ON IMPROVING FPT K -VERTEX COVER, WITH
APPLICATIONS TO SOME COMBINATORIAL PROBLEMS

by
Peter J. Taillon

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

Ottawa-Carleton Institute for Computer Science

School of Computer Science

CARLETON UNIVERSITY

Ottawa, Ontario
January, 2007

© Copyright by Peter J. Taillon, 2007

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-23301-6

Our file Notre référence

ISBN: 978-0-494-23301-6

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

Table of Contents

List of Tables	vi
List of Figures	vii
Abstract	viii
Acknowledgements	ix
Chapter 1 Introduction	1
Chapter 2 Background	6
2.1 Graph theory terms	6
2.2 Classical complexity	6
2.2.1 <i>NP</i> -completeness	6
2.2.2 Reductions	8
2.3 Parameterized languages and fixed-parameter tractability	10
2.3.1 Fixed-parameter tractability	10
2.3.2 Reductions	10
2.3.3 Evaluating FPT algorithms	11
2.3.4 Parameterized intractability	12
2.4 Algorithms for the k -VERTEX COVER problem	14
2.4.1 Preliminary definitions	14
2.4.2 Graph kernelization	15
2.4.3 Combinatorial tree search	17
2.4.4 Algorithm summary	18
2.4.5 Algorithms for bipartite graphs	21
2.4.6 Maximum independent set algorithms	21
2.5 Graph partitioning and fixed-parameter algorithms	22
2.5.1 Preliminary definitions	22

2.5.2	Separators and approximation	23
2.5.3	Separators and FPT algorithms	24
2.5.4	Edge-separators and FPT algorithms	27
2.5.5	Parameterized hardness for GRAPH k -CUT	28
Chapter 3	Parallel k-VERTEX COVER	30
3.1	Background	30
3.1.1	Multiple sequence alignment and the VERTEX COVER problem	30
3.1.2	Parallel models of computation	31
3.1.3	Search techniques and optimization	32
3.2	Relationship between processors and parameter in FPT algorithms .	35
3.3	Framework overview	37
3.3.1	Parallel FPT k -vertex cover algorithm	38
3.3.2	Algorithm description	39
3.3.3	Asymptotic analysis	41
3.4	Experimental results	42
3.4.1	Methodology	42
3.4.2	Data sets	42
3.4.3	Sequential and parallel-simulation k -vertex cover algorithm per- formance	44
3.4.4	Parallel k -vertex cover algorithm performance	46
3.5	Summary	47
Chapter 4	Parallel Memorization	48
4.1	Background	49
4.2	Memorization and the parallel framework	51
4.2.1	Asymptotic analysis	52
4.3	Refined parallel memorization	53
4.3.1	Effective table generation	53
4.3.2	Improved parallel memorization	56
4.4	Parallel CGM memorization	59

4.4.1	Distributed lookup table	59
4.5	Summary	61
Chapter 5	k-VERTEX COVER and Graphs of Bounded Degree	63
5.1	Background	63
5.2	Improving k -vertex cover algorithms in graphs of small degree	64
5.2.1	Determining strategic vertices	67
5.2.2	Processing the separator set	70
5.2.3	Asymptotic analysis	73
5.3	Predictive performance analysis	74
5.4	Algorithm for k -VERTEX COVER problem in general graphs	77
5.5	Algorithm for Maximum Independent Set problem for graphs of small degree	78
5.6	Summary	79
Chapter 6	The CONSTRAINED MINIMUM BIPARTITE VERTEX COVER Problem	80
6.1	Background	81
6.1.1	Matching theory	81
6.1.2	Algorithms for CONSTRAINED MINIMUM BIPARTITE VERTEX COVER	82
6.1.3	Details of the Chen and Kanj algorithm	82
6.2	CGM algorithm for CONSTRAINED MINIMUM BIPARTITE VERTEX COVER	85
6.2.1	Local memory and exponential algorithms	86
6.2.2	CGM Gallai-Edmonds decomposition	86
6.2.3	CGM Dulmage-Mendelsohn decomposition	88
6.2.4	Search tree branching	89
6.3	Gallai-Edmonds decomposition for general graphs	90
6.3.1	EREW PRAM Gallai-Edmonds algorithm for general graphs .	91
6.3.2	CGM Gallai-Edmonds decomposition for general graphs . . .	93

6.3.3	Applications of the Gallai-Edmonds decomposition	94
6.4	Summary	95
Chapter 7	k-VERTEX COVER and the MAXIMUM CLIQUE Problem	96
7.1	Background	96
7.2	Improved MAXIMUM CLIQUE algorithms	97
7.2.1	Vertex cover-based maximum clique algorithm	98
7.2.2	Heuristic maximum clique algorithm	100
7.3	Experimental results	101
7.3.1	Methodology	101
7.3.2	Data sets	103
7.3.3	FPT k -vertex cover maximum clique algorithm performance .	104
7.3.4	Maximum clique heuristic performance	104
7.4	Discussion of experimental results	108
7.5	Summary	109
Chapter 8	Conclusion and Future Work	110
8.1	Contribution summary	110
8.2	Future work	113
	Bibliography	115
	Appendix A Glossary	130
	Appendix B Symbols and notation	135
	Appendix C k-Vertex Cover Branching Rules	136

List of Tables

Table 2.1	k -Vertex cover algorithm complexities	18
Table 2.2	Maximum independent set algorithm complexities	21
Table 3.1	Protein sequences and conflict graph sizes	43
Table 3.2	Random graph and grid graph data	43
Table 5.1	Component detection experiments: average number of candidate pairs (average separator size, average component size).	75
Table 5.2	Complexities for recent k -vertex cover algorithms	76
Table 5.3	Complexity assuming hypothetical component sizes for vertex cover	77
Table 5.4	Complexity assuming hypothetical component sizes for indepen- dent set	78
Table 7.1	Biological graph instances (1)	103
Table 7.2	Biological graph instances (2)	104
Table 7.3	k -Vertex cover algorithm performance on bio-data (1)	105
Table 7.4	Heuristic performance on bio-data (2)	105
Table 7.5	Performance of exact algorithms on DIMACS	106
Table 7.6	Performance of heuristic on DIMACS	107
Table 7.7	Performance of heuristic on large DIMACS instances	107

List of Figures

Figure 1.1	Relationship between combinatorial graph problems	3
Figure 3.1	Search tree load-balance	39
Figure 3.2	Search path traced by processor using branching rules	40
Figure 3.3	Average sequential wall clock times	44
Figure 3.4	Processor overloading	45
Figure 3.5	Average parallel wall clock times	45
Figure 3.6	Average relative speedup on bio-data graphs	46
Figure 3.7	Average relative speedup of random and grid graph	46
Figure 4.1	Parallel lookup table generation	52
Figure 4.2	Improved parallel table generation	56
Figure 5.1	Tree search with explicit component processing	66
Figure 5.2	Converting a bounded degree graph into a flow network	68
Figure 5.3	Processing degree-2 separator vertices	71
Figure 5.4	Processing separator adjacent to degree-2 vertex	71
Figure 5.5	Degenerate degree-3 and degree-4 branching	73
Figure 6.1	Example of Gallai-Edmonds decomposition for a general graph	83
Figure 6.2	Example of Dulmage-Mendelsohn decomposition for a bipartite graph	84
Figure 7.1	Neighborhood complementation	98
Figure 7.2	Two-level sampling and complementation	98

Abstract

VERTEX COVER, INDEPENDENT SET, CLIQUE are three of the original problems shown to be *NP*-complete by Karp. As fundamental graph-theoretic problems, all three arise in diverse research domains, including computational biology, VLSI design, telecommunications, to name a few. In classical complexity these closely-related problems are considered equivalent in terms of computational difficulty. In parameterized complexity, there exist efficient fixed-parameter algorithms only for the VERTEX COVER problem, with the other two problems being considered intractable. The research for this thesis will focus on improvements to the state-of-the-art for solving the k -VERTEX COVER problem. Advances made to FPT k -vertex cover algorithms can provide better algorithms for solving the INDEPENDENT SET and CLIQUE problems, despite their intractability status. We introduce a general parallel framework for FPT k -vertex cover algorithms, and an improved sequential algorithm for graphs of bounded degree. We explore the applicability of this new sequential algorithm for finding a maximum independent set in a graph of bounded degree. We undertake the first investigation of parallel memorization, and develop a time- and space-efficient parallel algorithm for generating the subgraph lookup table. We investigate a variation of the VERTEX COVER problem for bipartite graphs, known as the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem. We apply parallel techniques to the current best sequential algorithm, whereby we develop CGM algorithms for the Gallai-Edmonds and Dulmage-Mendelsohn decompositions, both integral components for solving this problem. Finally, we develop both exact and heuristic algorithms for finding maximum cliques. Whenever possible, we supplement our analysis with computational experiments to underscore the effectiveness of our algorithms.

Acknowledgements

Cicero said friendship makes prosperity more shining and lessens adversity by dividing and sharing it. A doctoral thesis is a work that spans many years, and throughout that time there are so many people one has the pleasure of meeting, and better yet, calling friend. Although I want to omit no one, I am limited by time and space, the two classical resources in computation. I particularly want to thank for their inspiration and support Dr. Nader Bshouty, Dr. Mike Fellows, Robert Kane, Dr. Mike Langston, Dr. Doron Nussbaum, Dr. Shana Poplack, Dr. Irwin Reichstein, Dr. Jorg Sack, Dr. Brett Stephens, Ryan Taylor, and Dr. Petr Zizler. I also want to thank the always helpful administrative staff in the School of Computer Science. A special acknowledgement goes to former fellow students Dr. Dan Ghica, Dr. Kenji Imasaki, and Dan Wevrick.

There is a person who should receive very special mention. Lorraine Byerley has stood by me through this whole experience, many times sacrificing her own happiness as I worked to achieve my goal. I am happily looking forward to a life of making up for lost time.

As usual my family has been a constant source of strength for me, and I am deeply indebted to my parents, and sisters, Monique, Anne, Marie, and Michelle, for their unquestioning love and support.

This thesis is dedicated to my father, William Vincent Taillon, who passed away on November 22nd, 2004.

Démerdez-vous

-Motto, French Foreign Legion

Chapter 1

Introduction

Whatever you think you can do or believe you can do, begin it. Action has magic, grace and power in it.

-Johann Wolfgang Von Goethe

The following three graph-theoretic decision problems were originally shown to be *NP*-complete by Karp [111], and all three are fundamental in important applications such as VLSI design, resource allocation, computational biology, to name a few.

VERTEX COVER

Instance: A graph, $G = (V, E)$, and positive integer $K \leq |V|$.

Question: Is there a vertex cover of size K or less for G , i.e., a subset, $VC \subseteq V$, such that $|VC| \leq K$, and for each edge $(u, v) \in E$, at least one of u or v belongs to VC ?

INDEPENDENT SET

Instance: A graph, $G = (V, E)$, and positive integer $K \leq |V|$.

Question: Does G contain an independent set of size K or more, i.e., a subset, $I \subseteq V$, such that $|I| \geq K$, and for all $u, v \in I$, $(u, v) \notin E$?

CLIQUE

Instance: A graph, $G = (V, E)$, and positive integer $K \leq |V|$.

Question: Does G contain a clique of size K or more, i.e., a subset $C \subseteq V$ such that $|C| \geq K$, and for all $u, v \in C$, $(u, v) \in E$?

Downey and Fellows [65] developed parameterized tractability as a framework for studying classes of problems that are *NP*-complete and yet for which there exist

efficient algorithms that decide the problem with an exponential complexity that is a function of some fixed parameter. Such problems are called *Fixed-Parameter Tractable*, or FPT. For example, the parametric formulation of the VERTEX COVER problem would be:

k-VERTEX COVER

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

Parameter: A positive integer k .

Question: Is there a vertex cover of size k or less for G ?

The parameter, k , is a fixed constant independent of $|V|$. The best FPT algorithm that solves the *k*-VERTEX COVER problem has complexity $O(k|V| + 1.2738^k)$ [45]. Although there are many problems that are fixed-parameter tractable, not all parameterized problems are in the class *FPT*. The *W*-hierarchy consists of classes of problems that are not likely to be fixed-parameter tractable, with the class $W[1]$ representing the lowest level of intractability. Unlike VERTEX COVER, the closely-related *k*-INDEPENDENT SET and *k*-CLIQUE problems are complete for $W[1]$.

Problems that are in *FPT* possess the characteristic that they can be reduced in polynomial time to a subproblem of size bounded by a function of the fixed parameter, k . This process is called kernelization, and the new problem instance can be exhaustively analyzed for a solution. The combinatorial search phase usually consists of generating a search tree of depth bounded by parameter k . In the case of the *k*-VERTEX COVER problem, effectively all algorithms are based on some form of combinatorial analysis. An algorithm inspects the neighborhoods of selected vertices in the graph and makes branching decisions based on the potential minimum subsets of vertices required to form a cover for that neighborhood. A bounded-depth search tree is generated for the combinations of subsets derived in the previous step.

The research in this thesis focuses on improvement to the state-of-the-art for solving the *k*-VERTEX COVER problem and two related combinatorial problems, MAXIMUM INDEPENDENT SET and MAXIMUM CLIQUE (see Figure 1.1). We also concentrate our efforts on establishing a parallel framework for future FPT algorithms,

Figure 1.1: Relationship between VERTEX COVER, INDEPENDENT SET, and CLIQUE: (a) undirected graph, $G = (V, E)$; (b) minimum vertex cover, $VC \subseteq V$, and maximum independent set, $V \setminus VC$ (triangle-shaped vertices), of graph G ; (c) maximum independent set of $\bar{G} = (V, \bar{E})$ corresponding to maximum clique of G .

with notable emphasis on the k -VERTEX COVER problem. Advances made to FPT k -vertex cover algorithms can provide (indirectly) better algorithms for solving the INDEPENDENT SET and CLIQUE problems, despite their intractability status. With a need for algorithms that apply to solving practical real-world problems, greater and greater emphasis is being placed on parallel computation to bridge the gap between theory, and efficiency and practicality.

We introduce the first parallel FPT algorithm that solves the k -VERTEX COVER problem. This algorithm produces inherently well-balanced subproblems distributed across the processors. (Published in *Journal of Computer and System Sciences*, 2003 [38] and *Proceedings of CCGrid*, 2003 [39].) An implementation performed well on artificial graph instances and graphs derived from multiple sequence alignment datasets, demonstrating excellent speedup as well as solving the largest instances of k -vertex cover known to the FPT community at the time of publication. Having established this parallel FPT framework, we note that any improved sequential solution leads to an improved parallel algorithm, and hence we will restrict our attention to the sequential model.

We undertake the first investigation of parallel memorization. We develop a refined technique for generating the subgraph lookup table in a parallel setting, resulting in a time- and space-efficient algorithm. We also introduce the first CGM algorithm for parallel memorization.

We devise a new FPT k -vertex cover algorithm by efficiently solving the problem

in graphs of small bounded degree, a state to which all current best algorithms eventually reduce their original problem instance. Much work has gone into divide-and-conquer FPT algorithms based on planar separators (e.g., [6, 54]). The complexity of each of these algorithms is a function of the separator sizes for the graph, and so these techniques apply only to graph classes with good separator theorems, e.g., planar graphs or other constrained classes of graphs. With the goal of improving the algorithm for general graphs, we propose the idea of using minimum cuts as a way of inducing components in bounded degree graphs. The components are induced by processing the separator using efficiently adapted branching rules, a technique we refer to as *strategic branching*. Exhaustively solving these small components leads to a much faster algorithm overall. On its own, this result provides an improved k -vertex cover algorithm for the special case where the input graphs have maximum degree bounded by 4. Our analysis indicates that, when a suitably small separator can be found, the complexity of the k -vertex cover algorithm is noticeably improved.

The improved k -vertex cover algorithm can potentially provide an interesting corollary. Recall the close relationship between the VERTEX COVER and INDEPENDENT SET problems. When there exists a suitably small separator, the new k -vertex cover algorithm can have a complexity comparable to the best algorithm for solving the MAXIMUM INDEPENDENT SET problem for graphs of degree bounded by 4.

We investigate a variant of the VERTEX COVER problem restricted to bipartite graphs, known as the CONSTRAINED BIPARTITE MINIMUM VERTEX COVER problem. This problem has many important applications in VLSI and fault-tolerant circuits [99] and the current best FPT algorithm for this is due to Chen and Kanj [43]. We apply parallel techniques to the algorithm of Chen and Kanj, whereby we develop CGM algorithms for the Gallai-Edmonds and Dulmage-Mendelsohn decompositions, both integral components for solving this problem.

The CLIQUE problem, like VERTEX COVER, is one of the original *NP*-complete problems and a wealth of approaches have been used to solve it [26]. We introduce a graph sampling technique that we combine with our improved k -vertex cover algorithm to develop a very fast algorithm for finding a maximum clique in a graph.

We also develop a heuristic for the maximum clique problem that is based on calculating an independent set, rather than a vertex cover. (Published in *Proceedings of Algorithmic Aspects in Information and Management (AAIM)*, 2006 [179].) In experiments this heuristic proves to be very accurate and extremely fast compared to state-of-the-art algorithms [189, 145, 70, 153].

The thesis is outlined as follows. In Chapter 2 we provide a brief overview of classical and parameterized complexity, and a literature review of FPT algorithms for the k -VERTEX COVER problem. We also discuss relevant graph separation and cutting algorithms, and parallel search techniques. We describe our parallel framework in Chapter 3, as well as our FPT parallel k -VERTEX COVER algorithm and experimental results. In Chapter 4, we investigate parallel memorization and its application to our framework. In Chapter 5 we describe a unique approach to accelerating k -vertex cover algorithms by cutting small, bounded degree residual graphs. We discuss how this improvement can also be extended to provide a better algorithm for solving the MAXIMUM INDEPENDENT SET problem. We investigate parallel techniques for the CONSTRAINED BIPARTITE MINIMUM VERTEX COVER in Chapter 6. In Chapter 7 we describe a new approach for finding a maximum clique in an unweighted, undirected graph. We describe both exact and heuristic algorithms, and tabulate their performance when pitted against the DIMACS clique benchmark suite. Finally, we present our conclusions in Chapter 8 followed by an exposition of possible future avenues of research arising from our contributions.

Chapter 2

Background

As knowledge increases, wonder deepens.

-Charles Morgan

2.1 Graph theory terms

Let $G = (V, E)$ be an undirected graph and let $n = |V|$ and $m = |E|$. For $v \in V$, we use $N(v)$ to denote the set of vertices adjacent to v , and $N[v] = N(v) \cup \{v\}$. A graph is r -regular if $\forall v \in V, \deg(v) = r$. The *density* of a graph is defined as follows: if G is an undirected graph then the density is $m/(n(n-1)/2)$; if G is a directed graph then the density is given by $m/(n(n-1))$. A *subgraph* of G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A subgraph H is an *induced subgraph* of G if $E(H)$ contains all edges of G that connect vertices in $V(H)$. A graph is *connected* if there exists a path in G between each pair of vertices in V . A *component* of a graph is a maximal connected subgraph. Three vertices, x, y, z form a *bridge* of a vertex v if v, x, y, z form a cycle. A *triangle* is a cycle of length 3.

2.2 Classical complexity

In this section we provide an overview of classical complexity theory founded on the concepts of languages and reducibility.

2.2.1 NP-completeness

The theory of NP-completeness applies to decision problems. These problems are restricted to questions with *yes* or *no* answers. An abstraction of a decision problem, Π , consists of a set of instances, D_Π , of which a subset are *yes*-instances, $Y_\Pi \subseteq D_\Pi$. The problem Π is specified by a generic instance, consisting of one or more objects, and an associated question, with a *yes-no* answer to the question that pertains to the

objects. An instance, I , belongs to D_Π if it represents a valid instantiation of the objects. Instance I belongs to Y_Π if the answer to the question for that instance is *yes*.

To establish a formal computational framework, the decision problem is recast as a language recognition problem: given a string, $x \in \Sigma^*$, defined over some finite alphabet, Σ , we wish to determine if $x \in L$, where $L \subseteq \Sigma^*$ is a language consisting of all *yes*-instances for a problem Π . The string x is an instantiated instance of the problem Π . A deterministic Turing machine (*DTM*), $\mathcal{M}$, accepts an input if its computation terminates in the accepting state, i.e., the computation of $\mathcal{M}$ determines $x \in L$. A necessary requirement for $\mathcal{M}$ is that it halts for all input strings defined over Σ . The language recognized by $\mathcal{M}$ is defined by:

$$L(\mathcal{M}) = \{x \in \Sigma^* : \mathcal{M} \text{ accepts } x\}$$

In this framework, recognizing a language and solving the corresponding decision problem are equivalent. The time complexity of a deterministic Turing machine computation is the minimum number of steps required before entering a halting state, as a function of $|x|$. The complexity class P is defined as follows:

Definition 1

$$P = \{L : \text{there exists a DTM, } \mathcal{M}, \text{ that runs in polynomial time,} \\ \text{such that } L = L(\mathcal{M})\}$$

Thus the class P consists of all decision problems for which there exists an algorithm that can solve the problem in polynomial time.

Following the model espoused in [82], a nondeterministic Turing machine (*NDTM*) has the same structure as a deterministic Turing machine except it is endowed with a guessing module. This module nondeterministically generates a string, y , called a certificate, that verifies for input x if $x \in L$. The time complexity of a nondeterministic Turing machine computation is the minimum number of steps required for the guessing and checking phases, taken over all accepting computations. The complexity class NP is defined as follows:

Definition 2

$$NP = \{L : \text{there exists a NDTM, } \mathcal{M}', \text{ that runs in polynomial time,} \\ \text{such that } L = L(\mathcal{M}')\}$$

Thus the class NP consists of all decision problems for which there exists an algorithm that, given input x and a certificate y , verifies in time polynomial in the size of x that x is a *yes*-instance. Given these definitions, $P \subseteq NP$, but it is unknown whether this containment is strict.

2.2.2 Reductions

The key notion in the theory of NP -completeness is that of polynomial transformation (also known as *many-one* or *m-reduction*, denoted $\leq_m^p$):

Definition 3 [111] *A polynomial transformation from $L_1 \subseteq \Sigma_1^*$ to $L_2 \subseteq \Sigma_2^*$ is a function $f : \Sigma_1^* \rightarrow \Sigma_2^*$ such that:*

- *function f is polynomial-time computable;*
- *for all $x \in \Sigma_1^*$, $x \in L_1 \iff f(x) \in L_2$.*

The term many-one denotes that the function is not necessarily injective, i.e., several instances of L_1 may map to a single instance of L_2 . Polynomial transformations are transitive, i.e., given three languages L_1, L_2, L_3 , if $L_1 \leq_m^p L_2$ and $L_2 \leq_m^p L_3$ then $L_1 \leq_m^p L_3$.

Definition 4 [111] *A language, L , is hard for some complexity class, $\mathcal{C}$, if for all $L' \in \mathcal{C}$, $L' \leq_m^p L$. If language L is hard for $\mathcal{C}$ and $L \in \mathcal{C}$ then L is $\mathcal{C}$ -complete.*

In the seminal paper by Cook [48] the SATISFIABILITY problem (*SAT*) was shown to be the first NP -complete problem:

SATISFIABILITY

Instance: a set U of variables and a collection C of clauses over U .

Question: Is there a satisfying truth assignment for C ?

SAT is clearly in NP as a NDTM, $\mathcal{M}$, can nondeterministically guess a truth assignment and verify that this is a satisfying assignment in polynomial time.

Each language in NP has associated with it a polynomial time nondeterministic Turing machine program that recognizes it in polynomial time. Cook designed a generic transformation, f_L , that uses a description of an nondeterministic Turing machine, $\mathcal{M} = (\Sigma, \mathcal{Q}, q_0, q_F, \delta)$, that decides a language L , and maps it to an instance of SAT , such that: $x \in L$ (i.e., there is an accepting computation of $\mathcal{M}$ on x in a number of steps polynomial in the size of x) if and only if $f_L(x) \in SAT$ (i.e., there is a satisfying truth assignment for the clauses of $f_L(x)$). An accepting computation of $\mathcal{M}$ on x can be described at each discrete step in terms of the instantaneous description of the machine. The input variables of the expression $f_L(x)$ model $\mathcal{M}$ at time i , $0 \leq i \leq p(|x|)$ (where $p(|x|)$ is a polynomial):

- the state ($Q[i, j]$, $0 \leq j \leq |\mathcal{Q}| - 1$);
- the read/write head position ($H[i, j]$, $-p(|x|) \leq j \leq p(|x|) + 1$);
- the tape contents ($S[i, j, k]$, $-p(|x|) \leq j \leq p(|x|) + 1$, $0 \leq k \leq |\Sigma| - 1$).

Groups of clauses are used to ensure that the machine is in exactly one state at a given time step, the read/write head is scanning exactly one square, the transition from state q_{α_i} at time i to $q_{\alpha_{i+1}}$ at time $i + 1$ is legal according to function δ , and so on. Special variables model the initialization and terminating configurations of $\mathcal{M}$. The transformation can be accomplished in time polynomial in the size of the description of $\mathcal{M}$. Any satisfying truth assignment to $f_L(x)$ would need to specify the certificate value as well as the accepting computation path.

The class of NP -complete problems are those for which there currently exists no deterministic polynomial time algorithm that solves the problem, leading to the conjecture that $P \neq NP$. To show a language L is complete for NP requires proving that L is both hard for NP and is a member of NP , under m -reductions. Showing a language L is hard for NP requires proving a reduction from some problem, L' , that is hard for NP , to L .

2.3 Parameterized languages and fixed-parameter tractability

In this section we provide an overview of the parameterized analogs of the concepts from classical complexity including the counterpart to *NP*-completeness called *parametric intractability*.

2.3.1 Fixed-parameter tractability

In their seminal work, Downey and Fellows introduce a parameterized complexity framework based on languages consisting of two components, a main part and one or more parameters:

Definition 5 [60] *A parameterized language is a set $L \subseteq \Sigma^* \times \Sigma^*$ where Σ is a fixed alphabet.*

Without loss of generality, they consider parameterized languages to be defined as $L \subseteq \Sigma^* \times \mathbb{N}$. For some fixed k , the k^{th} slice of L is $L_k = \{\langle x, k \rangle : \langle x, k \rangle \in L\}$. The class of problems that are fixed-parameter tractable, *FPT*, is defined as follows:

Definition 6 [60] *A parameterized problem L is (strongly uniformly) fixed-parameter tractable if there is a constant α and an algorithm Φ such that Φ decides if $(x, k) \in L$ in time $f(k)|x|^\alpha$ where $f : \mathbb{N} \rightarrow \mathbb{N}$ is an arbitrary function.*

2.3.2 Reductions

Unlike classical reductions, parameterized reductions require that the languages in question have the same parameterized complexity. The parameterized counterpart to the classical reduction is defined as follows:

Definition 7 [60] *Let L and L' be two languages. Language L reduces to language L' by a standard parameterized m -reduction (denoted $\leq_m^{\text{FPT}}$) if there are functions $k \mapsto k'$ and $k \mapsto k''$ from $\mathbb{N}$ to $\mathbb{N}$ and a function $\langle x, k \rangle \mapsto x'$ from $\Sigma^* \times \mathbb{N}$ to Σ^* such that:*

1. $\langle x, k \rangle \mapsto x'$ is computable in time $k''|x|^c$;
2. $\langle x, k \rangle \in L$ if and only if $\langle x', k' \rangle \in L'$.

From the definition we see that in (parameterized) polynomial time, the k^{th} slice of language L can be reduced to the k'^{th} slice of language L' . Most concrete reductions are m -reductions of the form $(x, k) \mapsto (x', f(k))$, with x' depending on x and k , a running time $h(k)|x|^\alpha$, and f and h are recursive. Similarly to classical complexity, we have the following result:

Lemma 1 [60] *If the parameterized problem L reduces to the parameterized problem L' , and if L' is fixed-parameter tractable, then L is fixed-parameter tractable.*

One of the fundamental characteristics of problems in the class FPT is the concept of kernelizability:

Definition 8 [60] *A parameterized problem L is kernelizable if there is a parametric transformation of L to itself that satisfies:*

1. *The running time of the transformation of (x, y) into (x', y') , where $n = |x|$, is bounded by a polynomial $q(n, k)$ (so that in fact, this is a polynomial transformation of L to itself),*
2. $k' \leq k$,
3. $|x'| \leq h(k)$, where h is an arbitrary function.

We mention in passing that various definitions for parallel parameterized complexity classes have also been proposed. For example, see Bodlaender, Downey, and Fellows [23], or Cesati and Di Ianni [33].

2.3.3 Evaluating FPT algorithms

Downey and Fellows introduced the notion of *klam* value as a means of evaluating and comparing FPT algorithms [65]. Consider an algorithm, A , that solves a problem $\Pi \in FPT$. The *klam* value of A is the largest k such that:

1. Π can be solved by A in time $n^{O(1)} + f(k)$, where n is the size of the problem instance;
2. $f(k) \leq \mathcal{U}$, where $\mathcal{U}$ is taken as a bound on the maximum number of steps of any computation.

The value for $\mathcal{U}$ is universally accepted as 10^{20} . For example, consider the algorithm from the previous section with complexity $O(2^k)$ and the algorithm of Balasubramanian, Fellows, and Raman with complexity $O(1.324718^k)$ [15]. The first algorithm has a klam value of 66 ($2^{66} = 7.378697629 \times 10^{19} \leq 10^{20}$) while the second has a value of 163 ($1.324718^{163} = 8.056015735 \times 10^{19} \leq 10^{20}$).

2.3.4 Parameterized intractability

Given the framework from the previous section, there exist many NP -complete problems that are well-solved given a fixed parameter, e.g., VERTEX COVER [43], PLANAR DOMINATING SET [8], and FEEDBACK VERTEX SET [60]. Yet not all problems are fixed-parameter tractable. Recall that the class NP captures the notion of verifiability, i.e., given a solution we can verify that the problem instance is in fact a *yes*-instance, in polynomial time. Problems that are NP -complete are considered equally difficult to solve.

In parameterized complexity, there are problems for which there currently exist no better algorithm than an exhaustive approach that enumerates all subsets of a fixed size k . For example, the best algorithms for solving the k -CLIQUE, k -INDEPENDENT SET, and k -DOMINATING SET problems have complexity $O(n^{k+1})$. These problems, among many others, are deemed *fixed-parameter intractable*. There are in fact varying degrees of intractability and problems not likely to be fixed-parameter tractable are arranged in classes within what is called the *W-hierarchy*. Downey and Fellows [62, 63, 65] define the *W-hierarchy* in terms of constant-depth Boolean circuits required to verify a weight- k binary vector corresponding to a solution for a parameterized problem. The *W-hierarchy*, analogous to the class NP , captures the notion of the complexity of verifying a solution. In contrast, it provides a much finer distinction between the logical complexity of different problems.

Decision circuits are Boolean circuits with a single output gate. A decision circuit, C , consists of two types of gates: *large* (unbounded fan-in), and *small* (fan-in ≤ 2). The *depth* of a circuit C is defined as the maximum number of gates on any input-to-output path in C .

Definition 9 [63] *The weft of a circuit C is the maximum number of large gates on*

any input-output path in C .

Definition 10 [63] *A parameterized problem $\langle L, k \rangle$ belongs to $W[t]$, $t \geq 1$, if there exists a $k' = g(k)$ such that $\langle L, k \rangle$ reduces to the weight- k' satisfiability problem for constant depth, weft- t circuits.*

Intuitively, solutions for problems in $W[t]$ are easier to verify than those in $W[t + 1]$, $t \geq 1$, leading to the hierarchy:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \dots \subseteq W[P]$$

Let L and L' be parameterized problems. If L' is $W[t]$ -complete, we can prove L is $W[t]$ -hard by showing a reduction $L' \leq_m^{FPT} L$. To prove L is a member of $W[t]$ we show there exists an FPT algorithm that, given input $\langle x, k \rangle$, can derive a circuit C of constant depth h , and weft t , such that C accepts an input vector x' of weight- k' , where $k' = g(k)$, if and only if $\langle x, k \rangle \in L$.

For example, consider the problem k -INDEPENDENT SET ($W[1]$ -complete) and k -DOMINATING SET ($W[2]$ -complete). The DOMINATING SET problem is defined as follows: given a graph, $G = (V, E)$, a dominating set is a subset of vertices, $DS \subseteq V$, such that every vertex $v \in V$ is adjacent to some $u \in DS$. The decision version of the problem is NP -complete. We can describe a parameterized instance of the k -INDEPENDENT SET, $\langle G = (V, E), k \rangle$, by the expression:

$$\prod_{(x,y) \in E} (\bar{x} + \bar{y}) \quad (2.1)$$

The graph G has an independent set of size k if and only if expression (2.1) has a satisfying assignment of weight k . Similarly, we can describe a parameterized instance of k -DOMINATING SET, $\langle G = (V, E), k \rangle$, by the expression:

$$\prod_{x \in V} \sum_{y \in N[x]} y \quad (2.2)$$

The graph G has a dominating set of size k if and only if expression (2.2) has a satisfying assignment of weight k . Translating these two expression into their corresponding Boolean circuits, the circuit complexity (specifically the large gate depth) needed to

verify a solution for an instance of the k -DOMINATING SET problem is greater than that required to verify a solution for an instance of the k -INDEPENDENT SET problem. Although both problems are parametrically intractable, the k -DOMINATING SET problem is in a sense harder than the k -INDEPENDENT SET problem.

2.4 Algorithms for the k -VERTEX COVER problem

All FPT algorithms for the k -VERTEX COVER problem consist roughly of two phases: a series of kernelization operations followed by some form of exhaustive search. While the first phase is performed in polynomial time, the second phase is exponential in the size of the parameter and so requires the most time to complete. The exhaustive search generally takes the form of a bounded-depth tree search. Recent work in this area has focused on formulating branching rules based on identifying well-characterized local vertex-adjacency patterns, and inferring the potential local minimum vertex covers for these patterns.

2.4.1 Preliminary definitions

Search trees are combinatorial structures where each node of the search tree represents a new subproblem instance derived from the instance residing at its parent. The root node of the search tree is initialized with the kernelized problem instance. Let $\langle G = (V, E), k \rangle$ be a problem instance, where G is a graph and $k \in \mathbb{N}$. Given some operation on $\langle G, k \rangle$, e.g., vertex/edge deletion/insertion, we denote by $\langle G', k' \rangle$ the resulting problem instance.

We follow the exposition found in [142] regarding the asymptotic analysis of recurrence relations. For our purposes, we are interested in bounding the number of nodes in a bounded-depth search tree that is generated by recursive calls on smaller subproblems. The tree size can be described by homogeneous linear recurrence equations with constant coefficients and asymptotic solutions for such equations are determined by the roots of their characteristic polynomial.

Consider a recursive algorithm, A , that solves a problem of size n . During execution, A recursively solves subproblems of size $n - d_1, \dots, n - d_j$, where the sequence $(d_1, \dots, d_j)$ is called the branching vector of the recursion. This corresponds to the

recurrence:

$$C(n) \leq C(n - d_1) + \dots + C(n - d_j)$$

The characteristic polynomial for this equation is given by:

$$z^d = z^{d-d_1} + \dots + z^{d-d_j}$$

where $d = \max\{d_1, \dots, d_j\}$. If α is a root of the polynomial equation with maximum value then $C(n)$ is bounded by $O(\alpha^n)$, i.e., up to a polynomial factor (α is the positive real root).

In terms of asymptotic complexity notation, we mention a convention introduced by Woeginger [188], that ignores polynomial factors. For example, a k -vertex cover algorithm with time complexity $O(2^k k^2)$ would be written as $O^*(2^k)$. In this thesis, we will use either notation to describe tree search complexity depending on the circumstance.

2.4.2 Graph kernelization

When applicable, kernelization rules can be used to reduce the size of a graph and the parameter without requiring any recursive decision making. For example, it is always the case that any vertex of degree one can have its neighbor added to the vertex cover, without exceeding the requirement for a minimum cover. This operation reduces the size of the graph by two vertices, and the size of the parameter by one. To begin, we present one of the first kernelization algorithms described as follows [30]:

Let $VC \subseteq V$ denote a set of vertices forming a vertex cover. Given an input graph, $G = (V, E)$, $n = |V|$, $m = |E|$, and a parameter k consider all vertices v such that $\deg(v) > k$. If the number of such vertices is greater than k , then output “No” (clearly there would not be a sufficient number of vertices to form a cover for all edges). Let b be the number of vertices found in the previous step. For all v found above, add v to VC , and remove v and any incident edges from G . The new parameter is $k' = k - b$.

After the kernelization phase either a vertex cover, $VC \subseteq V$, $|VC| \leq k$, is generated, or it is determined that it is impossible for G to have a k -vertex cover, or a kernelized problem, $\langle G', k' \rangle$, is generated where $|V'| \in O(k^2)$ and $k' \leq k$. This algorithm has complexity $O(n^2)$, as it is required to compute the degree of each vertex.

Chen, et al. [43] were the first to discover a technique to produce a linear kernel, based on a theorem of Nemhauser and Trotter [140]. Since this result there have been two notable approaches for generating linear kernels. Several sophisticated kernelization algorithms have been developed recently based on linear programming [2, 3]. Consider the optimization version of the VERTEX COVER problem. It can be restated as an integer programming formulation where given $G = (V, E)$, and assigning $X_u \in \{0, 1\}$ to each vertex $u \in V$ so that the following conditions hold:

1. $\sum_u X_u$ is minimized;
2. $X_u + X_v \geq 1$, for all $(u, v) \in E$.

The objective function is the size of the vertex cover and the set of all feasible solutions consist of functions from V to $\{0, 1\}$ that satisfy condition (2). Consider a relaxation of the integer programming problem, where $X_u \geq 0$ rather than $X_u \in \{0, 1\}$. The value of the objective function returned by the linear programming problem is a lower bound on the objective function returned by the corresponding integer programming problem. The linear programming problem can be recast as a network flow problem, and solved efficiently in that domain. The graph is assumed to be preprocessed, i.e., consist of at most $O(k^2)$ vertices. Kernelization based on Dinic's maximum flow algorithm, with complexity $O(m\sqrt{n})$ given a bipartite graph with m edges and n vertices, would then have complexity $O(k^3)$. The graph G can be kernelized based on the solution to the linear programming problem. Let P , Q , and R be three subsets of the vertices G , where $P = \{u \in V | X_u > 0.5\}$, $Q = \{u \in V | X_u = 0.5\}$, and $R = \{u \in V | X_u < 0.5\}$. Then there exists an optimal vertex cover, VC , such that $VC \supset P$ and $VC \cap R = \emptyset$. After LP-kernelization the graph has size at most $2k$ vertices.

The technique known as *crown reduction* [47, 2] generates independent sets with specific properties, using randomly chosen maximum matchings, and can be described as follows: consider a graph, $G = (V, E)$, and parameter, k . A *crown*, (H, I) , consists of two sets of vertices, H and I such that:

1. I is non-empty independent set.

2. Let $H = N(I)$. There exists a matching M on the edges connecting I and H , such that all elements in H are matched.

Given an instance $\langle G, k \rangle$, G has a k -vertex cover if and only if $\langle G \setminus (I \cup H), k - |H| \rangle$ has a $(k - |H|)$ -vertex cover. Implementations of the crown reduction kernelization have been shown to be very fast and effective in practice compared to other (linear) kernelization algorithms [2]. With respect to edges with one endpoint in I and one endpoint in H , the induced subgraph $G(I \cup H)$ is bipartite. The maximum matching problem on a bipartite graph can also be recast as a network flow problem and again this can then be solved efficiently using Dinic's algorithm. After crown reduction, the graph has size at most $3k$ vertices.

2.4.3 Combinatorial tree search

We will now see a simple example of a branching rule based on analyzing the neighborhood of a designated vertex in the graph. Unlike kernelization rules, branching rules will always generate several possible candidate solutions that form a local cover. The objective is to trade-off the number of branches generated with the number by which the parameter is reduced in each branch. A simple, recursive search tree algorithm for the k -VERTEX COVER problem is described in [66]. Given a graph, $G = (V, E)$, and parameter, k , choose some vertex, v . It must be the case that either $v \in VC$ or $N(v) \subseteq VC$. These two possibilities determine two subsequent branches in the search tree:

1. $\{v\} \cup VC$; replace $\langle G, k \rangle$ with $\langle G' = G \setminus \{v\}, k' = k - 1 \rangle$;
2. $N(v) \cup VC$; replace $\langle G, k \rangle$ with $\langle G' = G \setminus N[v], k' = k - |N(v)| \rangle$.

The tree will size bounded by $O(2^k)$, where k is a constant.

Interleaving tree search and kernelization

A common technique to speedup the search tree phase is to use *interleaving*: at each branching step of the search tree algorithm, we rerun the kernelization phase to see if the intermediary problem instance can be further reduced [65, 66]. Niedermeier

Algorithm	Complexity	Klam value
Buss, Goldsmith [30]	$O(kn + 2^k k^{2k+2})$	66
Papadimitriou, Yannakakis [147]	$O(n + 3^k kn)$	41
Balasubramanian, et al. [15]	$O(kn + 1.324718^k k^2)$	163
Downey, et al. [66]	$O(kn + 1.31951^k k^2)$	166
Niedermeier, Rossmanith [142]	$O(kn + 1.29175^k)$	179
Stege, Fellows [174]	$O(kn + 1.2906^k 2.5k)$	180
Chen, et al. [43]	$O(kn + 1.2852^k)$	183
Chandran, Grandoni [37]	$O(kn + 1.2745^k k^4)$	189
Chen, et al. [45]	$O(kn + 1.2738^k)$	189

Table 2.1: Complexities for k -vertex cover algorithms for general graphs.

and Rossmanith [143] provide an analysis of this technique to show that the overall complexity of the tree search does not increase by a factor matching the cost of the kernelization algorithm because the instances in fact become sufficiently smaller at each branching step. This general technique can be used to eliminate the polynomial factor from the cost of the tree search for any k -vertex cover algorithm. For example, the complexity of the current best algorithm for solving the k -VERTEX COVER problem can be reduced from $O(kn + 1.2852^k k)$ to $O(kn + 1.2852^k)$.

2.4.4 Algorithm summary

We summarize the complexities of algorithms for solving the k -VERTEX COVER problem in Table 2.1. In this section we provide a brief description of each contribution.

One of the first algorithms to use kernelization for finding a k -vertex cover is attributed to Buss and Goldsmith in [30]. Given an instance, $\langle G = (V, E), k \rangle$, they observed that $\forall v \in V$ if $\deg(v) > k$ then it must be the case that $v \in VC$. Applying this approach reduces the original input $\langle G, k \rangle$ to a new instance $\langle G', k' \rangle$ where $|E'| \leq k^2$, and so $|V'| \leq 2k^2$. Until recently this technique has been used extensively in all k -vertex cover algorithms. After applying their kernelization technique, a k -vertex cover can be determined by examining all possible subsets of vertices of size at most k' . Given that the total number of edges of G' is bounded by kk' , this implies that there can be no more than $2kk'$ vertices in G' . This yields an algorithm of complexity $O(kn + 2^k k^{2k+2})$.

Papadimitriou and Yannakakis [147] designed an algorithm to show that k -VERTEX COVER is in P when the cover size k is bounded by $O(\log n)$, where $n = |V|$. The algorithm first generates a maximal matching M in the input graph $G = (V, E)$. If $|M| > k$ then it is clear that a cover, bounded in size by k , cannot exist. If $2 \cdot |M| \leq k$ then the $2 \cdot |M|$ vertices form a cover. Otherwise, exhaustively cycle through the $3^{|M|}$ subsets and determine whether one of these subsets forms a cover of G with size at most k . This step is justified as follows: let U be the set of endpoints of the M edges in the matching. Consider each $(x, y) \in M$: either $x \in VC$ or $y \in VC$ or both $x, y \in VC$. Once a subset of U is picked in a vertex cover, the remainder of the cover is uniquely determined: for every $w \in (V \setminus U)$, $w \in VC$ if and only if $\exists(v, w) \in E$ and $v \notin VC$. The algorithm has a complexity of $O(n + 3^k kn)$.

In [15], Balasubramanian, Fellows, and Raman describe two algorithms. This seminal paper is the first to use non-trivial branching rules to reduce the complexity of the tree search phase. These rules use explicit case-based processing to analyze the neighborhoods of vertices based on their degree and the edges between vertices. The algorithms have overall complexities of $O(kn + (\sqrt{3})^k k^2)$ and $O(kn + 1.324718^k k^2)$, respectively. The kernelization phase uses Buss' kernelization algorithm [30] and introduces two new kernelization rules for processing degree-2 vertices.

The algorithm of Downey, Fellows, and Stege [66] presents three new approaches: it introduces a set of kernelization rules that result in adding edges to the graph instance G' (the application of these rules generally does not result in a smaller parameter but the increase in the degree of the vertices benefits the tree search branching phase); the algorithm interleaves kernelization steps with search tree branching; finally, they cascade branching rule application resulting in tighter branching complexity. This approach is later emulated in [43, 42]. The overall complexity of the algorithm is given by $O(kn + 1.31951^k k^2)$.

In [142], Niedermeier and Rossmanith improve on the two previous results. They adapt the approach introduced by Balasubramanian, et al. where the goal of the search tree phase is to concentrate on processing vertices of small degree, i.e., 2, 3, 4, 5. The key idea of the paper is the extension made to processing higher-degree

vertices (i.e., 4, 5) to reduce the overall complexity, in particular investigating adjacencies beyond the immediate neighborhood of the selected vertex v , in the form of cycle detection. The kernelization rules are those introduced by Buss and Balasubramanian, et al. The algorithm has a complexity of $O(kn + 1.29175^k)$.

The algorithm of Stege and Fellows [174] extends the kernelization rules described in [66] where edges are added to the graph. As in the algorithm in [66], the interleaving of kernelization rules is a key component of the tree-search phase in that it guarantees the invariant $4 \leq \deg(v) < k$, $\forall v \in E'$. The algorithm has a complexity of $O(kn + \max\{1.25542^k k^2, 1.2906^k 2.5k\})$. The first expression results from the possibility that vertices have had their degree increased significantly by the edge-adding process.

Chen, Kanj, and Jia [43] develop the first kernelization algorithm to produce a linear kernel. The basis of this result comes from a theorem of Nemhauser and Trotter [140]. They also introduce a systematic approach to processing vertices of degree 2, called *vertex folding*, that reduces k by one yet incurs no branching cost. Basing their work on these results, they develop algorithms with the following complexities: for graphs where the vertex degree is bounded 3, $O(kn + 1.273^k)$; for graphs where the vertex degree is bounded by 4, $O(kn + 1.277^k)$; finally, for graphs of arbitrary degree, $O(kn + 1.2852^k)$.

Chandran and Grandoni [37] introduced an algorithm that requires exponential space with time complexity $O(kn + 1.2745^k k^4)$, that uses a refinement of the memorization technique of Robson [159]. Memorization (similar in concept to *memoization*) was used by Robson [159] in his landmark paper on solving the MAXIMUM INDEPENDENT SET problem. The main idea is to precompute solutions for all sufficiently small induced subgraphs derived from the instance, and store these in a lookup table. During the exhaustive tree search phase of the algorithm, when a subproblem instance size drops below some threshold, the precomputed solution can be looked up in polynomial time. This approach has since been adopted as a general technique to improve algorithms by trading off time complexity for space complexity. Recently, this technique has been applied with success to improve the complexity of fixed-parameter tractable algorithms, notably those solving the k -VERTEX COVER problem [37].

Finally, the current best polynomial-space result is due to Chen, Kanj, and Xia

Algorithm	Degree bound 3	Degree bound 4	General
Robson [159]	—	—	$O(1.224^n)$
Chen, et al. [43]	$O(1.174^n)$	$O(1.201^n)$	—
Beigel [21]	$O(1.125^n)$	$O(1.171^n)$	$O(1.223^n)$
Robson [160]	—	—	$O(1.1892^n)$
Fürer [81]	—	—	$O(\text{poly}(n)1.1120^n)$

Table 2.2: Complexities for recent maximum independent set algorithms.

[45] based on a generalization of their folding technique, and has time complexity $O(kn + 1.2738^k)$.

2.4.5 Algorithms for bipartite graphs

While most research has focused primarily on k -VERTEX COVER problem in general graphs, an important variation is the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem: given a bipartite graph, $G = (U \cup L, E)$, and integers k_U, k_L , we must decide if there exists a minimum vertex cover for G with at most k_U vertices from U and at most k_L vertices from U_L . This problem has many applications, including fault-tolerant reconfigurable arrays [99].

Fernau and Niedermeier [74] introduced the first FPT algorithm for the case where the cover is not necessarily minimum, with time complexity $O(1.3999^{k_U+k_L} + (k_U + k_L)n)$. Chen and Kanj [44] provide an elegant solution for the case where the cover is assumed to minimum. They use two well-known results from graph theory, the Gallai-Edmonds Structure Theorem and Dulmage-Mendelsohn decomposition, to develop an algorithm with time complexity $O(1.26^{k_U+k_L} + (k_U + k_L)n)$.

2.4.6 Maximum independent set algorithms

Because of its close relationship to MINIMUM VERTEX COVER we include a brief review of contributions for solving the MAXIMUM INDEPENDENT SET problem.

Tarjan and Trojanowski [180] introduced the first algorithm for finding a maximum independent set in a general graph with complexity $O(2^{n/3})$. For several years the best result for solving the MAXIMUM INDEPENDENT SET problem was due to

Robson [159], with complexity $O(2^{0.292n}) \approx O(1.224^n)$, based on his technique of memorization and requiring exponential space. Using their k -vertex cover algorithm for graphs of degree bounded, Chen, et al. [43] describe algorithms that find a maximum independent set with complexity $O(1.174^n)$ in graphs of degree bounded by 3, and $O(1.201^n)$ in graphs of degree bounded by 4. Beigel presented algorithms in [21] with running times $O(1.125^n)$, $O(1.171^n)$, and $O(1.223^n)$, for graphs of degree bounded by 3, graphs of degree bounded by 4, and general graphs, respectively. Robson [160] announced an algorithm with complexity $O(1.1892^n)$ for general graphs. Very recently, Fürer [81] published an algorithm for finding maximum independent sets in graphs of maximum degree 3, with complexity $O(\text{poly}(n)1.1120^n)$. These complexities are summarized in Table 2.2.

2.5 Graph partitioning and fixed-parameter algorithms

Researchers have attempted to incorporate various concepts from graph separation as a means of improving the complexity of FPT graph algorithms, and recently a great deal of work has focused on developing FPT algorithms for the PLANAR DOMINATING SET problem. For general graphs, k -DOMINATING SET is in $W[2]$, although there exist FPT algorithms for this problem when the input is restricted to planar graphs. We note that algorithms for solving the PLANAR DOMINATING SET problem can easily be extended to solve the k -VERTEX COVER problem in planar graphs.

2.5.1 Preliminary definitions

Let $G = (V, E)$ be an undirected graph with n vertices and m edges. A *subgraph* of G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A subgraph H is an *induced subgraph* of G if $E(H)$ contains all edges of G that connect vertices in $V(H)$.

Let S be class of graphs closed under the subgraph relation and let $f(n)$ be a non-negative function. An $f(n)$ -separator theorem for S is defined as a theorem of the following form:

Theorem 1 [126] *There exists constants $\alpha < 1$ and $\beta > 0$ such that if G is any n -vertex graph in S then the vertices of G can be divided into three sets, A, B, C*

such that no edge joins a vertex in A with a vertex in B , $\max(|A|, |B|) \leq \alpha n$ and $|C| \leq \beta f(n)$. The vertex set C is called an α -separator.

There exist separator theorems for many classes of graphs, e.g., planar graphs [126], graphs of bounded genus [84], chordal graphs [85], graphs with an excluded minor [11]. An $o(n)$ -separator theorem can be used to design divide-and-conquer algorithms for solving graph problems, if such a theorem exists for that class of graphs. For example, this approach have been successfully incorporated into algorithms that solve VLSI layout problems [125], finding approximate solutions to NP -complete problems [14], Gaussian elimination [127], graph embedding [128].

2.5.2 Separators and approximation

The first use of separators for approximating a solution to an NP -complete problem comes from Lipton and Tarjan [126] who use a planar separator theorem to obtain an approximation scheme for MAXIMUM INDEPENDENT SET. Their algorithm can find an independent set of size at least $(1 - O(1/\sqrt{f(n)}))$ times the maximum independent set size, in $O(n \max\{\log n, 2^{f(n)}\})$ time.

The central idea behind the result of Baker [14] is to decompose a planar graph into subgraphs that are r -outerplanar. We require the following definitions:

Definition 11 [14] *A planar embedding of a graph G is called outerplanar if all vertices lie on the boundary of the outer face of G . A graph G is called outerplanar if it admits an outerplanar embedding in the plane.*

Definition 12 [14] *A planar embedding of a graph G is called r -outerplanar if for $r = 1$, the embedding is outerplanar and, for $r > 1$, inductively removing all vertices on the boundary of the outer face and their incident edges, the embedding of the remaining subgraph is $(r - 1)$ -outerplanar. The outerplanarity number of a graph G is the minimum r such that G is r -outerplanar.*

Every planar graph is r -outerplanar for some r , and given a planar graph $G = (V, E)$ with n vertices, a r -outerplanar embedding for G with minimal r can be found in time polynomial in n [22]. An r -outerplanar graph can easily be decomposed into

two subgraphs with $2r$ common boundary vertices, i.e., separator vertices that are included in both subgraphs. Baker's strategy consists of recursively decomposing the graph into subgraphs, and applying a dynamic programming approach to the decomposition. For each subgraph, H , a table with 2^{2r} entries is used to keep track of the maximum independent sets for each combination of boundary vertices in the set. For a planar graph with n vertices, the algorithm can find an independent set of size at least $r/(r+1)$ times the optimal independent set size in $O(8^r rn)$ time.

2.5.3 Separators and FPT algorithms

Subexponential FPT algorithms for solving the VERTEX COVER problem in planar graphs are generally based on the notion of tree decomposition, as introduced by Robertson and Seymour:

Definition 13 [156] *Let $G = (V, E)$ be a graph. A tree decomposition of G is a pair $\langle \{X_i | i \in I\}, T \rangle$, where each $X_i \subseteq V$, is called a bag, and T is a tree with the elements of I as nodes.*

1. $\cup_{i \in I} X_i = V$;
2. for every edge $(u, v) \in E$ there is some node $i \in I$ such that $\{u, v\} \subseteq X_i$;
3. for all nodes $i, j, k \in I$, if j lies on the path between i and k in T then $X_i \cap X_k \subseteq X_j$.

The width of $\langle \{X_i | i \in I\}, T \rangle$ is $\max\{|X_i| | i \in I\} - 1$. The treewidth of G is the minimum k such that G has a tree decomposition of width k .

A tree decomposition of a graph attempts to identify an underlying tree-like structure of a given graph, and efficient algorithms can be developed that exploit this structure. The classes of graphs with bounded treewidth are important as many NP-complete problems become solvable in polynomial time for these classes.

Alber, Bodlaender, Fernau, Niedermeier [6, 7, 8] extend Baker's approach of exploiting the outerplanarity property of planar graphs to develop a subexponential algorithm for the DOMINATING SET problem. The results can easily be extended

to the VERTEX COVER problem. Alber, et al. prove the following important results that are used in subsequent subexponential algorithms.

Proposition 1 [6] *If a planar graph, $G = (V, E)$, has a k -dominating set then it is at most $3k$ -outerplanar.*

Proposition 2 [6] *An r -outerplanar graph has treewidth of at most $3r - 1$.*

Theorem 2 [6] *If a tree decomposition of width at most ω of a graph is known then a minimum vertex cover can be determined in time $2^\omega n$ where n is the number of nodes of the tree decomposition.*

The authors show that a graph with a k -vertex cover has treewidth $O(\sqrt{k})$. Combining this with Theorem 2 yields a subexponential complexity for the algorithm. This leads to the following result:

Theorem 3 [6] *Let k denote an upper bound on the size of a minimum vertex cover of a planar graph G . The treewidth of G is at most $4\sqrt{3k} + 5$ and the corresponding tree decomposition can be constructed in $O(\sqrt{kn})$. The MINIMUM VERTEX COVER problem can be solved on G in time $O(c^{\sqrt{k}}n)$ where $c = 2^{4\sqrt{3}}$.*

The idea behind the algorithm is to use separators to split the input graph in a layerwise manner and then employ a tree decomposition on each component to solve the MINIMUM DOMINATING SET problem exactly on each component. The local optimal solutions are stored in a table associated with each tree decomposition. A bottom-up dynamic programming approach is then used to merge the solutions of the local decompositions into a global solution. The algorithm has complexity $O(3^{6\sqrt{34k}}n)$. In a similar manner, the algorithm can easily be adapted to find a k -vertex cover in a planar graph requiring $O(2^{4\sqrt{3}\sqrt{k}}n)$ steps [6]. The algorithm proceeds as follows:

1. Decompose the input graph using the layerwise separation property defined in Baker [14], using separator sets $S_1, \dots, S_l$, into components $G_1, \dots, G_{l+1}$.
2. Construct tree decompositions for each component $G_1, \dots, G_{l+1}$.
3. Merge the small, local tree decompositions into a global decomposition for G using separator sets $S_1, \dots, S_l$.

The separator sets are bounded in size by $O(\sqrt{k})$ and the components are each of outerplanarity $O(\sqrt{k})$, so the treewidth of G_i is $O(\sqrt{k})$. The dynamic programming algorithm proceeds as follows:

1. For each bag, X_i , in T generate $2^{|X_i|}$ vertex cover candidate sets associated with the induced subgraph $G(X_i)$; store these sets in a table A_i .
2. Proceeding from the leaves of T to the root, compare tables A_i and A_j of two tree nodes connected by an edge $(i, j) \in E(T)$, determining the best vertex cover for the subgraph associated with those bags.

Before proceeding to the next set of results, we require more definitions from the work of Robertson and Seymour:

Definition 14 [156] *A graph H is a minor of a graph G if H can be obtained from a subgraph of G by contracting zero or more edges.*

Definition 15 [156] *A graph class $\mathcal{E}$ is a minor-closed class if the minor of any graph in $\mathcal{E}$ is also a member of $\mathcal{E}$. A minor-closed graph class $\mathcal{E}$ is H -minor-free if $H \notin \mathcal{E}$.*

Demaine, Hajiaghayi, and Thilikos [54] extend the result of Alber, et al. [8] to some non-planar graph classes. They prove that for a single-crossing graph H , the treewidth of any H -minor-free graph G having a k -dominating set is bounded by $O(\sqrt{k})$, leading to an algorithm with $O(3^{6\sqrt{34k}} n^{O(1)})$ complexity. The algorithm uses a graph summation operation to obtain local treewidth of *clique-sum* graphs [97]. The class of clique-sum graphs includes all planar graphs, as well as $K_{3,3}$ -minor-free and K_5 -minor-free graphs.

In contrast to previous work, Fomin and Thilikos [78, 79] base their algorithm on branchwidth rather than treewidth. Robertson and Seymour defined the notion of branchwidth as follows:

Definition 16 [157] *Let $G = (V, E)$ be a graph. A branch decomposition of G is a pair (T, τ) , where T is a tree with vertices of degree 1 or 3 and τ is a bijection from the set of leaves of T to $E(G)$. The order of an edge e in T is the number of vertices $v \in V(G)$ such that there are leaves t_1, t_2 in T in different components of $T(V(T), E(T) - e)$ with $\tau(t_1)$ and $\tau(t_2)$ both containing v as an endpoint.*

Definition 17 [157] *The width of (T, τ) is the maximum order over all edges of T , and the branchwidth of G , $bw(G)$, is the minimum width over all branch decompositions of G .*

Although treewidth and branchwidth both measure the global connectivity of a graph, branchwidth can be more suitable to actual implementations.

Fomin and Thilikos [78, 79] show that any n -vertex planar graph, G , has branchwidth $\leq 2.122\sqrt{n}$ and treewidth $\leq 3.182\sqrt{n}$. One immediate benefit is that optimal branch decompositions of a planar graph can be constructed in $O(n^4)$ time using an algorithm of Robertson and Thomas [158]. The dynamic programming techniques for tree decomposition translate in a straightforward manner to branch decompositions. Their result applies to generalizations of planar graphs, e.g., $K_{3,3}$ -minor-free or K_5 -minor-free graphs. Their algorithm for planar dominating set has complexity $O(2^{15.13\sqrt{k}}k + k^4 + n^3)$ and their algorithm for planar vertex cover has complexity $O(2^{4.5\sqrt{k}}k + k^4 + kn)$.

Finally, Demaine, Fomin, Hajiaghayi, and Thilikos [55] introduce a framework for designing fixed-parameter algorithms of complexity $O(2^{O(\sqrt{k})}n^{O(1)})$. The framework applies to classes of graphs excluding a fixed graph H as a minor. This general category of graphs includes planar graphs, bounded-genus graphs, single-crossing-minor-free graphs, and any class of graphs closed under the taking of minors. The authors extend the planar graph k -dominating set algorithm to bounded-genus graphs. Using dynamic programming and results from the Robertson-Seymour graph-minor theory as a basis, the algorithm repeatedly removes the handles from an input graph to reduce the bounded-genus graph to a planar graph.

2.5.4 Edge-separators and FPT algorithms

In the previous section we reviewed various approaches to graph separation and its application to fixed-parameter algorithm development. Most of the results were restricted to planar graphs or other constrained graph families. We now turn our attention to a more general problem of disconnecting an input graph into some fixed number of components by removing a subset of edges. Consider the following *NP*-complete problem:

GRAPH k -CUT

Instance: A graph, $G = (V, E)$, edge-weighting function, $w : E \rightarrow \mathbb{N}$, and positive integers k, m .

Question: Is there a set $C \subseteq E$, with $\sum_{e \in C} w(e) \leq m$, such that graph $G' = G - C$, obtained by removing the edges of the cut C , has at least k connected components?

The case where $k = 2$ is solvable using a maximum-flow/minimum-cut algorithm [50]. We briefly outline the hardness proof from [88] for arbitrary k , which reduces the MAXIMUM CLIQUE problem to the GRAPH k -CUT problem: the $(0, 1)$ -weighted GRAPH k -CUT problem is equivalent to finding a partition of V into k components such that the number of edges having both endpoints in the same component is maximum. If graph, $G = (V, E)$, has a clique of size $m = |V| - (k - 1)$ vertices, then any solution to the GRAPH k -CUT problem will exhibit such a clique as one of its k components. By cycling through the values $k = 2, \dots, |V|/2$, each time running the k -cut algorithm and checking if one of the components forms a clique, one could thus find a maximum clique in a graph in polynomial time.

A problem similar to GRAPH k -CUT is that of determining a separator set that partitions the graph into a given number of components. More specifically, given a graph, $G = (V, E)$, with weighted (or unweighted) vertices, the MINIMUM k -SEPARATOR problem consists of finding a set of vertices of minimum weight such that their removal will disconnect the graph into exactly k components. The complexity of this problem is open [88]. Even the more modest problem of finding a minimum cut in a undirected graph that separates exactly an arbitrary set of k vertices is *NP*-hard [71].

2.5.5 Parameterized hardness for GRAPH k -CUT

Given this result, one may ask whether the parameterized version of this problem is fixed-parameter tractable, that is where the number of components, k , is fixed. Downey, et al. [59] prove that the GRAPH k -CUT problem is hard for $W[1]$, using a reduction from k -CLIQUE. This proof establishes that determining a set of edges of

bounded weight, whose removal disconnects the graph into exactly $k + 1$ components, is at least as hard as finding a k -clique. They derive a new graph, $G' = (V', E')$ from the original graph, $G = (V, E)$, by augmenting G' with new edges in such a manner that k of the $k + 1$ components correspond to the k -clique elements in G .

Consider the more modest problem of disconnecting a fixed-sized piece of the graph:

CUTTING k VERTICES FROM A GRAPH

Instance: A graph, $G = (V, E)$, an edge-weighting function, $w : E \rightarrow \mathbb{N}$, and positive integers, k, m .

Parameter: k .

Question: Is there a set of k vertices $V' \subseteq V$, such that $\sum_{(u,v) \in E: u \in V', v \in V - V'} w((u, v)) \leq m$?

In [59] the authors prove that even cutting a fixed number of vertices from a graph is hard for $W[1]$, using a simplified version of the k -CLIQUE reduction for the GRAPH k -CUT hardness proof. Finally, we mention that Marx [134] shows several variants of separation and cutting problem to be tractable or intractable, depending on their parametric formulation.

Chapter 3

Parallel k -VERTEX COVER

Help thy brother's boat across, and lo! Thine own has reached the shore.

-Proverb

In this chapter we describe a parallel framework for FPT k -vertex cover algorithms, and introduce the first coarse-grained FPT parallel algorithm that solves the k -VERTEX COVER problem. In experiments, our implementation demonstrated excellent speedup and solved the largest instances of k -vertex cover on general graphs (at the time of publication). Stemming from our work, recent advances in parallel FPT have focused on improving kernelization for preprocessing the graph prior to the parallel search phase. Subsequently Abu-Khzam, et al. [1, 2] have been able to solve problems larger than our initial experiments, clearly demonstrating in an experimental setting the advantage of reducing the initial graph instance to one of linear size. The research in this chapter has been published in *Journal of Computer and System Sciences*, Vol.67, 2003.

3.1 Background

3.1.1 Multiple sequence alignment and the VERTEX COVER problem

We now provide a quick overview highlighting the importance of k -VERTEX COVER to computational biology, specifically multiple sequence alignment (cf.([186, 94, 169])). We do so to motivate the use of our parallel algorithm, and to illustrate the means by which we generated some experimental data sets.

Sequence alignments are a very useful computational tool because they can be used to infer evolutionary relationships among genes and proteins. Consider a sequence of symbols encoded over a finite alphabet, where each symbol in the sequence represents

a constituent amino acid in that protein. Finding an alignment requires determining a correspondence between identical substrings in the two sequences, while respecting the relative order of the amino acids in each sequence. The quality of an alignment is determined based on a scoring function that measures how closely the two sequences are matched. In the case of multiple alignment, the requirement is to find the best overall alignment given a set of several sequences.

Proteins that are closely related have more similar amino acid sequences for an orthologous protein than more distantly related proteins. A typical biological experiment involves a large set of amino acid sequences for an orthologous protein from distantly related organisms which are suspected to have a common ancestor. The task is to remove a minimum number of sequences from the set that contradict the common ancestor hypothesis.

The sequences in a data set can be aligned using a hierarchical multiple alignment program that generates pairwise alignments for all of the input sequences, and then ranks the scores of the alignments (e.g., the ClustalW package [183]). In some cases, multiple alignments achieve a poor overall score due to conflicts between two or more sequences. One way to resolve such conflicts is to exclude some sequences from the sample. Define a *conflict graph* as a graph where every sequence is a vertex and every edge is a conflict between two sequences. A conflict may be defined when the pairwise alignment of these two sequences has a score below a given threshold. The goal is to remove the fewest possible sequences that will eliminate all conflicts, which is equivalent to finding a minimum vertex cover in the graph [175].

3.1.2 Parallel models of computation

In recent years, practical parallel algorithms mainly assume one of two computing models, either the *Coarse-Grained Multicomputer* (CGM) [52], or the *Bulk-Synchronous Parallel Model* (BSP) [184], eschewing the *Parallel Random Access Model* (PRAM) [104] because of its less-than-realistic characteristics.

As our focus is on parallel computation and fixed-parameter algorithms for k -VERTEX COVER, we describe the model which will be the basis of our algorithms in

this work. The *Coarse-Grained Multicomputer* model, or CGM, consists of p processors with $O(n/p)$ local memory per processor, connected with an arbitrary network [52]. It is generally assumed that $n/p \geq p^\epsilon$, for some $\epsilon > 0$. A CGM algorithm consists of local computation steps separated by global communication steps. Each communication round consists of routing a single h -relation where $h = O(n/p)$, i.e., each processor sends $O(n/p)$ data and receives $O(n/p)$ data. The running time of a CGM algorithm is the sum of the times for each round of computation and communication.

The ancestor of the CGM model is the *Bulk-Synchronous Parallel* model [184]. A BSP computer is a collection of processor/memory modules connected by a router capable of delivering messages point-to-point between processors. A BSP computation consists of a sequence of supersteps separated by barrier synchronization. During a computational superstep, the processors perform local computations on data resident in their memory. During a communication superstep, data is exchanged among the processors. A BSP computer uses four parameters to model its performance: N , the problem size; p , the number of processors; L , the minimum time between synchronization steps; g , the ratio of the system computational capacity (number of computational operations) per unit time divided by the overall system communication capacity (number of messages of unit size that can be delivered the router) per unit time. An h -relation is performed in a superstep when each processor sends or receives at most h fixed-size messages. The time of a superstep in which an h -relation is performed is $\max(gh + w, L)$, where w is the time required for the local computations during the superstep. The overall running time of the algorithm is the sum of the running times for each of its supersteps.

A communication round in the CGM model is equivalent to a superstep in the BSP model with $h = gn/p$. A computation round in the CGM model is equivalent to a computation superstep in the BSP model.

3.1.3 Search techniques and optimization

In this section, we will summarize relevant search techniques that have been applied to solve discrete optimization problems, many of which belong to the class of *NP*-hard problems. This area has been extensively studied, for example see [89, 123].

It is often the case that a discrete optimization problem can be recast into the problem of finding a path in a graph, beginning at an initial node and ending at a goal node. This state-space graph can also be interpreted as a computation tree and we will use the terms tree and state-space graph interchangeably. Depending on the problem, either any solution path in the state-space graph is satisfactory, or there is a cost associated with each edge in the path and the objective is to determine the minimum solution. Two commonly used state-space search techniques are Depth-First Search and Best-First Search.

Depth-first search

In Depth-First Search, or *DFS*, the algorithm begins at an initial state, from which it generates a set of successor states. A successor state is selected and it in turn is expanded, with the process repeated recursively until no possible expansion is possible. The algorithm reaches a terminal state when it can no longer expand the current state. If this state is a goal state then the algorithm has determined a solution. If this state is not a goal state then one of its siblings is selected for expansion (backtracking). In Directed-DFS, or Ordered Backtracking, the algorithm imposes some order on the successor states. The previous DFS techniques use heuristic information that is local to the successors being expanded. If global information is available regarding the best solution found so far, then this can be used to prune branches from the search tree, i.e., preempt the search of a subtree rooted at the successor node in question.

DFS has long been considered well-suited to large-scale parallel implementation [89, 121, 122]. A notable characteristic is the low communication overhead: a subtree rooted at a given node can be searched in a manner independent of its sibling nodes. In the case of simple DFS no communication, aside from termination detection, is required at all.

Depth-first branch-and-bound search

The Depth-First Branch-and-Bound technique, or *DFBB*, uses a heuristic function to determine whether the cost associated with a given successor node exceeds that of the best solution found so far. If it does so, then the subtree rooted at that node is not

searched. The Depth-Bounded DFS technique uses a predetermined cost or depth to limit exploration of the search space. If an exhaustive search of this subspace yields no solution, then the limit is revised and the search continues.

Best-first search

As the name implies, Best-First Search, or *BFS*, dynamically arranges the successor nodes in an order predicated by a heuristic function. BFS algorithms expand the most promising node currently available on the search frontier. This is in marked contrast to DFS where the subtree rooted at a given successor node is fully explored before any of its siblings are.

Hybrid search algorithms

With the A^* algorithm, also known as Best-First Branch-and-Bound, or *BFBB*, a successor node, i , is selected for expansion using a heuristic evaluation function, $f(i) = g(i) + h(i)$, where $g(i)$ is the cost of the path from the initial state to node i , and $h(i)$ is the estimated cost to the nearest goal state. The Iterative-Deepening A^* technique, or *IDA**, performs a BFBB search with a cost-bound function used to limit the scope of the search. For each iteration the bound is increased to the minimum value that exceeds the previous bound.

While DFS terminates once a first solution is found, DFBB, A^* , and *IDA** traverse as much as is necessary of the search space to return the optimal solution.

Workload balancing

As a static assignment of subtrees to processors can result in processors becoming idle during execution, many researchers have investigated dynamic load-balancing schemes [76, 150, 124]. Equally important are the work-splitting strategies whereby a search space is divided amongst the processors [106, 112, 124, 155, 154].

In [154], a work-splitting strategy is described called *Search-Frontier Splitting*. Assuming p processors, the strategy operates in two phases as follows:

1. Task-splitting phase: nodes on the search frontier are generated in a distributed manner, with at least p nodes being generated with identical cost values (in

practice, the number of frontier nodes may exceed some constant factor of p). These nodes represent the roots of subtrees to be searched in the next phase.

2. Asynchronous search phase: using DFS or DFBB, each processor expands the nodes for which it is responsible. If a processor becomes idle, it requests work from an active processor: this can be accomplished either through request-broadcasting, or a dedicated master processor to schedule jobs. When a solution is found it is broadcast to all other processors.

It is reported in [154] that load-balancing is seldom an issue when using search-frontier splitting, as the nodes from the initial distribution phase are sufficient to keep the processors active.

The phenomenon of superlinear speedup has been widely reported (e.g., [135, 136, 100, 138]), and many researchers have derived analytical models to predict performance of search algorithms (e.g., [32, 106, 112]). In [151], Rao and Kumar investigated two approaches to depth-first search for discrete optimization problems: (i) simple backtracking, (ii) heuristic backtracking. They conclude from their analysis that a simple backtracking approach yields at least linear speedup on average, contingent on a nonuniform distribution of solutions in the search space. This conclusion derives from the fact that, assuming a nonuniform distribution, i.e., that only a small number of subtrees contain (possibly many) solutions, searching several subtrees simultaneously is more effective. In contrast, given a uniform distribution of solutions in the search tree, a sequential algorithm has a higher probability of selecting and searching a subtree in which a solution can be found.

3.2 Relationship between processors and parameter in FPT algorithms

It is interesting to conjecture how the number of processors influences the effectiveness of tree search-based fixed-parameter algorithms, such as those for k -VERTEX COVER. We will see that, although there is no possibility of reducing the exponential base of the complexity of the algorithm using practical parallelism, an improvement is reflected in the overall klam^1 rating for a given problem. By practical parallelism

¹See Chapter 2, Section 2.3.3.

one understands this to mean a model where the number of processors available, p , is independent of the size of the input. This is in contrast to theoretical models, such as Parallel Random Access Memory models (PRAM), that assume that the number of processors can scale as a polynomial function of the input size.

We first argue that, given a sequential tree search algorithm for k -VERTEX COVER, the effect of using p processors to solve the problem can be at most an asymptotically linear speedup. Let $T_{\text{seq}}(k) \in O(c^k)$, denote the complexity of the current best sequential tree search algorithm for k -VERTEX COVER, where $1 < c < 2$ is a constant. A tree search algorithm takes as input a kernelized instance, $\langle G, k \rangle$, from which it generates subproblems, $\langle G_1, k_1 \rangle, \langle G_2, k_2 \rangle, \dots, \langle G_i, k_i \rangle$, where $k_1, k_2, \dots, k_i \leq k$, and all subproblems are bounded in size by some function $f(k)$. In what follows, we assume a parallel platform with p processors, denoted $P_0, P_1, \dots, P_{p-1}$, and some arbitrary interconnection mechanism. An ideal parallel approach to solve the problem would be to map the individual subproblems to distinct processors.

Observation 1 *A processor, P_i , $0 \leq i \leq p - 1$, has the capability for supporting overloading, that is to say a physical processor can simulate multiple processors implemented as processes.*

For small c , a processor P_i is capable of simulating up to c virtual processors with negligible slowdown in its own local runtime, with respect to the overall computation.

Observation 2 *Consider the subproblem allocation scheme from the parallel approach described above. To ensure that no processor remains idle upon completion of its initial subtree search, we can assume each processor maintains a small queue of pending subproblem instances that can be transferred to idle processors in $O(f(k))$ time.*

This implies all processors can be kept active during the tree search. If the sequential tree search has complexity $O(T_{\text{seq}}(k))$, then given p processors, the same problem can be solved in $O(T_{\text{seq}}(k)/p)$ time. From Observation 1, we can bound the number of processors p by $c^{i-1} < p \leq c^i$, for some $0 < i \leq k$. From Observation 2, we know the processors are never idle and so their work is optimal. Thus, $T_{\text{seq}}(k)/p = c^k/c^i \in O(c^{k-\log_c p})$. In an asymptotic sense, for tree-based search algorithms such

as k -VERTEX COVER, the number of processors influences the klam rating of the problem as a linear function.

This exposition is meant to underline two important facts regarding parallel computation and tree search-based algorithms. Firstly, to a certain extent the asymptotic complexity of a sequential algorithm, i.e., the exponential base, becomes less important when viewed in terms of the impact of parallel implementation of the algorithm. Certainly within the realm of k -VERTEX COVER the relative complexities between algorithms is secondary to the number of processors available. Secondly, the argument underlines the need for experimentation. Although we have defined asymptotic limits, instances in a practical setting can exhibit surprising performance results. Asymptotic analysis is not always the definitive measure of the performance of an algorithm.

3.3 Framework overview

The parallel FPT framework consists of two phases, corresponding to kernelization and bounded-tree search. We assume a platform of p processors, $P_0, P_1, \dots, P_{p-1}$. In the first phase, each processor locally performs kernelization on a copy of the problem instance. In the second phase, the processors select a common starting point (e.g., a specific vertex) and then each deterministically executes a series of $O(\log p)$ branching rules producing a unique problem instance. These subproblems represent the leaves of the corresponding breadth-first search tree of depth $O(\log p)$. Finally, each processor performs a randomized depth-first search using its unique problem instance. When a processor finds a solution all other processors are notified to stop. If a processor exhausts its search space and determines no solution then it halts.

We note that any improvements to the sequential algorithm, either in the kernelization or search tree phases, results in an improved parallel algorithm. The load distribution phase that assigns the subproblems to the processors remains unchanged and the autonomous, randomized search phase would use the new sequential algorithm.

3.3.1 Parallel FPT k -vertex cover algorithm

The coarse-grained parallel k -vertex cover algorithm combines aspects of the two sequential FPT algorithms of Balasubramanian, et al. described in [15]. (See Appendix C.) The first algorithm of Balasubramanian, et al. combined Buss' kernelization technique with a 3-level, depth-first search strategy that produces a 3-ary search tree (referred to as *Theorem-1* in [15]). The second algorithm combined Buss' kernelization technique with case-based branching rules to determine a k -vertex cover (referred to as *Theorem-2* in [15]). We base our work on the algorithms of Balasubramanian, et al. because many of the branching rules are common to more recent algorithms (described in Chapter 2, Section 2.4), or ancestors thereof.

As discussed in Section 3.1.3, a crucial requirement of successful parallel combinatorial search is effective load-balancing. Extensive research has gone into developing algorithms that optimize distribution of subproblems should one or more of the processors become idle [123, 124]. Recall from Chapter 2, Section 2.3.3 that a *klam* value describes the amount of time required to solve a problem instance, as a function of the parameter, k . We propose to use the parameter as a metric for estimating the amount of time required of a processor to solve a given subproblem. During the first phase, our algorithm adapts *Theorem-1* branching rules to guarantee that subproblems assigned to the processors are approximately equal in size. We assume the initial graph instance, $\langle G, k \rangle$, has been kernelized to produce a new instance $\langle G', k' \rangle$, and let $VC \subseteq V$ be a partial cover G .

1. Randomly select a vertex, $v \in V'$.
2. Starting from v , perform a depth-first search traversing at most three edges.
3. Let (v, v_1, v_2, v_3) be the sequence of vertices visited by Step (2). We consider the following cases:
 - (a) if the sequence forms a chain $(v, \dots, v_i)$, and v_i has degree one, then process without branching;
 - (b) if the sequence forms a triangle (v, v_1, v_2) , then branch on either $\{v, v_1\} \cup VC$, or $\{v, v_2\} \cup VC$, or $\{v_1, v_2\} \cup VC$;

Figure 3.1: Search tree load-balance after $O(\log p)$ steps. The parameter values at each processor, prior to the start of the randomized search phase, are approximately equal.

- (c) otherwise branch on $\{v, v_2\} \cup VC$, or $\{v_1, v_3\} \cup VC$, or $\{v_1, v_2\} \cup VC$.

Each of the three possible branching alternatives uniformly reduce the parameter by two. See Figure 3.1. Because the size of the parameter roughly predicates the amount of work required to process the subproblem and the parameter values are large, the distribution of work is relatively equal when all processors begin their search phase.

During the second phase, each processor autonomously executes a randomized version of algorithm of *Theorem-2*. As with the kernelization phase, newer algorithmic developments can easily be integrated into this phase to provide overall improvement to the performance.

3.3.2 Algorithm description

We assume a CGM model comprising p processors, P_i , $0 \leq i \leq p - 1$, and $O(n/p)$ local memory. The parallel algorithm consists of the following steps:

1. All processors, P_i , $0 \leq i \leq p - 1$, together perform parallel kernelization on the problem instance $\langle G = (V, E), k \rangle$, and the resulting instance $\langle G', k' \rangle$ is then broadcast to all processors; let VC_{kern} be the set of vertices determined by the kernelization phase to be in the vertex cover set, VC .
2. Each processor, P_i , $0 \leq i \leq p - 1$, locally and deterministically executes the search tree phase of the *Theorem-1* algorithm on its instance of $\langle G', k' \rangle$ as

Figure 3.2: Search path traced by processor P_i using branching rules of *Theorem-1* and *Theorem-2*.

follows: P_i selects exactly the branching nodes that lead it to leaf i at depth $\log_3 p$ of the search tree; each processor now has a unique problem instance, $\langle G''_i, k''_i \rangle$.

3. Each processor, P_i , $0 \leq i \leq p - 1$, locally performs a fully-random depth-first search of the subtree rooted at leaf i starting with instance $\langle G''_i, k''_i \rangle$, searching for a vertex cover, VC_i in graph G''_i (see Figure 3.2).
4. When a processor, P_i , finds a solution it outputs the set $VC_{\text{kernel}} \cup VC_i$ and signals all other processors to terminate.

Step (2) represents the load-balancing phase described earlier. In the event that the number of available processors, p , is not a power of three we can expand the search tree to some level, j , $3^{j-1} < p < 3^j$, and perform a nearly-uniform distribution of subproblems across the processors. In the worst case, all but one processor will be assigned three subproblems. We will see in Section 3.4.3 that experimental evidence shows that in fact three virtual processors per physical processor is optimal in terms of the trade-off between processor overloading and search efficiency.

3.3.3 Asymptotic analysis

We now estimate the performance that can be expected from our parallel approach. Let Λ be the path described by the sequential algorithm traversing the search tree. The sequential processing time is determined by l_{seq} , the number of nodes in Λ which need to be traversed until the first solution is encountered. The parallel algorithm essentially sets p equally spaced starting points on Λ , and then starts p search processes, one at each starting point. Let Λ_i be the portion of Λ assigned to processor P_i , and let l_i be the number of nodes in Λ which processor P_i needs to traverse until it finds a first solution. The parallel time is determined by $l_{par} = \min_{0 \leq i \leq p-1} l_i$, i.e., the minimum number of nodes that a processor has to visit until it reaches a solution node. The possible relative speedup observed corresponds to the ratio (l_{seq}/l_{par}) .

A balls-in-bins model can be used to predict the relative speedup that could be expected for the search algorithm. Consider p processors and a path Λ of length L in which a sequential algorithm traverses the search tree, and assume that there are m solutions in the search tree which are randomly, uniformly distributed over the search path Λ . Consider an array of p rows and $n = L/p$ columns. The i th row corresponds to Λ_i and the entire array corresponds to Λ . We mark m random array elements as solutions and measure l_{seq} and $l_{par} = \min_{0 \leq i \leq p-1} l_i$.

The expected number of nodes in Λ that need to be traversed by the sequential algorithm is given by $E[l_{seq}] = \frac{L}{m+1}$. The expected number of nodes $l_{par} = \min_{1 \leq i \leq p} l_i$ that need to be traversed by the parallel algorithm is bounded by $E[l_{par}] \leq \frac{L/p}{m+1} + p$ [56]. Therefore, we obtain an expected relative speedup

$$E[s_p] \geq \frac{1}{\frac{1}{p} + \frac{m+1}{L/p}} \quad (3.1)$$

Equation 3.1 is only a lower bound on the expected relative speedup $E[s_p]$.

A near-linear relative speedup for low density m/L is consistent with the bound on $E[s_p]$ derived above. For $m \ll L/p$ the second part of the denominator becomes negligible and we get an expected relative speedup $E[s_p]$ of approximately p . It is important to note that the above lower bound on $E[s_p]$ is only a coarse lower bound. The actual relative speedup can be considerably better. Furthermore, as the discussion in [151] suggests, the uniform distribution of the m solutions over

the array examined above does not constitute a good scenario. When solutions are non-uniformly distributed, the processor whose search path starts close to a cluster has a high probability of finding a solution much faster than in the uniform case. Therefore it can be expected that the relative speedup observed is often better in the non-uniform case than in the uniform case.

3.4 Experimental results

We describe in this section the experimental results of our algorithms on both real-world data sets and benchmark data sets.

3.4.1 Methodology

For a given problem instance, the sequential baseline was chosen as the minimum of two sequential runtime measurements:

1. an implementation running purely sequentially on a single CPU (referred to as *Code-s*);
2. an implementation running as 3 virtual processors on a single CPU (referred to as *Code-p*).

All sequential times were measured as wall clock time in seconds. All parallel times were measured as the wall clock time between the start of the first process and the termination of the last process. All times include the time taken to read the input graph from a file and write the solution into a file. The speedup measured was relative to the minimum baseline sequential time (more will be said about this in Section 3.4.3).

Our experimental platform consisted of a 32 node cluster with 1.8GHz Intel Xeon processors, 512MB RAM per node and 60GB of disk storage per node. All nodes were interconnected via a Gigabit Ethernet switch.

3.4.2 Data sets

We used two different types of graphs for our experiments with the objective of studying the performance of the algorithm using both real-world and artificial data.

Data set	Threshold score	$ V $	$ E $	$k = VC $	k'
Somatostatin	10	559	33652	273	255
WW	10	425	40182	322	318
Kinase	16	647	113122	497	397
SH2 (src-homology domain 2)	10	730	95463	461	397
Thrombin	15	646	62731	413	413
PHD (pleckstrin homology domain)	10	670	147054	603	603

Table 3.1: Protein sequences and corresponding conflict graph sizes. The column k' denotes the parameter post kernelization.

Graph	$ V $	$ E $	$k = VC $	k'
Random	220	2155	122	122
Grid	289	544	145	145

Table 3.2: Random graph and grid graph data. The column k' denotes the parameter post kernelization.

Our biological data consisted of amino acid sequences collected from the National Center for Biotechnology Information database [139]. Several protein modules that comprise large families of sequences were chosen for alignments. The data sets selected are listed in Table 3.1. The sequences in each data set were aligned using ClustalW (see [40]), a hierarchical multiple alignment program. A conflict graph, i.e., the input for the minimum vertex cover problem, was created by modeling all sequences in the data set as vertices and inducing edges between sequences whose alignment had a score below a given threshold. The threshold values used are summarized in Table 3.1, together with the sizes of the corresponding conflict graphs and the values of k and the post-kernelized parameter, k' .

We also tested our algorithm on random graphs and grid graphs (see Table 3.2). We show results for one random graph and one grid graph which are typical for the results obtained in our experiments for these classes of graphs.

Figure 3.3: Average sequential wall clock times.

3.4.3 Sequential and parallel-simulation k -vertex cover algorithm performance

We executed the purely sequential code on a single processor of the parallel machine, and measured the wall clock time (Figure 3.3, first set of vertical bars). Each data point in Figure 3.3 represents the average of five experiments. We selected data sets Somatostatin, WW, PHD, Random and Grid (Tables 3.1 and 3.2) because they would complete on a single processor in a reasonable amount of time. We observe that the wall clock times for the code shown in Figure 3.3 do not strictly increase with k or k' , indicating that the structure of the graphs is clearly an important factor for the performance. We observed that the code that simulated several virtual processors on a single processor often run faster than the purely sequential code. The simulated parallel code, while exploring the search tree from multiple starting points, would often find a solution quicker than its sequential counterpart. For most of our data sets, as illustrated in Figure 3.4, the minimum average wall clock time is achieved by using three virtual processors (as before, each data point in Figure 3.4 represents the average of five experiments). We conclude that, exploring the search tree from multiple starting points as proposed in our parallel FPT method can also lead to

Figure 3.4: Code-p on one processor: average sequential wall clock time as a function of the number of simulated virtual processors.

Figure 3.5: Average parallel wall clock times.

improvements of sequential FPT methods. Figure 3.3 compares the average wall clock time for Code-s on one processor with the minimum average wall clock time for Code-p on one processor in simulation mode. In almost all cases we observe that Code-p simulated on one processor runs faster than Code-s on one processor for the same data set.

For the relative speedup measurements in the following Section 3.4.4, the sequential baseline (i.e., time for one processor) was therefore set to be the minimum of the wall clock times measured for Code-s and one processor simulations of Code-p.

Figure 3.6: Average relative speedup: (a) Somatostatin and (b) WW.

Figure 3.7: Average relative speedup: (a) random graph and (b) grid graph.

3.4.4 Parallel k -vertex cover algorithm performance

We executed our implementation on 27 processors of our parallel machine and measured the average parallel wall clock time. We chose this number to limit our physical processors to the largest power of three within the maximum number available in the cluster. This also facilitated running several time-consuming experiments simultaneously. Figure 3.5 shows the results for the Somatostatin, WW, Kinase, SH2, Thrombin and PHD data sets. Each data point represents the average of ten experiments. We observe that our parallel FPT method is able to solve problem instances of size $k \geq 400$ in less than 1.5 hours. We observe that the parallel wall clock times shown in Figure 3.5 do not strictly increase with either k or k' . For the data sets described, the structure of the graphs is clearly an important factor for the performance. Most interestingly, the PHD data set with $k = k' = 603$ can be solved in under ten minutes, on average.

We executed our code on 1, 3, 9, and 27 processors of our parallel machine and measured the average relative speedup with respect to parallel wall clock time. Figure 3.6 shows the results for the Somatostatin and WW data sets and Figure 3.7 shows the results for the random and grid graphs. Each data point represents the average of 20 experiments. For both cases in Figure 3.6, we observe that the average relative speedup does not grow monotonically with respect to the number of processors. For 27 processors, the average relative speedup is larger than 20. For a smaller number of processors we observed some deviation in the average relative speedup caused by considerable variations in individual running times. Depending on the structure of the graph and the vertices selected, events can occur where the search happens to find a solution nearly instantaneously.

For Figure 3.7, we observe that the average relative speedup grows monotonically in both cases. For the random graph data set in Figure 3.7(a), the slope of the average relative speedup curve is considerably lower. We observed the same effect for other random graphs. For the grid graph data set in Figure 3.7(b), we note that there exists exactly two solutions. As discussed in Section 3.3.3 the number of solutions in the search tree is also very important for the relative speedup. We conjecture that this is the reason why, in Figure 3.7(b), the slope of the average relative speedup curve is very close to linear.

3.5 Summary

In this chapter we studied the potential of parallelism when applied to the exponential-cost bounded-tree search phase of FPT algorithms. We introduced the first parallel CGM FPT algorithm that solves the k -VERTEX COVER problem and provided the first experimental examination of parallel FPT techniques. An implementation of the algorithm demonstrated excellent speedup, and solved problem instances with $k \geq 400$ in typically less than 1.5 hours, at the time of publication representing the largest reported solved instances.

Chapter 4

Parallel Memorization

No memory is ever alone; it's at the end of a trail of memories, a dozen trails that each have their own associations.

-Louis L'Amour

In this section we present the first investigation of the well-know memorization technique in a theoretical parallel setting. Memorization (similar in concept to *memoization*) was used by Robson [159] in his landmark paper on solving the MAXIMUM INDEPENDENT SET problem. The main idea is to store sufficiently small subgraphs and their solutions in a lookup table. In the course of the exhaustive tree search phase of the algorithm, when a subproblem instance is encountered whose size is below some threshold, the table can be consulted to see if the instance has already been solved once before: if it has, then the previously stored solution can be extracted in polynomial time; if the instance is not in the table then a solution is generated and stored for future reference. In this manner a solution for a given subproblem is only ever determined once. Recently, this technique has been adapted with some success to improve the complexity of fixed-parameter tractable algorithms, notably those solving the k -VERTEX COVER problem [144, 37].

In this section we adapt the standard parameterized parallel framework [38, 1] to solve the k -VERTEX COVER problem using memorization with precomputation (the approach used in [144]). Our technique makes efficient use of the processors for generating the lookup table, while reducing the space requirement of the tables. The new approach applies to any future standard k -vertex cover algorithm using memorization and precomputation. We describe a CGM-style algorithm for parallel memorization assuming a distributed lookup table.

4.1 Background

Robson [159, 160] introduced a technique similar to dynamic programming, called *memorization*, to efficiently solve the MAXIMUM INDEPENDENT SET problem. Given an n -vertex graph, $G = (V, E)$, the main idea is to maintain a lookup table, T , of small subgraphs of bounded size derived from G and their corresponding maximum independent sets. The algorithm uses a recursive search tree to find a maximum independent set $I \subseteq V$ in G , through a series of complex branching rules similar in structure to those derived for k -VERTEX COVER. For example, during the initial stages of the algorithm, $\forall v \in V$, where $\deg(v) > 8$, the algorithm creates two subproblems: (i) v is added to I and the recursion continues on subgraph $G(V \setminus N[v])$; (ii) v is assumed to not be in I and the recursion processes subgraph $G(V \setminus v)$. Let $G_i = (V_i, E_i)$ denote a subproblem where $|V_i| \leq \alpha n$ and $0 < \alpha < 1$ is a constant. When graph G_i is generated during recursion this subproblem is solved exactly to find some maximum independent set, I_i . The pair $\langle G_i, I_i \rangle$ is then stored in a lookup table. If instance G_i is generated later, as a subproblem in another branch of the recursion tree, then the maximum independent set I_i associated with G_i can be extracted in polynomial time. In this way subproblems of bounded size are solved exactly once. Let $T(n)$ be the size of a search tree generated by a recursive algorithm on a graph of order n . A value for α can be chosen that balances the number of entries in the table, i.e., the maximum number of small subgraphs of order at most αn with degree less than 8, and the initial search tree cost, $T(n - \alpha n)$. The algorithm has complexity $O(1.211^n)$ requiring exponential space [159]. This was recently improved to $O(1.1892^n)$ [160].

Memorization can be adapted to the k -VERTEX COVER problem, but only in the case where the underlying algorithm is *standard*, that is to say, uses vertex-deletion operations during the tree search. Algorithms that use vertex-folding, and as a result introduce new vertices during the tree search, cannot be used in conjunction with memorization (e.g., [43, 42, 45]).

In the case of the parameterized k -VERTEX COVER problem, the parameter can be used to bound the size of the subgraphs. Niedermeier and Rossmanith [144] show how to reduce the complexity of their k -VERTEX COVER algorithm, with running time

$O(kn + 1.29175^k k)$ using polynomial space, to one with complexity $O(kn + 1.2832^k k)$ and exponential space. Recall that the input to the tree search phase is a kernelized instance, $\langle G', k' \rangle$, where G' consists of at most $2k'$ vertices. The initial problem, $\langle G, k \rangle$, is a yes-instance if and only if G' has a cover of at most k' vertices.

In the first phase, the algorithm generates all induced subgraphs of G' of size up to some $s = \alpha k'$, and stores all precomputed solutions in the table, each of which has a cover size at most $\alpha k'/2$. In the second phase, the tree search proceeds as usual on instance $\langle G', k' \rangle$ with the following change: when the value of the parameter at some level i in the tree drops below the threshold, $k'_i \leq \alpha k'/2$, the solution corresponding to the subgraph is extracted in polynomial time from the lookup table. This contrasts with Robson's approach where the subgraph size is measured against the threshold.

Denote by $T(k')$ the size of a search tree of depth k' . Let $T(k' - \alpha k'/2)$ be the cost to generate a search tree to some intermediate depth, $k' - \alpha k'/2$, and let $T(\alpha k'/2) \cdot 2^{\binom{2k'}{\alpha k'}}$ denote the cost of building the table of precomputed solutions for induced subgraphs of size up to $\alpha k'$. The time complexity of the algorithm is bounded by,

$$T(k') \leq T(k' - \alpha k'/2) \cdot O(k) + O\left(T(\alpha k'/2) \cdot \binom{2k'}{\alpha k'}\right)$$

Choosing a value of $\alpha = 0.05245457$ minimizes the overall running time by equating of $T(k' - \alpha k'/2)$ and $T(\alpha k'/2) \cdot (4/\alpha^\alpha (2 - \alpha)^{2-\alpha})^{k'}$, to within constant factors, where the last term is a bound using Stirling's formula.

Substituting α into $T(k - \alpha k/2)$ and assuming the current best standard algorithm for k -VERTEX COVER, gives a running time $O^*(1.29175^{k - \alpha k/2})$, and produces an overall complexity of $O(kn + 1.2831^k k)$. This is an improvement over the polynomial-space algorithm of Chen, et al. [43] with complexity $O(kn + 1.2852^k)$.

The recently-published algorithm of Chandran and Grandoni [37] combines the idea of Robson whereby subproblems of bounded size are solved exactly once and stored for future lookup, with the parameter bounding approach of Niedermeier and Rossmanith. Like Robson, the initial branching in the tree consists of processing vertices of high degree, in this case degree greater than 6. They derive a careful strategy for branching on small components by deducing a lower bound on the size of the minimum vertex cover for each component. Further, they reduce the size of the

lookup table by only storing induced, connected subgraphs of order at most 6. The algorithm solves the k -VERTEX COVER problem with complexity $O(kn + 1.2759^k k^{1.5})$. An additional refinement can be made by observing that subgraphs have vertices of degree between 2 and 6 (kernelization will guarantee that no subgraph has vertices of degree 1), yielding a slightly better complexity of $O(kn + 1.2745^k k^4)$.

The approach of Niedermeier and Rossmanith can be seen to be effectively the same technique as that used by Robson, and Chandran and Grandoni, except the cost of initially processing a small subgraph instance is extracted from the complexity of the search tree phase, and done in a preprocessing step.

4.2 Memorization and the parallel framework

As we previously investigated parallel speedup on the search tree in Chapter 3, we turn our focus to efficient table-generation in a parallel setting. In what follows, we use as a basis the algorithm of Niedermeier and Rossmanith that relies on the precomputation of small subgraphs and corresponding solutions.

Parallel techniques for FPT algorithms have been investigated with notable success [38, 1]. These two frameworks differ in that Cheetham, et al. [38] uses sophisticated branching rules as a means of initially load-balancing the processors, whereas the approach of Abuk-Khzam, et al. [1] requires dynamic load-balancing during processing. We describe a simplified version of our parallel framework (Chapter 3, Section 3.3), similar to that used in [1, 16]. We assume there are p processors, labelled P_i , $0 \leq i \leq p - 1$, connected with some network mechanism. The processors expand in parallel from some kernelized root instance $\langle G', k' \rangle$, a search tree in breadth-first manner to some depth, j , until there are $O(p)$ leaves. To create a proper tree the processors, once at depth $O(\log p)$, must each have a unique subproblem and such that no subproblem is left unassigned. After $\log p$ branches, each processor P_i , begins a local search using the subproblem, $\langle G'_i, k'_i \rangle$, located at leaf i .

Adapting memorization to this framework in a straightforward manner would entail each processor, P_i , following a unique path to an intermediate depth, and then generating a local lookup table, T_i . See Figure 4.1. Assuming an initial input $\langle G = (V, E), k \rangle$, and α that has been determined *a priori*, we sketch the algorithm

Figure 4.1: Parallel branching and lookup table generation.

as follows:

1. In parallel, the processors calculate a kernelized graph instance, $\langle G' = (V', E'), k' \rangle$.
2. The processors build a search tree of depth $O(\log p)$, by branching (naïvely) on the vertices in V' , such that each processor, $P_0, P_1, \dots, P_{p-1}$, has a unique subproblem at a given leaf, i , $0 \leq i \leq p-1$.
3. Using the vertex set at leaf i , a processor P_i generates all induced, connected subgraphs up to some bounded size, $s = \alpha k$, and their corresponding minimum vertex covers, and stores them locally in table, T_i .
4. Each processor P_i , then processes its local subproblem as per a standard memorization algorithm, while referring locally to lookup table T_i .

4.2.1 Asymptotic analysis

Without loss of generality, let the table T_0 , generated at P_0 , be the largest lookup table generated over all processors, P_i , $0 \leq i \leq p-1$. Let t_0 be its size, with $T(t_0)$ being the time required to generate the table. This complexity includes the cost of generating all induced, connected subgraphs, and determining their minimum vertex covers. Because each processor generates its own lookup table locally, the algorithm has space complexity of size $O(p \cdot t_0)$ and time complexity $O(T(t_0))$.

4.3 Refined parallel memorization

We now describe a refinement to the lookup table generation phase that enables us to reduce its time and space complexity requirements. The refinement is based on a key observation that many subgraphs in the lookup table will be generated unnecessarily, i.e., they, in effect, will never be used later to lookup a solution. Our refinement proposes delaying the generation of the lookup table during the tree search.

4.3.1 Effective table generation

We begin with a simple illustration from which we derive a more general theorem. This initial exposition pertains to a sequential model, and in the next section we show how this leads to the improved parallel algorithm.

Consider the stage in the algorithm at which we generate the lookup table, T , containing all induced, connected subgraphs up to some bounded size, s , derived from a kernelized instance $\langle G' = (V', E'), k' \rangle$. Along with each subgraph entry we store an associated minimum cover. In what follows, let VC be a set of vertices representing the partial k -vertex cover at some step in the tree search.

Having generated T , we begin search tree branching as per the steps of some standard algorithm. Assume the first processing rule we apply is a naïve branching rule: for some vertex $v \in V'$, we generate two children in the search tree: (i) vertex $\{v\}$ is added to VC , generating subproblem $\langle G'' = G' \setminus \{v\}, k'' = k' - 1 \rangle$; (ii) vertices $N(v)$ are added to VC , generating subproblem $\langle G'' = G' \setminus (N[v]), k'' = k' - \deg(v) \rangle$.

We now consider all possible induced, connected subgraphs that can be derived from the remaining vertices in the subgraphs G'' located at these two children. We require three important lemmata:

Lemma 2 *Consider the subset of all subgraphs stored in T that contain at least one vertex from the set vertices $N[v]$. This subset of subgraphs will never be accessed by any lookup during subsequent steps of the algorithm.*

Proof. After the algorithm generates the lookup table T the search tree phase begins. The search tree is generated by removing combinations of vertices that are hypothesized to be in the vertex cover. After the first application of the branching

rule, the set of vertices $N[v]$ have been removed from G' . Any subsequent remaining subgraph derived from G' cannot contain these vertices, and so no future lookup will ever access any of the subgraphs containing those vertices. $\square$

Following the exposition in Section 4.1, let $s = \alpha k$, $0 < \alpha < 1/2$, so that the number of induced, connected subgraphs in table T is at most $O\left(\binom{2k}{\alpha k}\right)$. (In what follows, we will use either s or αk interchangeably when the context is clear.)

Lemma 3 *If we postpone the generation of table T by one level of the tree search, T will contain fewer induced, connected subgraphs, on the order of $O\left(\binom{2(k-1)}{\alpha(k-1)}\right)$.*

Proof. Follows directly from Lemma 2. $\square$

The next lemma describes the relationship described in Lemma 3 between the tables located at each child in the tree.

Lemma 4 *Let V' be a set of vertices, $v \in V'$, and let T_v represent a lookup table created by generating all induced, connected subgraphs from the set $V' \setminus \{v\}$. After branching on $\{v\}$, $N(v)$ as described above, the contents of the lookup table in the leftmost child, T_v , is a superset of the induced, connected subgraphs in the rightmost child, $T_{N(v)}$. This implies the following relationship:*

$$T \supset T_v \supset T_{N(v)}$$

Proof. Let $G' = (V', E')$ be a graph with $2k$ vertices and let $s > 2$. Given the set of vertices, V' , we can generate all induced, connected subgraphs of size at most s by enumerating all subsets of vertices and checking whether a given subset forms an induced, connected subgraph of G' , with time complexity:

$$\sum_{i=2}^s \binom{2k}{i} \in O\left(\binom{2k}{s}\right)$$

Consider two sets of vertices, $S_1 = V'$ and $S_2 = V' \setminus \{v\}$, and again consider enumerating all subsets of S_1 and of S_2 of sizes $2, 3, \dots, s$. The subsets generated by S_1 and S_2 will only differ in the subsets that contain vertex v .

If the subsets form an induced, connected subgraph of G , then clearly the subgraphs generated by sequences from S_1 form a superset of the subgraphs generated by the sequences in S_2 . $\square$

Lemmata 2, 3, 4 imply we can generate a smaller lookup table if we postpone its generation during the search tree phase. More importantly, although we branched and created two children at the root, we still need only generate one lookup table, and this table is in fact smaller than one that would have been built prior to the branching phase. The refinement can be extended in a more general way to take advantage of branching to a greater depth in the tree.

Lemma 5 *Let $G = (V, E)$ be a graph and let $S = (v_1, v_2, \dots, v_m)$ be a sequence of vertices drawn from V . Given an initial search tree, associate with each level, i , $1 \leq i \leq m$, a distinct branching rule that generates two children, $v_i \cup VC$ and $N(v_i) \cup VC$. We call the sequence S a branching sequence. The relationship between the lookup tables at each leaf of the search tree at depth m can be described as follows:*

$$\begin{aligned}
 T_{v_1, v_2, \dots, v_m} &\supset T_{v_1, v_2, \dots, v_{m-1}, N(v_m)} \\
 &\supset T_{v_1, v_2, \dots, N(v_{m-1}), v_m} \\
 &\supset T_{v_1, v_2, \dots, N(v_{m-1}), N(v_m)} \\
 &\vdots \\
 &\supset T_{N(v_1), N(v_2), \dots, N(v_{m-1}), N(v_m)}
 \end{aligned}$$

Proof. Follows from Lemmata 2, 3, 4. □

Lemma 5 argues that if you can find a branching sequence of vertices, the size of the lookup table can be reduced even further. These four lemmata indicate that it is possible to improve any k -vertex cover algorithm that uses memorization by determining a branching sequence, and branching on them appropriately at the beginning of the search tree phase. We can now state a more general theorem as follows:

Theorem 4 *Given a kernelized graph, $G' = (V', E')$, and a branching sequence S with m vertices, we can build a lookup table, T , of induced, connected subgraphs of size at most $\alpha(k - m)$, with corresponding minimum vertex covers, in time:*

$$O\left(T(\alpha(k - m)/2) \cdot \binom{2(k - m)}{\alpha(k - m)}\right)$$

where $T(\alpha(k - m)/2)$ is the time complexity to find a minimum cover in a subgraph of size at most $\alpha(k - m)$.

Figure 4.2: Improved parallel table generation.

Proof. Follows directly from Lemmata 2, 3, 4, 5. $\square$

Because we can reduce the time for generating the lookup table, the overall complexity can be reduced as well, in contrast to $O(T(\alpha k/2) \cdot \binom{2k}{\alpha k})$ of the original approach described in Section 4.2.

4.3.2 Improved parallel memorization

The refinement described in Section 4.3.1 can be used to develop an efficient parallel algorithm based on memorization. We assume p processors are available, and an initial instance $\langle G = (V, E), k \rangle$. We describe the algorithm as follows:

1. In parallel, processors, P_i , $0 \leq i \leq p-1$, calculate a kernelized graph instance, $\langle G' = (V', E'), k' \rangle$.
2. The processors calculate a branching sequence, $S = (v_1, v_2, \dots, v_m)$, drawn from V' .
3. The processors build a search tree of depth $O(\log p)$, by branching on the vertices in $S = (v_1, v_2, \dots, v_{\log_2 p})$, such that each processor, P_i , has a unique subproblem at a given leaf, i , $0 \leq i \leq p-1$:
 - (a) without loss of generality, assume the path from the root of the tree to processor P_0 consisted of applying the following rules: $v_1 \cup VC, v_2 \cup VC, \dots, v_m \cup VC$;

- (b) in parallel, the processors, P_i , $0 \leq i \leq p-1$, generate all induced, connected subgraphs with size up to some bound, s , and their minimum vertex covers, based on the vertex set of the instance at processor P_0 , and store these in a table, T_0 .
- 4. Each processor P_i , $0 \leq i \leq p-1$, then processes its local subproblem as per a standard memorization algorithm, referencing a copy of T_0 stored locally.

In Step (2), after $\log_2 p$ vertices have been added to the sequence, only vertices with degree > 6 are added to S . In Step (4), each processor begins its own local branching, using the remaining sequence vertices, $v_{\log_2(p+1)}, \dots, v_m$. This maximizes the number of vertices excluded from the table, while ensuring that local branching complexity is not affected by the choice of sequence vertices.

Asymptotic analysis

Without loss of generality, again let T_0 be the largest lookup table generated over all processors, P_i , $0 \leq i \leq p-1$, with size t_0 and let $T(t_0)$ be the time required to generate the table. The improved table generation algorithm has space complexity $O(t_0)$ and time complexity $O(T(t_0)/p)$.

This approach is an improvement to the straightforward parallel memorization algorithm described in Section 4.2 that would have each processor generate its own local lookup table after branching to a unique leaf.

Finding a branching sequence

A crucial step in the algorithm is the determination of the S of size $m \geq \log_2 p$, with which we initially build the search tree. It is not sufficient to branch on some arbitrary set, $S \subseteq V$, as this will not guarantee the property described in Lemma 5. Let $G = (V, E)$ be a graph for which we want to determine a branching sequence, S . We can determine S using the following algorithm.

1. While the set V is not empty
 - (a) select a vertex $v \in V$;

- (b) append v to sequence S ;
- (c) remove vertices $N[v]$ and adjacent edges from V .

2. Output branching sequence S .

Theorem 5 *Given an n -vertex graph, $G = (V, E)$, a branching sequence, S , can be found in time $O(|E|)$.*

Proof. Follows from the correctness of the algorithm. □

After $\log_2 p$ vertices have been added to S , Step (1)a can be restricted to the selection of vertices with degree greater than 6: these vertices can be the initial branching vertices at each processor during the local computation phase. In this way, the complexity of the local branching at each processor is unaffected and the generated table can be further reduced in size.

Generating induced, connected subgraphs in parallel

Step (3)b of the algorithm, wherein the induced, connected subgraphs are generated, can be performed efficiently in parallel. There has been extensive research in developing adaptive parallel algorithms for generating combinatorial objects, e.g., [5, 176, 177], where adaptive means the algorithm can run on a parallel model assuming some arbitrary number of processors. A set of combinatorial objects is subject to some total ordering. For example, a *lexicographic* ordering is defined as follows: let $A = (a_1, a_2, \dots, a_m)$ and $B = (b_1, b_2, \dots, b_m)$ be two objects. Object A lexicographically precedes B if and only if, for some $j \geq 1$, $a_i = b_i$ when $i < j$, and $a_j < b_j$. It is possible to define a ranking function, $rank(A)$, that, given an object A , returns the integer value of its rank in the total ordering. Similarly, we can define an unrank function, $unrank(j)$, which returns the j^{th} generated object.

Consider a platform with some number of processors, p , available, and let n_s be the number of instances to be generated. The processors can repeat the following steps for subsets of size $3, 4, 5, \dots, s$. We divide the task of generating the objects equally between the processors, so that processor P_i generates objects from interval $(i - 1)[n_s/p] + 1$ to $i[n_s/p]$. The last processor finishes the enumeration at the

object with rank n_s . Note that during the procedure, with the possible exception of initialization, the processors incur no communication cost.

A processor P_i can generate the subsets in its interval by iteratively using a successor function, and checking whether a given set of vertices indeed forms an induced, connected subgraph. If the set satisfies the criteria, the processor determines the minimum vertex cover for the corresponding subgraph, and both are stored locally.

4.4 Parallel CGM memorization

In this section we describe a CGM-style algorithm for managing the lookup table data structure, where we require the table, of size t_0 , to be distributed evenly across the processors.

4.4.1 Distributed lookup table

Recalling the definition from Chapter 3, Section 3.1.2, the CGM model comprises p processors, P_i , $0 \leq i \leq p - 1$, with local memory storage $n/p \geq p^\epsilon$, $\epsilon > 0$, and interconnected with an arbitrary communication fabric. The running time of a CGM algorithm is the sum of the times required for each computation and communication round. We assume a single copy of the lookup table T_0 is evenly distributed across the memory of the processors, thus requiring $O(t_0/p)$ memory per node.

In our distributed model, a table lookup is replaced with a query sent to a remote processor. The queries are stored in a queue and processed later during a synchronized bulk communication step. As the queries are independent within a given tree search, a processor can simply backtrack and continue searching instead of waiting for each query to be resolved. This also eliminates the possibility of a single processor being overwhelmed with query requests, and not being able to progress with its own tree search, by restricting query processing to a discrete phase rather than having the processors issue requests asynchronously. In what follows, we make the following assumptions:

- each processor has memory space sufficient to store a table segment of size $O(t_0/p)$, and the size of a search tree generated locally by a processor is bounded

by $O(t_0/p)$;

- the lookup table data structure has been created and indexed in lexicographic order as per the algorithm in Section 4.3.2, so that each processor can determine the table segment that would contain a subproblem solution;
- that initial branching has taken place, i.e., each processor, P_i , $0 \leq i \leq p-1$, has a unique subproblem instance, $\langle G'_i, k'_i \rangle$;
- each processor initially begins its local search from a randomly selected vertex in its own subproblem, $v_i \in V'_i$.

The algorithm consists of the following phases:

1. Each processor, P_i , $0 \leq i \leq p-1$, executes the standard memorization algorithm on its local subproblem instance $\langle G'_i, k'_i \rangle$. When a processor P_i requires a lookup table access it places a query request, q_i^j , in a local queue. Duplicate queries are ignored.
2. The processors execute a standard CGM parallel binary search to resolve the queries generated in Step (1).
3. Each processor P_i verifies a response r_i^j to a query q_i^j by combining the value r_i^j with the current partial cover for that subproblem to determine if a cover of size k'_i has been found. If a cover has been found, P_i notifies all other processors to terminate. Otherwise P_i continues processing its query responses.

Asymptotic analysis

Let $O(T(k) \cdot k)$ be the complexity for a sequential memorization algorithm for some input $\langle G, k \rangle$. In Step (1) of the CGM algorithm each processor P_i randomly selects a vertex from its unique subproblem $\langle G'_i, k'_i \rangle$ on which it initiates a randomized, depth-first search. The number of local tree-search computation operations in this step is bounded by $O(T(k)/p)$, generating at most $O(T(k)/p)$ queries. To resolve the queries in Step (2), we employ a standard CGM binary search technique (cf.([69])). A crucial property for solving a multiquery search problem is to ensure the queries

are evenly distributed across the processors. A standard technique is to replicate the table segment of some heavily-loaded processor across the other processors. In this way, the number of queries satisfied by the processors is approximately equal, at the expense of having each processor maintain two table segments in memory for the duration of the multisearch. CGM binary search can be accomplished using global sorting, local binary search, and broadcast, segmented broadcast and total exchange operations. Given an ordered sequence of n elements and a set of n queries, the search can be performed in time $O(\frac{n \log n}{p} + T_s(n, p))$, where $T_s(n, p) \in O(n/p)$ is the communication complexity of the sorting phase (cf.([37])). Finally, Step (3) has complexity $O(T(k)/p)$.

From the foregoing analysis, we derive the following theorem:

Theorem 6 *Let $O(T(k) \cdot k)$ be the sequential time for a memorization-based tree-search algorithm. We assume a CGM model comprising p processors, P_i , $0 \leq i \leq p-1$, with local memory storage $n/p \geq p^\epsilon$, $\epsilon > 0$, and interconnected with an arbitrary communication fabric. The CGM-based memorization algorithm has complexity $O(\frac{T(k) \cdot k}{p})$ using $O(1)$ communication rounds and $O(T(k)/p)$ -relations.*

4.5 Summary

We presented the first investigation of the well-known memorization technique in a theoretical parallel setting. Memorization was introduced by Robson [159] in his landmark paper on solving the MAXIMUM INDEPENDENT SET problem. This approach has since been adopted as a general technique to improve algorithms by trading off time complexity for space complexity. Recently, this technique has been applied with success to improve the complexity of fixed-parameter tractable algorithms, notably those solving the k -VERTEX COVER problem [144, 37].

We developed a general approach for using the memorization of precomputed subgraphs to solve the k -VERTEX COVER problem, based on the algorithm of Niedermeier and Rossmanith [144], posited in what has become the standard parameterized parallel framework [38, 1]. These two frameworks differ in that Cheetham, et al. [38] uses sophisticated branching rules as a means of initially load-balancing

the processors, whereas the approach of Abuk-Khzam, et al. [1] requires dynamic load-balancing during processing.

Our technique makes efficient use of the processors generating the lookup table, while at the same time requiring less space. The new approach applies to any future standard k -vertex cover algorithm using memorization and precomputation. We describe a CGM-style algorithm for parallel memorization assuming a distributed lookup table.

Chapter 5

k -VERTEX COVER and Graphs of Bounded Degree

Measure twice, cut once.

-Proverb

In this chapter we describe a new approach to improve algorithms for solving the k -VERTEX COVER problem. The algorithm uses graph cutting during the tree search phase as a basis for disconnecting and explicitly processing small portions of the residual graph. Our algorithm applies to graphs of small bounded degree, adapts existing k -vertex cover machinery, and incurs no additional complexity. Bounded degree graph instances are generated during tree search phases of the current best FPT k -vertex cover algorithms. Previous separator-based approaches were constrained to planar graphs, or graphs of bounded-genus, and required complex dynamic programming. We also investigate the applicability of our new algorithm to solving the MAXIMUM INDEPENDENT SET problem in graphs of small bounded degree.

5.1 Background

We recall from Section 2.4.2 two of the most recent and powerful kernelization algorithms. Kernelization using linear programming initially requires reformulating the VERTEX COVER problem as an integer programming problem, that in turn is simplified using linear programming [2, 3]. The latter can be recast as a network flow problem and solved efficiently using Dinic's algorithm with time complexity $O(m\sqrt{n})$ [58]. Kernelization through crown reduction requires finding an independent set, I , such that there exists a matching M on the edges connecting I and $H = N(I)$, and all the vertices in H are matched [47]. The subgraph induced by the vertices I and H is bipartite, and a maximum matching can be found by recasting the problem as a network flow instance, and again solved in $O(m\sqrt{n})$ [3].

There has been great success in deriving subexponential algorithms for planar graphs, and graphs from other restricted classes, using either planar separator theorems [6, 7, 8], or the bounded-treedwidth machinery of Robertson and Seymour [54, 55]. Unfortunately, these results cannot be carried over directly to general graphs, or even graphs of bounded degree. Further, unlike planar graphs or graphs of bounded genus, there exist no good separator theorems for graphs of bounded degree. Marx [134] showed that many graph separation problems are intractable even when they are reformulated with several parameters:

CUTTING l VERTICES: separate exactly l vertices from a graph by deleting at most k vertices. This problem is $W[1]$ -hard, even when parameterized on both l and k .

CUTTING INTO l COMPONENTS: generate at least l components by deleting k vertices. Again, this problem is $W[1]$ -hard even when parameterized on both l and k .

CUTTING l CONNECTED VERTICES: separate exactly l connected vertices by deleting at most k vertices. This problem is $W[1]$ -hard if only one of l or k is a parameter.

The graph cutting variants of the first two problems were already shown to be $W[1]$ -hard for parameter l by Downey, et al. [59].

In this chapter we differ slightly from the conventions of the thesis to this point, in that we adopt the notation of Woeginger [188], that ignores polynomial factors in the complexity. For example, the complexity $O(k^2 2^k)$ would be described as $O^*(2^k)$.

5.2 Improving k -vertex cover algorithms in graphs of small degree

Developing efficient algorithms for processing graphs of small bounded degree (e.g., 3 or 4) serve two important purposes: first, to provide algorithms of better complexity for input instances that are graphs of bounded degree, and second, as all k -vertex cover algorithms for general graph instances reduce the original graphs to ones of bounded degree, we will then have efficient subroutines that can be used during these phases of processing.

We describe a new algorithm for solving k -vertex cover based on recognizing the advantage of disconnecting a bounded-degree graph and explicitly processing components. The algorithm attempts to find small separators very quickly, by reusing the fast network-flow algorithm that is the basis of the linear kernelization routines. These separator vertices are then processed efficiently by recycling existing branching rules in a manner so as to disconnect the graph. There are notable characteristics to this approach:

- Unlike the separator-based approaches of Alber, et al. [6, 7, 8] and Demaine, et al. [54, 55] our algorithm is not restricted to planar or bounded-genus graphs, and does not require the components be approximately equal in size.
- Our algorithm also differs in that we are using separators merely as a means of inducing components in the subproblem, within as few steps as possible. We avoid the complex dynamic programming phase incurred by the algorithm of Alber, et al. and Demaine, et al. The space complexity is linear in the depth of the search tree, as the tables required for dynamic programming are not required.

As will be shown, this branching technique incurs no additional polynomial cost: the complexity of calculating the separator vertices is bounded by the complexity of the kernelization algorithms and so the general interleaving technique (cf.([143])) and its analytical implications still apply. Conventional k -vertex cover branching rules for general graphs are applied obviously, with emphasis on facilitating rule development and complexity analysis. We introduce the concept of *strategic branching* in which specific vertices are processed with the intent of affecting a particular state on the remaining graph after a certain number of steps. Consider a problem instance $\langle G' = (V', E'), k' \rangle$, where we wish to determine if G' has a vertex cover of size at most k' . The objective is to process a set of vertices, $S \subseteq V'$, $m = |S|$, such that after applying branching rules on each $s \in S$, the subproblems at the intermediate levels of the search tree consist of instances $\langle G''_i, k'_i - g(m) \rangle$, where the graph G''_i are the connected components of non-trivial size. In the next section we sketch how this set of vertices is found. Once the subproblem has been reduced to a graph consisting

Figure 5.1: Tree-search phase, contrasting the exploration cost in the original algorithm (node j) and the improved algorithm (node i : subproblem consists of a disconnected graph).

of two or more components, these components can be processed explicitly: as soon as the graph becomes disconnected, the smallest of the two components is solved exhaustively, wherein we determine k'' cover vertices. The remaining component is processed using the bound $k''' = k' - g(m) - k''$. (For example, see Figure 5.1.) We wish to ensure the branching rules used to process the separator are efficient as possible. The algorithm consists of the following steps:

1. Perform kernelization on the problem instance $\langle G = (V, E), k \rangle$ to generate kernelized instance $\langle G' = (V', E'), k' \rangle$.
2. The tree search comprises two distinct phases:
 - (a) generate a binary search tree by processing all vertices where $\deg(v) > 4$;
 - (b) let $\langle G'_i = (V'_i, E'_i), k'_i \rangle$ be an instance at some depth of the search tree, where $\forall v \in V'_i, \deg(v) \leq 4$:
 - i. determine a set of strategic vertices, $S_i \subseteq V'_i$;
 - ii. for each $s \in S_i$, apply the search tree branching rules to s until $S_i = \emptyset$; after this step, the remaining problem instance G'''_i will consist of components $C_1, \dots, C_j$, in increasing order of size;

- iii. find minimum cover in $C_1, \dots, C_{j-1}$; process C_j using bounded search.
3. If at any point in Step 2(a), $k'_i = 0$ and $E'_i \neq \emptyset$ (or accordingly, in Step 2(b) that $k''_i = 0$ and $E''_i \neq \emptyset$) then backtrack.

5.2.1 Determining strategic vertices

The separation strategy will focus on a distinct phase of the graph processing, that being when the graph consists of vertices such that $\forall v \in V', 2 \leq \deg(v) \leq 4$. As it can be expensive to compute separators, we propose to use the fast network flow algorithm crucial to the kernelization routines described in Section 5.1. Bui, Chaudhuri, Leighton, and Sipser [29] presented an algorithm that finds a minimum bisection in a random regular graph, G , under the assumption that the size of the bisector of G is small, by reformulating it as a network flow problem (cf.([162])). This approach was motivated by the fact that graphs that occur in practice rarely have a bisection width as large as the theoretical upper bound.

In our case, it is not necessary to achieve a split that generates a bisection in order to improve the algorithm complexity. Our algorithmic objective is to disconnect one (or more) small piece(s) of the bounded degree graph, using small separators. Due to the exponential nature of the algorithm, it is necessary to find separators that are sufficiently small for there to be improvement in the asymptotic complexity of the algorithm: we will assume the pieces contain $\Omega(\sqrt{2k})$ vertices and the separator has size $o(\sqrt{k})$, hence $O(\sqrt{k})$ cover vertices are required to process the separator. We note that the complexity is significantly improved if the smaller component, disconnected by the separator, is larger than $\sqrt{2k}$.

Minimum cuts and separators

We now summarize the technique described by Bui, et al. [29] for finding a minimum bisection in an unweighted graph with small bounded degree, based on a flow-network algorithm. The main idea is to convert the graph into a flow network and then use minimum (s, t) -cuts to determine a set of edges that represent the bisection of the input graph. Because finding a minimum bisection in a graph is *NP*-hard, the claim is that the algorithm finds such a cut most of the time, for a specific class of graphs

Figure 5.2: Converting a bounded degree graph into a flow network (for simplicity, edges represent opposing arcs): (a) graph with vertices s, t ; (b) corresponding flow network, with $p = 1$, $D(s, p) = \{a, b, c\}$, $D(t, p) = \{g, j, l\}$.

known to have a small bisection width. For example, consider a class of graphs, $G(n, d, b)$, that are n -node, d -regular graphs with a unique minimum bisector of size b . The main result of [29] states that, given $d \geq 3$ and $b(n) = o(n^{1-1/\lfloor d+1/2 \rfloor})$, the algorithm determines a unique minimum bisector for almost all graphs in $G(n, d, b)$, in polynomial time. It fails to find a minimum bisection when there is a minimum cut smaller than b that is not a bisection.

We can convert an undirected graph, $G = (V, E)$, into a flow network as follows: replace each edge, $(u, v) \in E$, with a pair of directed edges, (u, v) and (v, u) ; designate a source vertex s and a sink vertex t , $s, t \in V$; give each edge a symmetric capacity: for $u, v \in V$, $c(u, v) = c(v, u)$. If each edge is given unit capacity then a minimum (s, t) -cut algorithm will find the smallest set of edges that disconnects s from t . A graph where the degree is bounded by some small constant, d , will have a minimum cut that never exceeds some constant factor of d . To adapt a network flow algorithm to the separator problem requires modifying the input graph. We consolidate the source vertex, s , and the sink vertex, t , by contracting their respective neighborhoods to some distance, $p \geq 0$, and replacing them with two new vertices, s' and t' . Contracting a neighborhood around vertex s consists of a series of edge contractions where we replace vertex s with s' , and for all edges (s, x) , we delete vertex x and replace any edge (x, y) , $y \neq s$, with (s', y) . (Similarly for vertex t .) Let $D(s, p)$ represent the set of vertices collapsed into s' , the farthest vertex being at distance at most p from s . Given that $|D(s, p)| < d^{p+1}$, the number of edges connecting $D(s, p)$ with $V \setminus D(s, p)$ is at most d^{p+1} .

The capacities of edges connecting vertices in $D(s, p)$ and $V \setminus D(s, p)$ are determined as follows: let $v \notin D(s, p)$; an edge (s', v) is assigned capacity $|(u, v) \in E|$,

$\forall u \in D(s, p)$. (Similarly for t' and $D(t, p)$.) All other edges in the graph are assigned a unit value capacity. Finally, an edge (s', t') is added with capacity $|E|$, for all u, v where $u \in D(s, p)$ and $v \in D(t, p)$. For example, see Figure 5.2. We call the resulting graph *consolidated*.

The algorithm to find a minimum bisection consists of the following steps: for all pairs of vertices, $s, t \in V$, $s \neq t$, and s, t are least distance of $2p + 1$ apart, construct a consolidated graph, G' ; run a minimum (s, t) -cut on the graph G' , reconsolidate the graph, and extract the minimum cut. The algorithm as stated solves $O(k^2)$ maximum-flow minimum-cut problems and has polynomial complexity.

Improved cutting strategy

It is essential that the separator be sufficiently small, and that the small component(s) be of some nontrivial size. We propose the following algorithm as a means of reducing the complexity of finding a satisfactory minimum cut while guaranteeing an advantageous size for small components:

1. Perform kernelization on the problem instance $\langle G = (V, E), k \rangle$ to generate kernelized instance $\langle G' = (V', E'), k' \rangle$.
2. For some fixed number of iterations:
 - (a) randomly choose a vertex $v_1 \in V'$ and designate it to be source vertex, s ;
 - (b) determine a vertex v_2 , maximally distant from v_1 , to be sink vertex, t ;
 - (c) consolidate the neighborhoods around s and t by contracting their respective neighborhoods to some distance, $p \geq 0$, ensuring $D(s, p) = D(t, p) = O(\sqrt{2k})$; replace vertices s and t with the two new vertices, s' and t' ;
 - (d) use a maximum flow algorithm to determine a minimum (s', t') -cut of the consolidated graph;
 - (e) save all vertices from the minimum cut that satisfy the separator and component size constraints.
3. Merge all separators found in Step (2) in a strategic branching set.

We can use Dinic’s algorithm to determine the minimum (s, t) -cut in Step (2)d, recalling from Section 5.1 that maximum flow algorithms are fundamental to two state-of-the-art kernelization routines. As the consolidated graph has integer capacities, a more suitable choice would be the algorithm of Goldberg and Rao [87] with complexity $O(\min(n^{2/3}, m^{1/2})m \log(n^2/m) \log U)$, where U is the largest capacity edge in the flow network. This algorithm could also serve well for the kernelization-phase maximum flow algorithm incurring only a small additional cost in complexity as compared to the algorithm of Dinic for bipartite graphs.

We note that during the search tree phase most algorithms assume an interleaving of branching and kernelization steps. Generating the separator set for strategic branching is thus asymptotically no more expensive than in-tree kernelization.

5.2.2 Processing the separator set

Unlike the approaches described in Section 5.1 that use dynamic programming on tree decompositions, we will adapt existing branching rules for processing the vertices that constitute the separator set. In what follows, we will make use of two underlying k -vertex cover algorithms for processing separators in graphs of maximum degree 4: that of Chen, et al. [43], with branching complexity $O^*(1.277^k)$, and that of Balasubramanian, et al. [15], with branching complexity $O^*(1.324217^k)$. We refer the reader to Appendix C for details of the relevant branching rules.

Branching rule complexity

Branching rules are derived by considering combinations of subsets of vertices that could form a vertex cover in a small subgraph. The order in which rules are applied determines the complexity bound as the rules assume vertices in the subgraph have some known degree: the algorithms generally process all vertices of degree δ using rule set $\mathcal{X}$, before attempting to process vertices of degree γ using rule set $\mathcal{X} + 1$, where the latter set is constructed assuming the graph contains no vertices of degree δ . For example, the branching rules in the Chen, et al. algorithm assume all vertices of degree 2 in the graph be processed, before applying any branching rule for degree-3 or degree-4 vertices.

Figure 5.3: Processing degree-2 separator vertices to generate components A and C .

Figure 5.4: Processing separator adjacent to degree-2 vertex to generate components A and C .

We will show the branching rules either apply directly to processing the subgraph containing a separator vertex (and the cut edges), or that one (or two) rules can be applied to vertices adjacent to the separator resulting in its removal. The idea behind our processing will be to shift the role of separator vertex (as necessary) between vertices in a subgraph so as to enable us to invoke branching rules that remove the cut-set edge(s).

We begin by first presenting, for the sake of completeness, a series of lemmata that establish we can efficiently process vertices in a separator using the branching outlined in Appendix C. (We do not consider reduction rules as they require no branching and so process a separator vertex with ideal efficiency.)

In what follows, let $\langle G = (V, E), k \rangle$ be a graph with $|V| \leq 2k$ vertices, where $\forall v \in V, 2 \leq \deg(v) \leq 4$, and let $S \subseteq V$ be a separator of size m , as determined by our algorithm in Section 5.2.1. Because of the superior complexity, we adapt the branching rules of the Chen, et al. algorithm whenever possible.

Lemma 6 *Let $S \subseteq V$ be a set of separator vertices, where $\forall s \in S, \deg(s) = 2$. Then the set S can be processed with branching cost $O^*(1.277^m)$.*

Further, if $\exists s \in S, \deg(s) > 2$, and $\exists v \in N(s)$ such that $\deg(v) = 2$, then the set S can be processed with branching cost $O^(1.277^m)$.*

Proof. In the first case, we apply the folding rule to vertex s , branch on the new

vertex, s_0 , and remove the edges from the cut-set (see Figure 5.3). In the second case, we can equally apply the folding rule: vertex v would fold s and v into v_0 , and so the cut-set edges become adjacent to v_0 . Branching on v_0 or $N(v_0)$ removes the cut-set edges. $\square$

We can now limit our attention to separator vertices that have degree 3, or degree 4. We note that all Chen, et al. rules can be used directly to process the separator, S , within that algorithm's stated complexity, in the following cases: any degree-3 vertex adjacent to at least one degree-4 vertex; and any degree-4 vertex adjacent to at least one degree-3.

Lemma 7 *Let $S \subseteq V$ be a set of separator vertices, where $\forall s \in S$, $\deg(s) = 3$, and for each s , $\exists v \in N(s)$ such that $\deg(v) = 4$. Then the set S can be processed with branching cost $O^*(1.277^m)$.*

Further, let $S \subseteq V$ be a set of separator vertices, where $\forall s \in S$, $\deg(s) = 4$, and for each s , $\exists v \in N(s)$ such that $\deg(v) = 3$. Then the set S can be processed in time $O^(1.277^m)$.*

Proof. From the correctness of the Chen, et al. algorithm. $\square$

This leaves two degenerate cases (see Figure 5.5):

- separator vertices of degree 3, with only degree-3 neighbors (Figure 5.5a);
- separator vertices of degree 4, with only degree-4 neighbors (Figure 5.5b, 5.5c).

The algorithm of Chen, et al. presumes a certain amount of flexibility in processing the vertices, and in fact guarantees these cases do not arise (or, do so at most once). Thus it has no explicit branching rules for these cases. We use instead appropriate branching rules from the venerable algorithm of Balasubramanian, Fellows, Raman [15]. Remark that, if we assume the naïve branching rule, i.e., for degree-3 vertex, branching on $(k-1)$, $(k-3)$, and for degree-4 vertices, branching on $(k-1)$, $(k-4)$, the respective branching complexities are bounded by 1.465571^k and 1.380278^k , respectively. In contrast, the branching rules of the Balasubramanian, et al. algorithm are bounded by 1.324217^k , and so we take the liberty of assuming these rules are available in this case.

Figure 5.5: Degenerate degree-3 and degree-4 branching cases.

Lemma 8 *Let $S \subseteq V$ be a set of separator vertices and $\forall s \in S$ either $\deg(s) = 3$ and $\forall v \in N(s)$, $\deg(v) = 3$, or $\deg(s) = 4$ and $\forall v \in N(s)$, $\deg(v) = 4$. Then the set S can be processed in time $O^*(1.324217^m)$.*

Proof. From the correctness of the Balasubramanian, et al. algorithm. $\square$

We can now present a theorem that establishes that we can process vertices in a separator using the branching rules outlined in Appendix C.

Theorem 7 *Given a graph, $G = (V, E)$, $\forall v \in V$, $2 \leq \deg(v) \leq 4$, and an integer parameter k , the branching rules of the k -vertex cover algorithms of Chen, et al. [43] and Balasubramanian, et al. [15] can be applied in any order to process some subset of vertices, $S \subseteq V$, $|S| \leq k$, to find a partial vertex cover of G , with overall complexity $O(1.324718^k)$.*

Proof. From Lemmata 6–8. $\square$

Although we bound the processing of the separator by a worst branching complexity, we will see that, for sufficiently small separators, this will not negatively influence processing. Further, in more realistic circumstances, the better branching rules would also come into play leading to more efficient processing.

5.2.3 Asymptotic analysis

We now consider an asymptotic analysis of the improved k -vertex cover algorithm in graphs with degree bounded by 4.

Theorem 8 *Consider a graph, $G = (V, E)$, and integer parameter k , where $\forall v \in V$, $2 \leq \deg(v) \leq 4$. Using the strategic branching scheme and the adapted branching rules, let $S \subseteq V$ represent a separator determined by the algorithm in Section 5.2.1, and let $\langle G' = (C_1 \cup C_2), k - g(m) \rangle$ be a subproblem, consisting of two connected components at some intermediate node of the search tree, derived from applying at most $m = |S|$ branching rules to the separator. Let $n_{C_1} = |V(C_1)|$, $n_{C_2} = |V(C_2)|$, and without loss of generality we assume that $n_{C_1} \leq n_{C_2}$.*

We can determine if G has a vertex cover of size bounded by k in time $O(kn + c_1^m(c_2^{k'} + c_2^{k-g(m)-k'}))$, where c_1, c_2 are constants, $k' = (3(n_{C_1})+1)/4$, m is the separator size, and $g(m) \leq k$ is a function of the number of cover vertices found after m applications of the branching.

Proof. The expression c_1^m describes the number of leaves of the intermediate search tree generated after m branching rules have been applied. The constant c_2 can be 1.2738 [45] or 1.2745 [37] for example, depending on the underlying algorithm. The value $(3(n_{C_1})+1)/4$ represents a worst case bound on the cover sought for component C_1 . This result comes from approximating a maximum independent set of size $\lceil n/d \rceil$ for a graph G of bounded degree, d , by repeatedly adding the vertex with the smallest degree to the independent set. Because the branching rules can reduce the parameter by anywhere between 2 and 8, the function $g(m)$ bounds the parameter reduction after m applications of the rules. $\square$

5.3 Predictive performance analysis

To gain some intuition as to the existence of small separators, we conducted experiments using randomly generated graphs of varying sizes with vertices of minimum degree 2 and maximum degree 4. Table 5.1 lists five groups of ten randomly generated graphs, and the average number of separator and component pairs that fit the criteria in Section 5.2.1. In each group, a given graph exhibits several sufficiently small separators, capable of disconnecting sufficiently large subgraphs. For example, consider the graphs in group RG.1: each graph had, on average, at least two sets of strategic vertices consisting of approximately 13 separator vertices, that would each

Graph group	$ V $	Number of candidates
RG.1	375	2.3 (13.5, 47.0)
RG.2	755	10.1 (18.3, 47.3)
RG.3	1120	4.5 (24.1, 69.0)
RG.4	1499	7.4 (27.7, 93.5)
RG.5	1845	12.3 (24.2, 57.7)

Table 5.1: Component detection experiments: average number of candidate pairs (average separator size, average component size).

disconnect subgraphs with 47 vertices.

In this section we analyze the impact of combining strategic branching with the more recent state-of-the-art algorithm of Chen, et al. [45]. From Section 5.2.2 we know that most of the branching rules from the previous algorithm of Chen, et al. [43], and some rules from the Balasubramanian, et al. [15] algorithm, can be used to process the separator more or less directly.

In practice it is often the case that FTP algorithms perform much better than a worst case analysis would indicate and so we will focus on the expected performance. We use the following formula (based on Theorem 8) to model the search tree phase of the strategic branching algorithm:

$$c_1^m \left(c_2^{(3(\sqrt{2k})+1)/4} + c_2^{k-E[g(m)]-(3(\sqrt{2k})+1)/4} \right) \quad (5.1)$$

where $m = \sqrt{k}$, $c_2 = 1.2738$, and $\sqrt{2k}$ is the minimum size of the smallest component. Although our results are independent of the kernel size, we can assume it is linear and hence $|V| \leq 2k$ (see Chapter 2, Section 2.4.2).

We now turn to the question of determining the values for c_1 and $g(m)$. At a given node in the tree any of the branching rules from Appendix C may be invoked during separator processing resulting in the creation of 2, 3, or 4 children. We assume a worst case branching of 4 children (Balasubramanian, et al. Rule (5)c), hence $c_1 = 4$, leading to a tree with at most 4^m leaves. To determine the expected value of k_i at some leaf i of the search tree, i.e., $k - E[g(m)]$, we consider a path from the root of the tree to node i and the associated decremental values on each edge in the path. The sample space consists of the set of values $\{4, 5, 8\}$ and we define a random variable K_j , $1 \leq j \leq m$, to be the number of vertices added to the cover at some step j of the

Algorithm	Complexity	Klam value
Chen, et al. [43]	$O^*(1.2852^k)$	183
Chen, et al. [43] ¹	$O^*(1.277^k)$	188
Chandran, Grandoni [37]	$O^*(1.2745^k)$	189
Chen, et al. [45]	$O^*(1.2738^k)$	190

Table 5.2: Summary of complexities for recent k -vertex cover algorithms. (¹: restricted to graphs of degree bounded by 4.)

algorithm. The probability distribution is given by,

$$\Pr[K_j = 4] = 1/2,$$

$$\Pr[K_j = 5] = 1/4,$$

$$\Pr[K_j = 8] = 1/4.$$

The expected number of vertices added to the cover at a given node in the tree is,

$$E[K_j] = 4(1/2) + 5(1/4) + 8(1/4) = 5.25,$$

and so,

$$\sum_{j=1}^m E[K_j] = 5.25m,$$

i.e., the expected remaining parameter value for the subproblem at leaf i is $k - 5.25m$.

The complexity of Equation 5.1 converges to $O^*(1.2738^k)$ as $k \rightarrow \infty$ because the effect of cutting off and processing a small piece of the graph contributes little when k grows very large. Nonetheless, closer analysis shows the impact on the complexity is in fact more relevant within a practical range of values for k . In our analysis we fix k within a small range, specifically $200 \leq k \leq 4,000$, and study the effect on the complexity assuming separators generate two or more components.

As an initial comparison, Table 5.2 contains the complexities for various k -vertex cover algorithms. Let $T_S(k)$ denote the running for the strategic branching algorithm with parameter value k . For a fixed value of $k_{\max}$, we derive an exponential bound on $T_S(k)$ by determining some expression $T(k)$ such that $T_S(k)/T(k) \approx 1$, i.e., for all $k \leq k_{\max}$ the strategic branching algorithm is bounded by $T(k)$. Table 5.3 shows the overall complexity of Equation 5.1 in terms of $T(k)$, assuming various separator size

		Number of components			
		2	3	4	5
Value for $k_{\max}$	200	$O^*(1.261^k)$	$O^*(1.251^k)$	—	—
	250	$O^*(1.263^k)$	$O^*(1.251^k)$	—	—
	500	$O^*(1.266^k)$	$O^*(1.258^k)$	$O^*(1.250^k)$	—
	1,000	$O^*(1.269^k)$	$O^*(1.263^k)$	$O^*(1.257^k)$	$O^*(1.251^k)$
	2,000	$O^*(1.270^k)$	$O^*(1.266^k)$	$O^*(1.262^k)$	$O^*(1.258^k)$
	4,000	$O^*(1.271^k)$	$O^*(1.269^k)$	$O^*(1.266^k)$	$O^*(1.263^k)$

Table 5.3: Complexity assuming various hypothetical splits for k -vertex covers in graphs of maximum degree 4.

and split combinations. Cases where the entry is missing indicate that the separator would be too large and processing very nearly, or completely, solves the instance.

Having examined the table of complexity values, two items are worthy of mention. First, in most cases the increase in the complexity for the overall algorithm is potentially dramatic. Each splitting scenario represents an improvement over the complexity of $O^*(1.2738^k)$ of the original algorithm. Secondly, we note that in case the projected complexity would be worse than the original algorithm, due to the absence of sufficiently small separators, we can merely process the graph as per the original algorithm, i.e., with no special consideration for components.

5.4 Algorithm for k -VERTEX COVER problem in general graphs

We can improve the complexity of the algorithm for processing general graphs using the strategic branching approach for accelerating processing of graphs of degree bounded by 4. In the case of general graphs, branching rules systematically remove vertices of high degree, i.e., degree > 4 . Indeed, this is a step common to the current best k -vertex cover algorithms. Once all vertices of large degree have been removed, the improved algorithm for graphs of degree bounded by 4 can be used as a subroutine for processing the remaining subproblems in the lower branches of the search tree. As noted earlier, a search tree has most of its nodes in the lower branches, so this in fact will significantly improve the complexity of the algorithm for general graphs.

		Number of components			
		2	3	4	5
Value for n	200	$O^*(1.190^n)$	$O^*(1.183^n)$	—	—
	250	$O^*(1.191^n)$	$O^*(1.183^n)$	—	—
	500	$O^*(1.194^n)$	$O^*(1.188^n)$	$O^*(1.182^n)$	—
	1,000	$O^*(1.196^n)$	$O^*(1.191^n)$	$O^*(1.187^n)$	$O^*(1.183^n)$
	2,000	$O^*(1.196^n)$	$O^*(1.194^n)$	$O^*(1.191^n)$	$O^*(1.188^n)$
	4,000	$O^*(1.197^n)$	$O^*(1.196^n)$	$O^*(1.194^n)$	$O^*(1.191^n)$

Table 5.4: Complexity assuming various hypothetical splits for maximum independent sets in graphs of maximum degree 4.

5.5 Algorithm for Maximum Independent Set problem for graphs of small degree

The result from Section 5.3 has a surprising corollary. Although k -INDEPENDENT SET $\in W[1]$, Chen, et al. [43] propose using their k -vertex cover algorithm to find a maximum independent set in a graph of degree bounded by 4. For several years the best result for this problem was due to Robson [159] with complexity $O^*(2^{0.296n}) \approx O^*(1.227^n)$ restricted to graphs of maximum degree 4. The complexity for solving the problem using the k -vertex cover algorithm of Chen, et al. is $O^*(1.277^{3/4n} \approx 1.201^n)$. Beigel [21] subsequently presented an algorithm with complexity $O^*(1.171^n)$ for graphs of maximum degree 4, and Robson [160] published an algorithm for general graphs with complexity $O^*(1.18921^n)$. Very recently, Fürer [81] published an algorithm for finding maximum independent sets in graphs of maximum degree 3, with complexity $O(\text{poly}(n)1.1120^n)$.

Table 5.4 summarizes the complexities of the k -vertex cover algorithm with strategic branching (Table 5.3) when applied to the MAXIMUM INDEPENDENT SET problem in graphs of degree bounded by 4, assuming various split combinations. In all cases, the strategic branching is an improvement over the results of Chen, et al. and in some cases is an improvement over the recent result of Robson.

5.6 Summary

We described a new approach to improve algorithms for solving the k -VERTEX COVER problem. The algorithm uses graph cutting during the tree search phase as a basis for disconnecting and explicitly processing small parts of the residual graph. Unlike previous methods that use separator theorems for restricted graph classes and require complex dynamic programming, this algorithm applies to graphs of small bounded degree, adapts existing k -vertex cover machinery, and incurs no additional complexity. Such graph instances are generated during the tree search phases of the current best FPT k -vertex cover algorithms. Any future advances in sequential algorithms for the k -VERTEX COVER problem can incorporate strategic branching, thus further improving their complexity.

We also investigated the applicability of our new algorithm to solve the MAXIMUM INDEPENDENT SET problem in graphs of small bounded degree. Depending on the connected components that are generated, the new k -vertex cover algorithm has complexity comparable to the current best algorithm for the MAXIMUM INDEPENDENT SET problem.

Chapter 6

The CONSTRAINED MINIMUM BIPARTITE VERTEX COVER Problem

Every composer knows the anguish and despair occasioned by forgetting ideas which one had no time to write down.

-Hector Berlioz

In this section we look at a two-parameter variation of the k -VERTEX COVER problem restricted to bipartite graphs. Let $G = (U \cup L, E)$ be an n -vertex bipartite graph. The CONSTRAINED BIPARTITE VERTEX COVER problem requires determining a vertex cover consisting of at most k_1 vertices from U and at most k_2 vertices from L .

The problem was initially studied by Fernau and Niedermeier [74], who developed an algorithm with complexity $O(1.40^{k_1+k_2} + (k_1 + k_2)n)$. A slightly different variation of the problem, called the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem, requires determining a minimum vertex cover with the bounds k_1 and k_2 . Chen and Kanj [44] provide an algorithm for this problem with complexity $O(1.26^{k_1+k_2} + (k_1 + k_2)n)$.

In this section we investigate the applicability of parallel techniques to the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem. The algorithm of Chen and Kanj uses two important results from bipartite matching theory, the Gallai-Edmonds Structure Theorem and the Dulmage-Mendelsohn decomposition. We develop parallel CGM algorithms for these procedures and we investigate an alternate method for determining the Gallai-Edmonds decomposition using matrix operations. As the problem and the algorithm are very different from the VERTEX COVER algorithms seen so far, we go into some detail in the exposition.

6.1 Background

6.1.1 Matching theory

Before proceeding, we require some basic definitions from matching theory.

Given a graph, $G = (V, E)$, a *matching*, M , in G is a set of pairwise non-adjacent edges. A vertex is *matched* if it is incident to an edge in the matching. Otherwise the vertex is *unmatched*, or *exposed*. A *maximum matching* is a matching that contains the largest possible number of edges. The matching number, $\nu(G)$, of a graph is the size of a maximum matching. A *maximal matching* is a matching M of G with the property that if any edge not in M is added to M , it is no longer a matching. The *deficiency* of a graph G is the number of vertices exposed in a maximum matching, i.e., $n - 2\nu(G)$. Given a matching M , an *alternating path* is a path in which the edges belong alternatively to the matching and not to the matching. An *augmenting path* is an alternating path that starts from and ends on unmatched vertices. Clearly a matching is maximum if and only if it does not contain an augmenting path. A *perfect matching* is a matching which covers all vertices of the graph, i.e., every vertex of the graph is incident to exactly one edge of the matching. Every perfect matching is both maximum and maximal. A matching is said to be *near-perfect* if it covers all but exactly one vertex. A graph G is called *factor-critical* if the graph $G(V \setminus \{v\})$ has a perfect matching, $\forall v \in V$. A vertex is called *noncritical* if it is exposed in at least one maximum matching, otherwise it is *critical*. In other words, a vertex v is noncritical if $\nu(G(V \setminus \{v\})) = \nu(G)$ and vertex v is critical if $\nu(G(V \setminus \{v\})) < \nu(G)$. An *elementary* bipartite graph, $G = (U \cup L, E)$, has exactly two minimum vertex covers, the set of vertices U or the set L , and every edge in G lies in some perfect matching. An *odd* (*even*) component of a graph is a connected component whose vertex set has odd (even) cardinality.

The problem of finding a maximum matching in a bipartite graph with n vertices and m edges can be solved in time $O(n^{1.5}\sqrt{m/\log n})$ [12]. Finding a maximum matching in a general graph is also polynomial time solvable, e.g., with complexity $O(m\sqrt{n})$ [185]. Parallel algorithms for the maximum matching problem in bipartite graphs have been investigated from a theoretical perspective [91, 86]. In terms of

practical models, there have been CGM algorithms developed for a restricted version of the bipartite maximum matching problem, where the graph is assumed to be convex [27, 173]. A bipartite graph, $G = (U \cup L, E)$, is *convex* if there is an ordering of the vertices $L = (l_1, l_2, \dots, l_r)$ such that for any $u \in U$, if there exist edges $(u, l_i) \in E$ and $(u, l_j) \in E$, $i \leq j$, then $(u, l_m) \in E$, $i \leq m \leq j$. Most attempts at parallelization have focused on adapting existing sequential algorithms to a parallel setting, and not surprisingly, exhibit only small speedup [168]. A CGM parallel algorithm for the general case of bipartite maximum matching poses a much more difficult problem.

6.1.2 Algorithms for CONSTRAINED MINIMUM BIPARTITE VERTEX COVER

The CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem is defined as follows: given a bipartite graph, $G = (U \cup L, E)$, and integers k_U , k_L , we must decide if there exists a minimum vertex cover for G with at most k_U vertices from U and at most k_L vertices from U_L . This problem has many applications, including fault-tolerant reconfigurable arrays [99].

Fernau and Niedermeier [74] provided an FPT algorithm for the case where the cover is not necessarily minimum, with complexity $O(1.3999^{k_U+k_L} + (k_U + k_L)n)$. Chen and Kanj [44] provide an solution for the case where the cover is assumed to minimum. Their algorithm, with time complexity $O(1.26^{k_U+k_L} + (k_U + k_L)n)$, is based on two well-known results from graph theory, the Gallai-Edmonds Structure Theorem and Dulmage-Mendelsohn decomposition.

6.1.3 Details of the Chen and Kanj algorithm

Before describing the algorithm of Chen and Kanj for the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem, we first introduce the main algorithmic components.

Gallai-Edmonds Structure Theorem

Consider the following decomposition of a general graph $G = (V, E)$ and denote by D the set of all vertices in G not covered by at least one maximum matching. Let A be

Figure 6.1: Example of Gallai-Edmonds decomposition for a general graph, $G = (V, E)$. (Figure adapted from [133].)

the set of vertices in $V \setminus D$ adjacent to at least one vertex in D , and let $C = V \setminus (A \cup D)$. The Gallai-Edmonds Structure Theorem can be stated as follows:

Theorem 9 [133] *If $G = (V, E)$ is a graph and D , A , and C are subsets of V as defined previously, then:*

1. *each component of the subgraph induced by C has a perfect matching;*
2. *each component of the subgraph induced by D is factor-critical;*
3. *the deficiency of G equals $c(D) - |A|$, where $c(D)$ denotes the number of connected components in the subgraph induced by D ;*
4. *every maximum matching of G contains a perfect matching of each component of C , a near perfect matching of each component of D , and matches all the vertices of A with vertices in distinct components of D .*

If the graph is bipartite, i.e., $G = (U \cup L, E)$, then D forms an independent set, and A is a set of vertices in every minimum cover of G .

Dulmage-Mendelsohn Decomposition

A Dulmage-Mendelsohn decomposition of an elementary bipartite can be described as follows:

Figure 6.2: Example of Dulmage-Mendelsohn decomposition on subgraph induced by vertex set C , given bipartite graph, $G = (U \cup L, E)$. (Figure adapted from [133].)

Theorem 10 [133] *A elementary bipartite graph, $G = (U \cup L, E)$ with a perfect matching can be decomposed into elementary subgraphs, or blocks, $B_j = (U_j \cup L_j, E_j)$, $j = 1, 2, \dots, i$, such that every edge in G from subgraph B_x to subgraph B_y , $x < y$, must have one endpoint in U_x and the other endpoint in L_y .*

As the edges of the elementary subgraphs are those in any perfect matching, the edges with endpoints in different blocks, or *interblock* edges, are necessarily those not contained in any perfect matching (for example, see Figure 6.2).

Algorithm overview

With an understanding of the decomposition phases, we can now proceed to the algorithm itself. Let VC be the set of cover vertices for bipartite graph, $G = (U \cup L, E)$. In what follows, we summarize the main steps from the algorithm of Chen and Kanj:

1. Given a parameterized problem, $\langle G = (U \cup L, E), k_U, k_L \rangle$, perform standard kernelization:
 - (a) a vertex $v \in U$, where $\deg(v) > k_L$, is added to the cover, $\{v\} \cup VC$, and $k_U = k_U - 1$;
 - (b) a vertex $v \in L$, where $\deg(v) > k_U$, is added to the cover, $\{v\} \cup VC$, and $k_L = k_L - 1$;
2. Graph G can be further kernelized using the Gallai-Edmonds Structure Theorem, such that G is partitioned into three sets of vertices, D, A, C :

- (a) D is an independent set in G in which no vertex is contained in any minimum vertex cover;
 - (b) A is in the intersection of all minimum vertex covers, so add the vertices of A to VC ; $A \cup VC$, $k_U = k_U - |A \cap U|$, $k_L = k_L - |A \cap L|$;
 - (c) the subgraph induced by C has a perfect matching and consists of at most $2(k_U + k_L)$ vertices.
3. Apply the Dulmage-Mendelsohn decomposition to the bipartite subgraph $G'(C) = (U' \cup L', E')$, i.e., the subgraph induced by C . Subgraph G' can be decomposed into elementary subgraphs, $B_i = (U_i \cup L_i, E_i)$, $i = 1, 2, \dots, r$ such that any edge in G' from B_i to B_j , $i < j$, must have one endpoint in U_i and the other endpoint in L_j .

The minimum vertex cover for graph G' is the union of the minimum covers of the blocks, $B_1, B_2, \dots, B_r$.

4. Use a search tree to process the combinations of U_i and L_i parts of the blocks $B_1, B_2, \dots, B_r$. A directed acyclic graph, D , is used to model the interdependencies of the blocks in G' : for every block in B_i in G' with $d_i = |U_i| = |L_i|$ vertices, there is a corresponding node B_i in D with weight d_i ; if two nodes B_i and B_j in G' are joined by an interblock edge, i.e., between vertices U_i and L_j , then there is a corresponding directed edge from B_i to B_j in D .

The search tree phase uses the weighted nodes in the DAG for branching decisions. Branching rules add the corresponding U_i or L_i vertices to the cover, and remove their corresponding block, B_i , from G' .

6.2 CGM algorithm for CONSTRAINED MINIMUM BIPARTITE VERTEX COVER

In this section we describe a CGM parallelization of the algorithm of Chen and Kanj, specifically the decomposition and search tree phases. We assume the graph has been kernelized in the standard way, i.e., all vertices of degree greater than k_U and k_L have been removed and added to the cover, VC .

Recall the definition of CGM from Chapter 3, Section 3.1.2. The CGM model comprises p processors, P_i , $0 \leq i \leq p-1$ with $O(n/p)$ local memory storage such that $n/p \geq p^\epsilon$, $\epsilon > 0$, and interconnected with an arbitrary communication fabric. Computation and communication are separated into synchronous super-steps. In a communication round, each processor can send or receive $O(n/p)$ data words.

6.2.1 Local memory and exponential algorithms

In the algorithms that follows, we can adopt a less rigid definition of the memory requirements for the CGM model: specifically, we assume a given problem instance, $\langle G, k_1, k_2 \rangle$, never exceeds the memory limit of a single processor, i.e., a processor can store a copy of an entire instance in memory. This is a reasonable assumption as the instances are kernelized and so bounded in size by some function, $f(k_1 + k_2)$.

6.2.2 CGM Gallai-Edmonds decomposition

From the definition of the sets C , A , and D , we know that with respect to a maximum matching M , the vertices in C are perfectly matched, and Theorem 9, property (1), indicates there is no edge in any maximum matching M connecting vertices from A with C . Hence, vertices reachable from an unmatched vertex via an even, alternating path are exactly those in set D . Similarly, those vertices reachable from an unmatched vertex v via an odd, alternating path are exactly those in A .

A CGM algorithm, assuming p processors can be describe as follows. Let M be a maximum matching for G' and let $V_O \subseteq (U' \cup L')$ be the set of unmatched vertices with respect to M . Each processor, P_i , $0 \leq i \leq p-1$, has a local copy of the problem instance, $\langle G' = (U' \cup L', E'), k'_U, k'_L \rangle$, and a set of vertices, V_{O_i} , of size $\lceil |V_O|/p \rceil$.

1. Each processor, P_i , $0 \leq i \leq p-1$, builds an alternating forest with trees rooted at $v_{i,l} \in V_{O_i}$, $l = 1, 2, \dots, |V_{O_i}|$ where every tree node is associated with a vertex in $(U' \cup L')$, and every tree edge is associated with an edge in E' :
 - (a) Each processor, P_i , initially expands from vertex, $v_{i,l}$, in breadth-first fashion as follows (the root vertex of each tree is defined to be at level 1):

- i. let u be a node at an odd level, associated with a vertex u : then node u has $\deg(u)$ children corresponding to its neighboring vertices in G' ;
 - ii. let x be a node at an even level, associated with a vertex x : then node x has one child, y , corresponding to the edge (x, y) in matching M ;
 - iii. P_i maintains a list of visited vertices, and stops expanding a subtree if it encounters a vertex previously visited, or if no edges can be further explored;
 - iv. each processor, P_i , locally maintains subtree instances that can be transferred to idle processors. As each processor has a local copy of the graph, this requires only transferring the vertex that is the root of the given subtree and level index.
2. Each processor, P_i , $0 \leq i \leq p - 1$, forms a local set D_i from the vertices in the even levels of the trees, and forms the set of vertices in A_i from the vertices in the odd levels of the trees.
3. The processors, P_i , $0 \leq i \leq p - 1$, merge their sets A_i , D_i , to form the sets A , D , C :
- (a) if $|(U' \cup L') \setminus (A \cup D)| > 2(k'_U + k'_L)$ then $\langle G', k'_U, k'_L \rangle$ is a *No*-instance;
 - (b) if $|U' \setminus (A \cup D)| = k'_U$ then $\langle G', k'_U, k'_L \rangle$ is a *Yes*-instance;
 - (c) if $|L' \setminus (A \cup D)| = k'_L$ then $\langle G', k'_U, k'_L \rangle$ is a *Yes*-instance;
 - (d) otherwise, add set A to the cover, $A \cup VC$, and the remaining subproblem is $\langle G' \setminus (A \cup D), k'_U - |A \cap U'|, k'_L - |A \cap L'| \rangle$.

Asymptotic analysis

Steps (1), (2) are bounded by the time required to generate the breadth-first alternating trees, $O((|V| + |E|)[|V_O|/p])$. Step (3) can be accomplished using integer sorting, which requires $O(|V|/p)$ local computation and $O(1)$ communication rounds with at most $O(|V|/p)$ -relations [36].

Theorem 11 *Let $\langle G' = (U' \cup L', E'), k'_U, k'_L \rangle$ be an instance of the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem where G' has at most $n = 2(k_U + k_L)^2$*

vertices and at most $m = (k_U + k_L)^2$ edges. Let M be a maximum matching for G' with a set, $V_O \subseteq U \cup L$, of unmatched vertices.

Assuming a CGM model comprising p processors, P_i , $0 \leq i \leq p-1$, then instance G' can be either decided, or decomposed into sets D, A, C , in $O((n+m)\lceil |V_O|/p \rceil)$ local computations steps and $O(1)$ communication rounds using at most $O(n/p)$ -relations.

6.2.3 CGM Dulmage-Mendelsohn decomposition

Recall from Theorem 6.1.3 the set of edges not contained in any perfect matching are interblock edges. The Dulmage-Mendelsohn decomposition identifies subgraphs, or blocks of vertices, that are elementary, i.e., for which their constituent edges are in some perfect matching. The algorithm only requires testing each edge, $(u, v) \in E'$, where $(u, v) \notin M$, to see if the graph $G' \setminus \{u, v\}$ has a perfect matching.

We can describe a CGM algorithm as follows. We assume that each processor, P_i , $0 \leq i \leq p-1$, has a local copy of the graph, $G' = (U' \cup L', E')$. We can in fact use the maximum matching, M , from the Gallai-Edmonds decomposition phase. Let $M_C \subseteq M$ be the set of edges matching vertices from set C in G' . The edges of M_C form a perfect matching of $G(C)$.

The set of edges, E_I , not matched by the perfect matching M_C are distributed evenly across the processors in sets, E_{I_i} , $0 \leq i \leq p-1$, and each processor has a copy of M_C .

1. Each processor P_i , $0 \leq i \leq p-1$, tests its set of edges, E_{I_i} , as follows:
 - (a) select an edge $(u, v) \in E_{I_i}$;
 - (b) remove vertex u and v from G' ;
 - (c) check if the graph $G' \setminus \{u, v\}$ has an augmenting path with respect to M ;
if there is exists an augmenting path, then remove (u, v) from E_{I_i} .
2. All processors, P_i , evenly distribute all remaining interblock edges E_{I_i} , $0 \leq i \leq p-1$, determined in Step (1).
3. Each processor P_i using its set E_{I_i} , determines the blocks induced by E_{I_i} as follows:

- (a) given an edge $(u, v) \in E_{I_i}$, the processor associates a block, B_u , with vertex u and a block, B_v , with vertex v ;
- (b) for each block, P_i performs a breadth-first search, ensuring never to cross any interblock edge. The set of vertices reachable from u (or v) form block B_u (or B_v , respectively).

Asymptotic analysis

Step (1) requires $O(|E| \cdot |E_{I_i}|/p)$ time, as the existence of an augmenting path can be checked in linear time. Step (2) consists of a communication round with an $(|E_{I_i}|)$ -relation. In Step (3), each processor can perform a local breadth-first search to determine the blocks induced by E_{I_i} in time $O((|V| + |E|)/p)$.

Theorem 12 *Let $\langle G' = (U' \cup L', E'), k'_U, k'_L \rangle$ be an instance of the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem where G' has at most $n = 2(k_U + k_L)$ vertices and at most $m = (k_U + k_L)^2$ edges. Let M be a maximum matching for G with a set, $E_I \subseteq U \cup L$, edges not in M .*

Assuming a CGM model comprising p processors, P_i , $0 \leq i \leq p - 1$, a Dulmage-Mendelsohn decomposition for a kernelized bipartite graph G' can be determined in $O(m \lceil m/p \rceil)$ local computations steps and $O(1)$ communication rounds with at most $O(m/p)$ -relations.

6.2.4 Search tree branching

The tree search for the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem is notably different from that of the k -VERTEX COVER problem, in two particular ways: (i) the branching is guided by inspecting suitably weighted nodes in the directed, acyclic graph, D ; (ii) in branching decisions, the indegree and outdegree of the DAG nodes are secondary to their weight. Branching in the tree search phase is based on three rules applied to D :

1. Initially, all nodes B_i in D with weight greater than 3 are branched upon: this creates two subproblems in the tree: (i) $U_i \cup VC$, $G - B_i$, $D - B_i$; (ii) $L_i \cup VC$, $G - B_i$, $D - B_i$.

2. At this next phase, D consists of weight-2 and weight-1 nodes, and we can:
 - (a) branch on all weight-2 node with non-zero indegree and non-zero outdegree;
 - (b) branch on any weight-1 node with three incoming edges coming from weight-2 nodes, or branch on any weight-1 node with three outgoing edges going to weight-2 nodes.
3. In the final phase, the graph can be resolved in polynomial time with no branching required: D now consists only of weight-1 nodes, and weight-2 nodes having either only outgoing edges, or only incoming edges.

Due to the restricted structure of the problem and the unique use of the DAG to direct the branching in the graph, the framework of Chapter 3 can not be readily adapted to solving the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem. Nonetheless, naïve branching based on the rules outlined above is possible: assuming p processors, P_i , $0 \leq i \leq p-1$, an initial sequence of nodes, $S \subseteq D$, can be processed to form a partial search tree of depth $O(\log p)$. Each processor P_i would then process a unique subproblem according to the criteria described in Rules (1)-(3) above.

6.3 Gallai-Edmonds decomposition for general graphs

In this section we examine parallel approaches to Gallai-Edmonds decomposition for general graphs. Recall from Theorem 9 the Gallai-Edmonds decomposition determines subsets of C , A , D of V induced by a maximum matching, M , in a general graph $G = (V, E)$: in the non-degenerate case, M matches the vertices of A with vertices in distinct components of D (for example, see Figure 6.1); if M is perfect then $A = \emptyset = D$ and $C = G$.

We describe an alternate method for finding the Gallai-Edmonds decomposition, differing from the one presented in Section 6.2.2 in that a maximum matching is not required *a priori*.

6.3.1 EREW PRAM Gallai-Edmonds algorithm for general graphs

Let $G = (V, E)$ be an n -vertex graph. Cheriyan [46] describes a randomized Monte Carlo EREW PRAM algorithm for finding a Gallai-Edmonds decomposition with complexity $O(n \log^2 n)$ requiring $\tilde{O}(n^\omega)^1$ processors, where $\omega = 2.376$ denotes the current best complexity for multiplying two $n \times n$ matrices [49].

In what follows, we give a brief overview of the background and the algorithm. The decomposition algorithm first identifies the set of non-critical vertices, D . Given D , the sets A and C can be deduced. The algorithm is based on a key result of Tutte, which defines a random skew symmetric adjacency matrix for a n -vertex graph, $G = (V, E)$, as follows:

1. with each edge $(i, j) \in E$, we associate a distinct variable x_{ij} ;
2. let $\tilde{B} = \tilde{B}(x_{ij})$ be an $n \times n$ matrix with the following entries:

$$\tilde{B}_{ij} = \begin{cases} x_{ij} & \text{if } i > j \text{ and } (i, j) \in E \\ -x_{ij} & \text{if } i < j \text{ and } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

The determinant of $\tilde{B}(x_{ij})$ is a polynomial in x_{ij} . Tutte proved that the graph G has a perfect matching if and only if the determinant of $\tilde{B}(x_{ij})$ is not identically zero. Lovász [132] developed an algorithm for deciding whether a graph G has a perfect matching: replace each variable with an independently chosen random number from $\{1, 2, \dots, q-1\}$, where $q = n^{O(1)}$ is a prime number, and compute the determinant of the resulting matrix over the field of integer modulo q . Let B denote the resulting random matrix. With high probability, the $\det(B) \neq 0 \pmod{q}$ if and only if $\det(\tilde{B}(x_{ij}))$ is not identically zero if and only if the graph G has a perfect matching.

Lovász generalized the result of Tutte by establishing a relationship between the maximum matching of the graph and the rank of the matrix:

Proposition 3 *Let $\tilde{B} = \tilde{B}(x_{ij})$ be the skew symmetric adjacency matrix of a graph G . Then $\text{rank}(\tilde{B}) = 2\nu(G)$.*

¹The notation $\tilde{O}(\cdot)$ is the asymptotic complexity that ignores poly-logarithmic factors.

The following lemma provides a means of bounding the probability of failure for a class of randomized algorithm:

Lemma 9 Zippel-Schwartz [192, 167] *If $p(x_1, x_2, \dots, x_m)$ is a nonzero polynomial of degree d with coefficients in a field and S is a subset of the field, then the probability that p evaluates to zero on a random element $(s_1, s_2, \dots, s_m) \in S^m$ is at most $d/|S|$.*

Finally, by combining Proposition 3 and Lemma 9, we have:

Proposition 4 Lovász [132] *Let $B = \tilde{B}(x_{ij})$ be a random skew symmetric adjacency matrix of a graph G , where w_{ij} are independent random numbers from $\{1, 2, \dots, W\}$. Then $\text{rank}(B) \leq 2\nu(G)$, and with probability at least $1 - n/W$, $\text{rank}(B) \leq 2\nu(G)$.*

Fix $W = n^{O(1)}$ and let q be a prime such that $W < q = n^{O(1)}$. Cheriyan's algorithm is described by the following Lemma:

Lemma 10 [46] *Let $B = \tilde{B}(w_{ij})$ be a random skew symmetric adjacency matrix of a graph $G = (V, E)$, where each w_{ij} is an independent random number from $\{1, 2, \dots, W\}$. Let e_j denote a vector with a 1 in position j , and zeroes everywhere else, and let $[e_j^B]$ denote an $(n+1) \times n$ matrix formed by adding the $(n+1)^{\text{th}}$ row e_j to an $n \times n$ matrix B . Then considering some vertex $x \in E$ with index j in B we have:*

1. *if x is noncritical, then with probability at least $1 - (2n/W)$ the rank of matrix $[e_j^B]$ is greater than that of B ;*
2. *if x is critical, then with probability at least $1 - n/W$ the rank of matrix $[e_j^B]$ equals that of B .*

The algorithm then consists of the following steps:

1. Build a random skew symmetric adjacency matrix B of graph G :
 - (a) where each edge $(i, j) \in E$ is assigned a random weight, $w_{ij} \in \{1, 2, \dots, W\}$;
 - (b) for each edge $(i, j) \in E$, $i > j$: $B_{ij} = w_{ij}$, $B_{ji} = -w_{ij}$; and 0 if there is no edge.
2. Compute the rank r of B over $\mathbb{Z}_q$.

3. For each vector e_j , $j = 1, 2, \dots, n$, compute the rank r_j of matrix $[e_j^B]$ over $\mathbb{Z}_q$.
4. Let D be the set of vertices j with $r_j > r$: from D , define subsets A and C as per the Gallai-Edmonds decomposition.

With high probability, the Gallai-Edmonds decomposition of G is given by A , C , and D . Because the algorithm is Monte Carlo, it is possible to return an incorrect decomposition. This can be verified in sequential time, $\tilde{O}(m + n)$.

Sequentially, the algorithm has complexity $\tilde{O}(n \cdot n^\omega)$. An EREW PRAM algorithm with running time $O(n \log^2 n)$, using $\tilde{O}(n^\omega)$ processors easily follows from the algorithm above.

6.3.2 CGM Gallai-Edmonds decomposition for general graphs

The underlying theory behind the Gallai-Edmonds algorithm can be adapted to a coarse-grained setting. We assume graph $G = (V, E)$ is initially distributed across the processors, P_i , $0 \leq i \leq p - 1$. As per our caveat in Section 6.2.1, we assume each processor has sufficient memory to store an entire $n \times n$ matrix corresponding to the kernelized graph.

1. Each processor P_i , $0 \leq i \leq p - 1$ generates a block of the skew-symmetric adjacency matrix, B , using the edge set E .
2. In parallel, processors P_i , $0 \leq i \leq p - 1$, calculate the rank, r , of matrix B .
3. Matrix B is broadcast to all processors, P_i , $0 \leq i \leq p - 1$.
4. Each processor, P_i , $0 \leq i \leq p - 1$, locally calculates the rank of augmented matrix $[e_{ij}^B]$, for a given set of vectors, e_{ij} , $j = 1, 2, \dots, \lceil n/p \rceil$. Let D_i be the set of vertices whose corresponding augmented matrix has rank $r_{ij} > r$.
5. All processors P_i , $0 \leq i \leq p - 1$, merge their sets D_i , and generate sets D , A , and C .

Asymptotic analysis

Because the graph G is assumed to be a kernelized instance, the number of vertices is at most $n = (k_U + k_L)^2$. Step (1) can be done in $O(n^2/p)$ time. In Step (2), the rank of an $n \times n$ matrix A can be determined by LU decomposition, where $A = LU$ is decomposed into the product of two matrices, L , which is lower triangular, and U which is upper triangular. The rank of matrix A is equal to the number of non-zero rows in L and U . An LU decomposition of an $n \times n$ matrix, with pivoting, can be performed in $O(n^3/p)$ local computation steps and $O(p)$ communication rounds requiring $O(n/p)$ -relations [83]. Step (3) is a communication round with an $O(n^2)$ -relation. In Step (4), for each augmented matrix, we check whether a given vertex v is noncritical, by determining if the rank of the augmented matrix increases. This requires calculating the rank of an $(n + 1) \times n$ matrix $\begin{bmatrix} B \\ e_j \end{bmatrix}$, which can be done in time $O(n^\omega)$ [103]. Step (5) can be accomplished using integer sorting, which requires $O(n/p)$ local computation and $O(1)$ communication rounds with at most $O(n/p)$ -relations [36].

Theorem 13 *Let $G = (V, E)$ be a graph with at most $n = (k_U + k_L)^2$ vertices. Assuming a CGM model comprising p processors, P_i , $0 \leq i \leq p-1$, a Gallai-Edmonds decomposition for graph G can be determined in $O((n/p) \cdot n^\omega)$ local computation steps and $O(p)$ communication rounds requiring at most $O(n^2)$ -relations.*

6.3.3 Applications of the Gallai-Edmonds decomposition

The Gallai-Edmonds Structure Theorem can be used as a more general technique for improving other FPT algorithms, as it imposes structure on the graph. Consider the search tree phase of a k -vertex cover algorithm with a (general) kernelized graph $G' = (V', E')$ as input. Let A , D and C be the three sets of vertices generated by the Gallai-Edmonds decomposition of G' . The vertices in A form a general set of strategic branching vertices, analogous to those described in Chapter 5. Processing the vertices in A disconnects G' into the components induced by the vertices in C and D . From Theorem 9, we know the components in C have a perfect matching and the components in D have a near-perfect matching, implying a tight lower bound on

the minimum vertex cover in each component. These tight bounds provide a means of bounding the search when processing the individual components.

6.4 Summary

In this section we investigate the applicability of parallel techniques to the **CONSTRAINED MINIMUM BIPARTITE VERTEX COVER** problem. The algorithm of Chen and Kanj uses two important results from bipartite matching theory, the Gallai-Edmonds Structure Theorem, and the Dulmage-Mendelsohn decomposition. We develop CGM algorithms for both decompositions. The tree search phase, which is computationally the most expensive phase, is readily parallelizable, although it is difficult to adapt the parallel framework from Chapter 3 to the algorithm.

We investigated parallel techniques for the Gallai-Edmonds decomposition in general graphs, and describe an alternate means of computing the Gallai-Edmonds decomposition for bipartite graphs using matrix operations.

Chapter 7

k -VERTEX COVER and the MAXIMUM CLIQUE Problem

Everything should be made as simple as possible, but not one bit simpler.

-Albert Einstein

In this section we describe an improved algorithm for solving the MAXIMUM CLIQUE problem in a graph using a novel combination of sampling with a parameterized k -vertex cover algorithm. We also examine a very effective heuristic for finding a large clique that combines our sampling approach with fast independent set approximation. In experiments using the DIMACS benchmark, the heuristical approach established new lower bounds on the size of the largest clique for four instances and provides the first optimal solution for an instance unsolved until now. The heuristic competitively matched the accuracy of the current best exact algorithm in terms of correct solutions, while requiring a fraction of the runtime. The research in this chapter has been published in *Proceedings of the 2nd International Conference on Algorithmic Aspects in Information and Management (AAIM 2006)*.

7.1 Background

Let $G = (V, E)$ be an undirected graph. A *clique* is a subset, $C \subseteq V$, such the vertices of C form a complete subgraph in G . A wealth of different approaches have been used to tackle the problem, including graph coloring [190] and integer programming formulations [170], not to mention heuristics (for a survey, see Pelillo [149]).

The *complement graph* of G is the graph $\overline{G} = (V, \overline{E})$, where $\overline{E} = \{(i, j) | i, j \in V, (i, j) \notin E\}$. C is a clique in G if and only if C is an independent set in $\overline{G}$. Finally, a graph G has a vertex cover of size k if and only if the complement graph $\overline{G}$ has a clique of size $n - k$. Recall from our section on parameterized complexity theory

that k -CLIQUE is $W[1]$ -complete [65]. Abhu-Khzam, Langston, Shanbhag [1], Abhu-Khzam, et al. [2], and later Baldwin, et al. [16], describe experiments that use a parallel FPT k -vertex cover implementation for solving the k -CLIQUE problem by determining a minimum vertex cover in the complemented input graph.

The algorithm of Wood [189] is based on a branch-and-bound approach that uses a fractional coloring procedure [17] as an upper bound heuristic. Östegård [145] uses a strategy that is similar to dynamic programming, where the problem is solved for subsequently increasing numbers of nodes. An optimal value from previous computations is used as the minimal value for the current problem. The algorithm of Fahle [70] introduces simple cost-based domain filtering [77]. The approach restricts the candidate set by eliminating vertices that can not be used to extend the clique under construction, or by fixing certain vertices that will be in the current clique. The algorithm of Regin [153], also based on constraint programming, proposed two new upper bounds on the largest clique and a new search strategy used to control the search for an optimal solution. We direct the reader to Bomze, et al. [26] for a thorough survey on exact algorithms.

7.2 Improved MAXIMUM CLIQUE algorithms

In this section, we introduce a graph sampling technique that we combine with a k -vertex cover algorithm to develop an improved algorithm for a finding maximum clique in a graph. We will later describe a heuristic for the maximum clique problem that is not based on determining a vertex cover, but instead on calculating an independent set.

Let $G = (V, E)$ be a graph and assume $v \in V$ is a member of a clique, $C \subseteq V$, of G . In what follows, we use $N(v)$ to denote the set of vertices adjacent to some vertex v , $N[v]$ to denote the set $N(v) \cup \{v\}$, and $G(N[v])$ is the subgraph induced by $N[v]$. The algorithms exposted in this section are based on the following theorem.

Theorem 14 *Let $G = (V, E)$ be a graph and assume $v \in V$ is a member of a clique, $C \subseteq V$, of G . If C is the largest clique of which v is a member, then the maximum independent set of the complement of the subgraph $G(N[v])$ is exactly C .*

Figure 7.1: Determining a local maximum clique: (a) subgraph around vertex v , with local maximum clique $\{v, 1, 2, 3, 4, 5\}$; (b) extracted subgraph $G(N[v])$; (c) minimum vertex cover for $\overline{G}(N[v])$ is $\{6, 7\}$ with corresponding maximum independent set of $\{v, 1, 2, 3, 4, 5\}$.

Figure 7.2: Determining a local maximum clique with 2-level sampling, assuming sample vertices $v, 3$: (a) subgraph around vertex v , with local maximum clique $\{v, 1, 2, 3, 4, 5\}$; (b) extracted subgraph $G(N[v] \cap N[3])$; (c) corresponding maximum independent set of $\{v, 1, 2, 3, 4, 5\}$.

Proof. The proof follows from the definition of a clique and the relationship between the MINIMUM VERTEX COVER, MAXIMUM INDEPENDENT SET and MAXIMUM CLIQUE problems. $\square$

What makes Theorem 14 effective is that the parameterized approaches of Abhu-Khzam, Langston, Shanbhag [1], Abhu-Khzam, et al. [2], and Baldwin, et al. [16], solve the MAXIMUM CLIQUE problem by searching the entire complemented graph. Depending on the maximum degree of the graph, $\Delta(G)$, it can be asymptotically more efficient to process some subset of complemented subgraphs.

7.2.1 Vertex cover-based maximum clique algorithm

We can in fact derive an algorithm whereby we extract the subgraph $G(N[v])$, form its complement, $\overline{G}(N[v])$, and determine a minimum vertex cover for this subgraph

(see Figure 7.1). A maximum independent set for $\overline{G}(N[v])$ is the largest of all cliques in G of which v is a member. Although determining such a local maximum clique remains of exponential time complexity, the benefit arises from limiting the size of the complemented subgraph that is searched. If our candidate vertex v is a member of the maximum clique then we can determine the maximum clique without having to search the entire complemented graph $\overline{G} = (V, \overline{E})$.

Algorithm description

The algorithm consists of the following steps:

1. Given an input graph, $G = (V, E)$, randomly choose some vertex, v , $\deg(v) > 2$, and extract the subgraph $G(N[v])$.
2. Generate the complement of subgraph $G(N[v])$, denoted $\overline{G}(N[v])$.
3. Determine a minimum vertex cover of subgraph $\overline{G}(N[v])$.
4. Generate the maximum clique by extracting the corresponding maximum independent set.
5. Repeat at Step (1) for some fixed number of iterations.

For some instances, the subgraph $G(N[v])$ extracted in Step (2) may be almost as large as the graph G . In such a case, we use modified versions of steps (2), (3), (4), repeated some fixed number of times:

- 2a. Randomly choose a vertex, $z \in G(N[v])$, $\deg(z) > 2$, and extract the subgraph $G(N[v] \cap N[z])$;
- 2b. generate the complemented subgraph $\overline{G}(N[v] \cap N[z])$;
- 2c. determine a minimum vertex cover of subgraph $\overline{G}(N[v] \cap N[z])$.
3. Generate the maximum clique by extracting the corresponding maximum independent set.
4. Repeat at Step (2a) for some fixed number of iterations.

This two-level sampling assumes that if the initial guess of v was correct then we are presented with even better probability of guessing a vertex that is also in the clique, $z \in N(v)$, within the smaller subgraph. The approach accelerates the process of calculating the minimum vertex cover by further restricting the size of the complemented subgraph (see Figure 7.2). In the case of the heuristic algorithm, it helps generate more precise solutions by limiting the subset of vertices that are inspected by the heuristic.

If we iterate through all vertices, solving for the minimum vertex cover in each subgraph, we will find the maximum clique for the graph. On the other hand, one can compromise between accuracy and speed: the more random guesses that are made, the more confident one becomes that the largest clique has been (or will be) found. As we will see, this contrasts with other search algorithms that can fail to find a clique within some allotted time, or can not improve on the largest clique no matter how much time is allowed.

7.2.2 Heuristic maximum clique algorithm

As with most heuristics, one is willing to trade-off the cost of attempting to find an optimal solution for an algorithm of lower complexity that returns a possibly suboptimal solution quickly. Because examining the complemented subgraph still requires time exponential in the maximum degree of the graph, it is interesting to consider heuristic solutions to those subproblems.

Algorithm description

In our initial conception, we considered a simple vertex cover heuristic to accelerate the algorithm, in place of the exact k -vertex cover algorithm. The heuristic consisted of removing the vertex with highest degree (and all incident edges), and adding it to the vertex cover. The final implemented version of the algorithm consisted of the same steps as shown in Section 7.2.1, except that we use the heuristic below to calculate an independent set (cf. ([148, 43])) for subgraph $\overline{G}(N[v] \cap N[z])$.

1. Select vertex, $w \in N[v] \cap N[z]$, where w has minimum degree.

2. Add w to the independent set.
3. Remove $N[w]$ from $\overline{G}(N[v] \cap N[z])$.
4. Repeat at Step (1) until $\overline{G}(N[v] \cap N[z])$ has no edges.

Ideally, any sophisticated heuristic/approximation scheme could be used in this case, producing possibly improved results (within approximation limits, of course).

7.3 Experimental results

We describe in this section the experimental results of our algorithms on both real-world data sets and benchmark data sets.

7.3.1 Methodology

We use the following methodology in the experimental phase of our research. In the exact parameterized setting, we compare our algorithm that combines k -vertex cover with sampling, to a conventional k -vertex cover-based approach. Both are parallel implementations and so we make a relative comparison between these two FPT algorithms.

We next compare a (sequential) heuristic version of our algorithm to the current best exact implementations for the MAXIMUM CLIQUE problem. We adopt this approach for the following reasons:

1. While there are no current experimental evaluations between heuristic algorithms, Regim [153] recently published an admirable tabulation of exact algorithms.
2. We wish to investigate the quality of the solutions obtained by the heuristic approach, while noting the time required to obtain the solutions. Clearly the exact algorithms perform much more work than the heuristic and so run time comparisons, while interesting, are not meant to indicate superiority.

We found that in practice the vertex cover heuristic, described in Section 7.2.2, generally induced unspectacular clique sizes compared to the known solutions (similar

results are observed in [19]). In contrast, we experimented with the simple independent set heuristic and it produced surprisingly better results. The choice of this heuristic for our experiments was based on its simplicity and ease of implementation, although ultimately any sophisticated heuristic/approximation would do. This particular heuristic in fact corresponds to the simple approach: select a vertex with highest degree (in the original graph), add it to the clique, delete all its non-neighboring vertices, and repeat until the remaining graph is empty (cf. (Johnson [107])).

In general, algorithms for finding maximum cliques rely on massive experimentation to determine their effectiveness. To expedite our experiments, we make use of two separate platforms. Our first experimental platform consisted of a shared-memory SunFire 6800, configured with 20 900MHz UltraSPARC-III processors and 20GB of memory. Our second experimental platform consisted of a 2.0GHz Intel Xeon processor with 512MB RAM and 60GB of disk storage. Execution time was measured as wall clock time in seconds and includes the time taken to read the input graph from a file and output the solution size.

A *run* consists of a single execution of the code, given some sampling parameters, out of which the size of the largest clique found is output. An *experiment* consists of some number of runs on a given instance and taking the largest clique produced out of all the runs. During a run on instance, $G_i = (V_i, E_i)$, the code generates a random sample set of vertices, $S_1 \subseteq V_i$, $|S_1| = \alpha_1|V_i|$. For each vertex $v \in S_1$, we generate a second random sample set, $S_2 \subseteq N(v)$, $|S_2| = \alpha_2|S_1|$. Finally, for each $z \in S_2$ we extract the subgraph induced by $G(N[v] \cap N[z])$, complement the subgraph, and extract the maximum independent set. This clique is then checked in the original graph to ensure its correctness.

It is difficult to directly compare the algorithm of Section 7.2.1 with the results in [1, 2, 16], as they have implemented a different kernelization algorithm. Instead we use a readily-available FPT k -vertex cover algorithm described in [38] and compare it against itself, i.e., we measure the performance of our k -vertex cover code running on a complemented graph instance versus our new maximum clique algorithm. This provides a relative performance comparison that carries over to any k -vertex cover algorithm. Because of the time required to calculate an exact minimum vertex cover,

Graph	Largest clique	Density	$ V $	$ E $
Somatostatin	14	0.22	559	33652
WW	22	0.45	425	40182
Thrombin (10)	24	0.19	646	40717

Table 7.1: Graph instances derived from experimental biological data (1).

we restrict our experiment to consist of one run for the parallel k -vertex cover algorithm using 9 processors on the first platform. This is compared to one run of the maximum clique algorithm on the same input with sampling parameters $\alpha_1 = 0.10$ and $\alpha_2 = 0.05$. We justify using the parallel platform in this series because the time to find a minimum vertex cover of a complemented graph can become significantly large and we wish to conduct the experiments in a reasonable time.

Given the shorter execution time of the heuristic algorithm, each experiment is repeated sequentially 50 times on each graph instance using our second platform. We consider an experiment to consist of 10 runs of the heuristic, and the time for an experiment is taken as the average time of the 10 runs. We take as sampling parameters $\alpha_1 = \alpha_2 = 0.10$. We compare the heuristic performance with implementations of the algorithms of Wood [189], Östegard [145], Fahle [70], and Regin [153]. Our platform closely resembles that of Regin¹ so we can draw directly upon his observations to evaluate our algorithm against previous work. In Section 7.3.4 we present a table adapted from [153] that includes measurements for all four algorithms.

7.3.2 Data sets

We used two different types of graphs for our experiments with the objective of studying the performance of the algorithms using both real-world and public domain benchmark data. The first set of graphs, Table 7.1, comprise a small subset selected from our experiments in [38, 39]. The second set, Table 7.2, are those cited in Baldwin, et al. [16] from which the authors derived phylogenetic trees based on proteins domains. The third set comprise the atendentious DIMACS benchmark [57].

¹His platform was an Intel Pentium IV, 2.0GHz, 512MB RAM.

Graph	Largest clique	Density	$ V $	$ E $
globin10	12	0.16	972	75,386
globin15	23	0.32	972	149,473
pp_sh2-3	3	0.05	148	494
pp_sh2-10	17	0.27	726	69,982
sh2-5	7	0.08	839	26,612
sh2-10	23	0.37	839	129,697

Table 7.2: Graph instances derived from experimental biological data (2).

7.3.3 FPT k -vertex cover maximum clique algorithm performance

The objective of this experiment is to demonstrate the efficacy of the algorithm from Section 7.2.1. We selected the instances *WW*, *Somatostatin*, and *Thrombin*, and used the parallel k -vertex cover code described in [38, 39], running on 9 processors.

For each graph instance we measure the total execution time, the time required before a clique matching the optimal size is found, and the number of times a clique matching the optimal size is found. We emphasize the relative speed with which an optimal clique is found rather than the actual run times. In Table 7.3 we observe that the sampling algorithm completed in a shorter time than that required to find a minimum vertex cover in a complemented graph. The experiment with the *Thrombin* instance highlights one of the benefits of the new maximum clique algorithm. The new algorithm generated several cliques of size 23 within 48 hours of starting and ultimately found several cliques matching the optimal size before termination. In contrast, the algorithm searching for a minimum vertex cover in the complemented graph had in that same time only processed two cover instances (sizes $k = 623$ and $k = 621$). The execution on the complemented graph was terminated after 2 weeks, never having completed its search.

7.3.4 Maximum clique heuristic performance

The software we developed for these experiments used libraries of graph and set manipulation routines made available by Östegard [146].

We see from Table 7.4 that the heuristic performed excellently on real-world data, with the exception of instance *Sh2-10*. In the latter case the average clique size

Graph	Largest clique	Conventional Total time (hr)	Total time (hr)	Sampling and local complementation	
				Time when clique matching largest if first found	Number of cliques matching largest size
Somatostatin	14	02:18:37	00:29:40	00:03:03	7
WW	22	43:34:15	19:33:12	02:45:01	21
Thrombin (10)	24	DNF	198:42:11	116:39:45	10

Table 7.3: Summary of k -vertex cover-based maximum clique algorithm results using biological data sets. The algorithm with sampling and local complementation used sampling parameters $\alpha_1 = 0.10$, $\alpha_2 = 0.05$. *DNF* denotes the execution was terminated after two weeks.

Graph	Largest clique	Average clique size	Average run time (s)	Average experiment run time (s)	Number of times largest was found
globin10	12	12.00	4.52	45.22	50
globin15	23	22.57	29.87	298.70	50
pp.sh2-3	3	1.81	0.01	0.10	50
pp.sh2-10	17	16.92	6.81	68.06	50
sh2-5	7	6.26	0.27	2.72	48
sh2-10	23	22.00	28.91	289.10	11

Table 7.4: Summary of heuristic algorithm results using biological data sets. The experiments were repeated 50 times for each graph instance.

produced by the heuristic was very close to the optimal.

We follow the methodology of Regin for testing the performance of the heuristic algorithm against the DIMACS clique benchmark set, i.e., code execution on a given instance taking longer than 14,400 seconds is terminated. Table 7.5 summarizes the performance data for Wood, Östegard, Fahle, and Regin’s algorithms for the benchmark (adapted from [153]). The timing measurements have been scaled appropriately to compensate for different processor speeds.

We divide the data sets into two groups, based on the amount of time they require to complete the series of experiments within the 4 hour time limit. For each graph instance we measure the average clique size found during the 50 experiments, the average time of all the runs, the average time for the experiments, and a count of the number of times that an experiment found a clique that matched the largest known size. Table 7.6 consists of instances that were sampled with $\alpha_1 = \alpha_2 = 0.10$. The same methodology is used to test the algorithm on instances in Table 7.2. As the instances in Table 7.7 were large, we limited the sampling to $\alpha_1 = 0.01$ and $\alpha_2 = 0.05$. In the tables, we list in boldface entries of particular interest.

In comparing the results in Table 7.5 and Table 7.6 we see the heuristic algorithm consistently outperforms the algorithms of Wood, Östegard, and Fahle, and is faster

DIMACS		Wood		Ostegard		Fahle		Regin	
Graph	K	K	Time (s)	K	Time (s)	K	Time (s)	K	Time (s)
brock400.1	27		fail		fail	> 24	fail	27	11,340.80
brock400.2	29		fail		fail	> 29	fail	29	7,910.60
brock400.3	31		fail		fail	> 24	fail	31	4,477.23
brock400.4	33		fail		fail	> 25	fail	33	6,051.77
brock800.1	23		fail		fail	> 21	fail	> 21	fail
brock800.2	24		fail		fail	> 20	fail	> 20	fail
brock800.3	25		fail		fail	> 20	fail	> 20	fail
brock800.4	26		fail		fail	> 20	fail	> 20	fail
hamming10-2	512	512	0	512	0.84	512	5.16	512	1.04
hamming10-4	≥ 40		fail		fail	> 32	fail	> 40	fail
johnson16-2-4	8	8	13.05	8	0.09	8	7.91	8	3.80
johnson32-2-4	≥ 16		fail		fail	> 16	fail	> 16	fail
johnson8-2-4	4	4	0	4	0	4	0	4	0
johnson8-4-4	14	14	0	14	0	14	0.03	14	0
keller4	11	11	1.23	11	0.17	11	2.53	11	0.50
keller5	27		fail		fail	> 25	fail	> 27	fail
keller6	≥ 59		fail		fail	> 43	fail	> 54	fail
MANN_a9	16	16	0	16	0	16	0	16	0
MANN_a27	126	126	46.95		fail	126	10,348.87	126	18.48
MANN_a81	≥ 1,100		fail		fail	> 996	fail	≥ 1,100	fail
p_hat500-1	9	9	0.91	9	0.10	9	0.60	9	2.30
p_hat500-2	36	36	17.81	36	142.93	36	203.93	36	32.69
p_hat500-3	50		fail		fail	> 48	fail	50	12,744.70
p_hat700-1	11	11	2.69	11	0.22	11	2.67	11	6.01
p_hat700-2	44		fail		fail	44	2,086.63	44	255.79
p_hat700-3	≥ 62		fail		fail	> 54	fail	≥ 62	fail
p_hat1000-1	10	10	18.88	10	1.95	10	16.43	10	27.80
p_hat1000-2	46		fail		fail	> 44	fail	46	16,845.70
p_hat1000-3	≥ 68		fail		fail	> 50	fail	≥ 66	fail
p_hat1500-1	12		fail		fail	12	119.77	12	480.84
p_hat1500-2	≥ 65		fail		fail	> 52	fail	≥ 63	fail
san1000	15	15	43.59	15	0.17	15	3,044.09	15	102.80
san400.0.5.1	13	13	0.75	13	0	13	6.74	13	1.19
san400.0.7.1	40	40	13.25		fail	40	425.99	40	23.28
san400.0.7.2	30	30	415.12	30	168.70	30	159.72	30	67.53
san400.0.7.3	22		fail		fail	22	617.07	22	273.23
san400.0.9.1	100		fail		fail	100	7,219.53	100	1,700.00
sanr200.0.7		18	22.50	18	4.70	18	24.99	18	4.30
sanr200.0.9			fail		fail	> 41	fail	42	150.08
sanr400.0.5		13	22.55	13	2.21	13	23.09	13	17.12
sanr400.0.7			fail		fail	21	15,925.00	21	3,139.11

Table 7.5: Performance comparison of recent maximum clique algorithms. *K* denotes the largest known clique size, or the largest clique found at the time of termination. *Fail* indicates execution exceeded the four hour time constraint.

than that of Regin while producing a competitive number of optimal or similar answers. While the shorter run times are not unexpected, the quality of the results are quite surprising. In general, when the heuristic fails it does so for the same instances as does the current best algorithm, while generating comparable suboptimal answers in a fraction of the time. For example, the solution to the *Brock800.1* instance matches the current best suboptimal answer, but required much less time to compute (80 seconds). Another such instance is *P_hat700-3*, where the heuristic found a solution matching the best lower bound in less than a minute. Regin's algorithm exceeded the 4 hour time limit while finding a solution of the same size. There are many cases where the heuristic finds the largest clique extremely quickly, whereas the algorithm of Regin either fails to terminate within the 4 hour time limit (e.g., *Johnson32-2-4*, *Keller5*) or takes noticeably longer (e.g., *San400.0.9.1*, *Sanr400.0.7*).

Graph	Largest clique	Average clique size	Average run time (s)	Average experiment run time (s)	Number of times largest was found
brock400_1	27	24.08	8.17	81.68	0
brock400_2	29	24.35	8.17	81.70	17
brock400_3	31	24.61	8.13	81.32	22
brock400_4	33	26.49	8.19	81.92	47
brock800_1	23	20.81	225.41	2,254.14	0
brock800_2	24	20.78	223.14	2,231.40	0
brock800_3	25	21.13	245.43	2,454.32	0
brock800_4	26	20.55	221.80	2,218.06	0
hamming10-2	512	512.00	1,900.45	19,004.48	50
hamming10-4	≥ 40	36.00	439.04	4,390.44	0
johnson16-2-4	8	8.00	0.09	0.90	50
johnson32-2-4	≥ 16	16.00	25.30	253.04	50
johnson8-2-4	4	4.00	0.01	0.08	50
johnson8-4-4	14	14.00	0.02	0.22	50
keller4	11	11.00	0.20	2.00	50
keller5	27	27.00	106.67	1,066.74	50
MANN_a9	16	16.00	0.01	0.14	50
MANN_a27	126	125.00	24.95	249.48	0
p_hat500-1	9	9.00	0.60	6.00	50
p_hat500-2	36	35.66	6.59	65.92	50
p_hat500-3	50	49.02	21.81	218.10	37
p_hat700-1	11	10.29	1.94	19.38	50
p_hat700-2	44	43.99	24.74	247.44	50
p_hat700-3	≥ 62	60.26	4.50	44.96	25
p_hat1000-1	10	10.00	6.81	68.14	50
p_hat1000-2	46	45.72	91.62	916.20	50
p_hat1000-3	≥ 68	65.08	331.23	3,312.30	0
p_hat1500-1	12	11.22	35.58	355.76	45
p_hat1500-2	65	64.09	514.32	5,143.22	34
san1000	15	10.48	5.09	50.94	29
san400_0.5_1	13	11.60	2.46	24.60	50
san400_0.7_1	40	39.94	6.63	66.26	50
san400_0.7_2	30	29.62	6.46	64.56	50
san400_0.7_3	22	20.31	6.43	64.28	50
san400_0.9_1	100	100.00	16.42	164.18	50
sanr200_0.7	18	17.77	0.45	4.54	50
sanr200_0.9	42	41.37	1.08	10.76	50
sanr400_0.5	13	12.58	1.91	19.12	50
sanr400_0.7	21	20.82	6.46	64.56	50

Table 7.6: Summary of heuristic algorithm results using DIMACS benchmark. The experiments were repeated 50 times for each graph instance.

Graph	Largest clique	Average clique size	Average run time (s)	Average experiment run time (s)	Number of times largest was found
keller6	≥ 59	55.00	1,007.27	10,072.70	0
MANN_a81	$\geq 1,100$	1,096.00	13,961.40	139,614.00	0

Table 7.7: Summary of heuristic algorithm results using large DIMACS instances. The experiments were repeated 50 times for each graph instance.

Most importantly, the heuristic improves on the best lower bound for four as yet unsolved instances, and solves a fifth instance optimally. For the instance *Brock800_3*, it found a clique of size 22. The previous best lower bound for these graphs was 20. For the instances *Brock800_2* and *Brock800_4* the heuristic found cliques of size 21 in each graph. The heuristic found a clique of size 55 in the *Keller6* instance, where previously the best suboptimal clique size was 54. For the instance *P_hat1500-2* the heuristic found a clique matching the optimal size of 65. The best lower bound until now was size 63 as determined by Regin.

We observe two cases of oversampling. When solving the *Hamming10-2* instance

(Table 7.6), the algorithm found a clique matching the largest known size very quickly, but it continued to search through its rather large sample set, incurring an unnecessarily long runtime. In the case of the *MANN_a81* instance (Table 7.7), the heuristic found a relatively large clique in one run within the time limit, although the experiment exceeded the time limit.

7.4 Discussion of experimental results

It is important to note that the effectiveness of the algorithms are a function of the maximum clique size and the sampling rate. Cases where the maximum clique size is small and there are many such cliques, or the maximum clique is large (in relation to the vertex set), are clearly the most favorable circumstances. The contrary situations can be ameliorated with higher sampling at the cost of the longer execution time. Unless we iterate through all the vertices and solve each subgraph exactly, as in the algorithm introduced in Section 7.2.1, sampling is not guaranteed to find the maximum clique. The motivation for the experimental work is to verify the effectiveness of the approach in practice, particularly using real-world data sets, and to a certain extent the experiments do speak for themselves.

As mentioned in Section 7.2.2, the vertex cover heuristic generally produced unspectacular clique sizes compared to those generated by the independent set heuristic. The dominant factor determining the quality of a solution is the probability of error at each step, dependent on the number of choices available. The subtle algorithmic difference was significant in that the vertex cover heuristic required many more steps to determine a cover for the subgraph and, at each step in the heuristic, there were many possible choices of vertices of high degree. On the other hand, the independent set heuristic required fewer steps to complete and, at each step, there were generally very few vertices of minimum degree to choose from. (Alternately, in the original subgraph, the vertices of high degree were generally members of the clique.) Although one may call into question the variety and hardness of the DIMACS instances given the success of the simple heuristic, experiments on the real-world data sets nonetheless demonstrate some practicality of the approach. Finally, given more sophisticated heuristic/approximation schemes, one can expect higher quality solutions, again at

the expense of more time.

7.5 Summary

We described an improved algorithm for solving the MAXIMUM CLIQUE problem in a graph using a novel combination of sampling with a parameterized k -vertex cover algorithm.

We also experimented with a very effective heuristic for finding a large clique that combines our sampling approach with fast independent set approximation. In experiments using the DIMACS benchmark, the heuristical approach established new lower bounds for four instances and provides the first optimal solution for an instance unsolved until now. The heuristic competitively matched the accuracy of the current best exact algorithm in terms of correct solutions, while requiring a fraction of the runtime.

Chapter 8

Conclusion and Future Work

*One moment's high, and glory rolls on by; Like a streak of lightning;
That flashes and fades in the summer sky*

-“Marathon”, Rush

We now summarize the contributions made in this thesis, and outline several possible areas for future research. With a need for algorithms that apply to solving practical real-world problems, greater and greater emphasis is being placed on parallel computation to bridge the gap between theory, efficiency and practicality. Our work has focused on establishing a parallel framework for future FPT algorithms, with a notable emphasis on the k -VERTEX COVER problem. k -VERTEX COVER is a core problem in parameterized complexity, and is a fundamental tool for solving other intractable problems such as the MAXIMUM INDEPENDENT SET and MAXIMUM CLIQUE problems.

8.1 Contribution summary

The contributions to parallel FPT are:

- We develop the first parallel FPT algorithm to solve the k -VERTEX COVER problem (published in *Journal of Computer and System Sciences*, 2003 [38]).
 - In practical experiments, this algorithm solved the largest instances of k -vertex cover known to the FPT research community at the time of publication.
 - This algorithm uses an implicit load-balancing strategy that serves as a framework for future k -vertex cover algorithms.

- We undertake the first investigation of parallel memorization.
 - We developed a refined technique for generating the lookup table in a parallel setting, resulting a more time and space efficient algorithm.
 - We also introduced the first CGM algorithm for parallel memorization.

The contributions to the VERTEX COVER problem are:

- We develop an improved sequential FPT k -vertex cover algorithm for graphs where the maximum degree of the vertices is bounded by 4.
 - This algorithm uses minimum cuts for generating components and is faster and more flexible compared to previous separator-based approaches that use complex dynamic programming and tree-decomposition. The new algorithm adapts traditional branching rules and calculates a minimum cut by using a network flow algorithm that is already a subroutine of the kernelization algorithm.
 - All current best k -vertex cover algorithms for general graphs reduce the input graph to one of bounded degree during processing. Our bounded-degree algorithm can therefore be used as a subroutine to improve algorithms for general graphs.
 - When a suitably small separator can be found in the graph, the k -vertex cover algorithm can be used to solve the MAXIMUM INDEPENDENT SET problem in graphs of degree bounded by 4, with complexity comparable to the current best algorithm.

The contributions to the CONSTRAINED BIPARTITE MINIMUM VERTEX COVER problem are:

- We present CGM algorithms for parallelizing the algorithm of Chen and Kanj [44].
 - This includes developing parallel algorithms for the Gallai-Edmonds decomposition and the Dulmage-Mendelsohn decomposition in bipartite graphs.

- We also present a CGM algorithm for computing a Gallai-Edmonds decomposition in a general graph.

The contributions to the MAXIMUM CLIQUE problem are:

- We make use of an observation on the relationship between a maximum clique in a graph, and the maximum independent set in its complement. We develop a new sampling strategy to exploit this property, and the practical implications of our observation are put to use in the new algorithms that follow (published in *Algorithmic Aspects in Information and Management (AAIM)*, 2006 [179]).
- We develop an improved exact algorithm for solving the MAXIMUM CLIQUE problem using a novel combination of sampling with our k -vertex cover algorithm.
 - This algorithm has a superior time complexity compared to previous FPT approaches that search for a minimum vertex cover in the complement of the input graph.
 - In practical experiments, the new algorithm outperformed the conventional FPT approach when run on datasets derived from biological research.
- We develop a new heuristic for finding a maximum clique based on our sampling approach.
 - In experiments using the DIMACS benchmark, the heuristic established new lower bounds for four instances and provides the first optimal solution for an instance unsolved until now.
 - The heuristic competitively matched the accuracy of the current best exact algorithm in terms of correct solutions. In almost all cases, the heuristic required a fraction of the time to find the solutions.
 - The heuristic was found to perform equally well on real-world datasets derived from biological research.

8.2 Future work

The contributions in this thesis can be further extended in several interesting and fruitful directions. One possible avenue is to investigate combining memorization with automorphism detection. Because the subgraphs being stored in the lookup table are small, it is likely that many will in fact possess the same structure. Given a set of small subgraphs that are isomorphic, it would be very beneficial to store a single copy, with its corresponding minimum cover, rather than each instance. This would reduce the size of the table significantly.

From practical experience, it is evident that not all problem instances, considered equivalent under the fixed-parameter tractable umbrella, require the same time bounds to solve. For example, in the case of k -vertex cover instances, it is frequently observed that two graphs, $\langle G'_1, k'_1 \rangle$ and $\langle G'_2, k'_2 \rangle$, where $k'_1 = k'_2$, can have widely divergent runtimes. As heard time and again, asymptotic complexity analysis does not capture the practical effort required to solve certain instances. One might ask if there exists a means of further classifying instances within *FPT* for certain problems.

On the surface, the CONSTRAINED MINIMUM BIPARTITE VERTEX COVER problem looks like a good candidate to which one can apply the memorization technique to improve the complexity. Unlike the k -VERTEX COVER problem, there are certain limitations. The result of Chandran and Grandoni relies on an important theorem of Robson that bounds the number of induced connected subgraphs where the minimum degree is 2 and the maximum degree is 6. Because the algorithm of Chen and Kanj uses a weighted, directed, acyclic graph to guide the branching in the search tree phase, there is no non-trivial degree bound on the vertices in the bipartite graph. Similarly, there are no useful bounds on the in-degree and out-degree of the node in the DAG. It would be very useful to derive tight asymptotic bounds on the number of induced, connected subgraph in bipartite graph, in a nature similar to Robson's result. It should also may be possible to improve the complexity of the search tree phase, either asymptotically or in practice, by exploiting a separator theorem for directed, acyclic graphs (e.g., [152]).

The results from the MAXIMUM CLIQUE research indicate that the heuristic would be beneficial to any exact algorithm by providing a very fast, accurate lower bound

on the largest clique. It would be important to investigate how much better these exact algorithms perform, given this preprocessing step that helps them converge more quickly to a solution.

Bibliography

- [1] F.N. Abu-Khzam, M.A. Langston, P. Shanbhag. “Scalable parallel algorithms for difficult combinatorial problems: A case study in optimization”. In *Proceedings of the International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, pp.563–568, 2003.
- [2] F.N. Abu-Khzam, R.L. Collins, M.R. Fellows, M.A. Langston, W.H. Suters, C.T. Symons. “Kernelization algorithms for the vertex cover problem: Theory and experiments”. In *Proceedings of the ACM-SIAM Workshop on Algorithm Engineering and Experiments (ALENEX 2004)*, pp.62–69, 2004.
- [3] F.N. Abu-Khzam, M.A. Langston, W.H. Suters. “Fast, effective vertex cover kernelization: A tale of two Algorithms”. In *Proceedings of the International Conference on Computer Systems and Applications (ICCSA 2005)*, 2005.
- [4] A. Aho, J. Hopcroft, J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [5] S.G. Akl, D.R. Gries, I. Stojmenović. “An optimal parallel algorithm for generating combinations”. In *Information Processing Letters*, Vol.33, pp.135–139, 1990.
- [6] J. Alber, H. Fernau, R. Niedermeier. “Parameterized complexity: Exponential speed-up for planar graph problems”. In *Journal of Algorithms*, Vol.52, pp.26–56, 2004.
- [7] J. Alber, H. Fernau, R. Niedermeier. “Graph separators: A parameterized view”. In *Journal of Computer and System Science*, Vol.67, pp.808–832, 2003.
- [8] J. Alber, H.L. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier. “Fixed-parameter algorithms for dominating set and related problems on planar graphs”. In *Algorithmica*, Vol.33, pp.461–493, 2002.
- [9] L. Aleksandrov, H. Djidjev. “Linear algorithms for partitioning embedded graphs of bounded genus”. In *SIAM Journal of Discrete Mathematics*, Vol.9, pp.129–150, 1996.
- [10] F. Alizadeh. “A sublinear-time randomized parallel algorithm for the maximum clique problem in perfect graphs”. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 1991)*, Vol.2, pp.188–194, 1991.
- [11] N. Alon, P.D. Seymour, R. Thomas. “A separator theorem for graphs with an excluded minor and its applications”. In *Proceedings of the ACM Symposium on the Theory of Computing (STOC 1990)*, pp.293–299, 1990.

- [12] H. Alt, N. Blum, K. Mehlhorn, M. Paul. "Computing a maximum cardinality matching in a bipartite graph in time $O(n^{1.5}\sqrt{m/\log n})$ ". In *Information Processing Letters*, Vol.37, pp.237–240, 1991.
- [13] R. Anderson, J.C. Setubal. "A parallel implementation of the push-relabel algorithm for the maximum flow problem". In *Journal of Parallel and Distributed Computing*, Vol.29, pp.17–26, 1995.
- [14] B.S. Baker. "Approximation algorithms for *NP*-complete problems on planar graphs". In *Journal of the ACM*, Vol.143, pp.153–180, 1994.
- [15] R. Balasubramanian, M.R. Fellows, V. Raman. "An improved fixed-parameter algorithm for vertex cover". In *Information Processing Letters*, Vol.65, pp.163–168, 1998.
- [16] N.E. Baldwin, R.L. Collins, M.R. Leuze, M.A. Langston, C.T. Symons, B.H. Voy. "High-performance computational tools for motif discovery". In *Proceedings of the IEEE International Workshop on High Performance Computational Biology*, April, 2004.
- [17] E. Balas, J. Xue. "Weighted and unweighted maximum clique algorithms with upper bounds from fractional coloring". In *Algorithmica*, Vol.15, pp.397–412, 1996.
- [18] R. Bar-Yehuda, S. Even. "A local-ratio theorem for approximating the weighted vertex cover problem". In *Annals of Discrete Mathematics*, Vol.25, pp.27–48, 1985.
- [19] R. Bar-Yehuda, V. Dabholkar, K. Govindarajan, D. Sivakumar. "Randomized local approximations with applications to the MAX-CLIQUE problem". Technical Report 93-30, University at Buffalo, 1993.
- [20] D.J. Becker, T. Sterling, D. Savarese, J.E. Dorband, U.A. Ranawak, C.V. Packer. "Beowulf: A parallel workstation for scientific computation". In *Proceedings of the International Conference on Parallel Processing (ICPP 1995)*, 1995.
- [21] R. Beigel. "Finding maximum independent sets in sparse and general graphs". In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA 1999)*, pp.856–857, 1999.
- [22] D. Bienstock, C.L. Monma. "On the complexity of embedding planar graphs to minimize certain distance measures". In *Algorithmica*, Vol.5, pp.93–109, 1990.
- [23] H.L. Bodlaender, R.G. Downey, M.R. Fellows. "Applications of parameterized complexity to problems of parallel and distributed computation". Unpublished extended abstract, 1994.

- [24] H.L. Bodlaender. “A partial k -arboretum of graphs with bounded treewidth”. In *Theoretical Computer Science*, Vol.209 , pp.1–45, 1998.
- [25] B. Bollobás. *Extremal Graph Theory*. Academic Press, 1978.
- [26] I.M. Bomze, M. Budinich, P.M. Pardalos, M. Pelillo. “The maximum clique problem”. In *Handbook of Combinatorial Optimization*, Du, Pardalos (Eds.), Vol.A, Kluwer, pp.1–74, 1999.
- [27] P. Bose, A. Chan, F. Dehne, M. Latzel. “Coarse grained parallel maximum matching in convex bipartite graphs”. In *Proceedings of the Parallel Processing Symposium (IPPS 1999)*, pp.125–129, 1999.
- [28] A. Brügger, A. Marzetta, K. Fukuda, J. Nievergelt. “The parallel search bench ZRAM and its applications”. In *Annals of Operations Research*, Vol.90, pp.45–63, 1999.
- [29] T.N. Bui, S. Chaudhuri, F.T. Leighton, M. Sipser. “Graph bisection algorithms with good average case behavior”. In *Combinatorica*, Vol.7, pp.171–191, 1987.
- [30] J.F. Buss, J. Goldsmith. “Nondeterminism within P ”. In *SIAM Journal of Computing*, Vol.22, pp.560–572, 1993.
- [31] L. Cai, J. Chen, R.G. Downey and M.R. Fellows. “On the parameterized complexity of short computation and factorization”. In *Archive For Mathematical Logic*, Vol.36, pp.321–337, 1997.
- [32] L. Carter, L. Stockmeyer, M. Wegman. “The complexity of backtrack searches”. In *Proceedings of the ACM-SIAM Symposium on Theory of Computing (STOC 1985)*, pp.449–457, 1985.
- [33] M. Cesati, M. Di Ianni. “Parameterized parallel complexity”. In *Proceedings of the International Euro-Par Conference*, pp.892–896, 1998.
- [34] M. Cesati. “PERFECT CODE is $W[1]$ -complete”. In *Information Processing Letters*, Vol.81, pp.163–168, 2002.
- [35] M. Cesati. “The Turing way to parameterized complexity”. In *Journal of Computer and System Sciences*, Vol.67, pp.654–685, 2003.
- [36] A. Chan, F. Dehne. “A note on coarse grained parallel integer sorting”. In *Proceedings of the Annual International Symposium on High Performance Computers (HPCS 1999)*, pp.261–267, 1999.
- [37] L.S. Chandran, F. Grandoni. “Refined memorization for vertex cover”. In *Information Processing Letters*, Vol.93, pp.125–131, 2005.

- [38] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, P.J. Taillon. "Solving large FPT problems on coarse grained parallel machines". In *Journal of Computer and System Sciences*, Vol.67, pp.691–706, 2003.
- [39] J. Cheetham, F. Dehne, A. Rau-Chaplin, U. Stege, P.J. Taillon. "A parallel FPT application for clusters". In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pp.70–77, 2003.
- [40] J. Cheetham, F. Dehne, S. Pitre, A. Rau-Chaplin, P.J. Taillon. "Parallel CLUSTAL W for PC clusters". In *Proceedings of the International Conference on Computational Science and Its Applications (ICCSA 2003)*, LNCS 2667, pp.300–309, 2003.
- [41] J. Cheetham. Private communication, 2004.
- [42] J. Chen, L. Liu, W. Jia. "Improvement on vertex cover for low-degree graphs". In *Networks*, Vol.35, pp.253–259, 2000.
- [43] J. Chen, I. Kanj, W. Jia. "Vertex cover: Further observations and further improvements". In *Journal of Algorithms*, Vol.41, pp.280–301, 2001.
- [44] J. Chen, I.A. Kanj. "Constrained minimum vertex cover in bipartite graphs: complexity and parameterized algorithms". In *Journal of Computer and System Sciences*, Vol.67, pp.833–847, 2003.
- [45] J. Chen, I.A. Kanj, G. Xia. "Simplicity is beauty: Improved upper bounds for vertex cover". Technical Report 05-008, DePaul University, 2005.
- [46] J. Cheriyan. "Randomized $\tilde{O}(M(|V|))$ algorithms for problems in matching theory". In *SIAM Journal of Computing*, Vol.26, pp.1635–1655, 1997.
- [47] B. Chor, M. Fellows, D. Juedes. "Linear kernels in linear time, or How to save k colors in $O(n^2)$ steps". In *Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer (WG 2004)*, LNCS 3353, pp.257–269, 2004.
- [48] S.A. Cook. "The complexity of theorem-proving procedures." In *Proceedings of the ACM-SIAM Symposium on Theory of Computing (STOC 1971)*, pp.151–158, 1971.
- [49] D. Coppersmith, S. Winograd. "Matrix multiplication via arithmetical progression". In *Journal of Symbolic Computations*, Vol.9, pp.251–280, 1990.
- [50] T.H. Cormen, C.E. Leiserson, R.L. Rivest. *Introduction to Algorithms*. MIT Press, 1990.
- [51] D.G. Corneil, F.F. Dragan, E. Köhler. "On the power of BFS to determine a graph's diameter". In *Networks*, Vol.42, pp.209–222, 2003.

- [52] F. Dehne. “Guest editor’s introduction”. In *Algorithmica Special Issue on Coarse Grained Parallel Algorithms*, Vol.24, pp.173–176, 1999.
- [53] F. Dehne, M. Fellows, F. Rosamond. “An FPT algorithm for set splitting”. In *Proceedings of the 29th Workshop on Graph Theoretic Concepts in Computer Science (WG 2003)*, LNCS 2880, pp.180–191, 2003.
- [54] E. Demaine, M. Hajiaghayi, D.M. Thilikos. “Exponential speedup of fixed parameter algorithms on $K_{3,3}$ -minor-free or K_5 -minor-free graphs”. In *Algorithmica*, Vol.41, pp.245–267, 2005.
- [55] E. Demaine, F.V. Fomin, M. Hajiaghayi, D.M. Thilikos. “Subexponential parameterized algorithms on graphs of bounded-genus and H -minor-free graphs”. In *Journal of the ACM*, Vol.52, pp.866–893, 2005.
- [56] L. Devroye. Private communication, 2001.
- [57] DIMACS clique benchmarks.
ftp://dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique.
- [58] E.A. Dinic. “Algorithm for solution of a problem of maximum flows in networks with power estimation”. In *Soviet Math. Dokl.*, Vol.11, pp.1277–1280, 1970.
- [59] R.G. Downey, V. Estivill-Castro, M.R. Fellows, E. Prieto, F.A. Rosamond. “Cutting up is hard to do: The parameterized complexity of k -cut and related problems”. In *Computing the Australasian Theory Symposium (CATS 2003)*, Electronic Notes in Theoretical Computer Science, Elsevier, Vol.78, 2003.
- [60] R.G. Downey, M.R. Fellows. “Fixed-parameter tractability and completeness”. In *Congressus Numerantium*, Vol.87, pp.161–187, 1992.
- [61] R.G. Downey, M.R. Fellows. “Parameterized computational feasibility”. In *Feasible Mathematics II*, Clote, Remmel (Eds.), Birkhauser, pp.219–244, 1995.
- [62] R.G. Downey, M.R. Fellows. “Fixed parameter tractability and completeness I: Basic theory”. In *SIAM Journal of Computing*, Vol.24, pp.873–921, 1995.
- [63] R.G. Downey, M.R. Fellows. “Fixed parameter tractability and completeness II: Completeness for $W[1]$ ”. In *Theoretical Computer Science A*, Vol.141, pp.109–131, 1995.
- [64] R.G. Downey, M.R. Fellows, K.W. Regan. “Parameterized circuit complexity and the W hierarchy”. In *Theoretical Computer Science A*, Vol.191, pp.91–115, 1998.
- [65] R.G. Downey, M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1998.

- [66] R.G. Downey, M.R. Fellows, U. Stege. "Parameterized complexity: A framework for systematically confronting computational intractability". In *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*, Roberts, Kratochvil, Nesetril (Eds.), *AMS-DIMACS Proceedings Series*, Vol.49, AMS, pp.49–99, 1999.
- [67] P. Erdős, R.L. Graham, E. Szemerédi. "On sparse graphs with dense long paths". Technical Report STAN-CS-75-504, Computer Science Department, Stanford University, 1975.
- [68] S. Even. *Graph Algorithms*. Computer Science Press, 1979.
- [69] A. Fabri, O. Devillers. "Scalable algorithms for bichromatic line segment intersection problems on coarse grained multicomputers". In *International Journal of Computational Geometry and Applications*, Vol.6, pp.487–506, 1996.
- [70] T. Fahle. "Simple and fast: Improving a branch-and-bound algorithm for maximum clique". In *10th Annual European Symposium (ESA 2002)*, pp.485–498, 2002.
- [71] U. Feige, R. Krauthgamer, K. Nissim. "On cutting a few vertices from a graph". In *Discrete Applied Mathematics*, Vol.127, pp.643–649, 2003.
- [72] M.R. Fellows. "On the complexity of vertex set problems". Technical Report, Computer Science Department, University of New Mexico, 1988.
- [73] M.R. Fellows. Private communication, 2004.
- [74] H. Fernau, R. Niedermeier. "An efficient exact algorithm for constraint bipartite vertex cover". In *Journal of Algorithms*, Vol.38, pp.374–410, 2001.
- [75] L.S. Filotti, G.L. Miller, J. Reif. "On determining the genus of a graph in $O(V^{O(g)})$ steps". In *Proceedings of the ACM-SIAM Symposium on Theory of Computing (STOC 1979)*, pp.27–37, 1979.
- [76] R.A. Finkel, U. Manber. "DIB - A distributed implementation of backtracking". In *ACM Transactions of Programming Languages and Systems*, Vol.9, pp.235–256, April 1987.
- [77] F. Focacci, A. Lodi, M. Milano. "Cost-based domain filtering". In *Proceedings of CP 1999*, LNCS 1713, pp. 189–203, 1999.
- [78] F.V. Fomin, D.M. Thilikos. "A simple and fast approach for solving problems on planar graphs". In *21st International Symposium on Theoretical Aspects of Computer Science (STACS 2004)*, LNCS 2996, pp.56–67, 2004.
- [79] F.V. Fomin, D.M. Thilikos. "New upper bounds on the decomposability of planar graphs". In *Journal of Graph Theory*, Vol.51, pp.53–81, 2006.

- [80] F.V. Fomin, K. Høie. "Pathwidth of cubic graphs and exact algorithms". In *Information Processing Letters*, Vol.97, pp.191–196, 2006.
- [81] M. Fürer. "A faster algorithm for finding maximum independent sets in sparse graphs". In *Proceedings of Theoretical Informatics: 7th Latin American Symposium (LATIN 2006)*, pp.491–501, 2006.
- [82] M. Garey, D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [83] A.V. Gerbessiotis. "Algorithmic and practical considerations for dense matrix computations on the BSP model". Technical Report PRG-TR-32-97, Computing Laboratory, University of Oxford, 1997.
- [84] J.R. Gilbert, J.P. Hutchinson, R.E. Tarjan. "A separator theorem for graphs of bounded genus". In *Journal of Algorithms*, Vol.5, pp.391–407, 1984.
- [85] J.R. Gilbert, D.J. Rose, A. Edenbrandt. "A separator theorem for chordal graphs". In *SIAM Journal of Algorithms and Discrete Methods*, Vol.5, pp.306–313, 1984.
- [86] A.V. Goldberg, S.A. Plotkin, D.B. Shmoys, E. Tardos. "Using interior point methods for fast parallel algorithms for bipartite matching and related problems". In *SIAM Journal of Computing*, Vol.21, pp.140–150, 1992.
- [87] A.V. Goldberg, S. Rao. "Beyond the flow decomposition barrier". In *Proceedings of the 38th IEEE Symposium on Foundations of Computer Science (FOCS 1997)*, pp.2–11, 1997.
- [88] O. Goldschmidt, D.S. Hochbaum. "A polynomial algorithm for the k -cut problem for fixed k ". In *Mathematics of Operations Research*, Vol.19, pp.24–37, 1994.
- [89] A. Grama, V. Kumar. "State-of-the-art in parallel search techniques for discrete optimization problems". In *IEEE Transactions on Knowledge and Data Engineering*, Vol.11, 1999.
- [90] M. Grötschel, L. Lovász, A. Schrijver. "The ellipsoid method and its consequences in combinatorial optimization". In *Combinatorica*, Vol.1, pp.169–197, 1981.
- [91] L.K. Grover. "Fast parallel algorithms for bipartite matching". In *Proceedings of the 2nd Integer Programming and Combinatorial Optimization Conference (IPCO 1992)*, pp.367–384, 1992.
- [92] U.I. Gupta, D.T. Lee, J.Y.-T. Leung. "Efficient algorithms for interval graphs and circular-arc graphs". In *Networks*, Vol.12, pp.459–467, 1982.

- [93] . S.K.S. Gupta, C.-H. Huang, P. Sadayappan, R.W. Johnson. "A framework for generating distributed-memory parallel programs for block recursive algorithms". In *Journal of Parallel and Distributed Computing*, Vol.34, pp.137–153, 1996.
- [94] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. University of Cambridge, 1997.
- [95] J. Hasselberg, P.M. Pardalos, G. Vairaktarakis. "Test case generators and computational results for the maximum clique problem". In *Journal of Global Optimization*, Vol.3, pp.463–482, 1993.
- [96] J. Håstad. "Clique is hard to approximate within $n^{1-o(1)}$ ". In *Proceedings of the Symposium on Foundations of Computer Science (FOCS 1996)*, pp.627–636, 1996.
- [97] M. Hajiaghayi, N. Nishimura, P. Ragde, D.M. Thilikos. "Fast approximation schemes for $K_{3,3}$ -minor-free or K_5 -minor-free graphs". In *Proceedings of the Euroconference on Combinatorics, Graph Theory and Applications 2001*, 2001.
- [98] E. Hartuv, A. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, R. Shamir. "An algorithm for clustering cDNAs for gene expression analysis". In *Proceedings of the International Conference on Computational Molecular Biology (ICCB 1999)*, pp.188–197, 1999.
- [99] N. Hasan, C.L. Liu. "Minimum fault coverage in reconfigurable arrays". In *Proceedings of the 18th International Symposium on Fault-Tolerant Computing (FTCS 1988)*, pp.348–353, 1988.
- [100] D.P. Helmbold, C.E. McDowell. "Modeling speedup (n) greater than n ". In *Proceedings of the International Conference on Parallel Processing (ICPP 1988)*, pp.8–12, 1988.
- [101] D. Hochbaum. *Approximation Algorithms For NP-Hard Problems*. PWS, 1997.
- [102] W.-L. Hsu. "Maximum weight clique algorithms for for circular-arc graphs and circle graphs". In *SIAM Journal of Computing*, Vol.14, pp.224–231, 1985.
- [103] O.H. Ibarra, S. Moran, R. Hui. "A generalization of the fast LUP matrix decomposition algorithm and applications". In *Journal of Algorithms*, Vol.3, pp.45–56, 1982.
- [104] J. Jaja. *An Introduction to Parallel Algorithms*. Addison-Wesley, 1992.
- [105] A. Jagota, L. Sanchis. "Adaptive, restart, randomized greedy heuristics for maximum clique". In *Journal of Heuristics*, Vol.7, No.6, pp.565–585, 2001.
- [106] V.K. Janakiram, D.P. Agrawal, R. Mehrotra. "A randomized parallel backtracking algorithm". In *IEEE Transactions on Computers*, Vol.C-37, No.12, 1988.

- [107] D.S. Johnson, M.A. Trick (Eds.). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. AMS Press, DIMACS Series Vol.26, 1996.
- [108] D.S. Johnson. "The NP-Completeness column: An ongoing guide". In *Journal of Algorithms*, Vol.8, pp.438–448, 1987.
- [109] D.R. Karger. "Minimum cuts in near-linear time". In *Journal of the ACM*, Vol.47, pp.46–76, 2000.
- [110] E. Kaltofen, V. Pan. "Processor efficient parallel solution of linear systems over an abstract field". In *Proceedings of the 3rd ACM Symposium on Parallel Algorithms and Architectures (SPAA 1991)*, pp.180–191, 1991.
- [111] R. Karp. "Reducibility among combinatorial problems". In *Complexity of Computer Computations*, Mille, Thatcher (Eds.), Plenum Press, pp.85–104, 1972.
- [112] R. Karp, Y. Zhang. "Randomized parallel algorithms for backtrack search and branch-and-bound computation". In *Journal of the ACM*, Vol.40, pp. 765–789, 1993.
- [113] G. Karypis, V. Kumar. "Multilevel k -way partitioning scheme for irregular graphs". In *Journal of Parallel and Distributed Computing*, Vol.48, pp.96–129, 1998.
- [114] G. Karypis, E.-H. Han, V. Kumar. "Chameleon: Hierarchical clustering using dynamic modeling". In *IEEE Computer*, Vol.32, pp.68–75, 1999.
- [115] B.N. Khoury, P.M. Pardalos, D.-Z. Du. "A test generator for the Steiner problem in graphs". *Manuscript*, 1992.
- [116] T. Kloks, D. Kratsch, H. Müller. "Finding and counting small induced subgraphs efficiently". In *Information Processing Letters*, Vol.74, pp.115–121, 2000.
- [117] D.L. Kreher. *Combinatorial Algorithms: Generation, Enumeration, and Search*. CRC Press, 1998.
- [118] B. Krishnamurthy. "Constructing test cases for partitioning heuristics". In *IEEE Transactions on Computers*, Vol.C-36, pp.1112–1114, 1987.
- [119] S. Khuller. "The vertex cover problem". In *ACM SIGACT News*, Vol.33, pp.31–33, 2002.
- [120] J. Kneis, D. Mölle, S. Richter, P. Rossmanith. "Algorithms based on the treewidth of sparse graphs". In *Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005)*, LNCS 3787, pp.385–396, 2005.

- [121] V. Kumar, V.N. Rao. "Parallel depth-first search on multiprocessors part I: Implementation". In *International Journal of Parallel Programming*, Vol.16, pp.479–499, 1987.
- [122] V. Kumar, V.N. Rao. "Parallel depth-first search on multiprocessors part II: Analysis". In *International Journal of Parallel Programming*, Vol.16, pp.501–519, 1987.
- [123] V. Kumar, A. Grama, A. Gupta, G. Karypis. *Introduction to Parallel Computing: Algorithm Design and Analysis*. Benjamin Cummings, 1993.
- [124] V. Kumar, A. Grama, V.N. Rao. "Scalable load balancing techniques for parallel computers". In *Journal of Parallel and Distributed Algorithms*, Vol.22, pp.60–79, 1994.
- [125] C.E. Leiserson. "Area-efficient graph layouts (for VLSI)". In *Proceedings of the IEEE Symposium on Foundations of Computer Science (STOC 1980)*, pp.270–280, 1980.
- [126] R.J. Lipton, R.E. Tarjan. "A separator theorem for planar graphs". In *SIAM Journal of Applied Mathematics*, Vol.36, pp.177–189, 1979.
- [127] R.J. Lipton, D.J. Rose, R.E. Tarjan. "Generalized nested dissection". In *SIAM Journal of Numerical Analysis*, Vol.16, pp.346–358, 1979.
- [128] R.J. Lipton, R.E. Tarjan. "Application of a planar separator theorem". In *SIAM Journal of Computing*, Vol.9, pp.615–627, 1980.
- [129] D. Lokshtanov, C. Sloper. "Fixed parameter set splitting, obtaining a linear kernel and improving running time". In *Proceedings of the Conference on Algorithms and Complexity in Durham (ACiD 2005)*, pp.105–113, 2005.
- [130] L. Lovász. "Normal hypergraphs and the weak perfect graph conjecture". In *Journal of Discrete Mathematics*, Vol.2, pp.253–267, 1972.
- [131] L. Lovász. "On the Shannon capacity of a graph". In *IEEE Transactions on Information Theory*, Vol.25, pp.1–7, 1979.
- [132] L. Lovász. "On determinants, matchings and random algorithms". In *Fundamentals of Theory of Computation*, Budach, L. (Ed.), Akademie-Verlag, pp.565–574, 1979.
- [133] L. Lovász, M.D. Plummer. *Matching Theory*. North-Holland, 1986.
- [134] D. Marx. "Parameterized graph separation problems". In *International Workshop on Parameterized and Exact Computation (IWPEC 2004)*, LNCS 3162, pp.71–82, 2004.

- [135] R. Mehrotra, E. Gehring. "Superlinear speedup through randomized algorithms". In *Proceedings of the International Conference on Parallel Processing (ICPP 1985)*, pp.291–300, 1985.
- [136] B. Monien, O. Vornberger, E. Spekenmeyer. "Superlinear speedup for parallel backtracking". Technical Report 30, University of Paderborn, 1986.
- [137] B. Monien, R. Preis. "Upper bounds on the bisection width of 3- and 4-regular graphs". In *MFCS 2001*, LNCS 2136, pp.524–536, 2001.
- [138] V. Nageshwara, V. Kumar. "Superlinear speedup in state-space search". In *Proceedings of the 1988 Foundation of Software Technology and Theoretical Computer Science*, LNCS 338, Springer-Verlag, 1988.
- [139] National Center For Biotechnology Information. <http://www.ncbi.nlm.nih.gov>.
- [140] G.L. Nemhauser, L.E. Trotter. "Vertex packing: Structural properties and algorithms". In *Mathematical Programming*, Vol.8, pp.232–248, 1975.
- [141] J. Nešetřil, S. Poljak. "On the complexity of the subgraph problem". In *Commentationes Mathematicae Universitatis Carolinae*, Vol.26, pp.415–419, 1985.
- [142] R. Niedermeier, P. Rossmanith. "Upper bounds for vertex cover further improved". In *Proceedings of the 16th Symposium on Theoretical Aspects in Computer Science (STACS 1999)*, LNCS, 1999.
- [143] R. Niedermeier, P. Rossmanith. "A general method to speed up fixed-parameter-tractable algorithms". In *Information Processing Letters*, Vol.73, pp.125–129, 2000.
- [144] R. Niedermeier, P. Rossmanith. "On efficient fixed-parameter algorithms for weighted vertex cover". In *Journal of Algorithms*, Vol.47, pp.63–77, 2003.
- [145] P.R.J. Östegård. "A new algorithm for the maximum-weight clique problem". In *Nordic Journal of Computing*, Vol.8, No.4, pp.424–436, 2001.
- [146] P.R.J. Östegård. Private communication, 2004.
- [147] C.H. Papadimitriou, M. Yannakakis. "On Limited Nondeterminism and the Complexity of the V-C Dimension". In *Journal of Computer and System Sciences*, Vol.53, pp. 161–170, 1996.
- [148] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.
- [149] M. Pelillo. "Heuristics for maximum clique and independent set". In *Encyclopedia of Optimization*, Floudas, Pardalos (Eds.), Kluwer Academic, Vol.2, pp.411–423, 2001.

- [150] A.G. Ranade. "Optimal speedup for backtrack search on a butterfly network". In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures (SPAA 1991)*, 1991.
- [151] V.N. Rao, V. Kumar. "On the efficiency of parallel backtracking". In *IEEE Transactions on Parallel and Distributed Systems*, Vol.4, pp.427–437, 1993.
- [152] R. Ravi, A. Agrawal, P. Klein. "Ordering problems approximated: Single-processor scheduling and interval graph completion". In *Proceedings of the 18th International Colloquium on Automata, Languages and Programming*, pp.751–762, 1991.
- [153] J.-C. Regin. "Solving the maximum clique problem with constraint programming". In *Fifth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 2003.
- [154] A. Reinefeld. "Scalability of massively parallel depth-first search". In *Parallel Processing of Discrete Optimization Problems*, Pardalos, Ramakrishnan, Resende (Eds.), AMS Press, DIMACS Series, Vol.22, pp.305–322, 1995.
- [155] A. Reinefeld, V. Schnecke. "Work-load balancing in highly parallel depth-first search". In *Proceedings Scalable High Performance Computing Conference (SH-PCC 1994)*, IEEE Computer Science Press, pp.295–303, 1994.
- [156] N. Robertson, P.D. Seymour. "Graph minors. II. Algorithmic aspects of tree-width". In *Journal of Algorithms*, Vol.7, pp.309–322, 1986.
- [157] N. Robertson, P.D. Seymour. "Graph minors. X. Obstructions to tree-decomposition". In *Journal of Combinatorial Theory Ser. B*, Vol.52, pp.153–190, 1991.
- [158] N. Robertson, R. Thomas. "Call routing and the ratcatcher". In *Combinatorica*, Vol.14, pp.217–241, 1994.
- [159] J.M. Robson. "Algorithms for maximum independent sets". In *Journal of Algorithms*, Vol.7, pp.425–440, 1986.
- [160] J.M. Robson. "Finding a maximum independent set in time $O(2^{n/4})$ ". Technical Report 1251-01, LaBRI, Université Bordeaux I, 2001.
- [161] C. Roth-Korostensky. *Algorithms For Building Multiple Sequence Alignments and Evolutionary Trees*. Ph.D. Thesis, ETH Zürich, Institute of Scientific Computing, February 2000.
- [162] A.L. Rosenberg, L.S. Heath. *Graph Separators, With Applications*. Kluwer Academic/Plenum Publishers, 2001.

- [163] L.A. Sanchis. "Test case construction for *NP*-hard problems". In *Proceedings of the Allerton Conference on Communication, Control, and Computing*, September 1988.
- [164] L.A. Sanchis. "On the complexity of test case generation for *NP*-hard problems". In *Information Processing Letters*, Vol.36, November 1990.
- [165] L.A. Sanchis. "Test case construction for the vertex cover problem". In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol.15, 1994.
- [166] P. Scheffler. *Die Baumweite von Graphen als ein Maß für die Kompliziertheit algorithmischer Probleme*. Ph.D. Thesis, Akademie der Wissenschaften der DDR, Berlin, 1989.
- [167] J.T. Schwartz. "Fast Probabilistic Algorithms for Verification of Polynomial Identities". In *Journal of the ACM*, Vol.27, pp.701-717, 1980.
- [168] J.C. Setubal. "Sequential and parallel experimental results with bipartite matching algorithms". Technical Report IC-96-09, Institute of Computing, University of Campinas, 1996.
- [169] J.C. Setubal, J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing, 1997.
- [170] N.Z. Shor. "Dual quadratic estimates in polynomial and Boolean programming". In *Computational Methods in Global Optimization*, Pardalos, Rosen (Eds.), Ann. Oper. Res., Vol.25, pp.163-168, 1990.
- [171] M. Sipser. "Borel sets and circuit complexity". In *Proceedings of the ACM Symposium on Theory of Computing (STOC 1983)*, pp.61-69, 1983.
- [172] N.J.A. Sloane. "Unsolved problems in graph theory arising from the study of codes". In *Graph Theory Notes of New York*, Vol.18, pp.11-20, 1989.
- [173] J. Soares, M. Stefanos. "BSP/CGM algorithm for maximum matching in convex bipartite graphs". *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2003)*, pp.167-176, 2003.
- [174] U. Stege, M.R. Fellows. "An improved fixed-parameter tractable algorithm for vertex cover". Technical Report 318, Department of Computer Science, ETH Zürich, 1999.
- [175] U. Stege. *Resolving Conflicts From Problems in Computational Biology*. Ph.D. Thesis, No.13364, ETH Zürich, 2000.
- [176] I. Stojmenović. "On random and adaptive parallel generation of combinatorial objects". In *International Journal of Computer Mathematics*, Vol.42, pp.125-135, 1992.

- [177] I. Stojmenović. "Listing combinatorial objects in parallel". In *International Journal of Parallel, Emergent and Distributed Systems*, Vol.21, pp.127–146, 2006.
- [178] V. Strassen. "Gaussian elimination is not optimal". In *Numerische Mathematik*, Vol.14, pp.354–356, 1969.
- [179] P.J. Taillon. "A new approach for solving the MAXIMUM CLIQUE problem". In *Proceedings of the 2nd Annual Conference Algorithmic Aspects in Information and Management (AAIM 2006)*, pp.279–290, 2006.
- [180] R.E. Tarjan, A.E. Trojanowski. "Finding a maximum independent set". In *SIAM Journal of Computing*, Vol.6, pp.537–546, 1977.
- [181] C. Thomassen. "The graph genus problem is NP-complete". In *Journal of Algorithms*, Vol.10, pp.568–576, 1989.
- [182] C. Thomassen. "The genus problem for cubic graphs". In *Journal of Combinatorial Theory Ser. B*, Vol.69, pp.52–58, 1997.
- [183] J.D. Thompson, D.G. Higgins, T.J. Gibson. "CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice". In *Nucleic Acids Research*, Vol.22, pp.4673–4680, 1994.
- [184] L. Valiant. "A bridging model for parallel computation". In *Communication of the ACM*, Vol.33, No.8, August 1990.
- [185] V.V. Vazirani. "A theory of alternating paths and blossoms for proving the correctness of the $O(\sqrt{V}E)$ maximum matching algorithm". In *Combinatorica*, Vol.14, pp.71–109, 1994.
- [186] M.S. Waterman. *Introduction to Computational Biology*. Chapman and Hall, 1995.
- [187] D.B. West. *Introduction to Graph Theory*. Prentice-Hall, 1996.
- [188] G.J. Woeginger. "Exact algorithms for NP-hard problems: A survey". In *Proceedings of the 5th International Workshop on Combinatorial Optimization-Eureka, You Shrink!*, Junger, Reinelt, Rinaldi (Eds.), LNCS 2570, Springer-Verlag, pp.184–207, 2003.
- [189] D.R. Wood. "An algorithm for finding maximum cliques in a graph". In *Operations Research Letters*, Vol.21, pp.211–217, 1997.
- [190] J. Xue. *Fast Algorithms For Vertex Packing and Related Problems*. Ph.D. Thesis, GSIA, Carnegie Mellon University, 1991.

- [191] E.A. Yildirim, X. Fan-Orzechowski. “On extracting maximum stable sets in perfect graphs using the Lovász’s theta function”. In *Journal of Computational Optimization and Applications*, Vol.32, pp.1–30, 2005.
- [192] R.E. Zippel. “Probabilistic algorithms for sparse polynomials”. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (EUROSAM 1979)*, LNCS 72, pp.216–226, 1979.

Appendix A

Glossary

alternating path: given a matching M in a graph $G = (V, E)$, an alternating path is a path in which the edges belong alternatively to the matching and not to the matching.

augmenting path: an augmenting path is an alternating path that starts from and ends on unmatched vertices.

breadth-first search: a search that considers neighbors of a vertex, i.e., outgoing edges of the vertex's predecessor in the search, before any outgoing edges of the vertex. Extremes are searched last.

BSP: a Bulk-Synchronous Parallel model consists of a collection of processor/memory modules connected by a router capable of delivering messages point-to-point between processors [184]. A BSP computation consists of a sequence of supersteps separated by barrier synchronization. During a computational superstep, each processor performs local computations on data resident in their memory.

CGM: the Coarse-Grained Multicomputer model, or CGM, comprises p processors, P_i , $0 \leq i \leq p - 1$ with $O(n/p)$ local memory storage such that $n/p \geq p^\epsilon$, $\epsilon > 0$, and interconnected with an arbitrary communication fabric [52]. Computation and communication are separated into synchronous super-steps.

clique: in a graph, $G = (V, E)$, a clique is a subset, $C \subseteq V$, such that if $u, v \in C$ then $(u, v) \in E$.

complexity class: a set of computational problems with the same bounds $\Theta(n)$ on time and space, for deterministic and nondeterministic machines.

component: a component of a graph is a maximal connected subgraph.

connected graph: a graph, $G = (V, E)$, is connected if there exists a path in G between each pair of vertices in V .

critical vertex: a vertex is critical if it is in every maximum matching.

deficiency: the deficiency of a graph $G = (V, E)$ is the number of vertices exposed in a maximum matching, i.e., $n - 2\nu(G)$.

depth-first search: any search that considers outgoing edges of a vertex before any neighbors of the vertex, that is, outgoing edges of the vertex's predecessor in the search. Extremes are searched first.

density: an undirected graph, $G = (V, E)$, with n vertices and m edges has density $m/(n(n-1)/2)$; if G is a directed graph then the density is given by $m/(n(n-1))$.

directed graph: a graph whose edges are ordered pairs of vertices. That is, each edge can be followed from one vertex to another vertex.

directed acyclic graph, DAG: a directed graph with no path that starts and ends at the same vertex.

elementary bipartite graph: a bipartite graph, $G = (U \cup L, E)$, with exactly two minimum vertex covers, the set of vertices U or the set L , and every edge in G lies in some perfect matching.

factor-critical: a graph G is factor-critical if the graph $G(V - v)$ has a perfect matching, $\forall v \in V$.

FPT: fixed-parameter tractable, a class of problems that are *NP*-complete and yet for which there exist efficient algorithms that decide the problem with an exponential complexity that is a function of some fixed parameter.

independent set: in a graph, $G = (V, E)$, an independent set is a subset, $I \subseteq V$, such that for all $u, v \in I$, $(u, v) \notin E$.

induced subgraph: a subgraph H is an induced subgraph of G if $E(H)$ contains all edges of G that connect vertices in $V(H)$.

kernelization: a property of problems in *FPT* such that they can be reduced in polynomial time to a subproblem of size bounded by a function of the fixed parameter.

klam: a quantitative measure of the effectiveness of a fixed-parameter algorithm.

Las Vegas algorithm: a randomized algorithm that, for a fixed input, upon termination either returns an output guaranteed to be correct or reports failure, the latter with small probability.

matching: given a graph, $G = (V, E)$, a matching, M , in G is a set of pairwise non-adjacent edges. A vertex is matched if it is incident to an edge in the matching. Otherwise the vertex is unmatched, or exposed.

maximal matching: a maximal matching is a matching M of a graph $G = (V, E)$ with the property that if any edge not in M is added to M , then it is no longer a matching.

maximum matching: a maximum matching is a matching that contains the largest possible number of edges. The matching number, $\nu(G)$, of a graph is the size of a maximum matching.

memorization: a technique that uses exponential space to improve the time complexity of an algorithm. Subproblems of bounded size and their solutions are stored in a data structure for later retrieval during an exhaustive recursion search.

Monte Carlo algorithm: a randomized algorithm that, for a fixed input, upon termination may give incorrect results with small probability.

near-perfect matching: a matching is said to be near-perfect if it covers all but exactly one vertex.

noncritical vertex: a vertex is noncritical if it is exposed in at least one maximum matching, otherwise it is critical.

***NP*:** the complexity class of decision problems for which answers can be checked by an algorithm whose run time is polynomial in the size of the input. *NP* is the class that a nondeterministic Turing machine accepts in polynomial time.

***NP*-complete:** the complexity class of decision problems for which answers can be checked for correctness, given a certificate, by an algorithm whose run time is polynomial in the size of the input (that is, it is in *NP*) and no other *NP* problem is more than a polynomial factor harder.

nondeterministic Turing machine: a Turing machine which has more than one next state for some combinations of contents of the current cell and current state. An input is accepted if at least one move sequence leads to acceptance.

odd (even) component: an odd (even) component of a graph is a connected component whose vertex set has odd (even) cardinality.

***P*:** the complexity class of languages that can be accepted by a deterministic Turing machine in polynomial time.

PRAM: a shared memory model of computation, where typically the processors all execute the same instruction synchronously, and access to any memory location occurs in unit time.

perfect matching: a perfect matching is a matching which covers all vertices of the graph, i.e., every vertex of the graph is incident to exactly one edge of the matching.

regular graph: a graph is r -regular if $\forall v \in V, \deg(v) = r$.

subgraph: a subgraph of G is a graph H such that $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$.

Turing machine: a model of computation consisting of a finite state machine controller, a read-write head, and an unbounded sequential tape. Depending on the current state and symbol read on the tape, the machine can change its state and move the head to the left or right.

vertex cover: in a graph, $G = (V, E)$, a vertex cover is a subset, $VC \subseteq V$, such that for each edge $(u, v) \in E$, at least one of u or v belongs to VC .

W -hierarchy: classes of problems that are not likely to be fixed-parameter tractable, with the class $W[1]$ representing the lowest level of intractability.

Appendix B

Symbols and notation

Symbol or notation	Meaning
$\tilde{O}(\cdot)$	Asymptotic complexity that ignores poly-logarithmic factors
$O^*(\cdot)$	Asymptotic complexity that ignores polynomial factors
$G \setminus \{v\}$	Graph generated after removing vertex v and its incident edges from graph G
$\delta(G)$	Smallest degree taken over all vertices in graph G
$\Delta(G)$	Largest degree taken over all vertices in graph G
$G(V)$	Graph induced by the set of vertices C
$N(v)$	Set of vertices adjacent to a vertex v
$N[v]$	$N(v) \cup \{v\}$
$\nu(G)$	Matching number of graph G

Rule no. (cf.([43]))	Branching Vector	Branching Number
(1)(ii)	(2,4)	1.272019650
(A)(i)	(3,3)	1.259921050
(A)(ii)	(3,4,8)	1.276266656
(B1)	(2,4)	1.272019650
(B2)(i)	(3,7,5)	1.263738861
(B2)(ii)	(3,7,6,10)	1.271441913
(B3)	(3,8,4)	1.276266656
Rule no. (cf.([15]))	Branching vector	Branching number
(4)d	(3,3,7)	1.324717957
(5)a	(4,4,4)	1.316074013
(5)c	(4,4,5,8)	1.324717957

Appendix C

k -Vertex Cover Branching Rules

In this section we outline in detail the relevant branching rules of the Chen, et al. [43] and the Balasubramanian, et al. [15] algorithms, restricted to graphs of maximum degree bounded by 4.

(Chen, et al.) Rule 1: Degree-2 vertex

folding:

$$1. \{v_0\} \subseteq VC$$

$$2. N(v_0) \subseteq VC$$

$$C(k) \leq 1 + C(k-1) \text{ -or-}$$

$$C(k) \leq 1 + C(k-2) + C(k-4)$$

(Balasubramanian, et al.) Rule 5: Degree 4

Subcase: (5)a

1. $\{1, 2, 3, 4\} \subseteq VC$
2. $N(3) \subseteq VC$
3. $N(4) \cup \{3\} \subseteq VC$

$$C(k) \leq 1 + 3C(k-4)$$

Subcase: (5)c

1. $\{1, 2, 3, 4\} \subseteq VC$
2. $N(3) \subseteq VC$
3. $N(4) \cup \{3\} \subseteq VC$
4. $N(\{1, 2\}) \cup \{3, 4\} \subseteq VC$

$$C(k) \leq 1 + 2C(k-4) + C(k-5) + C(k-8)$$