})

(2.2.1) if ((w_1, w_2) \in E)

return 1 + \text{mis}(G - N^*(v));

(2.2.2) if ((w_1, w_2) \notin E)

return \max\{1 + \text{mis}(G - N^*(v)), 2 + \text{mis}(G - N^*(w_1) - N^*(w_2))\};

(3) if (there exists v such that d(v) = 3),

(Suppose N(v) = \{w_1, w_2, w_3\})

(3.1) if ((w_1, w_2), (w_2, w_3), (w_1, w_3) \in E)

return 1 + \text{mis}(G - N^*(v));

(3.2) if ((w_1, w_2), (w_1, w_3) \in E) (or symmetric cases)

return \max\{1 + \text{mis}(G - N^*(v)), 2 + \text{mis}(G - N^*(w_2) - N^*(w_3))\};

(3.3) if ((w_1, w_2) \in E) (or symmetric cases)

(Suppose N_i = V - \{w_1, w_2, w_3\} - N(w_i), for i = 1, 2, 3)

Note |N_1|, |N_2| \leq |V| - 5, |N_3| \leq |V| - 6

(3.3.1) if (|N_1 \cap N_3| \leq |N_2 \cap N_3| = |V| - 6) (or symmetric cases)

return \max\{1 + \text{mis}(G - N^*(v)), 2 + \text{mis}(N_3)\};

(3.3.2) if (|N_1 \cap N_3|, |N_2 \cap N_3| \leq |V| - 7)

return \max\{1 + \text{mis}(G - N^*(v)), 2 + \text{mis}(N_1 \cap N_3), 2 + \text{mis}(N_2 \cap N_3)\};

...
(3.4) if \((w_i, w_j) \notin E\) for all \(i \neq j\)

(Suppose \(\overline{N}_i = V - \{w_1, w_2, w_3\} - N(w_i)\), for \(i = 1, 2, 3\)

Note \(|\overline{N}_i| \leq |V| - 6\) for \(i = 1, 2, 3\)

(3.4.1) if \(|\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| \geq |V| - 7\)

return \(\max\{1+\text{mis}(G - N^*(v)), 3+\text{mis}(\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)\}\);

(3.4.2) if \(|\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| = |V| - 8\) or \(|V| - 9\)

(3.4.2.1) if \(|\overline{N}_i \cap \overline{N}_j| \leq |\overline{N}_i \cap \overline{N}_2 \cap \overline{N}_3| + 1\) for all \(i \neq j\)

return \(\max\{1+\text{mis}(G - N^*(v)), 3+\text{mis}(\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)\}\);

(3.4.2.2) if \(|\overline{N}_1 \cap \overline{N}_2| \geq |\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| + 2\)

(or symmetric cases)

return \(\max\{1+\text{mis}(G - N^*(v)), 2+\text{mis}(\overline{N}_1 \cap \overline{N}_2), 3+\text{mis}(\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)\}\);

(3.4.3) if \(|\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| \leq |V| - 10\)

(3.4.3.1) if \(|\overline{N}_i \cap \overline{N}_j| \leq |\overline{N}_i \cap \overline{N}_2 \cap \overline{N}_3| + 1\) for all \(i \neq j\)

return \(\max\{1+\text{mis}(G - N^*(v)), 3+\text{mis}(\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)\}\);

(3.4.3.2) if \(|\overline{N}_1 \cap \overline{N}_2| \geq |\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| + 2\)

(or symmetric cases)

return \(\max\{1+\text{mis}(G - N^*(v)), 2+\text{mis}(\overline{N}_1 \cap \overline{N}_2), 3+\text{mis}(\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)\}\);

(3.4.3.3) if \(|\overline{N}_1 \cap \overline{N}_2|, |\overline{N}_1 \cap \overline{N}_3| \geq |\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| + 2\)

(or symmetric cases)

return \(\max\{1+\text{mis}(G - N^*(v)), 2+\text{mis}(\overline{N}_1 \cap \overline{N}_2), 2+\text{mis}(\overline{N}_1 \cap \overline{N}_3), 3+\text{mis}(\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)\}\);

(3.4.3.4) if \(|\overline{N}_1 \cap \overline{N}_2|, |\overline{N}_1 \cap \overline{N}_3|, |\overline{N}_2 \cap \overline{N}_3| \geq |\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| + 2\)

(For \(i = 1, 2, 3\), let \(u_{i1}, u_{i2} \in (\overline{N}_j \cap \overline{N}_k) - \overline{N}_i\),

where \(j, k \neq i\).)

(3.4.3.4.1) if \(|\overline{N}_j \cap \overline{N}_k| = |\overline{N}_j \cap \overline{N}_2 \cap \overline{N}_3| + 2\) and

\((u_{i1}, u_{i2}) \in E\) for some distinct \(i, j, k\)

return \(\max\{1+\text{mis}(G - N^*(v)), 2+\text{mis}(\overline{N}_1 \cap \overline{N}_2), 2+\text{mis}(\overline{N}_1 \cap \overline{N}_3), 3+\text{mis}(\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)\}\);

...
(3.4.3.4.2) if $|N_i \cap N_k| = |N_1 \cap N_2 \cap N_3| + 2$ and $(u_{41}, u_{42}) \not\in E$ for all distinct $i, j, k$
return max\(\{1 + \text{mis}(G - N^*(v))\}, 4 + \text{mis}(N_i \cap N_2 \cap N_3 - N^*(u_{41}) - N^*(u_{42}))\).

(3.4.3.4.3) if $|N_1 \cap N_2|, |N_1 \cap N_3| = |N_1 \cap N_2 \cap N_3| + 2$
(or symmetric cases)
return max\(\{1 + \text{mis}(G - N^*(v))\}, 4 + \text{mis}(N_i \cap N_2 \cap N_3 - N^*(u_{31}) - N^*(u_{32}))\),
\(2 + \text{mis}(N_2 \cap N_3), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\);.

(3.4.3.4.4) if $|N_1 \cap N_2| = |N_1 \cap N_2 \cap N_3| + 2$
(or symmetric cases)
return max\(\{1 + \text{mis}(G - N^*(v))\}, 4 + \text{mis}(N_i \cap N_2 \cap N_3 - N^*(u_{31}) - N^*(u_{32}))\),
\(2 + \text{mis}(N_2 \cap N_3), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\);.

(3.4.3.4.5) if $|N_i \cap N_j| \geq |N_1 \cap N_2 \cap N_3| + 3$ for $i \neq j$
return max\(\{1 + \text{mis}(G - N^*(v))\}, 2 + \text{mis}(N_1 \cap N_2), 2 + \text{mis}(N_1 \cap N_3), 2 + \text{mis}(N_2 \cap N_3), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\);.

Figure 3.3: The Original Algorithm (Part III) [8]
... 
(4) if (there exists v such that \(d(v) = 4\))
(4.1) if \((d(w) = 4\) for all \(w \in V\))
(4.1.1) if (there exist \(v, w\) such that \((v, w) \notin E\) and 
\(|N(v) \cap N(w)| \geq 2\))
(4.1.1.1) if \(|N(v) \cap N(w)| \geq 3\)
\[
\text{return } \max\{2+\text{mis}(G - N^*(v) - N^*(w)), \text{mis}(G - \{v, w\})\};
\]
(4.1.1.2) if \(|N(v) \cap N(w)| = 2\)
\[
\text{(Let } x, y \in (N(v) - N(w)), \text{)} \\
\text{(Let } q, r \in (N(w) - N(v)). \text{)}
\]
\[
\text{Let } \overline{N}(z) = V - \{z\} - N(z) \text{ for } z \in V. \\
\]
(4.1.1.2.1) if \(((x, y), (q, r) \in E) \text{ (or symmetric cases)}
\[
\text{return } \max\{2+\text{mis}(\overline{N}(v) \cap \overline{N}(w)), \text{mis}(G - \{v, w\})\};
\]
(4.1.1.2.2) if \(((x, y) \in E, (q, r) \notin E) \text{ (or symmetric cases)}
\[
\text{return } \max\{2+\text{mis}(\overline{N}(v) \cap \overline{N}(w)), 3+\text{mis}(\overline{N}(v) \cap \overline{N}(w) \cap \overline{N}(q) \cap \overline{N}(r)), \text{mis}(G - \{v, w\})\};
\]
(4.1.1.2.3) if \(((x, y), (q, r) \notin E, \text{ }) \text{ (or symmetric cases)}
\[
\text{|N(v) \cap N(w) \cap N(q) \cap N(r)| \geq |V| - 9)} \\
\]
\[
\text{return } \max\{3+\text{mis}(\overline{N}(v) \cap \overline{N}(w) \cap \overline{N}(x) \cap \overline{N}(y)), 3+\text{mis}(\overline{N}(v) \cap \overline{N}(w) \cap \overline{N}(q) \cap \overline{N}(r)), \text{mis}(G - \{v, w\})\};
\]
(4.1.1.2.4) if \(((x, y), (q, r) \notin E) \text{ (or symmetric cases)}
\[
\text{|N(v) \cap N(w) \cap N(q) \cap N(r)|,} \\ 
\text{|N(v) \cap N(w) \cap N(x) \cap N(y)| \leq |V| - 10)}
\]
\[
\text{return } \max\{2+\text{mis}(\overline{N}(v) \cap \overline{N}(w)), 3+\text{mis}(\overline{N}(v) \cap \overline{N}(w) \cap \overline{N}(x) \cap \overline{N}(y)), 3+\text{mis}(\overline{N}(v) \cap \overline{N}(w) \cap \overline{N}(q) \cap \overline{N}(r)), \text{mis}(G - \{v, w\})\};
\]
...

Figure 3.4: The Original Algorithm (Part IV) [8]
... (4.1.2) if (there exist $v, w$ such that $(v, w) \in E$ or $|N(v) \cap N(w)| \leq 1$)

(Suppose $N(v) = \{w_1, w_2, w_3, w_4\}$.
Let $\bar{N}_i = G - N(v) - N(w_i)$.)

(4.1.2.1) if $((w_1, w_i) \in E$ for $i = 2, 3, 4$) (or symmetric cases)
return 1;

(4.1.2.2) if $((w_1, w_2), (w_1, w_3), (w_2, w_3) \in E,$
$\quad (w_1, w_4), (w_2, w_4), (w_3, w_4) \notin E$)
(or symmetric cases)
return max$\{1 + \text{mis}(G - N^*(v)),$
$\quad 2 + \text{mis}(\bar{N}_1 \cap \bar{N}_4),$
$\quad 2 + \text{mis}(\bar{N}_2 \cap \bar{N}_4),$
$\quad 2 + \text{mis}(\bar{N}_3 \cap \bar{N}_4)\};$

(4.1.2.3) if $((w_1, w_2), (w_3, w_4) \in E,$
$\quad (w_1, w_3), (w_1, w_4), (w_2, w_3), (w_2, w_4) \notin E$)
(or symmetric cases)
return max$\{1 + \text{mis}(G - N^*(v)),$
$\quad 2 + \text{mis}(\bar{N}_1 \cap \bar{N}_3),$
$\quad 2 + \text{mis}(\bar{N}_2 \cap \bar{N}_3),$
$\quad 2 + \text{mis}(\bar{N}_1 \cap \bar{N}_4),$
$\quad 2 + \text{mis}(\bar{N}_3 \cap \bar{N}_4)\};$

(4.1.2.4) if $((w_1, w_2) \notin E$ for $i \neq j$)
return max$\{1 + \text{mis}(G - N^*(v)),$
$\quad 2 + \text{mis}(\bar{N}_1 \cap \bar{N}_2),$
$\quad 2 + \text{mis}(\bar{N}_1 \cap \bar{N}_3),$
$\quad 2 + \text{mis}(\bar{N}_1 \cap \bar{N}_4),$
$\quad 2 + \text{mis}(\bar{N}_2 \cap \bar{N}_3),$
$\quad 2 + \text{mis}(\bar{N}_2 \cap \bar{N}_4),$
$\quad 2 + \text{mis}(\bar{N}_3 \cap \bar{N}_4),$
$\quad 3 + \text{mis}(\bar{N}_1 \cap \bar{N}_3 \cap \bar{N}_4),$
$\quad 3 + \text{mis}(\bar{N}_1 \cap \bar{N}_2 \cap \bar{N}_4),$
$\quad 3 + \text{mis}(\bar{N}_2 \cap \bar{N}_3 \cap \bar{N}_4),$
$\quad 4 + \text{mis}(\bar{N}_1 \cap \bar{N}_2 \cap \bar{N}_3 \cap \bar{N}_4)\};$

Figure 3.5: The Original Algorithm (Part V) [8]
(4.2) if $(d(w) \geq 5$ for some $w \in V$) 
(Let $v, w$ be such that $d(v) = 4, d(w) \geq 5, (v, w) \in E.$) 
return $\max\{1+\text{mis}(G - N^*(w)), \text{mis}(G - \{w\})\};$

(5) if $(d(w) = 5$ for all $w \in V)$
(5.1) if $(|V| = 6)$ 
return 1;
(5.2) if $(|V| > 6)$ 
return $\max\{1+\text{mis}(G - N^*(v)), \text{mis}(G - \{v\})\};$

(6) if (there exists $w \in V$ with $d(w) \geq 6$) 
return $\max\{1+\text{mis}(G - N^*(w)), \text{mis}(G - \{w\})\}.$

Figure 3.6: The Original Algorithm (Part VI) [8]
int mis(G) {
    (0.1) if (|V| = 0), return 0;
    (0.2) if (there exists vertex v such that \(d(v) = 0\)),
            return 1 + mis(G - \{v\});
    (1) if (there exists v such that \(d(v) = 1\)), (Suppose \((v,w) \in E\))
            return mis(G - \{w\});
    (2) if (there exists v such that \(d(v) = 2\)),
        (2.1) if \((d(v) = 2 \text{ for all } v \in V)\)
            (Suppose G has m connected components \(C_i\) with \(|C_i| = n_i\))
            return \(\sum_{i=1}^{m} \frac{n_i}{2}\);
        (2.2) if (there exists \(v, w_1 \in V\) such that \(d(v) = 2, d(w_1) \geq 3\), \((v,w_1) \in E)\)
            (Suppose \(N(v) = \{w_1,w_2\}\))
            (2.2.1) if \(\{(w_1,w_2) \in E\}\)
                return 1 + mis(G - \(N^*(v)\));
            (2.2.2) if \(\{(w_1,w_2) \notin E, (w_1,w_2) \in E)\)
                return max{1 + mis(G - \(N^*(v)\)),
                            2 + mis(G - \(N^*(w_1) - N^*(w_2)\))};
    (3) if (there exists v such that \(d(v) = 3\)),
        (Suppose \(N(v) = \{w_1,w_2,w_3\}\))
        (3.1) if \(\{(w_1,w_2), (w_2,w_3),(w_1,w_3) \in E)\)
            return 1 + mis(G - \(N^*(v)\));
        (3.2) if \(\{(w_1,w_2), (w_1,w_3) \in E\} \text{ (or symmetric cases)}\)
            return max{1 + mis(G - \(N^*(v)\)), 2 + mis(G - \(N^*(w_2) - N^*(w_3)\))};
        (3.3) if \(\{(w_1,w_2) \in E\} \text{ (or symmetric cases)}\)
            (Suppose \(N_i = V - \{w_1,w_2,w_3\} - N(w_i), \text{ for } i = 1,2,3\)
            Note \(|\overline{N}_1|, |\overline{N}_2| \leq |V| - 5, |\overline{N}_3| \leq |V| - 6)\)
            (3.3.1) if \(|\overline{N}_1 \cap \overline{N}_3| \leq |\overline{N}_2 \cap \overline{N}_3| = |V| - 6\) \text{ (or symmetric cases)}
                return max\{1 + mis(G - \(N^*(v)\)), 2 + mis(\(\overline{N}_2\))\};
        (3.3.2) if \(|\overline{N}_1 \cap \overline{N}_3|, |\overline{N}_2 \cap \overline{N}_3| \leq |V| - 7\)
            return max\{1 + mis(G - \(N^*(v)\)), 2 + mis(\(\overline{N}_1 \cap \overline{N}_3\)),
                        2 + mis(\(\overline{N}_2 \cap \overline{N}_3\))\};
...
(3.4) if \((w_i, w_j) \notin E\) for all \(i \neq j\)
(Suppose \(N_i = V - \{w_1, w_2, w_3\} - N(w_i)\), for \(i = 1, 2, 3\)
Note \(|N_i| \leq |V| - 6\) for \(i = 1, 2, 3\)
(3.4.1) if \((|N_1 \cap N_2 \cap N_3| \geq |V| - 7)\)
return \(\max\{1 + \text{mis}(G - N^*(v)), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\}\);
(3.4.2) if \((|N_1 \cap N_2 \cap N_3| = |V| - 8\) or \(|V| - 9)\)
(3.4.2.1) if \((|N_i \cap N_j| \leq |N_i \cap N_j| + 1\) for all \(i \neq j\)
return \(\max\{1 + \text{mis}(G - N^*(v)), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\}\);
(3.4.2.2) if \((|N_1 \cap N_2| \geq |N_i \cap N_j| + 2)\)
(or symmetric cases)
return \(\max\{1 + \text{mis}(G - N^*(v)), 2 + \text{mis}(N_1 \cap N_2), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\}\);
(3.4.3) if \((|N_1 \cap N_2 \cap N_3| \leq |V| - 10)\)
(3.4.3.1) if \((|N_i \cap N_j| \leq |N_i \cap N_j| + 1\) for all \(i \neq j\)
return \(\max\{1 + \text{mis}(G - N^*(v)), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\}\);
(3.4.3.2) if \((|N_1 \cap N_2| \geq |N_i \cap N_j| + 2)\)
(or symmetric cases)
return \(\max\{1 + \text{mis}(G - N^*(v)), 2 + \text{mis}(N_1 \cap N_2), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\}\);
(3.4.3.3) if \((|N_1 \cap N_2|, |N_1 \cap N_3| \geq |N_1 \cap N_2 \cap N_3| + 2)\)
(or symmetric cases)
return \(\max\{1 + \text{mis}(G - N^*(v)), 2 + \text{mis}(N_1 \cap N_2), 2 + \text{mis}(N_1 \cap N_3), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\}\);
(3.4.3.4) if \((|N_1 \cap N_2|, |N_1 \cap N_3|, |N_2 \cap N_3| \geq |N_1 \cap N_2 \cap N_3| + 2)\)
return \(\max\{1 + \text{mis}(G - N^*(v)), 2 + \text{mis}(N_1 \cap N_2), 2 + \text{mis}(N_1 \cap N_3), 2 + \text{mis}(N_2 \cap N_3), 3 + \text{mis}(N_1 \cap N_2 \cap N_3)\}\);

(4) select a vertex \(v\) of maximum degree;
return \(\max\{\text{mis}(G - \{v\}), 1 + \text{mis}(G - N^*(v))\}\);

Figure 3.8: The Modified Algorithm (Part II)
3.1.1 Correctness and Running Time Analysis

We will prove the correctness and find an upper bound for the algorithm in Figures 3.7 and 3.8. Here, $L(n)$ denotes the maximum number of leaves of a search tree to solve a problem of size $n$, where $|V| = n$. The analysis of each rule is as follows.

(0.2) This rule is trivially true. The recurrence is $L(n) \leq L(n - 1)$.

(1) The set $\{v\}$ dominates $\{w\}$ in $N^*(v)$ (Lemma 2.2). The recurrence is $L(n) \leq L(n - 1)$.

(2.1) The graph consists of cycles only (Lemma 3.1). The recurrence is $L(n) \leq 1$.

(2.2.1) The set $\{v\}$ dominates $\{w_1\}$ and $\{w_2\}$ in $N^*(v)$ (Lemma 2.2). The recurrence is $L(n) \leq L(n - 3)$.

(2.2.2) The set $\{w_1, w_2\}$ dominates $\{w_1\}$ and $\{w_2\}$ in $N^*(v)$ (Lemma 2.2). The recurrence is $L(n) \leq L(n - 3) + L(n - 5)$.

(3.1) The set $\{v\}$ dominates $\{w_i\}$ in $N^*(v)$, for all $i = 1, 2, 3$ (Lemma 2.2). The recurrence is $L(n) \leq L(n - 4)$.

(3.2) The set $\{v\}$ dominates $\{w_i\}$ in $N^*(v)$, for all $i = 1, 2, 3$ (Lemma 2.2). The recurrence is $L(n) \leq L(n - 4) + L(n - 5)$.

(3.3.1) Note that $\overline{N_2} \cap \overline{N_3} = \overline{N_3}$ and $|\overline{N_3}| = |V| - 6$. If $J \subseteq V - N^*(v)$ such that $J \cup \{w_1, w_3\}$ is independent, we have $J \subseteq \overline{N_1} \cap \overline{N_3}$. Thus, $J \subseteq \overline{N_3} = \overline{N_2} \cap \overline{N_3}$. Therefore, $\{w_2, w_3\}$
dominates \{w_1, w_3\} in N^*(v) (Lemma 2.2). The recurrence is \(L(n) \leq L(n - 4) + L(n - 6)\).

(3.3.2) The recurrence is \(L(n) \leq L(n - 4) + 2L(n - 7)\).

(3.4.1) For \(i = 1, 2, 3\), since \(|\overline{N}_i \cap \overline{N}_2 \cap \overline{N}_3| \geq |V| - 7\) and \(|\overline{N}_i| \leq |V| - 6\), we have \(|\overline{N}_i - (\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)| \leq 1\). Thus, \{w_1, w_2, w_3\} dominates \{w_i, w_j\} in \{v, w_1, w_2, w_3\} for all \(i, j\). The recurrence is \(L(n) \leq L(n - 4) + L(n - 6)\).

(3.4.2) If \(|\overline{N}_i \cap \overline{N}_j| \leq |\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| + 1\) for some \(i, j\), then for all \(J \subseteq V - N^*(v)\) such that \(J \cup \{w_i, w_j\}\) is independent, only the following two cases occur. If \(|\overline{N}_i \cap \overline{N}_j - (\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)| = 1\), suppose \(z \in \overline{N}_i \cap \overline{N}_j - (\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3), J \cup \{w_1, w_2, w_3\} - \{z\}\) is also independent. If \(|\overline{N}_i \cap \overline{N}_j - (\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)| = 0\), \(J \cup \{w_1, w_2, w_3\}\) is independent. Thus, \{w_i, w_j\} is dominated by \{w_1, w_2, w_3\} in \(N^*(v)\).

For distinct \(i, j, k\), we have \(|\overline{N}_i| \geq |\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| + |\overline{N}_i \cap \overline{N}_j - (\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)| + |\overline{N}_i \cap \overline{N}_k - (\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)|\). For \(i = 1, 2, 3\), since \(|\overline{N}_i| \leq |V| - 6\) and \(|\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| \geq |V| - 9\), we have \(|\overline{N}_i \cap \overline{N}_j - (\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)| + |\overline{N}_i \cap \overline{N}_k - (\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3)| \leq 3\). Thus, we get \(|\overline{N}_i \cap \overline{N}_j| \leq |\overline{N}_1 \cap \overline{N}_2 \cap \overline{N}_3| + 2\) for exactly one pair of \(i \neq j\). Suppose 1, 2 is such a pair.

(3.4.2.1) By the explanation in (3.4.2) above, \{w_i, w_j\} is dominated by \{w_1, w_2, w_3\} in \(N^*(v)\) for all \(i \neq j\). \{v\} also dominates \{w_i\} for all \(i = 1, 2, 3\). The recurrence is \(L(n) \leq L(n - 4) + L(n - 8)\).
(3.4.2.2) By the explanation in (3.4.2) above, \{w_2, w_3\} and \{w_1, w_3\} are dominated by \{w_1, w_2, w_3\} in \textit{N}^*(v). \{v\} also dominates \{w_i\} for all i = 1, 2, 3. The recurrence is \(L(n) \leq L(n-4) + L(n-6) + L(n-8)\).

(3.4.3.1) same as (3.4.2.1). The recurrence is \(L(n) \leq L(n-4) + L(n-10)\).

(3.4.3.2) same as (3.4.2.2). The recurrence is \(L(n) \leq L(n-4) + L(n-6) + L(n-10)\).

(3.4.3.3) By the explanation in (3.4.2) above, \{w_2, w_3\} is dominated by \{w_1, w_2, w_3\}. The recurrence is \(L(n) \leq L(n-4) + 2L(n-6) + L(n-10)\).

(3.4.3.4) The recurrence is \(L(n) \leq L(n-4) + 3L(n-6) + L(n-10)\).

(4) The recurrence is \(L(n) \leq L(n-1) + L(n-5)\).

By Theorem 2.1, we find that \(L(n) \leq c \cdot 2^{0.406n}\) for some constant \(c > 0\). By the discussion of last chapter, we prove that the running time of the algorithm in Figures 3.7 and 3.8 is \(O^*(2^{0.406n})\) and the worst case recurrence is the recurrence for rule (4).

3.1.2 Improvement

Following the approach of Fomin, Grandoni and Kratsch \[3\], we assign weights to vertices of different degrees so as to take into account the effect of decreasing a vertex’s degree since the
The algorithm does not work well at vertices of degree 4 or higher (The worst recurrence is the one for rule (4)).

Let $G = (V, E)$ and $n_i$ be the number of vertices of degree $i$ in $G$. We will use the following measure $k(G)$ as the “size” of the graph:

$$k(G) = \sum_{i \geq 0} w_i n_i,$$

where $w_i \in [0, 1]$ is to be fixed. Note that $k(G) \leq n$.

To simplify the running time analysis, we will make the following assumptions:

1. We assume $w_0 = w_1 = 0$. The reason is that vertices with degree at most 1 can be removed from graph without branching by rule (0.2) or (1). These vertices only contribute to the running time by polynomial factors.

2. We assume $w_i = 1$, for all $i \geq 4$. Then, we only have to decide 2 weights $w_2$ and $w_3$.

3. Let $\Delta w_i = w_i - w_{i-1}$ be the decrease in weight of a vertex of degree $i$. We assume $\Delta w_2 \geq \Delta w_3 \geq \Delta w_4 \geq 0$. Since a vertex of degree 1 can be removed without branching, intuitively, the effect of decreasing the degree of a degree 2 vertex is more significant than that of a degree 4 vertex.

Let $L(k)$ be the maximum number of leaves of the search tree to solve a graph $G$ of size $k$. For graphs of size $k = 0$, the algorithm solves it in polynomial time without branching. We have $L(0) = 1$. Therefore, by the discussion in the previous
section, if $L(k) \leq 2^{\beta k}$ for some $\beta > 0$, then the running time of the algorithm is $O^*(2^{\beta n})$.

The recurrences for rules (2) and (3) are listed as follows.

(2.1) $L(k) \leq 1$.

(2.2.1) $L(k) \leq L(k - 2w_2 - w_3)$.

(2.2.2) $L(k) \leq L(k - 2w_2 - w_3) + L(k - 4w_2 - w_3)$.

(3.1) $L(k) \leq L(k - 4w_3)$.

(3.2) $L(k) \leq L(k - 4w_3) + L(k - 5w_3)$.

(3.3.1) $L(k) \leq L(k - 4w_3) + L(k - 6w_3)$.

(3.3.2) $L(k) \leq L(k - 4w_3) + 2L(k - 7w_3)$.

(3.4.1) $L(k) \leq L(k - 4w_3) + L(k - 6w_3)$.

(3.4.2.1) $L(k) \leq L(k - 4w_3) + L(k - 8w_3)$.

(3.4.2.2) $L(k) \leq L(k - 4w_3) + L(k - 6w_3) + L(k - 8w_3)$.

(3.4.3.1) $L(k) \leq L(k - 4w_3) + L(k - 10w_3)$.

(3.4.3.2) $L(k) \leq L(k - 4w_3) + L(k - 6w_3) + L(k - 10w_3)$.

(3.4.3.3) $L(k) \leq L(k - 4w_3) + 2L(k - 6w_3) + L(k - 10w_3)$.

(3.4.3.4) $L(k) \leq L(k - 4w_3) + 3L(k - 6w_3) + L(k - 10w_3)$.

For rule (4), suppose we branch at a vertex $v$ with $d(v) = d$. Note that $d \geq 4$ since rule (2) and (3) cannot be applied and $v$ is the vertex with maximum degree. Let $N(v) = \{u_1, u_2, ..., u_d\}$
and \( d_i = d(u_i) \). Let \( m_i (m_i(v)) \) be the number of vertices of degree \( i \) in \( N(v) \), i.e. \( m_i = |\{u \in N(v) | d(u) = i\}| \).

The analysis is divided into the following two cases.

1. We consider the subproblem for \( G - \{v\} \).
   The size of the problem decreases by \( w_d \) because of the removal of \( \{v\} \). The size also decreases by \( \sum_{i=4}^{d} m_i \Delta w_i \) due to the reduction in \( N(v) \). Thus, in this case, the total reduction in size is
   \[
   \Delta_1 = w_d + \sum_{i=4}^{d} m_i \Delta w_i.
   \]

2. We consider the subproblem for \( G - N^*(v) \).
   The size of the problem decreases by \( w_d \) because of the removal of \( \{v\} \). The size further decreases by \( \sum_{i=4}^{d} m_i w_i \) due to the removal of \( N(v) \). There may also be reduction in \( N^2(v) \), but we ignore this reduction here for simplicity. Thus, in this case, the total reduction in size is
   \[
   \Delta_2 \geq w_d + \sum_{i=4}^{d} m_i w_i.
   \]

By the analysis above, we get the following recurrences
\[
L(k) \leq L(k - \Delta_1) + L(k - \Delta_2) \tag{3.1}
\]
subject to \( \sum_{i=4}^{d} m_i = d \), which are dominated by the following
weaker recurrences

\[ L(k) \leq L(k - w_d - \sum_{i=4}^{d} m_i \Delta w_i) + L(k - w_d - \sum_{i=4}^{d} m_i w_i) \]  
(3.2)

subject to \( \sum_{i=4}^{d} m_i = d \).

For \( i \geq 5 \), we have \( \Delta w_i = 0 \) and \( w_i = 1 \). For \( d \geq 5 \), we have \( \Delta_1 = 1 + m_4 \Delta w_4 \) and \( \Delta_2 \geq 1 + m_4 w_4 + m_{\geq 5} \), where \( m_{\geq i} = |\{ u \in N(v) \mid d(u) \geq i \}| \).

By the above observation, we only need to calculate a finite number of recurrences since recurrences (3.2) for \( d \geq 5 \) are dominated by

\[ L(k) \leq L(k - 1 - m_4 \Delta w_4) + L(k - 1 - m_4 w_4 - m_{\geq 5}) \]  
(3.3)

subject to \( m_4 + m_{\geq 5} = 5 \).

The proof of the above will be given in Lemma 3.2 in a later section.

Now, we fix \( w_2, w_3 \in [0, 1] \), let \( S = \{ a_1 + a_2 w_2 + a_3 w_3 \mid a_1, a_2, a_3 \in \mathbb{N} \cup \{0\} \} \). Note that \( S \) is the set which contains all possible sizes for a graph and is well-ordered.

When we run through \( d \) from 4 to 5 and all possible \( m_i \) satisfying \( m_4 + m_{\geq 5} = d \), and also considering the recurrences for rules (2) and (3), we have

\[ L(k) \leq \max_{1 \leq i \leq m} \{ L(k - b_i) + L(k - c_i) \} \]  
(3.4)

where \( k \in S \), \( m \in \mathbb{N} \), \( b_i \) and \( c_i \) depends on \( w_2 \) and \( w_3 \). Let \( \beta_i \) be the largest roots of \( x^{-b_i} + x^{-c_i} - 1 = 0 \) and \( \beta = \max_{1 \leq i \leq m} \beta_i \).
Following the proof of Theorem 2.1, we have \( L(k) \leq c \cdot \beta^k \) for some constant \( c > 0 \). Thus the problem reduces to finding weights \( w_2 \) and \( w_3 \) which minimize \( \beta \).

To find good weights for the problem, we follow the approach of randomized local search technique in [4]. First, we fix initial weights \( w_2 \) and \( w_3 \) which satisfy the initial assumptions. These yield a solution \( \beta \). Then we randomly perturb the weights \( w_2 \) and \( w_3 \) by generating normal random variables. If the new weights \( w'_2 \) and \( w'_3 \) are feasible and yield a better solution, i.e. \( \beta' < \beta \), then we update the weights. If there are no improvement after a large number of steps, then we will decrease the variance of the random variables drawn and repeat the above process.

By the help of the computer, we numerically find that when \( (w_2, w_3) = (0.7100996, 0.98383497) \), we have \( \beta = 1.31590763... < 2^{0.39606n} \). Thus the running time of algorithm in Figures 3.7 and 3.8 is \( O^*(2^{0.39606n}) \). The worst case recurrence is the recurrence for rule (3.4.3.4). The improvement is not significant when comparing with the bound \( O^*(2^{0.406n}) \) obtained from standard analysis.

### 3.1.3 Using more weights

In the previous section, we use only two weights \( (w_2, w_3) \) to compute the running time. A nature question to ask is if we can find a better bound by using more weights. In this subsection, we will use four weights \( (w_2, w_3, w_4, w_5) \) to see if increasing the
number of weights results in a tighter upper bound.

In the following running time analysis, we will follow the notations and approach in section 3.1.2.

The initial assumptions will be changed to the following:

1. We assume \( w_0 = w_1 = 0 \).

2. We assume \( w_i = 1 \), for all \( i \geq 6 \). Then, we only have to compute finite and small number of weights.

3. Let \( \Delta w_i = w_i - w_{i-1} \) be the decrease in weight of a vertex of degree \( i \). We assume \( \Delta w_2 \geq \Delta w_3 \geq \ldots \geq \Delta w_6 \geq 0 \).

The recurrences for rule (2) to (4) are the same as in section 3.1.2. However, since \( w_i = 1 \) and \( \Delta w_i = 0 \) for \( i \geq 7 \), the recurrences (3.2) for \( d \geq 7 \) are dominated by the following recurrences

\[
L(k) \leq L(k-1-\sum_{i=4}^{6} m_i \Delta w_i) + L(k-1-\sum_{i=4}^{6} m_i w_i - m_{\geq 7}) \quad (3.5)
\]

subject to \( \sum_{i=4}^{6} m_i + m_{\geq 7} = 7 \).

We have a finite set of recurrences after fixing \( w_2, w_3, w_4, w_5 \in [0,1] \). By the help of the computer, we numerically find that when
\[
(w_2, w_3, w_4, w_5) = (0.53635019, 0.98383497, 0.99999999, 1),
\]
we get \( \beta = 1.31590764... < 2^{0.39606n} \). Thus the running time of algorithm in Figures 3.7 and 3.8 is \( O^*(2^{0.39606n}) \). The worst case recurrence is the recurrence for rule (4) of the case when \( d = 4 \),
3.2 The First Algorithm

In Figure 3.9, we give a simple algorithm which is discussed in [9]. This algorithm consists of only 5 reduction and branching rules. Thus, the running time analysis is much simpler than that of the algorithm in the previous section.

The correctness of the algorithm in Figure 3.9 follows from Lemma 2.1, 2.3 and 3.1.

3.2.1 Standard Analysis

Let $L(n)$ be the maximum number of leaves in the search tree to solve a problem of size $n$, where $|V| = n$. Note that $L(0) = 1$. If rule (1) or (2) is satisfied, we have $L(n) \leq L(n - 1)$. If rule (3) are satisfied, then $L(n) = 1$. If rule (4) is satisfied, note that by
int mis(G)
{
(0) if (|V| = 0), return 0;
(1) if (there exists a vertex v such that d(v) = 0), return 1 + mis(G — {v});
(2) if (there exists a vertex v such that d(v) = 1),
(Suppose w is the neighbour of v.) return mis(G — {v});
(3) if (for all v ∈ V, d(v) = 2),
(Suppose G has m connected components C_i with |C_i| = n_i.) return \( \sum_{i=1}^{m} \left\lfloor \frac{n_i}{3} \right\rfloor \);
(4) select a vertex v of maximum degree,
return max\{mis(G — {v}), 1+mis(G — N^*(v))\};
}

Figure 3.9: A simple algorithm that solves the maximum independent set problem, adopted from [9]

rule (3), we have to branch at a vertex v with d(v) ≥ 3. Thus, we have \( L(n) \leq L(n - 1) + L(n - 4) \).

Let \( \beta \) be the largest real root of \( x^{-4} + x^{-1} - 1 = 0 \). By numerical computation, we find that \( \beta = 1.3802... < 2^{0.465} \). Thus, from the discussion in the previous chapter, the running time of the algorithm in Figure 3.9 is \( O^*(2^{0.465n}) \).

3.2.2 Measure and Conquer

We will use the notations defined in section 3.1 throughout this thesis. The initial assumptions are the same as in section 3.1.2.

Note that for rules (1), (2) and (3), no branching is involved. Thus, these three rules only contribute to the running time by polynomial factors. Rule (4) is the one which contributes most to the running time.

Suppose at a step, rule (4) applies and we branch at a vertex
Note that \( d(v) \geq 3 \) since rules (2) and (3) cannot be applied and \( v \) is the vertex with maximum degree. The analysis of rule (4) is divided into two cases:

1. We consider the subproblem for \( G - \{v\} \).
   The size of the problem decreases by \( w_d \) because of the removal of \( \{v\} \). The size also decreases by \( \sum_{i=2}^{d} m_i \Delta w_i \) due to the reduction in \( N(v) \). Thus, in this case, the total reduction in size is
   \[
   \Delta_1 = w_d + \sum_{i=2}^{d} m_i \Delta w_i.
   \]

2. We consider the subproblem for \( G - N^*(v) \).
   The size of the problem decreases by \( w_d \) because of the removal of \( \{v\} \). The size further decreases by \( \sum_{i=2}^{d} m_i w_i \) due to the removal of \( N(v) \). There may also be reduction in \( N^2(v) \), but \( N^2(v) \) may be empty. Thus, in this case, the total reduction in size is
   \[
   \Delta_2 \geq w_d + \sum_{i=2}^{d} m_i w_i.
   \]

By the analysis above, we arrive at the following recurrences
\[
L(k) \leq L(k - \Delta_1) + L(k - \Delta_2)
\]
subject to \( \sum_{i=2}^{d} m_i = d \), which are dominated by the weaker
recurrences

\[ L(k) \leq L(k - w_d - \sum_{i=2}^{d} m_i \Delta w_i) + L(k - w_d - \sum_{i=2}^{d} m_i w_i) \quad (3.6) \]

subject to \( \sum_{i=2}^{d} m_i = d \).

Note that for \( i \geq 5 \), we have \( w_i = 1 \) and \( \Delta w_i = 0 \). By the above observation, it suffices to calculate a finite number of recurrences. A precise statement is given in the following lemma.

**Lemma 3.2** (Adopted from [3]). All recurrences (3.6) for \( d \geq 5 \) are dominated by the following recurrences

\[ L(k) \leq L(k - 1 - \sum_{i=2}^{4} m_i \Delta w_i) + L(k - 1 - \sum_{i=2}^{4} m_i w_i - m_{\geq 5}) \quad (3.7) \]

subject to \( \sum_{i=2}^{4} m_i + m_{\geq 5} = 5 \).

**Proof.** By the discussion above, since \( w_i = 1 \) and \( \Delta w_i = 0 \) for all \( i \geq 5 \), we have for \( d \geq 5 \), \( \Delta_1 = 1 + \sum_{i=2}^{4} m_i \Delta w_i \) and \( \Delta_2 \geq 1 + \sum_{i=2}^{4} m_i w_i + m_{\geq 5} \).

Now fix \( d > 5 \), let \( m_i \) be such that \( \sum_{i=2}^{4} m_i + m_{\geq 5} = d \), where \( m_{\geq 5} = \sum_{i=5}^{d} m_i \). We construct \( m_i' \) recursively as follows.

1. Set \( d' = 5 \). We first check on \( m_2 \).
   - If \( m_2 \geq d' \), then set \( m_2' = d' \), \( m_3' = m_4' = m_{\geq 5}' = 0 \) and end the process.
   - Otherwise, set \( m_2' = m_2 \) and update \( d' \) by \( d' = d' - m_2' \).

2. Then we check on \( m_3 \).
   - If \( m_3 \geq d' \), then \( m_3' = d' \), \( m_4' = m_{\geq 5}' = 0 \) and end the
process.

Otherwise, set \( m_3' = m_3 \), update \( d' \) by \( d' = d' - m_3' \) and go to step 1.

The process will end eventually since \( \sum_{i=2}^{4} m_i + m_{\geq 5} = d > 5 \).

Note that \( m_i' \leq m_i \) for all \( i \) and \( \sum_{i=2}^{4} m_i' + m_{\geq 5}' = 5 \) by our construction. We have \( \sum_{i=2}^{4} m_i' \Delta w_i \leq \sum_{i=2}^{4} m_i \Delta w_i \) and \( \sum_{i=2}^{4} m_i' w_i \leq \sum_{i=2}^{4} m_i w_i \). Thus, the recurrences

\[
L(k) \leq L(k - 1 - \sum_{i=2}^{4} m_i \Delta w_i) + L(k - 1 - \sum_{i=2}^{4} m_i w_i - m_{\geq 5})
\]

subject to \( \sum_{i=2}^{4} m_i + m_{\geq 5} = d \) are dominated by the weaker recurrences

\[
L(k) \leq L(k - 1 - \sum_{i=2}^{4} m_i' \Delta w_i) + L(k - 1 - \sum_{i=2}^{4} m_i' w_i - m_{\geq 5}')
\]

subject to \( \sum_{i=2}^{4} m_i' + m_{\geq 5}' = 5 \).

By the process in section 3.1.2, with initial weights \( w_2 = 0.5 \) and \( w_3 = 0.5 \), we numerically find that when

\((w_2, w_3) = (0.5966018, 0.9286447)\), we have \( \beta = 1.29043429... < 2^{0.36786} \). It follows that the running time of the algorithm in Figure 3.9 is \( O^*(2^{0.36786n}) \), which is significantly better than \( O^*(2^{0.465n}) \), derived from standard analysis. The worst case recurrence is when \( d = 3, \ m_2 = 0 \) and \( m_3 = 3 \). This algorithm also performs much better than the algorithm in Figures 3.7 and 3.8.
3.2.3 Using more weights

In this subsection, we hope we can obtain a better bound by using four weights \((w_2, w_3, w_4, w_5)\). The initial assumptions are the same as in section 3.1.3.

By the discussion in section 3.2.2, we have the same recurrences

\[
L(k) \leq L(k - w_d - \sum_{i=2}^{d} m_i \Delta w_i) + L(k - w_d - \sum_{i=2}^{d} m_i w_i) \tag{3.8}
\]

subject to \(\sum_{i=2}^{d} m_i = d\).

Since \(w_i = 1\) and \(\Delta w_i = 0\) for \(i \geq 7\), the recurrences (3.8) for \(d \geq 7\) are dominated by

\[
L(k) \leq L(k - 1 - \sum_{i=2}^{6} m_i \Delta w_i) + L(k - 1 - \sum_{i=2}^{6} m_i w_i - m_{\geq 7}) \tag{3.9}
\]

subject to \(\sum_{i=2}^{6} m_i + m_{\geq 7} = 7\).

Thus, we only have to calculate a finite number of recurrences. Our job remains to find weights \((w_2, w_3, w_4, w_5)\) which minimize \(\beta\). We numerically obtain

\((w_2, w_3, w_4, w_5) = (0.5966057, 0.9286457, 0.9999999, 1)\) which leads to \(\beta = 1.29043474... < 2^{0.36786}\). The worst case recurrence is when \(d = 4\), \(m_2 = m_3 = 0\) and \(m_4 = 4\). Thus the running time of the algorithm is \(O^*(2^{0.36786n})\).

In this algorithm, using more weights does not seem to help in finding a tighter bound. The result using four weights is a bit worse than using two weights in this algorithm. It may be
int mis(G)
{
(0) if (|V| = 0), return 0;
(1) if (there exists a vertex v such that d(v) = 0),
    return 1 + mis(G - {v});
(2) if (there exist two vertices v and w such that N*(w) \subseteq N*(v)),
    return mis(G - {v});
(3) if (for all v ∈ V, d(v) = 2),
    (Suppose G has m connected components C_i with |C_i| = n_i)
    return \sum_{i=1}^{m} \lceil \frac{n_i}{2} \rceil;
(4) select a vertex v of maximum degree,
    return max\{mis(G - {v}), 1+mis(G - N*(v))\};
}

Figure 3.10: The second algorithm, modified from [9]

because of the randomized local search technique we use to find the weights. It seems that no significant improvement can be made if we use more than two weights in this algorithm.

3.3 The Second Algorithm

This algorithm is modified from the algorithm in section 3.2. The concept of dominance [3] in Lemma 2.3 is applied in order to speed up the original algorithm. The modified algorithm is given in Figure 3.10. If we use standard analysis, we achieve the same running time $O^*(2^{0.465n})$ as in the algorithm in section 3.2. Thus, we try to apply “Measure and Conquer” to examine how the technique of dominance helps speed up the original algorithm.
3.3.1 Running Time Analysis

We will first use two weights and the initial assumptions are the same as in section 3.1.2.

Considering rule (4), we will divide the running time analysis into two cases:

1. We consider the subproblem for $G - \{v\}$.
   The total reduction in size is
   \[
   \Delta_1 = w_d + \sum_{i=2}^{d} m_i \Delta w_i.
   \]
   (same as in section 3.2.2)

2. We consider the subproblem for $G - N^*(v)$.
   The size of the problem decreases by $w_d$ because of the removal of $\{v\}$. The size further decreases by $\sum_{i=2}^{d} m_i w_i$ due to the removal of $N(v)$. There may also be reduction in $N^2(v)$. Note that for each vertex $w$ in $N(v)$, it must have at least one neighbour in $N^2(v)$, otherwise $w$ dominates $v$ and rule (2) applies. Therefore, there are at least $d$ edges between $N(v)$ and $N^2(v)$. The reduction for each edge between $N(v)$ and $N^2(v)$ is at least $\Delta w_d$ by initial assumption 3. Thus, the reduction in $N^2(v)$ is at least $d \Delta w_d$. In this case, the total reduction in size is
   \[
   \Delta_2 \geq w_d + \sum_{i=2}^{d} m_i w_i + d \Delta w_d.
   \]
By the above analysis, we arrive at the following recurrences

\[ L(k) \leq L(k - \Delta_1) + L(k - \Delta_2) \]

subject to \( \sum_{i=2}^{d} m_i = d \), which are dominated by a set of weaker recurrences

\[ L(k) \leq L(k - w_d - \sum_{i=2}^{d} m_i \Delta w_i) + L(k - w_d - \sum_{i=2}^{d} m_i w_i - d\Delta w_d) \]

subject to \( \sum_{i=2}^{d} m_i = d \).

Similar to the discussion in section 3.2.2, the recurrences (3.10) for \( d \geq 5 \) are dominated by the recurrences

\[ L(k) \leq L(k - 1 - \sum_{i=2}^{4} m_i \Delta w_i) + L(k - 1 - \sum_{i=2}^{4} m_i w_i - m_{\geq 5}) \] (3.11)

subject to \( \sum_{i=2}^{4} m_i + m_{\geq 5} = 5 \).

We only need to calculate a finite number of recurrences. We numerically find that when \((w_2, w_3) = (0.4276845, 0.9068486)\), we get \( \beta = 1.28519903... < 2^{0.36199} \). Thus the running time of the second algorithm in Figure 3.10 is \( O^*(2^{0.36199n}) \).

### 3.3.2 Using More Weights

In this section, we will use four weights \((w_2, w_3, w_4, w_5)\) to see if we can get a tighter upper bound. The initial assumptions are the same as in section 3.1.3.
The recurrences are the same as in the previous subsection:

\[ L(k) \leq L(k - w_d - \sum_{i=2}^{d} m_i \Delta w_i) + L(k - w_d - \sum_{i=2}^{d} m_i w_i - d\Delta w_d) \]

subject to \( \sum_{i=2}^{d} m_i = d \). Similar to the discussion in section 3.2.3, the recurrences (3.12) for \( d \geq 7 \) are dominated by the recurrences

\[ L(k) \leq L(k - 1 - \sum_{i=2}^{6} m_i \Delta w_i) + L(k - 1 - \sum_{i=2}^{6} m_i w_i - m_{\geq 7}) \]

subject to \( \sum_{i=2}^{6} m_i + m_{\geq 7} = 7 \).

Numerically, we find that when \((w_2, w_3, w_4, w_5) = (0.5390437, 0.8625855, 0.9736865, 1)\), we have \( \beta = 1.26666734... < 2^{0.34104} \). Thus the running time of the algorithm in Figure 3.10 is \( O^*(2^{0.34104n}) \). In this example, we find a tighter bound by using two more weights. By the above weights found, it seems sufficient to use only three weights for this algorithm. When comparing with the time complexity of the first algorithm, introducing the concept of dominance speeds up the original algorithm significantly.

### 3.4 The Third Algorithm

The third algorithm is modified from algorithms in [1, 3] and the second algorithm by introducing the concept of folding. The third algorithm is given in Figure 3.11.

Note that in this algorithm, vertices of degree less than or
CHAPTER 3. IMPROVEMENTS

int mis(G)
{
(0) if (|V| = 0), return 0;
(1) if (there exists a vertex v such that \(d(v) = 0\)),
return 1 + mis(G - \{v\});
(2) if (there exist two vertices v and w such that \(N^*(w) \subseteq N^*(v)\)),
return mis(G - \{v\});
(3) if (there exists a foldable vertex v with \(d(v) \leq 3\),
select such vertex of minimum degree and fold),
return 1+mis(G(v));
(4) select a vertex v of maximum degree,
return max{mis(G - \{v\}), 1+mis(G - N^*(v))};
}

Figure 3.11: The third algorithm, modified from [3, 1]

equal to 2 can be filtered out without branching. It is because vertices of degree 2 either dominate other vertices or are foldable. Thus, rule (2) or (3) applies. Therefore, we have to amend our initial assumptions to the following.

1. We assume \(w_0 = w_1 = w_2 = 0\). It is because vertices of degree less than or equal to 2 can be filtered out without branching.

2. We assume \(w_i = 1\), for all \(i \geq 7\). Then, we only have to compute finite and small number of weights.

3. Let \(\Delta w_i = w_i - w_{i-1}\) be the decrease in weight of a vertex of degree \(i\). We assume \(\Delta w_3 \geq \Delta w_4 \geq ... \geq \Delta w_7 \geq 0\).

However, folding (rule (3)) may increase the size of the graph. Therefore, we need to add a few constraints. Suppose at a step, we apply rule (3) and fold a vertex \(v\) with \(d(v) = d\). We need to consider cases when \(d = 2, 3\).
Figure 3.12: $v$ is not foldable in the first graph while rule (2) applies in the third and fourth graphs.

1. $d = 2$
   Let $N(v) = \{u_1, u_2\}$ and $d(u_i) = d_i$. Note that $u_1, u_2$ are non-adjacent, otherwise $v$ dominates $u_1$. Furthermore, the new vertex $u_{12}$ has degree at most $(d_1 - 1) + (d_2 - 1) = d_1 + d_2 - 2$. Thus we have for all $d_1, d_2 \geq 2$,
   $$w_2 + w_{d_1} + w_{d_2} - w_{d_1+d_2-2} = w_{d_1} + w_{d_2} - w_{d_1+d_2-2} \geq 0. \quad (3.14)$$

2. $d = 3$
   Let $N(v) = \{u_1, u_2, u_3\}$ and $d(u_i) = d_i$. Suppose $(u_1, u_2) \in E$ and $(u_1, u_3), (u_2, u_3) \notin E$. (Only this case is possible for folding, see Figure 3.12.) Note that the new vertex $u_{13}$ has degree at most $(d_i - 2) + (d_3 - 1) + 1 = d_i + d_3 - 2$ for $i = 1, 2$. Thus we have for all $d_1, d_2, d_3 \geq 3$,
   $$w_3 + w_{d_1} + w_{d_2} + w_{d_3} - w_{d_1+d_3-2} - w_{d_2+d_3-2} \geq 0. \quad (3.15)$$

**Lemma 3.3.** Constraints (3.14) and (3.15) for $d_i > 7$ is dominated by the constraints with $d_i$ replaced by 7.

*Proof.* If $d_i > 7$ for some $i$, we have $w_{d_i} = 1 = w_7$ and $w_{d_i+d_j-2} =$
\(1 = w_{d_j+5}\), where \(d_j \geq 2\).

Consider the constraint (3.14). Without loss of generality, we suppose \(d_1 > 7\). If we replace \(d_1\) by 7, we have the new constraint

\[ w_{d_2} - w_{d_2+5} + 1 \geq 0 \]

for \(d_2 \geq 2\). For \(d_2 \geq 2\), we have \(w_{d_1+d_2-2} = w_{d_2+5} = 1\). Thus,

\[ w_{d_1} + w_{d_2} - w_{d_1+d_2-2} = 1 + w_{d_2} - w_{d_2+5} \geq 0. \]

Consider the constraint (3.15). Without loss of generality, we suppose \(d_1 > 7\). If we replace \(d_1\) by 7, we have the new constraint

\[ w_3 + w_{d_2} + w_{d_3} - w_{d_3+5} - w_{d_2+d_3-2} + 1 \geq 0 \]

for \(d_2, d_3 \geq 3\). For \(d_2, d_3 \geq 3\), we have \(w_{d_1+d_3-2} = w_{d_3+5} = 1\). Thus,

\[
\begin{align*}
  w_3 + w_{d_1} + w_{d_2} + w_{d_3} - w_{d_1+d_3-2} - w_{d_2+d_3-2} \\
  = w_3 + w_{d_2} + w_{d_3} - w_{d_3+5} - w_{d_2+d_3-2} + 1 \\
  \geq 0.
\end{align*}
\]

Therefore, we only need to consider the constraint (3.14) for \(d_1, d_2 \in \{2, 3, 4, 5, 6, 7\}\) and the constraint (3.15) for \(d_1, d_2, d_3 \in \{3, 4, 5, 6, 7\}\).

The running time analysis for the rule (4) of the algorithm in Figure 3.11 is divided into the following two cases.
1. We consider the subproblem for $G - \{v\}$.

The size of the problem decreases by $w_d$ because of the removal of $\{v\}$. The size also decreases by $\sum_{i=3}^{d} m_i \Delta w_i$ due to the reduction in $N(v)$. Thus, the total reduction in size is

$$\Delta_1 = w_d + \sum_{i=3}^{d} m_i \Delta w_i.$$ 

2. We consider the subproblem for $G - N^*(v)$.

The size of the problem decreases by $w_d$ because of the removal of $\{v\}$. The size further decreases by $\sum_{i=3}^{d} m_i w_i$ due to the removal of $N(v)$. There may also be reduction in $N^2(v)$. Note that for each vertex $w$ in $N(v)$, it must have at least one neighbour in $N^2(v)$, otherwise $w$ dominates $v$ ($N^*(w) \subseteq N^*(v)$) and rule (2) applies. Moreover, if $w$ is in $N(v)$ such that $d(w) = 3$, then there must be exactly two edges between $w$ and $N^2(v)$. Otherwise, we can fold $w$ and rule (3) applies. Therefore, there are at least $d + m_3$ edges between $N(v)$ and $N^2(v)$. Since for each edge between $N(v)$ and $N^2(v)$, the reduction is at least $\Delta w_d$ by initial assumption 3. Thus, the reduction in $N^2(v)$ is at least $(d + m_3)\Delta w_d$. In this case, the total reduction in size is

$$\Delta_2 \geq w_d + \sum_{i=3}^{d} m_i w_i + (d + m_3)\Delta w_d.$$
By the analysis above, we get the following recurrences

\[
L(k) \leq L(k - w_d - \sum_{i=3}^{d} m_i \Delta w_i) + L(k - w_d - \sum_{i=3}^{d} m_i w_i - (d + m_3) \Delta w_d)
\]  

subject to \( \sum_{i=3}^{d} m_i = d \).

For \( i \geq 8 \), we have \( \Delta w_i = 0 \) and \( w_i = 1 \). By the discussion in section 3.2.2, we have recurrences (3.16) for \( d \geq 8 \) are dominated by

\[
L(k) \leq L(k - 1 - \sum_{i=3}^{7} m_i \Delta w_i) + L(k - 1 - \sum_{i=3}^{7} m_i w_i - m_{\geq 8})
\]  

subject to \( \sum_{i=3}^{7} m_i + m_{\geq 8} = 8 \).

By the help of the computer, we numerically find that when \( \mathbf{w} = (0.5196894, 0.8202818, 0.9465605, 0.9927809) \), we get \( \beta = 1.22662682... < 2^{0.29470} \). Thus the running time of the third algorithm in Figure 3.11 is \( O^*(2^{0.29470n}) \). The performance of the algorithm improves a lot by introducing the concept of folding to deal with degree 2 vertices in which dominance cannot be applied.

When compared with the upper bound found using standard analysis (\( O^*(2^{0.406n}) \)), we can design a measure for the problem which capture the characteristics of the algorithm. Thus, we can sometimes find a much better bound by using the method "Measure and Conquer".

\[\square\text{ End of chapter.}\]
Chapter 4

Lower Bounds

In chapter 3, we have illustrated how to achieve improvement on the upper bounds of the running time of four different algorithms by applying the technique of "Measure and Conquer". However, we do not know if the upper bounds found are tight. Therefore, in this chapter, we will investigate the lower bounds on running time of the four algorithms. The idea of the input graphs are based on examples given by Fomin, Grandoni and Kratsch in [3, 4].

4.1 Tarjan and Trojanowski's Algorithm

Consider the graph $G_l = (V, E)$ with $|V| = n = 7l$, where $l \geq 1$. $G_l$ consists of $l$ blocks $B_i$, where $i = 1, 2, ..., l$, with each block $B_i$ consisting of 7 vertices, namely $u_i, a_i, b_i, c_i, d_i, e_i, f_i$. For $i = 1, 2, ..., l$, $u_i$ is adjacent to all of $\{a_i, b_i, c_i, d_i, e_i, f_i\}$ and $\{a_i, b_i, c_i\}$ are adjacent to all of $\{d_i, e_i, f_i\}$. There are also edges $(a_i, a_{i+1}), (b_i, b_{i+1}), (c_i, c_{i+1}), (d_i, d_{i+1}), (e_i, e_{i+1}), (f_i, f_{i+1})$ for all $i = 1, 2, ..., l - 1$. By the above construction, we have $d(u_i) = 6$.
Figure 4.1: Input graph $G_l$ for $l = 3$, for the modified Tarjan and Trojanowski's Algorithm

for all $i = 1, 2, ..., l$, $d(a_i) = d(b_i) = d(c_i) = d(d_i) = d(e_i) = d(f_i) = 6$ for all $i = 2, 3, ..., l - 1$, $d(a_1) = d(b_1) = d(c_1) = d(d_1) = d(e_1) = d(f_1) = 5$ and $d(a_l) = d(b_l) = d(c_l) = d(d_l) = d(e_l) = d(f_l) = 5$. The graph $G_l$ for $l = 3$ is given in Figure 4.1.

Theorem 4.1. The running time of the algorithm in Figures 3.7 and 3.8 is $\Omega(2^2) = \Omega(2^{0.142n})$.

Proof. We apply the algorithm on $G_l$. Since all vertices of $G_l$ are of degree 5 or more, therefore rule (4) applies and the algorithm branches at a vertex of maximum degree. The algorithm may branch at $u_1$. If so, in the first case, $u_1$ is removed from graph and $d(a_1) = d(b_1) = d(c_1) = d(d_1) = d(e_1) = d(f_1) = 4$. Thus, in the next step, rule (4) still applies and the algorithm may branch at $u_2$ (see Figure 4.2 for the case $l = 3$). In the second
CHAPTER 4. LOWER BOUNDS

case, \( N^*(u_1) \) is removed and \( G_{l-1} \) is left. Therefore, at the next step, rule (4) applies and the algorithm may branch at \( u_2 \).

Now suppose we are at some step \( k \) and we have branched at \( \{u_1, u_2, ..., u_{k-1}\} \) in sequence, we may reach the following graphs.

1. The graph \( G_{l-k+1} \).
   Then the algorithm may branch at the vertex \( u_k \) which is of maximum degree.

2. The graph \( G_{l-k+1} \) together with some connected blocks \( B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, ..., B_j - \{u_j\} \). (There are edges connecting the blocks by construction.)
   Since the blocks are connected, all vertices are of degree at least 4. Thus, the algorithm may still choose to branch at \( u_k \).

3. The graph \( G_{l-k+1} \) together with isolated blocks \( B_h - \{u_h\} \).
   (They are not connected to any other blocks.) (possibly with some connected blocks \( B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, ..., B_j - \{u_j\} \).)
   Note that in the isolated blocks, there exists vertices of degree 3. (see Figure 4.3 for the case \( l = 3 \) and \( k = 2 \)) Rule (3.4) applies and eventually we will arrive at a subproblem with only \( G_{l-k+1} \). (possibly together with connected blocks \( B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, ..., B_j - \{u_j\} \) of all vertices of at least degree 4.) Thus, the algorithm may branch at \( u_k \).

Therefore, by repeating the above argument, the algorithm may branch at \( u_1, u_2, ..., u_l \). The algorithm may at least branch
for $l = \frac{n}{7}$ times resulting in at least $l = \frac{n}{7}$ leaves in the search tree to solve $G_l$. Thus, the running time of the algorithm in Figures 3.7 and 3.8 is $\Omega(2^{\frac{n}{7}}) = \Omega(2^{0.142n})$.

4.2 The First Algorithm

We consider the graph $G_l = (V, E)$ with $|V| = n = 5l$, where $l \geq 1$. $G_l$ consists of $l$ blocks $B_i$, where $i = 1, 2, ..., l$, with each block $B_i$ consisting of 5 vertices, namely $\{u_i, a_i, b_i, c_i, d_i\}$. For $i = 1, 2, ..., l$, $u_i$ is adjacent to all of $\{a_i, b_i, c_i, d_i\}$, $a_i$ is adjacent to $b_i$ and $c_i$ is adjacent to $d_i$. There are also edges $(a_i, a_{i+1}), (b_i, b_{i+1}), (c_i, c_{i+1}), (d_i, d_{i+1})$, for all $i = 1, 2, ..., l - 1$. By the above construction, we have $d(u_i) = 4$ for all $i = 1, 2, ..., l$, $d(a_i) = d(b_i) = d(c_i) = d(d_i) = 4$ for all $i = 2, 3, ..., l - 1$, $d(e_i) = d(f_i) = 3$ for all $i = 1, 2, ..., l$. The graph after removing $u_1$ is shown in Figure 4.2.
CHAPTER 4. LOWER BOUNDS

Figure 4.3: The graph after removing $u_1$ and $N^*(u_2)$.

$d(a_1) = d(b_1) = d(c_1) = d(d_1) = 3$ and $d(a_i) = d(b_i) = d(c_i) = d(d_i) = 3$. The graph $G_l$ for $l = 3$ is given in Figure 4.4.

**Theorem 4.2.** The running time of the first algorithm in Figure 3.9 is $\Omega(2^{5^3}) = \Omega(2^{0.2n})$.

**Proof.** We apply the algorithm on $G_l$. Since all vertices in $G_l$ are of degree at least 3, rule (4) applies and the algorithm branches at a vertex of maximum degree. Vertex $u_1$ is a possible candidate. Suppose the algorithm branches at $u_1$. In the first case, $u_1$ is removed. Thus, we have $d(a_1) = d(b_1) = d(c_1) = d(d_1) = 2$. Since all vertices are of degree 2 or more and there exists vertices of degree at least 3, rule (4) applies in the next step. In the second case, $N^*(u_1)$ is removed and $G_{l-1}$ is left. Therefore, in both subproblems, in the next step, the algorithm may branch...
Now suppose we are at some step $k$ and we have branched at $\{u_1, u_2, ..., u_{k-1}\}$ in sequence, we may reach the following graphs.

1. The graph $G_{l-k+1}$.
   Then the algorithm may branch at the vertex $u_k$ which is of maximum degree.

2. The graph $G_{l-k+1}$ together with some connected blocks $B_i - \{u_i\}$, $B_{i+1} - \{u_{i+1}\}$, ..., $B_j - \{u_j\}$. (There are edges connecting the blocks by construction.)
   Since the blocks are connected, all vertices are of degree at least 2 and there exists vertices of degree at least 3. Thus, the algorithm may still choose to branch at $u_k$.

3. The graph $G_{l-k+1}$ together with isolated blocks $B_h - \{u_h\}$. 

Figure 4.4: Input graph $G_l$ for $l = 3$, for the first algorithm at $u_2$. 

at $u_2$.
(They are not connected to any other blocks.) (possibly with some connected blocks \( B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, \ldots, B_j - \{u_j\} \).)

Note that in the isolated blocks, there exists vertices of degree 1. Rule (2) applies and eventually we will arrive at a subproblem with only \( G_{l-k+1} \). (possibly together with connected blocks \( B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, \ldots, B_j - \{u_j\} \) of all vertices of at least degree 2.) Thus, the algorithm may branch at \( u_k \).

Repeating the above argument, the algorithm may branch at \( u_1, u_2, \ldots, u_t \). The algorithm may at least branch for \( l = \frac{n}{5} \) times resulting in at least \( l = \frac{n}{5} \) leaves in the search tree to solve \( G_l \). The running time of the first algorithm in Figure 3.9 is \( \Omega(2^{\frac{n}{5}}) = \Omega(2^{0.2n}) \).

\[ \square \]

4.3 The Second Algorithm

**Theorem 4.3.** The running time of the second algorithm in Figure 3.10 is \( \Omega(2^{\frac{n}{3}}) = \Omega(2^{0.2n}) \).

**Proof.** We consider the same graph \( G_l \) in section 4.2. Let us apply the algorithm on \( G_l \). Note that rule (2) dominance does not apply since there are edges connecting the blocks \( B_i \). All vertices are of degree at least 3, thus rule (4) applies and the algorithm may branch at \( u_1 \). In the first case, \( u_1 \) is removed and \( d(a_1) = d(b_1) = d(c_1) = d(d_1) = 2 \). However, dominance cannot be applied after removal of \( u_1 \). In the second case, \( N^*(u_1) \) is
CHAPTER 4. LOWER BOUNDS

removed resulting in $G_{i-1}$. Thus, in both subproblems, rule (4) applies at the next step and the algorithm may branch at $u_2$.

Now suppose we are at some step $k$ and we have branched at \{$u_1, u_2, \ldots, u_{k-1}$\} in sequence, we may reach the following graphs.

1. The graph $G_{l-k+1}$.

Then the algorithm may branch at the vertex $u_k$ which is of maximum degree.

2. The graph $G_{l-k+1}$ together with some connected blocks $B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, \ldots, B_j - \{u_j\}$. (There are edges connecting the blocks by construction.)

Since the blocks are connected, all vertices are of degree at least 2 and there exists vertices of degree at least 3. Moreover, rule (2) dominance cannot be applied. Thus, the algorithm may still choose to branch at $u_k$.

3. The graph $G_{l-k+1}$ together with isolated blocks $B_h - \{u_h\}$. (They are not connected to any other blocks.) (possibly with some connected blocks $B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, \ldots, B_j - \{u_j\}$.)

Note that in the isolated blocks, there exists vertices of degree 1. Rule (2) applies and eventually we will arrive at a subproblem with only $G_{l-k+1}$. (possibly together with connected blocks $B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, \ldots, B_j - \{u_j\}$ of all vertices of at least degree 2 and dominance cannot be applied.) Thus, the algorithm may branch at $u_k$.

By iterating the above argument, the algorithm may branch at $u_1, u_2, \ldots, u_{l-1}$. There are at least $l - 1 = \frac{n}{5} - 1$ leaves in
the search tree to solve $G_i$. Therefore, the running time of the algorithm in Figure 3.10 is $\Omega(2^{\frac{9}{5}}) = \Omega(2^{0.2n})$.

4.4 The Third Algorithm

For this algorithm, the input graph is the same as in the example given by Fomin, Grandoni and Kratsch [3]. We consider the graph $G_l = (V, E)$ with $|V| = n = 6l$, where $l \geq 1$. $G_l$ consists of $l$ blocks $B_i$, where $i = 1, 2, \ldots, l$, with each block $B_i$ consisting of 6 vertices, namely $\{u_i, a_i, b_i, c_i, d_i, e_i\}$. For $i = 1, 2, \ldots, l$, $u_i$ is adjacent to all of $\{a_i, b_i, c_i, d_i, e_i\}$, $a_i$ is adjacent to $b_i$, $b_i$ is adjacent to $c_i$, $c_i$ is adjacent to $d_i$ and $e_i$ is adjacent to $a_i$. There are also edges $(a_i, a_{i+1})$, $(b_i, b_{i+1})$, $(c_i, c_{i+1})$, $(d_i, d_{i+1})$, $(e_i, e_{i+1})$, for all $i = 1, 2, \ldots, l - 1$. By the above construction, we have $d(u_i) = 5$ for all $i = 1, 2, \ldots, l$, $d(a_i) = d(b_i) = d(c_i) = d(d_i) = d(e_i) = 5$ for all $i = 2, 3, \ldots, l - 1$, $d(a_1) = d(b_1) = d(c_1) = d(d_1) = d(e_1) = 4$ and $d(a_i) = d(b_i) = d(c_i) = d(d_i) = d(e_i) = 4$. The graph $G_l$ for $l = 3$ is given in Figure 4.5.

**Theorem 4.4.** The running time complexity of the third algorithm in Figure 3.11 is $\Omega(2^{\frac{9}{5}}) = \Omega(2^{0.166n})$.

*Proof.* We apply the algorithm on the graph $G_l$. Note that neither dominance nor folding can be applied. Thus, rule (4) applies and the algorithm may branch at $u_1$. In the first case, $u_1$ is removed. As a result, $d(a_1) = d(b_1) = d(c_1) = d(d_1) = d(e_1) = 3$ but $a_1, b_1, c_1, d_1, e_1$ are not foldable. Dominance cannot be applied after the removal of $u_1$. In the second case, $N^*(u_1)$ is
removed and the remaining graph is $G_{l-1}$. In both cases, rule (4) applies and the algorithm may branch at $u_2$ at the next step.

Now suppose we are at some step $k$ and we have branched at $\{u_1, u_2, ..., u_{k-1}\}$ in sequence, we may reach the following graphs.

1. The graph $G_{l-k+1}$.

Then the algorithm may branch at the vertex $u_k$ which is of maximum degree.

2. The graph $G_{l-k+1}$ together with some connected blocks $B_i - \{u_i\}, B_{i+1} - \{u_{i+1}\}, ..., B_j - \{u_j\}$. (There are edges connecting the blocks by construction.)

Since the blocks are connected, rule (2) dominance cannot be applied. Moreover, Rule (3) folding cannot be applied. Thus, the algorithm may still choose to branch at $u_k$. 

Figure 4.5: Input graph $G_l$ for $l = 3$, for the third algorithm
3. The graph $G_{l-k+1}$ together with isolated blocks $B_h - \{u_h\}$.
(They are not connected to any other blocks.) (possibly with some connected blocks $B_i - \{u_i\}$, $B_{i+1} - \{u_{i+1}\}$, ..., $B_j - \{u_j\}$.)

Note that the isolated blocks are cycles with 5 vertices.

Rule (3) applies and eventually we will arrive at a subproblem with only $G_{l-k+1}$. (possibly together with connected blocks $B_i - \{u_i\}$, $B_{i+1} - \{u_{i+1}\}$, ..., $B_j - \{u_j\}$ in which dominance and folding cannot be applied.) Thus, the algorithm may branch at $u_k$.

By repeating the above argument, the algorithm may branch at $u_1, u_2, ..., u_{l-1}$. Therefore, there are at least $l - 1 = \frac{n}{6} - 1$ leaves in the search tree to solve the graph $G_l$. It follows that the time complexity of the algorithm in Figure 3.11 is $\Omega(2^{\frac{n}{6}}) = \Omega(2^{0.166n})$.

□ End of chapter.
Chapter 5

Conclusion

In this thesis, we investigated the performance of existing exact algorithms solving the maximum independent set problem. We discussed four algorithms and investigated the impact of applying "Measure and Conquer". The upper and lower bounds on the running time of the four algorithms investigated are listed in Table 5.1.

We first modify the algorithm of Tarjan and Trojanowski [8]. The modified algorithm is 2 pages long and complicated. By standard analysis, the running time is $O^*(2^{0.406n})$ which is not satisfactory when compared with the original algorithm of Tarjan and Trojanowski. After applying "Measure and Conquer", the improvement is little since the modified algorithm consists of too many branching rules even for low degree vertices.

Therefore, we turn our attention to simpler base algorithms. We start at the simplest algorithm (the first algorithm [9]) which branches at vertices of degree 3 or more. Since vertices of degree 1 or less can be filtered without branching, assigning weights on
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Upper bound (Standard Analysis)</th>
<th>Upper bound (2 weights used)</th>
<th>Upper bound (4 weights used)</th>
<th>Lower bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tarjan and Trojanowski's Algorithm (section 3.1)</td>
<td>$O^*(2^{0.406n})$</td>
<td>$O^*(2^{0.39606n})$</td>
<td>$O^*(2^{0.39606n})$</td>
<td>$\Omega(2^{0.142n})$</td>
</tr>
<tr>
<td>The First Algorithm (section 3.2)</td>
<td>$O^*(2^{0.465n})$</td>
<td>$O^*(2^{0.36786n})$</td>
<td>$O^*(2^{0.36786n})$</td>
<td>$\Omega(2^{0.2n})$</td>
</tr>
<tr>
<td>The Second Algorithm (section 3.3)</td>
<td>$O^*(2^{0.465n})$</td>
<td>$O^*(2^{0.36199n})$</td>
<td>$O^*(2^{0.34104n})$</td>
<td>$\Omega(2^{0.2n})$</td>
</tr>
<tr>
<td>The Third Algorithm (section 3.4)</td>
<td>$O^*(2^{0.406n})$</td>
<td>$-$</td>
<td>$O^*(2^{0.29470n})$</td>
<td>$\Omega(2^{0.166n})$</td>
</tr>
</tbody>
</table>

Table 5.1: Results on the four algorithms
vertices of different degrees helps us find a much tighter upper bound in the analysis. However, increasing the number of weights computed does not help in this case.

In order to improve the upper bound, the technique of dominance and folding is added to the first algorithm. The third algorithm is modified from Beigel's [1] (by imposing the constraints on the degree of the vertex we fold) and from Fomin, Grandoni and Kratsch's [3] (ignoring the technique of mirroring). We improved a lot on the asymptotic upper bounds by introducing the two techniques. However, the upper bounds we found are far from tight and there are huge gaps between the lower and the upper bounds. It seems possible to refine the measure and the running time analysis to find better upper bounds.

We have seen examples of applying the technique "Measure and Conquer" on algorithms solving the maximum independent problem. The impact is tremendous. The future trend to design exact algorithms for NP-hard problems will be designing a measure for the problem instance which can capture the characteristic of the algorithm.

☐ End of chapter.
Bibliography


