Parallel Fixed Parameter Tractable Problems

Shouwei Li

Heinz Nixdorf Institute
Department of Computer Science
Paderborn University

Schriftliche Arbeit
zur Erlangung des Grades eines Doktors der Naturwissenschaften

Paderborn December 2017

Reviewers: Prof. Dr. Friedhelm Meyer auf der Heide
Prof. Dr. Christian Scheideler

I would like to dedicate this thesis to my parents, my wife and our daughter.

Abstract

Parameterized complexity theory provides a refined classification of intractable problems on the basis of multivariate design and complexity analysis of deterministic algorithms. In brief, a problem is *fixed-parameter tractable* (FPT) if it has an algorithm that runs in time $\mathcal{O}(f(k) \cdot n^{\mathcal{O}(1)})$, where n is the input size, k is the parameter, and f is an arbitrary computable function of k and independent of n.

The study of parameterized complexity has been extended to parallel computing, which is broadly known as *parameterized parallel complexity*. This thesis focuses on the issue of which problems employ efficient fixed-parameter parallel algorithms. A problem is *fixed-parameter parallel-tractable* (FPPT) if it has an algorithm that runs in time $\mathcal{O}(f(k) \cdot (\log n)^{\alpha})$ using $\mathcal{O}(n^{\beta})$ parallel processors, where n is the input size, k is the parameter, f is an arbitrary computable function of k and independent of n, and α, β are constants independent of n and k.

The primary contributions of this thesis are summarized as follows. We propose an efficient parallel algorithm for the general MONOTONE CIRCUIT VALUE PROBLEM with n gates and an underlying graph of genus k. The problem is known to be in NC when the underlying graph of the input circuit is of genus 1, and such an embedding is given with the input. Unlike the previous work on this problem, our algorithm does not require a precomputed embedding to be given with the input. Hence, the monotone circuit value problem parameterized by genus is in FPPT. This result also implies that for some (not all) P-complete problems, it is possible to find an algorithm that makes the problem fall into NC by fixing one or more non-trivial parameters. If we confine ourselves to P-complete problems, an interesting analogy would be: *FPPT is with respect to P-complete what FPT is with respect to NP-complete*.

We extend the FPPT framework to a general kernelization method called *crown decomposition* that is used to cope with a number of NP-complete problems in FPT, such as the VERTEX COVER PROBLEM, the MAXIMUM SATISFIABILITY PROBLEM, *etc*. This result directly implies an efficient parallel algorithm for the parameterized vertex cover problem that outperforms the best known parallel algorithm for this problem: using $\mathcal{O}(m)$ instead of $\mathcal{O}(n^2)$ parallel processors, the running time improves from $4 \log n + \mathcal{O}(k^k)$ to

$\mathcal{O}(8^k + k \cdot \log^3 n)$, where m is the number of edges, n is the number of vertices, and k is an upper bound of the size of the sought vertex cover, since the crown structure admits a kernel with size at most $3k$. Thus, the parallel crown decomposition and the parameterized vertex cover problem are in FPPT.

Furthermore, we explore another parameter called *modular-width* that covers a significantly large class of graphs. We extend the study of modular-width to the parameterized parallel complexity and show that the WEIGHTED MAXIMUM CLIQUE PROBLEM and the MAXIMUM MATCHING PROBLEM are in FPPT when parameterized by modular-width. These results are of interests for several reasons. First, not only for P-complete and NP-complete problems but also for those problems that are still open for P-complete or NC, there are parameterized parallel algorithms with non-trivial parameters for them. Thus, FPPT is orthogonal to P-complete, NP-complete and probably some unknown classes between NC and P-complete (if P is not equal to NC). Second, there exist some parameters that make a large number of problems in FPPT, which are in different complexity classes in the traditional hierarchy.

Zusammenfassung

Parametrisierte Komplexitätstheorie beschreibt eine neu-definierte Klassifikation von schweren Problemen basierend auf einem mehrdimensionalen Design und der Komplexitätsanalyse von deterministischen Algorithmen. In Kurzform, ein Problem ist *fixed-parameter tractable* (FPT), wenn es einen Algorithmus gibt, der Laufzeit $\mathcal{O}(f(k) \cdot n^{\mathcal{O}(1)})$. Hierbei ist n die Eingabegröße, k der Parameter, f ist eine beliebige berechenbare Funktion abhängig von k und unabhängig von n.

Die Forschung von parametrisierter Komplexität wurde auf das parallele Rechnen erweitert und ist im Allgemeinen als *Parametrisierte Parallele Komplexität* bekannt. Diese Arbeit konzentriert sich auf die Fragestellung, welche Probleme effiziente parametrisierbare parallele Algorithmen erlauben. Ein Problem ist *fixed-parameter parallel-tractable* (FPPT), wenn es einen Algorithmus gibt, der Laufzeit $\mathcal{O}(f(k) \cdot (\log n)^{\alpha})$ mit $\mathcal{O}(n^{\beta})$ parallelen Prozessoren hat. Hierbei ist n die Eingabegröße, k der Parameter, f ist eine beliebige berechenbare Funktion abhängig von k und unabhängig von n, und α, β sind von n und k unabhängige Konstanten. Die Hauptbeiträge dieser Arbeit sind im Folgenden zusammengefasst.

Wir präsentieren einen effizienten parallelen Algorithmus für das generelle MONOTONE CIRCUIT VALUE PROBLEM mit n Gates und einem zugrunde liegenden Graphen mit k-begrenzendem Geschlecht. Das Problem ist bekannterweise in NC, wenn die Eingabe ein Geschlecht von 1 hat und mit einer fixen Einbettung gegeben ist. Im Gegensatz zu vorherigen Arbeiten für dieses Problem benötigt unser Algorithmus keine gegebene Einbettung. Aus diesem Grund ist das vom Geschlecht parametisierte monotone circuit value problem in FPPT. Insbesondere implizieren unsere Ergebnisse, dass für ein gegebenes (aber nicht jedes) P-vollständiges Problem ein Algorithmus gefunden werden kann, der das Problem in NC sein lässt durch Fixierung von einer oder mehreren Parametern. Wenn wir uns auf P-vollständige Probleme eingrenzen, existiert folgende Analogie: *FPPT für P-vollständige Probleme ist ähnlich wie FPT für NP-vollständige Probleme*.

Diese Arbeit erweitert das FPPT Framework zu einer generellen Kernelmethode, *crown decomposition*, welche viel in FPT genutzt wird für eine Anzahl von NP-vollständigen Problemen, wie z. B. das VERTEX COVER PROBLEM, das MAXIMUM SATISFIABILITY PROBLEM, *etc*. Aus diesem Ergebnis erhalten wir direkt einen effizienten parallelen Algorithmus, der

den besten bekannten parallelen Algorithmus für dieses Problem verbessert: Durch Nutzung von $\mathcal{O}(m)$ statt $\mathcal{O}(n^2)$ parallelen Prozessoren verbessert sich die Laufzeit von $4\log n + \mathcal{O}(k^k)$ auf $\mathcal{O}(8^k + k \cdot \log^3 n)$, wobei m die Anzahl der Kanten und n die Anzahl der Knoten des Eingabegraphs, sowie k eine obere Schranke für die Größe der gesuchten Knotenüberdeckung ist, da die Crown-Struktur eine Kernelgröße von höchstens $3k$ impliziert. Daher ist sowohl die parallele crown decomposition als auch die parametrisierte Knotenüberdeckung in FPPT.

Des Weiteren haben wir einen weiteren Parameter, die *modular-width*, untersucht, welche eine signifikant große Klasse von Graphen betrifft. Wir haben die Forschung von modular-width auf die parametrisierte parallele Komplexität erweitert und zeigen, dass die Probleme WEIGHTED MAXIMUM CLIQUE PROBLEM und MAXIMUM MATCHING PROBLEM in FPPT liegen für eine begrenzte modular-width. Diese Ergebnisse sind aus verschiedenen Gründen interessant: Auf der einen Seite scheinen nicht nur für NP-vollständige und P-vollständige Probleme, sondern auch für einige Probleme, für die noch nicht bekannt ist, ob sie P-vollständig oder in NC liegen, parametrisierte parallele Lösungen mit nicht trivialen Parametern zu existieren. Daher ist FPPT orthogonal zu P-Vollständigkeit, NP-Vollständigkeit und sogar zu einigen offenen Probleme der P-Vollständigkeit ist (wenn P ungleich NC ist). Auf der anderen Seite existieren vermutlich einige universale Parameter, die eine große Anzahl von Problemen in FPPT einordnen, die aber aus verschiedenen Komplexitätsklassen in der traditionellen Hierarchie kommen.

Table of contents

List of figures

Chapter 1

Introduction

One of the goals of computational complexity theory is to analyze and classify problems regarding the resource demand on time or space for an algorithm to cope with them. Conventionally, the asymptotic running time or space requirement of an algorithm is represented as a function of the input size n. This fundamental idea has lead to a diversity of complexity classes and a clean-cut intractability theory and brings forward many profound questions in theoretical computer science as well, such as whether P is equal to NP. However, measuring complexity only regarding the input size is often too coarse and has several drawbacks. For instance, several problems might have the same complexity under a broad view, nevertheless the number of instances that make the problem intractable might differ significantly from one to another. Moreover, this method suggests that any additional information about the input instance was ignored even if for some graph problems, the number of vertices, topology, hereditary that reflect the structural properties of the input graphs usually have a considerable impact on the resulting algorithms. After adopting these constraints, a number of problems probably become much easier than they typically are. Take the MAXIMUM INDEPENDENT SET PROBLEM as an example, it is known to be NP-hard for general graphs but can be solved in a polynomial time for special classes of graphs, such as claw-free graphs[52] and fork-free graphs [5]. Similarly, if the size of the solution to a problem is k and given with the input instance, and measure the resource demand not only regarding the input size n but also k, then several degrees of tractability for k might exist when k is restricted to fixed small numerical values. For example, the only result known to the maximum independent set problem under this circumstance is a trivial brute-force algorithm that runs in time $\mathcal{O}(n^{k+1})$, where n is the number of vertices of the input graph, and k is the size of the sought independent set. By contrast, the VERTEX COVER PROBLEM has an algorithm that runs in time $\mathcal{O}(1.274^k + kn)$, where n is the number of vertices of the input graph, and k is the size

of the sought vertex cover [14]. Consequently, these two problems seem to have different parameterized complexity.

Ever since the first systematic work on "*parameterized complexity*" depicted by Downey *et al.* [22], it has witnessed tremendous growth in the last two decades and become an active research area in theoretical computer science. This new branch of computational complexity theory takes a step backward and provides a refined classification of intractable problems on the basis of multivariate design and complexity analysis of deterministic algorithms. In classical computational complexity theory, a problem typically specified by the input instance and the question to be answered. Within this framework, besides the input instance and the question to be answered, people also have interests in other characteristics that constitute as parameters, such as cardinality of the final solution, treewidth of the input graph, *etc*. The parameterized version of problems are termed *fixed-parameter tractable* and comprise the class of FPT if they have an algorithm that runs in time $\mathcal{O}(f(k) \cdot n^{\mathcal{O}(1)})$, where n is the input size, k is the parameter, and f is an arbitrary computable function of k and independent of n. Problems in FPT are abundant, for many of which people are still competing to establish a record for better solutions. One remarkable example is the parameterized vertex cover problem, for which the best-known algorithm runs in time $\mathcal{O}(1.274^k + kn)$, where n is the number of vertices of the input graph, and k is the size of the sought vertex cover. The class FPT occupies the bottom of parameterized complexity hierarchy just as the class P in the classical polynomial hierarchy.

To further reduce the running time of sequential algorithms, the idea of parameterized complexity has been extended to parallel computing and this is broadly known as *parameterized parallel complexity*. A first attempt to formalize the concept has been pursued by Bodlaender *et al*. In a one page abstract [7], they introduced a class, known as *parameterized analog of NC* (PNC), which contains all parameterized problems that have a parallel deterministic algorithm runs in time $\mathcal{O}(f(k) \cdot (\log n)^{h(k)})$ using $\mathcal{O}(g(k) \cdot n^{\beta})$ parallel processors, where n is the input size, k is the parameter, f, g, and h are arbitrary computable functions of k and independent of n, and β is a constant independent of n and k. Shortly afterwards a more systematic work depicted by Cesati *et al*. [12]. They introduced another class, known as *fixed-parameter parallelizable* (FPP), which contains all parameterized problems that have a parallel deterministic algorithm runs in time $\mathcal{O}(f(k) \cdot (\log n)^{\alpha})$ using $\mathcal{O}(g(k) \cdot n^{\beta})$ parallel processors, where n is the input size, k is the parameter, f and g are arbitrary computable functions of k and independent of n, and α, β are constants independent of n and k.

1.1 Contributions of the Thesis

This thesis focuses on the issue of which problems employ efficient fixed-parameter parallel algorithms. The primary contributions of this thesis are summarized as follows.

We rename a class called *fixed-parameter parallel-tractable* (FPPT), which contains all parameterized problems that have a parallel deterministic algorithm that runs in time $\mathcal{O}(f(k) \cdot (\log n)^{\alpha})$ using $\mathcal{O}(n^{\beta})$ parallel processors, where n is the input size, k is the parameter, f is an arbitrary computable function of the parameter k and independent of n, and α, β are constants independent of n and k.

We initiate the study of FPPT with a typical P-complete problem — the general MONOTONE CIRCUIT VALUE PROBLEM (MCVP) and propose an efficient parallel algorithm that runs in time $\mathcal{O}((k+1) \cdot \log^2 n)$ using $\mathcal{O}(n^c)$ parallel processors, where n is the number of gates, k is the genus of the underlying graph, and $\mathcal{O}(n^c)$ is the best processor boundary for parallel matrix multiplication [47]. Our algorithm improves the result in [48], which showed that the problem is in NC when the underlying graph of input is of genus 1. However, their approach was non-constructive and assumed that a precomputed embedding is given with the input. Besides extending the genus from 1 to parameter k, our algorithm does not require a precomputed embedding is given with the input as well. In fact, it does not rely on such an embedding at all. Hence, the monotone circuit value problem parameterized by genus is in FPPT. This result also implies that for some (not all) P-complete problems, it is possible to find an algorithm that makes the problem fall into NC by fixing one or more non-trivial parameters. Hence, if we confine ourselves to P-complete problems, an interesting analogy would be: *FPPT is with respect to P-complete what FPT is with respect to NP-complete*.

These results are based on the following publication:

> *Faisal N. Abu-Khzam, Shouwei Li, Christine Markarian, Friedhelm Meyer auf der Heide, and Pavel Podlipyan. The monotone circuit value problem with bounded genus is in NC. In The International Computing and Combinatorics Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, Proceedings, pages 92–102, 2016.* [1]

Subsequently, we extend the FPPT framework to a general kernelization method called *crown decomposition* that is used in FPT to cope with a number of NP-complete problems, such as the VERTEX COVER PROBLEM, the MAXIMUM SATISFIABILITY PROBLEM, *etc*. We show that the crown decomposition can be computed in time $\mathcal{O}(k \cdot \log^3 n)$ using $\mathcal{O}(m)$ parallel processors, where m is the number of edges, n is the number of vertices of the input graph, and k is the size of the sought vertex cover. This result directly implies an

efficient parallel algorithm for the parameterized vertex cover problem that outperforms the best known parallel algorithm for this problem [12] : using $\mathcal{O}(m)$ instead of $\mathcal{O}(n^2)$ parallel processors, the running time improves from $4\log n + \mathcal{O}(k^k)$ to $\mathcal{O}(8^k + k \cdot \log^3 n)$, where m is the number of edges, n is the number of vertices of the input graph, and k is an upper bound of the size of the sought vertex cover. Since the crown structure admits a kernel with size at most $3k$, the parallel crown decomposition and the parameterized vertex cover problem are in FPPT.

These results are based on the following publication:

> *Faisal N. Abu-Khzam, Shouwei Li, Christine Markarian, Friedhelm Meyer auf der Heide, and Pavel Podlipyan. On the parameterized parallel complexity and the vertex cover problem. In The International Conference on Combinatorial Optimization and Applications, COCOA 2016, Hong Kong, China, December 16-18, Proceedings, pages 477–488, 2016.* [2]

Furthermore, we explore another parameter called *modular-width* that covers a significantly large class of graphs. It has been shown that several problems are in FPT when parameterized by modular-width [31]. We extend the study of modular-width to parameterized parallel complexity and show that the WEIGHTED MAXIMUM CLIQUE PROBLEM and the MAXIMUM MATCHING PROBLEM are in FPPT when parameterized by modular-width. These results are of interests for several reasons. First, not only for P-complete and NP-complete problems but also for those that are still open for P-complete or NC, are parameterized parallel algorithms with non-trivial parameters for them. Thus, FPPT is orthogonal to P-complete, NP-complete and probably some unknown classes between NC and P-complete (if P is not equal to NC). Second, there exist some parameters that make a large number of problems in FPPT, which are in different complexity classes in the traditional hierarchy.

These results are based on the following publication:

> *Faisal N. Abu-Khzam, Shouwei Li, Christine Markarian, Friedhelm Meyer auf der Heide, and Pavel Podlipyan. Modular-width: An auxiliary parameter for parameterized parallel complexity. In The International Frontiers of Algorithmics Workshop, FAW 2017, Chengdu, China, June 23-25, Proceedings, pages 139–150, 2017.* [3]

1.2 Preliminaries and Notation

1.2.1 Models of Computation

A problem is said to be *feasible* if there is a polynomial time algorithm to solve it (as stated for the first time by Edmonds [24]). This notation requires people to make an agreement on the machine model that is used to measure the resource demands on time or space for an algorithm. Since the theory of computational complexity was founded in the early nineteen-sixties, a large number of machine models has been proposed to capture the notion of effective computation.It is unrealistic to measure the complexity in theory with real computers because the results would depend on the contemporary hardware and technology. Through suitable machine models people attempts to provide a reasonable approximation of what one might expect if a real computer were used for the computations.

One of the most favorite models of computation in sequential algorithm design is the *random access machine* (RAM). Each RAM consists of a central processing unit; a read-only input tape; a write only output tape; and a random-access memory with the property that each memory cell can be accessed in one unit of times. The computation unit contains several simple instructions, such as moving data between memory cells either directly or indirectly; comparing and conditional branching; simple arithmetic instructions such as add, subtract, multiply, divide, and so on. A RAM program is sequential of these instructions. Execution starts with the first instruction and ends when a halt instruction is encountered. Typically, all instructions in the RAM model are assessed one unit of cost regardless of the length of the numbers being manipulated by the operation. The usual complexity measures of interest for RAM computations are time, in the form of the number of instructions executed, and space, in the form of the number of memory cells accessed. To prevent this notion of time from distorting our notion of feasibility, the model prohibits rapid generation of very large numbers. For example, the model with prohibiting numbers of super-polynomial length from being generated or tested in polynomial-time. Aside from these considerations, the power of the RAM model is essentially unchanged throughout a broad range of variations in the instruction set.

PRAM Model

The natural generalization of the RAM model to parallel computation is the *parallel random access machine* (PRAM) introduced independently by Fortune and Wyllie [30] and by Goldschlager [35]. The PRAM model consists of a collection of RAM processors that run in parallel and communicate via a common memory.

The basis PRAM model consists of an unbounded collection of numbered RAM processors and an unbounded collection of shared memory cells. Moreover, each processor has local memory and knowns its index and has instruction for direct and indirect read/write access to the shared memory. Rather than being on tapes, inputs and outputs to the computation are placed in shared memory to allow concurrent access. Instructions are executed in unit time, synchronized over all active processors. One popular model is the "*concurrent read exclusive write*" (CREW-PRAM) that multiple processors can read a memory cell but only one can write at a time.

Based on the RAM and the PRAM computation models, we have the following two complexity classes which are essential in this thesis:

- the class P is the set of all languages L that are decidable in sequential time $n^{\mathcal{O}(1)}$.
- the class NC is the set of all languages L that are decidable in parallel time $(\log n)^{\mathcal{O}(1)}$ and processors $n^{\mathcal{O}(1)}$.

The Boolean Circuit Model

Although the PRAM model is a natural parallel extension of the RAM model, it is not obvious that the model is reasonable for several reasons. For instance, does the PRAM model correspond, in capability and cost, to a physically implementable device? Is it fair to allow unbounded numbers of processors and memory cells? How reasonable is it to have unbounded size integers in memory cells? Is it possible to have unbounded numbers of processors accessing any portion of shared memory for a unit cost? Is synchronous execution of one instruction on each processor in unit time realistic?

To expose issues like these, it is useful to have a more primitive model that, although being less convenient for programming, is more closely related to the realities of physical implementation. Such a model is the *boolean circuit* [10]. The model is simple to describe and easy to analyze. Circuits are basic technology, consisting of simple logic gates connected by bit-carrying wires. They have no memory and no notion of state. Circuits avoid almost all issues of machine organization and instruction repertoire. Their computational components correspond directly with devices that we can fabricate.

The circuit model is still and idealization of real electronic computing devices. It ignores a number of important practical considerations such as circuit area, volume, pin limitations, power dissipation, packaging, and single propagation delay. Such issues are addressed more accurately by more complex VLSI models, but for many purposes, the boolean circuit model seems to provide an excellent compromise between simplicity and realism. For example, one feature of PRAM models that has been widely criticized as unrealistic and unimplementable

is the assumption of unit time access to shared memory. Consideration of (bounded fanin) circuit models exposes this issue immediately, since a simple fanin argument provides a lower bound of $\Omega(\log p)$ on time to combine bits from p sources.

Let $B_k = \{f \mid f : \{0,1\}^k \to \{0,1\}\}$ denote the set of all k-ary Boolean functions.

Definition 1 ([36]). *A boolean circuit C is a labeled finite oriented directed acyclic graph. Each vertex v in C has a type $\tau(v) \in \{I\} \cup B_0 \cup B_1 \cup B_2$. A vertex v with $\tau(v) = I$ has indegree 0 and is called an input. The inputs of C are given by a tuple $\langle x_1, \ldots, x_n \rangle$ of distinct vertices. A vertex v with out-degree 0 is called an output. The outputs of C are given by a tuple $\langle y_1, \ldots, y_m \rangle$ of distinct vertices. A vertex v with $\tau(v) \in B_i$ must have in-degree i and is called a gate.*

Note that fanin is less than or equal to two but fanout is unrestricted. Inputs and gates can also be outputs. Each circuit computes a well-defined function of its input bits as specified in the following definition.

The resource measures of interest for a circuit are its size and depth.

Definition 2. *The size of C, denoted by size(C), is the number of vertices in C. The depth of C, denoted by depth(C), is the length of the longest path in C form an input to the output.*

The computation model of circuits has one characteristic which makes it different from other familiar models such as the RAM: A boolean circuit has a fixed number of input gates. Thus an individual circuit can only work on inputs of one fixed length, and different circuits are required for different length inputs. From an algorithmic perspective, this is certainly a contradiction to our wish: one algorithm for a problem should be able to handle all possible lengths of inputs, or in other words, we want a uniform circuit family.

Definition 3. *A families of Boolean circuits $\{C_n : n \in \mathbb{N}\}$ is logspace uniform if there exists a deterministic Turing machine TM, such that TM runs in logarithmic space and for all $n \in \mathbb{N}$, TM outputs a description of C_n on input 1^n.*

Definition 4 ([36]). *Let $\{C_n\}$ be a boolean circuit family that computes the function $f_C : \{0,1\}^* \to \{0,1\}$. The language accepted by $\{C_n\}$ denoted L_C is the set $L_C = \{x \in \{0,1\}^* | f_C(x) = 1\}$.*

Circuits and PRAMs

We have alluded to the fact that many parallel models are equivalent when we consider feasible highly parallel algorithms. That is, if a problem has a feasible highly parallel solution on one model, then it also has one on any equivalent model. Originally the notion of feasible

and highly parallel came from the observations that certain problems had polylogarithmic running time and a polynomial number of processor solutions on many different models. In a triumph of circularity, all the models that support feasibly highly parallel algorithms became the "reasonable" parallel models. For any new parallel model to be considered reasonable, it must be able to simulate some existing reasonable model and vice versa. This is also done in [36].

Theorem 1 ([36]). *A function f from $\{0,1\}^*$ to $\{0,1\}^*$ can be computed by a logarithmic space uniform boolean circuit family $\{C_n\}$ with $depth(C_n) = (\log n)^{\mathcal{O}(1)}$ and $size(C_n) = n^{\mathcal{O}(1)}$ if and only if f can be computed by a CREW-PRAM M in time $t(n) = (\log n)^{\mathcal{O}(1)}$ and processors $p(n) = n^{\mathcal{O}(1)}$.*

Thus,

- polynomially time bounded log-space uniform families of circuits describe the class P.
- polylogarithmically depth-bounded log-space uniform families of circuits describe the class NC.

1.2.2 Graphic Metrics

This subsection describes some properties of graphs that are commonly known as graph metrics. Among several well known graph metrics, the ones reviewed in this subsection are those relevant to this thesis.

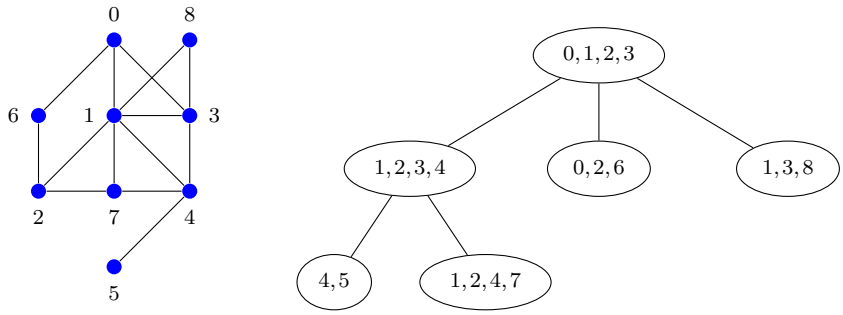
Treewidth

Definition 5. *A tree decomposition of a graph $G = (V, E)$ is a pair (T, Y), where T is a tree and $Y = \{Y_i : i \in V(T)\}$ is a collection of subsets of V satisfying the following:*

1. *for all $(u, v) \in E$, there exists an i such that $\{u, v\} \in Y_i$.*
2. *for all $i, j, k \in V(T)$, if j is on the path between i and k in T, then $Y_i \cap Y_k \subseteq Y_j$.*

The width *of a tree decomposition (T, Y), denoted by $w((T, Y))$, is $\max\{|Y_i| : Y_i \in Y\} - 1$. The* treewidth *of G, denoted by $tw(G)$ is the minimum of $w((T, Y))$ where (T, Y) is a tree decomposition of G.*

Tree decompositions of a graph G whose width is the same as $tw(G)$ are considered optimal tree decompositions. Fig. 1.1 shows a graph and the optimal tree decomposition of treewidth 3.



(a) A graph G (b) An optimal tree decomposition (treewidth = 3)

Fig. 1.1 Tree decomposition

Cliquewidth

The *Cliquewidth* is a parameter that describes the structural complexity of the graph; it is closely related to treewidth, but unlike treewidth it can be bounded even for dense graphs. It is defined by means of the following 4 operations on $k-$colored graphs:

1. create a new vertex v and colored i;
2. join all vertices of color i to other vertices of color j;
3. recolor all vertices i to color j;
4. take the disjoint union of G_1 and G_2.

Definition 6 ([18])**.** *The smallest number of colors needed to construct G is called the cliquewidth of G, denoted by $cw(G)$.*

Cliquewidth was introduced because many NP-hard probelms are polynomial time solvable on bounded cliquewidth graphs, whereas earlier width parmeters were not able to handle these graphs, for example, the treewidth of a clique on $n \geq 2$ vertices is $n-1$ while the cliquewidth is 2.

1.3 Organization of the Thesis

Chapter 2 discusses FPPT in detail and presents the efficient parallel algorithm for the monotone circuit value problem with n gates and an underlying graph of genus k. Chapter 3 extends the FPPT framework to crown decomposition, which is widely use to prove a problem in FPT. Chapter 4 shows the weighted maximum clique problem and the maximum matching problem are in FPPT when parameterized by modular-width. In Chapter 5, we conclude the thesis and propose several directions for future work.

Chapter 2

The MCVP Parameterized by Genus

2.1 Introduction

Boolean circuits are defined in terms of the logic gates they contain, which is called basis normally. A basis is complete if from which all other Boolean circuits can be constructed. For example, a well-known complete basis for Boolean circuits is the set {NOT, AND, OR}. The CIRCUIT VALUE PROBLEM (CVP) asks for computing the output of a given Boolean circuit on a given input. This problem has been shown to be complete for P with respect to logarithmic space reductions [43]. Some restricted variants of CVP are also well studied. For instance, the PLANAR CIRCUIT VALUE PROBLEM (PCVP) is a variant of CVP in which the underlying graph of the Boolean circuit has a planar embedding. Another variant is the MONOTONE CIRCUIT VALUE PROBLEM (MCVP) in which the Boolean circuit has only AND and OR gates. Unfortunately, both PCVP and MCVP are also shown to be P-complete [34]. Interestingly, if the Boolean circuit is simultaneously planar and monotone which is often referred to as PMCVP, then it can be evaluated in NC. The first NC algorithm for PMCVP was given in [58] that is based on the straight-line code parallel evaluation technique and runs in time $\mathcal{O}(\log^3 n)$ using $\mathcal{O}(n^6)$ parallel processors. Subsequently, a more sophisticated algorithm with the same running time but using only a linear number of parallel processors was presented in [55].

Planarity is a strong constraint for the circuit value problem and can be tested in NC. The first NC algorithm for planarity testing is due to JáJá [41]. Subsequently, Klein *et al.* described another constructive parallel algorithm, inspired by the highly efficient sequential algorithm resulting from the combined work of Lampel *et al.* [21], Even *et al.* [26] and Booth *et al.* [9]. Their approach differs significantly from the sequential algorithm that extends an embedding node by node. In contrast, they use a divide-and-conquer strategy, computing embeddings for subgraphs and combining these subgraphs to form embeddings

of larger subgraphs. In order to handle the numerous complications that arise in carrying out this approach, they made good use of an efficient data structure called *PQ-tree* [9], to represent all embeddings of each subgraph. They devised three new operations: *multiple reduction*, *intersection*, and *join* on PQ-trees, and showed that these three operations could be carried out efficiently in parallel.

In this chapter, we rename a class called *fixed-parameter parallel-tractable* (FPPT), which contains all parameterized problems that have a parallel deterministic algorithm runs in time $\mathcal{O}(f(k) \cdot (\log n)^{\alpha})$ using $\mathcal{O}(n^{\beta})$ parallel processors, where n is the input size, k is the parameter, f is an arbitrary computable function of the parameter k and independent of n, and α, β are constants independent of n and k. We initiate the study of FPPT with the general monotone circuit value, for which we propose an efficient parallel algorithm that runs in time $\mathcal{O}((k+1) \cdot \log^2 n)$ using $\mathcal{O}(n^c)$ parallel processors, where n is the number of gates, k is the genus of the underlying graph, and $\mathcal{O}(n^c)$ is the best processor boundary for parallel matrix multiplication [47]. Our algorithm for MCVP improves the previous result [48], which showed that the problem is in NC when the underlying graph of input is of genus 1. However, their approach was non-constructive and assumed that a precomputed embedding is given with the input. Besides extending the genus from 1 to the parameter k, our algorithm does not require a precomputed embedding is given with the input, and in fact, it does not rely on such an embedding at all. Thus, the MCVP is in FPPT. This result also implies that for some (not all) P-complete problems, it is possible to find an algorithm that makes the problem fall into NC by fixing one or more non-trivial parameters. Hence, if we confine ourselves to P-complete problems, an interesting analogy would be: *FPPT is with respect to P-complete what FPT is with respect to NP-complete*.

To achieve this result, we adopt the same strategy used to obtain the efficient parallel algorithm for the planarity problem in [42]. Formally, given an undirected graph, the planarity problem consists of determining whether there exists a clockwise edge ordering around each vertex such that the graph can be drawn on the plane without any edge crossing, and if so, constructing a planar embedding of the given graph. An essential component of our algorithm is the partitioning of a *directed acyclic graph* (DAG) of genus k into planar subgraphs. This could be of independent interest by itself since it applies to general DAGs, not only those that correspond to circuits (with only one sink node).

This chapter is organized as follows. Section 2.2 discusses FPPT in details. Section 2.3 describes the PQ-tree data structure and gives a detailed description of parallel operations on PQ-trees. Section 2.4 provides a general algorithm for partitioning any directed acyclic graph (not only a DAG representing a circuit) into planar subgraphs. Our algorithm and the main results are presented in Section 2.5.

2.2 Fixed-parameter Parallel-tractable (FPPT)

It is easy to observe that FPPT $\subseteq$ FPP $\subset$ PNC according to the definition. We also note here that FPP $\subseteq$ FPPT, because a parallel algorithm with runtime $\mathcal{O}(f(k) \cdot (\log n)^{\alpha})$ using $\mathcal{O}(g(k) \cdot n^{\beta})$ parallel processors can be simulated by another with running time $\mathcal{O}(g(k)f(k) \cdot (\log n)^{\alpha})$ using $\mathcal{O}(n^{\beta})$ parallel processors. Therefore, the definition of FPPT is a simplified version of FPP and emphasizes the demand for a polynomial number of processors. In fact, the parameter k may be treated like another input variable (rather than a constant), so a number of processors that varies as an arbitrary (super-polynomial) function of k is not desired. From a theoretical standpoint, an FPP-algorithm may give more information about the parameterized complexity of a problem than FPPT. If we are interested in the maximum possible size of k (as a function $k(n)$ of n), the problem is in NC as long as $k \leq k(n)$. For example, an algorithm with running time $\mathcal{O}(k \cdot \log^{\alpha} n)$ using $\mathcal{O}\left(2^k \cdot n^{\beta}\right)$ parallel processors is an NC-algorithm for $k = \mathcal{O}(\log n)$. Expressing its performance in the FPPT model $\mathcal{O}\left(k \cdot 2^k \cdot \log^{\alpha} n\right)$ using $\mathcal{O}(n^{\beta})$ processors would only yield the bound $k = \mathcal{O}(\log \log n)$.

Lemma 1 shows the relation between FPT and PNC.

Lemma 1 ([12]). *PNC is a subset of FPT.*

It is easy to observe that FPP $\subseteq$ PNC. Hence, we conclude that:

$$\text{FPPT} \subseteq \text{FPP} \subseteq \text{PNC} \subseteq \text{FPT}.$$

2.3 Parallel Operations on PQ-trees

Given a universal set $U = \{e_1, \ldots, e_n\}$, a PQ-tree is a tree-based data structure that represents all the permissible permutations over U, in which the leaves are elements of U and the internal nodes are distinguished by being labeled either as P-*nodes* or Q-*nodes*. A PQ-tree is *proper* exactly when each of the following three conditions holds:

- Every element of U appears precisely once as a leaf node;
- Every P-node has at least two children, and they might be arbitrarily permuted;
- Every Q-node has at least three children, and they are allowed only to be placed in reverse order.

Let T be a PQ-tree over the universal set U and $L(T)$ denotes the set of linear orders represented by T. We say that T generates $L(T)$. One element of $L(T)$ is obtained by reading off the leaves from left to right in order of they appear in T. The other elements are those linear orders that can be obtained by applying the three conditions mentioned above. Since there is no way to represent a PQ-tree over the empty set, we use a special null tree T_{null} to represent the empty set. With each linear ordering λ we associate the cyclic ordering $co(\lambda)$ obtained from λ by letting the first element of λ follow the last. Then the PQ-tree T represents the set of cyclic orderings $CO(T) = co(L(T))$.

Let A be a subset of the universal set U. A linear ordering $\lambda = e_1, \ldots, e_n$ of U *satisfies* the set A if all the elements of A are consecutive in λ. For a PQ-tree T, let

$$\Psi(T, A) = \{\lambda : \lambda \in L(T),\ \lambda \textit{ satisfies } A\} \tag{2.1}$$

Given any T and $A \subseteq U$, there is a PQ-tree $\hat{T}$ such that

$$L(\hat{T}) = \Psi(T, A) \tag{2.2}$$

called the *reduction* of T with respect to A. In order to parallelize the planarity testing algorithm presented in [9], Klein *et al.* introduced three new operations on PQ-trees: *multiple-disjoint-reduction, intersection* and *join* [42].

Given any T and $A_1, \ldots, A_k \subseteq U$, there is a PQ-tree $\hat{T}$ such that

$$L(\hat{T}) = \Psi(T, \{A_1 \ldots, A_k\}) \tag{2.3}$$

called the multiple-disjoint-reduce of T with respect to $\{A_1, \ldots, A_k\}$. They proposed algorithm MREDUCE($T, \{A_1, \ldots, A_k\}$) which modifies T to obtain a PQ-tree $\hat{T}$ such that $L(\hat{T}) = \Psi(T, \{A_1, \ldots, A_k\})$ if all subsets A_i's for $1 \leq i \leq k$ are pairwise disjoint. Their algorithm works in time $\mathcal{O}(\log n)$ using a linear number of parallel processors, where $n = |U|$. Note that if no ordering generated by T satisfies $\{A_1, \ldots, A_k\}$, the result of multiple-disjoint-reduce $\hat{T}$ is just the null tree T_{null}.

A PQ-tree $\hat{T}$ is the intersection of two PQ-trees T and T' over the same ground set if $L(\hat{T}) = L(T) \cap L(T')$. Klein *et al.* also proposed algorithm INTERSECT(T, T') to reduce the given PQ-trees simultaneously with respect to multiple sets that are not necessarily disjoint, using the multiple-disjoint-reduce as a subroutine. The algorithm modifies T' to be the intersection of the two original trees. INTERSECT can be computed in time $\mathcal{O}(\log^2 m)$ using m parallel processors, where m is the size of the ground set.

The last operation is join. Suppose $T_0, \ldots, T_k$ are PQ-trees over $A_0, \ldots, A_k$, respectively, and for some pairs (A_i, A_j) may overlap. We say that T is the *join* of T_0 with $T_1, \ldots, T_k$ if $CO(T) = CO(T_0)\ join\ (CO(T_1), \ldots, CO(T_k))$. To be more specific, we can compute a new PQ-tree T such that the cyclic ordering of T satisfies $A_0, \ldots, A_k$ simultaneously. The join of T_0 with $T_1, \ldots, T_k$ can be computed in time $\mathcal{O}(\log^2 n)$ using n parallel processors, where n is the total number of ground elements, using the multiple-disjoint-reduce and the intersection as subroutines.

Now, we describe how to represent the set of embeddings of a graph with a proper PQ-tree in Lemma 2.

Lemma 2. *For any node in a planar graph, all input edges and all output edges of the gate are placed consecutively in the cyclic ordering of the edges around the gate in the plane.*

Proof. Suppose C' is a graph. Let c be a node in C'. Assume that i_1 and i_2 are the two input edges of c and o_1 and o_2 are the two output edges of c, such that o_1 and o_2 interlace with i_1 and i_2 in the cyclic ordering of the edges around c. Suppose s is the single source of C' and t is the single sink of C', then there are four directed paths P_1, P_2, P_3 and P_4 in C': $P_1 = (c, o_1, \ldots, t)$, $P_2 = (c, o_2, \ldots, t)$, $P_3 = (s, \ldots, i_1, c)$, and $P_4 = (s, \ldots, i_2, c)$. These four paths cannot be embedded in a plane without having crossing edges. This contradicts with the fact that C' is a plane graph and concludes the proof. Please refer to Fig. 2.1. □

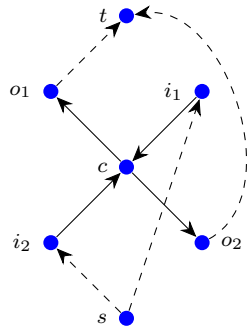
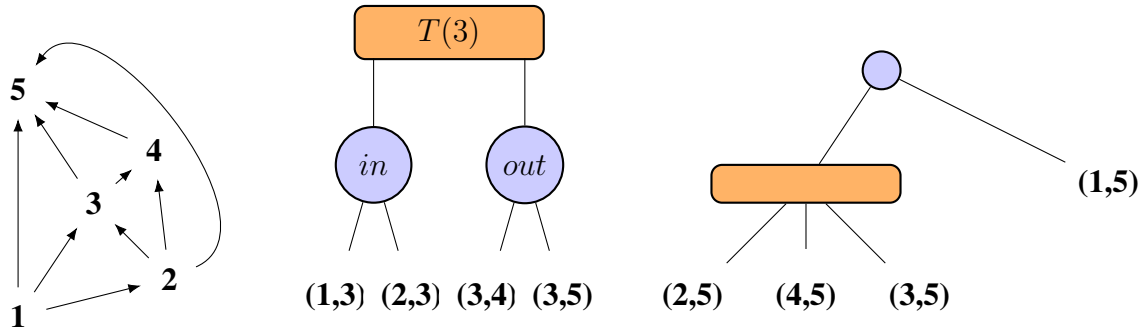


Fig. 2.1 The input edges must be consecutive for planar graph.

It has been shown that any planar graph can be represented by a valid PQ-tree [42]. Suppose $v \in G$, then we can directly construct a valid PQ-tree $T(v)$ corresponding to node v, and any cyclic ordering of the edges incident to v will be an arrangement of v, provided that the incoming edges and the outgoing edges are consecutive, respectively. In this case, we let $T(v)$ be the tree whose root is a Q-node with two P-nodes children, in and out, where the children of in are the incoming edges of v and the children of out are the outgoing edges of v. Fig. 2.2 illustrates an example of DAG and an initial PQ-tree $T(1)$ constructed from node 1.



(a) A DAG (b) PQ-tree for node 3 (c) A valid PQ-tree for the DAG

Fig. 2.2 An example of a DAG and a PQ-tree constructed for node 3 with P-nodes represented as circles, and a Q-node represented as a rectangle

2.4 Partitioning a Directed Acyclic Graph into Planar Subgraphs

Since the underlaying graph of the input monotone Boolean circuit is directed acyclic and has a genus k, we take a dived-and-conquer strategy to partitioning the graph into at most $k+1$ planar subgraphs. Thus, each piece is both monotone and planar, and we can take an NC algorithm to evaluate it.

The *genus* of a graph G is an integer that represent the maximum number of cuttings along non-intersecting closed simple curves without rendering the resultant manifold disconnected. It is equal to the number of handles on it. In particular, a graph has genus 0 if and only if it is planar. A solid torus has genus 1. See Fig. 2.3.

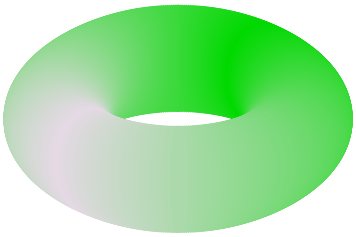


Fig. 2.3 Torus has genus 1

A block (also known as a biconnected component or 2-connected component) is a maximal biconnected subgraph. The *block decomposition* of a graph is the set of all the blocks of the graph.

Throughout the rest of this section, we make several assumptions. When we talk about genus or connectivity of a DAG, we also mean these terms of the underlying undirected

graph. If the DAG has multiple sinks, then we can eliminate all but one. We will claim this later. Therefore, throughout the rest of this chapter we assume that there is only one single sink t in the DAG. We also assume the input graph is 2-connected, which is especially needed for the relation between genus and block decomposition described in Lemma 3.

Lemma 3. [50] *The genus of a connected graph is the sum of the genus of its blocks.*

The assumption of 2-connected has no significant effect on our results since such a block decomposition can be obtained in $\mathcal{O}(\log n)$ time using $\mathcal{O}(n+m)$ parallel processors by the method of Tarjan *et al.* [57], and each block can be processed independently.

In this section we design an algorithm that partitions the input DAG with bounded genus into a series of planar subgraphs. The algorithm has five steps:

Algorithm 1 Parallel algorithm for MCVP

1: **procedure** ALG(G)
2: Transform the input graph G into a layered graph G' with procedure SPLIT(G).
3: Parallel construct PQ-tree for each node $v \in G'$ except the sink node t. Each PQ-tree represents a plane subgraph.
4: Take subgraphs that consist of nodes in consequent layers and apply the join operation on the PQ-trees in even layer with odd layer in parallel.
5: Contract the subgraphs represented by PQ-trees and update layer numbers.
6: Evaluate the planar sub-circuits in $k+1$ steps and output the result.
7: **end procedure**

The first step of our algorithm is to transform the input graph into a *proper layer graph* with the same genus:

Definition 7. *A proper layer graph is a graph $G = (V, E)$, with vertex set $V = V_1 \cup V_2 \cup \cdots \cup V_p$, $V_i \cap V_j = \emptyset$, and edge set $E = E_1 \cup E_2 \cup \cdots \cup E_{p-1}$, $E_i \subseteq V_i \times V_{i+1}$.*

To do so, we first flip the directions of edges in G to obtain G^* and let $d(t, v)$ be the length of the longest directed path from sink node t to each node v in G^*.

The layer number $\mathfrak{d}(v)$ of each node v defined as follows:

$$\mathfrak{d}(v) = d(t, v). \tag{2.4}$$

In this step, we assign to each node v of the given DAG G the layer number $\mathfrak{d}(v)$ and then add some dummy nodes to transform the input graph into a proper layered graph. Algorithm 2 formally describes this procedure.

Algorithm 2 Split of DAG

procedure SPLIT(G)
  Calculate the layer number $\mathfrak{d}(v)$ for every node $v \in G$.
  for all directed edges (u, v) in G **do**
    Let $l = \mathfrak{d}(u) - \mathfrak{d}(v) - 1$.
    if $l > 0$ **then**
      Add dummy nodes $n_1, \ldots, n_l$ and directed edges $(u, n_1), (n_1, n_2), \ldots, (n_l, v)$ to the graph G.
    end if
  end for
  Together with added dummy nodes and edges we obtain DAG G', such that for any edge (u, v), $l = \mathfrak{d}(u) - \mathfrak{d}(v) - 1 = 0$.
end procedure

Note that for any directed edge $(u, v) \in G'$, it holds that $l = 0$. In other words, all edges are situated between adjacent layers, i.e., no edges are crossing more than two layers and no edges in the same layer.

We still need to show that Algorithm 2 can be accomplished in NC time and the layered graph G' has the same genus as G. It is easy to see that the layer number for each node can be computed in NC time, and more precisely in NC^2 by using parallel topological sorting algorithms, such as that in [16]. So, we only need to observe that:

Lemma 4. *Graphs G and G' have the same genus.*

Proof. Since graph G' is obtained only by edge subdivision of graph G and the edge subdivision operation will not change the genus of a graph, then G and G' have the same genus. □

It is easy to see that the parallel construction of PQ-trees for each node $v \in G'$, except for the sink node t, takes only constant time with a linear number of parallel processors since we only need to rearrange the input and output edges of a node.

Next, we describe how to contract the subgraphs. We start with the original layered graph and let $G^{(0)} = G'$. In the i^{th} stage, we choose a collection of subgraphs of the graph $G^{(i)}$ in accordance with the layer number, contract these subgraphs and update the layer number. This results in the graph $G^{(i+1)}$.

For each node $v \in G^{(i+1)}$ and each stage $j \leq i$, we denote by $H^{(j)}(v)$ the subgraph of $G^{(j)}$ that was contracted over steps $j+1, \ldots, i$ forming v. We use $H(v)$ for $H^{(0)}(v)$. If $u \in H^{(j)}(v)$ for $v \in G^{(i+1)}$, we use u^{i+1} to denote v.

We choose our subgraphs to contract at each stage i such that the following properties hold:

- At most $\mathcal{O}(\log n)$ stages are needed;
- The sink node is never contracted with any other node;
- For each node $v \neq t$ in $G^{(i)}$, the subgraph $H(v)$ permits a PQ-tree representation of the set of its embeddings;
- The layer number is easy to update, following the contraction of the edges.

The second step of our algorithm can be viewed as a contraction process over the layers in parallel.

We first show that the algorithm terminates in $\mathcal{O}(\log n)$ stages. Then we show how the subgraphs are chosen and prove that our method of selecting the subgraphs satisfies the above properties.

Lemma 5. *Algorithm 1 terminates in $\mathcal{O}(\log n)$ stages.*

Proof. We already showed how to transform the input DAG into a layered DAG such that all directed edges go from layer $i+1$ to layer i in the layered DAG in Algorithm 2. To ensure that only $\mathcal{O}(\log n)$ stages are needed, we contract the nodes as follows. Suppose that there is one node u at layer $i+1$ and its neighbor set in layer i is $\{v_1, \ldots, v_m\}$. Moreover, assume that the other neighbor of each v_i in the neighbor set of u in layer $i+1$ is $\{u_1, \ldots, u_h\}$. Since the input is a monotone boolean circuit, there are only two input wires for each gate. If (u, v_i) is already an input wire for gate v_i, then there must exist another input wire coming from gate $\{u_1, \ldots, u_h\}$ to gate v_i. We will contract the edges incident to u, $\{v_1, \ldots, v_m\}$, and $\{u_1, \ldots, u_h\}$ together.

In each stage, either the number of layers is reduced by two (contract success) or the nodes in adjacent layers cannot be contracted to form a larger planar subgraph. Hence, for the latter, we cut all edges between layer $i+1$ and i. After cutting the edges between layer $i+1$ and i, the graph is split into two parts. One part is below layer i (including layer i) which has some hanging incoming edges, and the other part is above layer $i+1$ (including layer $i+1$) which has some hanging outgoing edges. It is easy to see that the first part is still a connected directed acyclic graph with sink node t. However, the second part could be a disconnected graph because the adjacent layers may not be a complete bipartite graph. So, we add a new sink node t' to the second part and draw all the hanging outgoing edges to t'. Apparently, this step guarantees that the second part is a connected directed acyclic graph (refer to Fig 2.4 for an illustration). Then, we handle these two subgraphs in parallel.

□

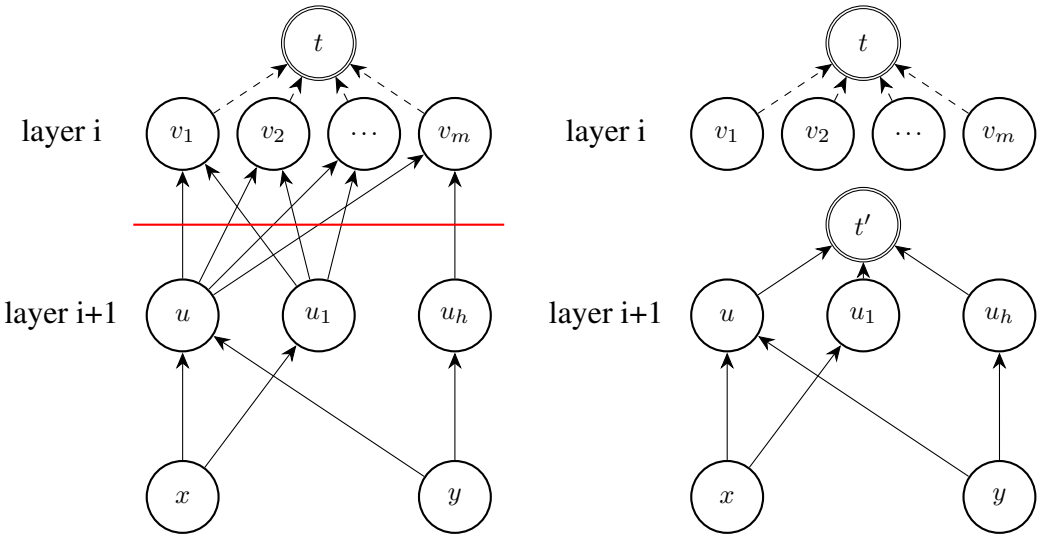


Fig. 2.4 Cut operation with a new sink node.

At every stage i, we compute these PQ-trees for each new node $v \in G^{(i+1)}$ in parallel from the PQ-trees for the nodes $H^{(i)}(v)$ identified to form v.

If a null tree T_{null} arises as $T(v)$ for some node v, then there are no arrangements of v. So, the candidate subgraphs cannot form a larger planar subgraph. Assume on the other hand that the contraction process continues until there is only one node left other than the sink node in G^i. Then every internal node is adjacent only to the sink node. Let $\{v_1, \ldots, v_m\}$ be these nodes. For $j = i, \ldots, m$, if $T(v_j)$ is not T_{null}, then there is a planar embedding for each internal node v_j. So both the input edges and the output edges form a consecutive subsequence, and the graph is planar.

We now show why the cut operation reduces the genus of both parts by at least 1 and how the circuit will be split into $k+1$ planar sub-circuits in Lemma 6 and Lemma 7.

Lemma 6. *The cut operation reduces the genus of subgraphs by at least* 1.

Proof. Suppose that some nodes in layer $i+1$ and i cannot be contracted, then either there exists at least one node u in layer $i+1$ such that its incoming edges from layer $i+2$ interlace with the outgoing edges to layer i, or there exists at least one node v in layer i such that its incoming edges from layer $i+1$ interlace with the outgoing edges to layer $i-1$. Irrespective of both of these scenarios, we do the following and obtain a new graph containing only two blocks. We delete all directed edges between layers i and $i+1$ and then add one new node t' as the sink node of the second part and the source node of the first part. Suppose the genus of the first block is g_1 and the genus of the second block is g_2. Then the genus of the first block is equal to the genus of the first part of the graph after cutting, and the genus of the second

block is equal to the genus of the second part of the graph after cutting. The cut operation will reduce the genus of the layered graph by 1.

Hence, according to Lemma 3 we have $g_1 + g_2 + 1 = k$. This implies that $g_1 + g_2 < k$. Since $g_1 \geq 0$ and $g_2 \geq 0$, then $g_1 < k$ and $g_2 < k$.

□

Lemma 7. *Algorithm 1 splits the input circuit with bounded genus k into $k+1$ planar circuits.*

Proof. We prove this lemma by induction. Without loss of generality, after the first cut, we observe that $g_1 \leq g_2$ and the genus of the layered graph reduces by 1. Now, we have two separate graphs. We consider different combinations of g_1 and g_2 as follows:

- if g_1 is 0, then it is a planar subgraph. This graph is represented by one PQ-tree. The other part will have genus $g_2 = k - 1$. In this case, the genus reduces by 1 and the number of subgraphs increases by 1.
- if g_1 is 1, then it is not a planar subgraph, and it will be cut into two subgraphs later. The other part will have genus $g_2 = k - 2$.

Hence, we conclude that every cut will reduce the genus, at least by 1, and hence there are at most $k+1$ planar subgraphs.

□

2.5 MCVP is in FPPT

In this section, we formally present our main result which can be summarized in the following theorem:

Theorem 2. *Given a general monotone boolean circuit with n gates and an underlying graph of genus k, it can be evaluated in time $\mathcal{O}((k+1) \cdot \log^2 n)$ using $\mathcal{O}(n^c)$ parallel processors, where $\mathcal{O}(n^c)$ is the best processors boundary for parallel matrix multiplication. Hence, the monotone circuit value problem parameterized by genus is in FPPT.*

Our main result follows from the lemmas above together with the parallel evaluation technique for layered planar monotone circuit value problem that runs in time $\mathcal{O}(\log^2 n)$ using a linear number of parallel processors [56]. By referring to Fig. 2.4, once the circuit with output t' has been evaluated, $u, u_1, \ldots, u_h$ are known as well, which then allows the inputs $v_1, v_2, \ldots, v_m$ of the next circuit to be set up in constant time so that the next circuit can be evaluated. By referring to Lemma 7, a genus k circuit might be split into at most $k+1$

subcircuits. Since each subcircuit it both planar and monotone and also layered, we can take the layered MCVP algorithm to evaluate it in $K+1$ steps. We note that the processor demand is bounded to $\mathcal{O}(n^c)$, which is the processor boundary for the parallel matrix multiplication algorithm. This is only because we need a parallel topological sorting algorithm to compute the layer numbers. Otherwise, our algorithm will only need a linear number of processors.

Chapter 3

Parallel Crown Decomposition and Parameterized Vertex Cover Problem

3.1 Introduction

In this chapter, we extend the FPPT framework to a kernelization method called *crown decomposition* that is used in FPT to cope with a number of NP-complete problems, such as the VERTEX COVER PROBLEM, the MAXIMUM SATISFIABILITY PROBLEM, *etc*. We show that the crown decomposition can be computed in time $\mathcal{O}(k \cdot \log^3 n)$ using $\mathcal{O}(m)$ parallel processors, where m is the number of edges, n is the number of vertices of the input graph, and k is the size of the sought vertex cover. Following the parallel crown decomposition procedure, we directly obtain an efficient parallel algorithm for the parameterized vertex cover problem that outperforms the best known parallel algorithm for this problem [12] : using $\mathcal{O}(m)$ instead of $\mathcal{O}(n^2)$ parallel processors, the running time improves from $4\log n + \mathcal{O}(k^k)$ to $\mathcal{O}(8^k + k \cdot \log^3 n)$, where m is the number of edges, n is the number of vertices of the input graph, and k is an upper bound of the size of the sought vertex cover. Since the crown structure admits a kernel with size at most $3k$, the parallel crown decomposition and the parameterized vertex cover problem are in FPPT.

This chapter is organized as follows. Section 3.2 shows that the crown decomposition can be computed in time $\mathcal{O}(8^k + k \cdot \log^3 n)$ using $\mathcal{O}(m)$ parallel processors, where m is the number of edges, n is the number of vertices of the input graph, and k is the size of the sought vertex cover. Section 3.3 shows the MAXIMUM MATCHING PROBLEM parameterized by the size of the matching is in FPPT, which is a crucial subroutine for computing the crown decomposition in parallel.

3.2 Parallel Crown Decomposition

Kernelization is a polynomial-time transformation that reduces an arbitrary instance (I, k) of a parameterized problem to an equivalent instance (I', k'), such that $k' \leq k$ and $|I'|$ is bounded by some function of k. The resulting instance is often referred to as a problem kernel of the instance. A problem is FPT if and only if it has a kernelization algorithm [23].

Crown decomposition is a general kernelization technique that used in FPT to cope with a number of NP-complete problems. The technique is based on the classical matching theorems of Kőnig and Hall. Recall that for a graph $G = (V, E)$, a *matching* M is a subset of E such that no two edges in M share a common vertex. A vertex that is incident to an element of M is said to be matched under M. A matching M *saturates* a set of vertices U when every vertex in U is matched under M. A matching is maximal if it is not contained in a larger matching. Such a matching is *maximum* if no matching of larger cardinality exists. If all vertices are matched under a matching, then it is a *perfect matching*. The problem of finding a maximum matching or a perfect matching, even in planar graphs, has received considerable attention in the field of parallel computation.

Definition 8 ([23])**.** *A crown decomposition of a graph $G = (V, E)$ is a partitioning of V into three parts C, H and R, such that*

1. *C is nonempty.*
2. *C is an independent set.*
3. *There are no edges between vertices of C and R. That is, H separates C and R.*
4. *Let E' be the set of edges between vertices of C and H. Then E' contains a matching of size $|H|$. In other words, G contains a matching of H into C.*

The set C can be seen as a crown put on head H of the remaining part R. A *straight* crown is a crown (C, H) that satisfies the condition $|C| = |H|$. A *flared* crown is a crown (C, H) that satisfies the condition $|C| > |H|$. These notitions are depicted in Fig. 3.1. Note that E' contains a matching of size $|H|$ implies that there is matching of H into C. This is a matching in the subgraph G', with the vertex set $C \cup H$ and the edge set E', saturating all the vertices of H.

For finding a crown decomposition in polynomial time, we use the following well known structure and algorithmic results. The first is Theorem 3 due to Kőnig.

Theorem 3. *In every undirected bipartite graph the size of a maximum matching is equal to the size of a minimum vertex cover.*

Theorem 4 (Hall's theorem)**.** *Let G be an undirected bipartite graph with bipartition (V_1, V_2). The graph G has a matching saturating V_1 if and only if for all $X \subseteq V_1$, then we have $|N(X)| \geq |X|$.*

Theorem 5 is due to Hopcroft and Karp.

Theorem 5. *Let G be an undirected bipartite graph with bipartition (V_1, V_2), on n vertices and m edges. Then we can find a maximum matching as well as a minimum vertex cover of G in time $\mathcal{O}(m\sqrt{n})$. Furthermore, in time $\mathcal{O}(m\sqrt{n})$ either we can find a matching saturating V_1 or an inclusion-wise minimal set $X \subseteq V_1$ such that $|N(x)| < |X|$.*

Given an undirected graph $G = (V, E)$, the parameterized vertex cover problem asks whether there is a set of vertices $V' \subseteq V$ of size at most k (k is the parameter), such that for every edge $(u, v) \in E$, at least one of its endpoints, u or v is in V'. In other words, the complement of V' is an independent set (i.e., a set that induces an edge-less subgraph).

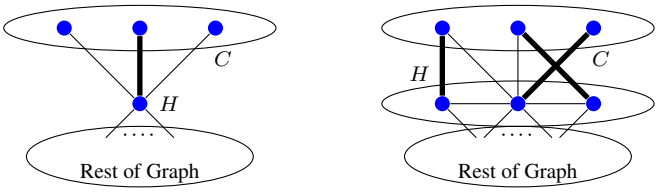


Fig. 3.1 Sample crowns (Bold edges denote a matching)

In the following, we present an efficient parallel algorithm for the parameterized vertex cover problem which is based on the parallel crown decomposition of the input graph.

Algorithm 3 Parallel algorithm for finding a crown.

procedure PARALLELCROWN(G)

Step 1: Find a maximal matching M_1 in parallel and identify the set of all unmatched vertices as the set O of outsiders.

Step 2: Find a maximum auxiliary matching M_2 of the edges between O and $N(O)$ in parallel.

Step 3: If every vertex in $N(O)$ is matched by M_2, then $H = N(O)$ and $I = O$ form a straight crown, and we are done.

Step 4: Let I_0 be the set of vertices in O that are unmatched by M_2. Repeat Steps 5a and 5b until $n = N$ so that $I_{N-1} = I_N$.

Step 5:

5a. Let $H_n = N(I_n)$.

5b. Let $I_{n+1} = I_n \cup N_{M_2}(H_n)$.

Step 6: $I = I_N$ and $H = H_N$ form a flared crown.

end procedure

Suppose that the rest graph G' is produced by removing vertices in I and H and their adjacent edges. The size of G' is n', thus $n' = n - |I| - |H|$ and the parameter size is $k' = k - |H|$. It is easy to observe that if a maximum matching of size greater than k is found, then there has no vertex cover of size at most k, and the algorithm returns a "no" instance. Therefor if either of the matchings M_1 and M_2 is larger than k, the process can be terminated. This fact also allows to place an upper bound on the size of the graph G'.

Theorem 6. *If both the matchings M_1 and M_2 are of size less than or equal to k, then the crown structure admits a kernel with size at most $3k$.*

Proof. Since the size of the matching M_1 is less than or equal to k, it contains at most $2k$ vertices. Thus, the set O contains at least $n - 2k$ vertices. Since M_2 is less than or equal to k, there are at most k vertices in O that are matched by M_2. Thus, there are at least $n - 3k$ vertices that are in O that are unmatched by M_2. These vertices are included in I_0 and are therefore in I. Thus the largest number of vertices in G that are not included in I and H is $3k$. In other words, the crown structure admits a kernel with size at most $3k$. □

We analyze the running time and processor utilization in the following. Apparently, the most expensive part of this procedure are Step 1 and Step 2.

Israeli *et al.* presented a parallel algorithm for the maximal matching problem which can be summarized by Lemma 8:

Lemma 8 ([40])**.** *A maximal matching in general graphs can be found in time $\mathcal{O}(\log^3 n)$ using $\mathcal{O}(m)$ parallel processors, where m is the number of edges and n is the number of vertices of the input graph.*

So we only need to show that the maximum matching M_2 in Step 2 can be constructed efficiently in parallel, which will be discussed in section 3.3 where we prove that the construction of such a maximum matching is in FPPT.

Consequently, since the crown structure admits a kernel with size at most $3k$, we can cope with the reduced instance of vertex cover in $\mathcal{O}(f(k))$-time. This finishes the proof of Theorem 7.

Theorem 7. *The parameterized vertex cover problem is in FPPT.*

3.3 Parameterized Maximum Matching Problem

In this section, we consider a parameterized version of the maximum matching problem stated below, as a subroutine of the parallel crown reduction procedure. Note that, in our definition of the parameterized maximum matching problem, we ask the question whether the cardinality of the maximum matching is equal to or less than parameter k. (This is to be contrasted with the usual parameterization of a maximization problem where one would ask for a solution of size k or more).

Input:	A graph $G = (V, E)$ and a positive integer k.
Problem:	Is there a matching M of size at most k?
	If yes, how to construct one?

To the best of our knowledge, this is the first time that the maximum matching problem is studied with an input parameter given, especially in a way that upper-bounds the size of the sought matching. In classical computational complexity, the maximum matching problem has the same complexity as the perfect matching problem, since the former can be easily reduced to the latter in logarithmic space. Suppose we want to check whether there is a matching with k edges in G. If such a matching existed, $2k$ vertices would have been matched and $n-2k$ would have been "free." We add $n-2k$ new vertices to G and create edges between these new vertices and all old vertices in G in order to obtain a new graph G'. Thus, G' has a perfect matching if and only if G has a matching with exactly k edges. Conversely, perfect matching parameterized by the number of matching edges is trivially FPPT by a simple reduction to our version of parameterized maximum matching (if $n \neq 2k$ then it is a no instance). So the two problems have the same parameterized parallel complexity with

respect to the parameter k. However, when the parameter is the number of perfect matchings, perfect matching falls in the class FPPT, more precisely in NC in this case [4], while it is not known yet whether maximum matching has an NC algorithm when the number of maximum matchings is bounded by a polynomial in n. This problem is open at this stage. The related studies on perfect matching, including NC algorithms for some special structures of graphs or parameters, such as: dense graphs [20], regular bipartite graphs [46], claw-free graphs [15], bipartite graphs with polynomially bounded number of perfect matchings [37], general graphs with polynomially bounded number of perfect matchings [4], bipartite planar and small genus graphs [49].

Suppose that G is a graph with minimum vertex cover bounded by a constant k. It is not hard to see that the maximum matching of G must be bounded to k as well (since the edges in a maximum matching are pairwise disjoint, at least one end of each edge should be included in the minimum vertex cover). Therefore, in order to cover all edges in a maximum matching, the minimum vertex cover must be greater than or equal to the size of a maximum matching.

Our algorithm is based on the augmenting path approach used in the classical *Blossoms* algorithm. We show that, with careful analysis, the parameterized maximum matching problem can be coped with in time $\mathcal{O}(f(k) \cdot \log^3 n)$ using $\mathcal{O}(m)$ parallel processors, where m is the number of edges and n is the number of vertices of the input graph. The algorithm is summarized in Algorithm 4.

Algorithm 4 Parallel algorithm for parameterized maximum matching

1: **procedure** FINDMAXIMUMMATCHING(G,k) ▷ Graph G and parameter k
2: Step 1: Find a maximal matching M_1 in parallel. If $|M_1| > k$, return false.
3: Step 2: Construct a new graph by merging the unmatched vertices into a new vertex S.
4: Step 3: Construct an alternating BFS tree rooted at S in parallel.
5: Step 4: Either find an augmenting path with respect to M_1, or construct a new graph with a maximum matching of cardinality $k-1$.
6: Step 5: Repeat Step 1 to Step 4 at most k rounds.
7: **end procedure**

It is well known that the size of any maximal matching is at least half the size of a maximum matching. Therefore, we note that the maximal matching produced in Step 1 is not less than $\frac{k}{2}$.

In Step 2, we make a transformation as follows. We merge all unmatched vertices into a single vertex S such that all edges connected to the unmatched vertices become incident to

S, thus making all augmenting paths in the graph G transfer into an odd alternating cycle in the new graph.

Next, we construct an alternating BFS tree with the following properties:

- S is the root (and layer 0);
- All vertices adjacent to S are in layer 1;
- All edges from odd layers to even layers are matching edges (elements of the matching M_1 constructed in Step 1).

Note that we also add the unmatched edges into the alternating BFS tree in Fig. 3.2 to help in the analysis. This will be made clear in the sequel.

The parallel alternating BFS tree can be constructed in time $\mathcal{O}(\log^2 n)$ using $\mathcal{O}(n^3)$ parallel processors [33]. However, the graph for which the BFS is constructed consists of at most $2k+1$ vertices because all unmatched vertices are merged into S. Thus, this step can be completed in time $\mathcal{O}(\log^2 k)$ using $\mathcal{O}(k^3)$ parallel processors. Consequently, we observe the following:

- All vertices in the tree are matched except for S. All edges between layer 0 and layer 1 are unmatched, otherwise S will be matched.
- If the nodes in layer 1 have no descendants (i.e., neighbors in a layer of a higher index), then they must be matched in this layer (e.g., b and c in Fig. 3.2). Otherwise, these nodes would be unmatched. We call such an edge type B edge. The unmatched edges in the same layer are also type B edges.
- The unmatched edges from odd layers to descendant layers are of type A (e.g., (d, f) and (a, k) in Fig. 3.2).
- The unmatched edges from even layers to the adjacent odd layer are of type C (e.g., (h, j) in Fig. 3.2).
- The depth of the tree is at most $2k$ because the size of the maximum matching is bounded by k.

Lemma 9. *If M_1 is not a maximum matching, then every alternating odd cycle must pass through at least one type B edge.*

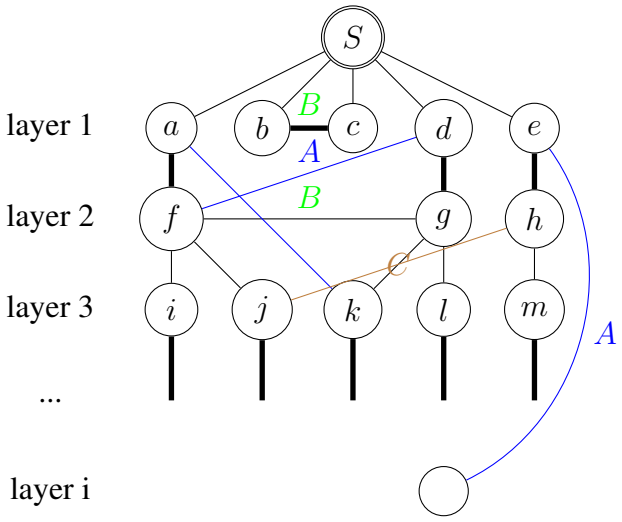


Fig. 3.2 An alternating BFS tree. (Bold edges denote matching edges)

Proof. Suppose there is an augmenting path $p = e_1, e_2, \ldots, e_h$ that consists of type A, type C, and matching edges. According to the observation above, the alternating path p will be transfered to an alternating odd cycle in Fig. 3.2. Thus, the initial vertex and the end vertex of p must be S and e_1 must be an unmatched edge between layer 0 and layer 1 (e.g., (S, e) in Fig. 3.2). For p to be an augmenting path, e_2 must be a matched edge. Since it is not allowed to take type B edges, we have to go further down (e.g., (e, h) in Fig. 3.2). Then, e_3 must be a type C edge (e.g., (h, j) in Fig. 3.2). In order to construct an augmenting path, e_4 must be an matching edge, which starts from an odd layer and goes future down in the tree. So the end point of e_4 must be in in even layer. Following this procedure, in some future layer of the path, there must has an edge back to previous layer and to S in the end. If this edge locates at odd layer, it must be a matched edge, so it is type B edge. If this edge locates at even layer, it must be an unmatched edge, so it is also a type B edge. So it is impossible to construct an augmenting path without a type B edge. □

Since each augmenting path transforms into an alternating odd cycle and increases the matching by 1, we only need to check if there are type B edges in the alternating BFS tree. If yes, we proceed by checking whether the alternating odd cycle comes from a valid augmenting path. Otherwise, we either get a maximum matching with cardinality at most k or reduce the graph to a new graph with the maximum matching of size bounded by $k-1$. Now, we analyze this procedure by considering the following two cases:
Case 1: Suppose an alternating odd cycle resulted from an augmenting path is $(u_1 - a - b - u_2)$ and the edge (a, b) is matched. a, b connect to different vertices u_1 and u_2 in S. Then we can obtain a larger matching by changing (a, b) to be unmatched and $(u_1, a), (u_2, b)$ to be

matched. Refer to Fig. 3.3(b) for an illustration. Similarly, if the edge (a,b) is an unmatched type B edge, then we can obtain a larger matching by changing $(c,a),(d,b)$ to be unmatched and $(u_1,c),(a,b),(u_2,d)$ to be matched. Refer to Fig. 3.3(c) for an illustration.

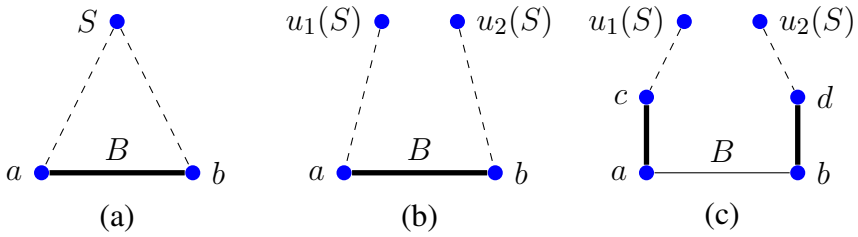


Fig. 3.3 An alternating odd cycle passes through type B edges. (a) an alternating odd cycle; (b) (a,b) is a matched type B edge; (c) (a,b) is an unmatched type B edge.

Case 2: Suppose that an alternating odd cycle transformed from an augmenting path is $(u-a-b-u)$ and the edge (a,b) is matched. Also a and b connect to the same vertices u and there is a alternating path from S to u. We do not know if there exists an augmenting path, neither how to find one, if there is any. For this case, the alternating odd cycle is called a blossom structure. Refer to Fig. 3.4 for an illustration. Since a blossom contains at least one matched edge and three nodes, we can shrink the blossom to a single vertex and the new graph obtained will have a maximum matching of cardinality at most $k-1$. Since the size of the maximum matching is bounded by k, at most k rounds are needed to construct a maximum matching.

Lemma 8 indicates that the parallel algorithm for maximal matching has a running time $\mathcal{O}(\log^3 n)$, and the depth of the alternating BFS tree is at most $2k$. In each round, the size of the matching will increase at least by one. Thus, the running time will be $\mathcal{O}(k \cdot \log^3 n)$ with $\mathcal{O}(m)$ parallel processors, where m is the number of edges, n is the number of vertices of the input graph, and k is an upper bound of the size of the sought maximum matching. Given all of the above, we can now state theorem 8.

Theorem 8. *Maximum matching parameterized by an upper bound on the matching size, is in FPPT.*

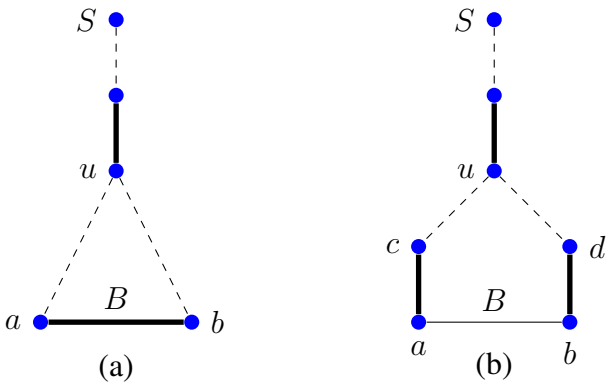


Fig. 3.4 Blossom structure.

Chapter 4

Parallel Algorithms Parameterized by Modular-width

4.1 Introduction

One of the most ubiquitous parameters studied in both sequential and parallel scenarios is *treewidth*, which roughly measures how a graph likes a tree. In general, algorithms for a problem on graphs of bounded treewidth are much more efficient than their counterparts in general graphs (e.g., see [6] and [8]). A celebrated meta-theorem of Courcelle [17] states that any problem expressible in monadic second-order logic is FPT when parameterized by the treewidth of the input graphs. Similarly, Cesati *et al.* showed that all problems involving *MS* (definable in monadic second-order logic with quantifications over vertex and edge sets) or *EMS properties* (that involve counting or summing evaluations over sets definable in monadic second-order logic) are FPP when restricted to graphs of bounded treewidth [12]. Moreover, Lagergren *et al.* presented an efficient parallel algorithm for the TREE DECOMPOSITION PROBLEM when the treewidth is a constant [44] . Hagerup *et al.* showed that the MAXIMUM NETWORK FLOW PROBLEM is in NC when the treewidth is a constant [39]. Despite the fact that these results are noteworthy, one major drawback of treewidth is that a large number of instances are excluded, since graphs of small treewidth are necessarily sparse. The notion of clique-width (as well as rank-width [54], boolean-width [11] and shrub-depth [32]), which is stronger than treewidth, tries to address this problem by covering a larger family of graphs, including many dense graphs. However, the price for this generality is exorbitant. Several problems which were shown in FPT when parameterized by treewidth become intractable when parameterized by these parameters, such as the MAXIMUM CUT PROBLEM, the CHROMATIC NUMBER PROBLEM, the HAMILTONIAN CYCLE PROBLEM, and the EDGE DOMINATING SET PROBLEM [27, 29, 28].

Consequently, a parameter called *modular-width* that covers a significantly larger class of graphs has been introduced by Gajarský *et al.* in [31]. They showed that several problems are in FPT when parameterized by modular-width, whereas these problems become intractable when parameterized by clique-width and shrub-depth, such as the chromatic number problem and the PARTITIONING INTO PATHS PROBLEM (hence the HAMILTONIAN PATH PROBLEM and the hamiltonian cycle problem).

In this chapter, we extend the study of modular-width to parameterized parallel complexity and show that the WEIGHTED MAXIMUM CLIQUE PROBLEM and the MAXIMUM MATCHING PROBLEM are in FPPT when parameterized by modular-width. These results are of interests for several reasons. First, not only for P-complete and NP-complete problems but also for those that are still open for P-complete or NC, there are parameterized parallel algorithms with non-trivial parameters for them. Thus, FPPT is orthogonal to P-complete, NP-complete and probably some unknown classes between NC and P-complete (if P is not equal to NC). Second, there exist some parameters that make a large number of problems in FPPT which are in different complexity classes in the traditional hierarchy.

Our algorithms are based on the following ideas/techniques: we use an algebraic expression to represent the input graph. Then, we construct the modular decomposition tree of the input graph and a computation tree that corresponds to the maximal strong modules in the maximal decomposition tree. For each node of the computation tree, we compute an optimal solution by giving the optimal solutions for the children of this node in the modular decomposition tree. The optimal solution for each node can be obtained by integer linear programming. Then, we use a bottom-up dynamic programming approach along with the modular decomposition tree to obtain the global solution. In order to explore and evaluate the tree efficiently, we use a parallel tree contraction technique due to Miller *et al.* [51].

4.2 Modular Decomposition

All graphs considered in this chapter are simple, undirected and loopless. We use the classical graph theoretic notations and definitions (e.g. see [38]). The neighborhood of a vertex x in a graph $G = (V, E)$ is denoted by $N(x)$. Given a subset of vertices $X \subseteq V$, $G[X]$ denotes the subgraph induced by X.

Let M be a subset of vertices of a graph G, and x be a vertex of $V \setminus M$. We say that vertex x *splits* M (or is a *splitter* of M) if M contains a neighbor and a non-neighbor of x. If x does not split M, then M is *homogeneous* with respect to x.

Definition 9. *Given a graph $G = (V, E)$, $M \subseteq V$ is called a* module *if M is homogeneous with respect to any vertex $x \in V \setminus M$ (i.e. $M \subseteq N(x)$ or $M \cap N(x) = \emptyset$).*

Let M and M' be disjoint sets. We say that M and M' are adjacent if any vertex of M is adjacent to all the vertices in M' and non-adjacent if the vertices of M are non-adjacent to the vertices of M'. Thus, it is not hard to observe that two disjoint modules are either adjacent or non-adjacent.

A module M is *maximal* with respect to a set S of vertices if $M \subset S$ and there is no module M' such that $M \subset M' \subset S$. If the set S is not specified, we assume that $S = V$. A module M is a *strong module* if it does not overlap with any other module. Note that, one-vertex subsets and the empty set are modules and are known as the *trivial* modules. A graph is called *prime* if all of its modules are trivial.

Definition 10. *Let $P = \{M_1, \ldots, M_k\}$ be a partition of the vertex set of a graph $G = (V, E)$. If for all i, $1 \leq i \leq k$, M_i is a non-trivial module of G, then P is a* modular partition *of G.*

A non-trivial modular partition $P = \{M_1, \ldots, M_k\}$ which only contains maximal strong modules is a *maximal modular partition*. Note that each undirected graph has a unique maximal modular partition [45]. If G (resp. $\overline{G}$) is not connected then its connected (resp. co-connected) components are the elements of the maximal modular partition.

Definition 11. *For a modular partition $P = \{M_1, \ldots, M_k\}$ of a graph $G = (V, E)$, we associate a* quotient graph $G_{/p}$*, whose vertices are in one-to-one correspondence with the parts of P. Two vertices v_i and v_j of $G_{/p}$ are adjacent if and only if the corresponding modules M_i and M_j are adjacent in G.*

The inclusion tree of the strong modules of G, called the *modular decomposition tree*, entirely represents the graph if the representative graph of each strong module is attached to each of its nodes (see Fig. 4.1). It is easy to observe that there are only three relations, $M \subseteq M'$, $M' \subseteq M$, or $M \cap M' = \emptyset$, for any two nodes M and M' in the modular decomposition tree. The *modular-width* is the maximum degree of the modular decomposition tree. An excellent feature of modular decomposition is that it can be computed in $\mathcal{O}(\log^2 n)$ time with $\mathcal{O}(n+m)$ parallel processors [19], thus the modular-width stays also within the same resource bounds.

Theorem 9 (Modular decomposition theorem [13])**.** *For any graph $G = (V, E)$, one of the following three conditions is satisfied:*

1. *G is not connected;*
2. *$\overline{G}$ is not connected;*
3. *G and $\overline{G}$ are connected and the quotient graph $G_{/P}$, where P is the maximal modular partition of G, is a prime graph.*

Theorem 9 indicates that, the quotient graphs associated with the nodes of the modular decomposition tree of the strong modules are of three types: an *independent set* if G is not connected (the node is labeled *parallel*); a clique if $\overline{G}$ is not connected (the node is labeled *series*); a prime graph otherwise.

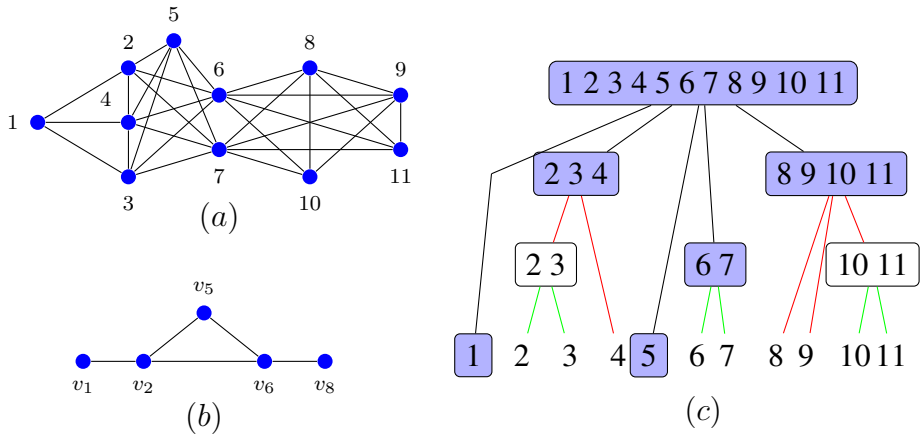


Fig. 4.1 (a) shows the graph G; $\{\{1\},\{2,3\},\{4\},\{5\},\{6,7\},\{9\},\{8,10,11\}\}$ is a modular partition of G. The maximal modular partition of G is $P = \{\{1\},\{2,3,4\},\{5\},\{6,7\},\{8,9,10,11\}\}$ and (b) represents its quotient graph. (c) is the modular decomposition tree of G. The maximal strong modules are in blue. The green edges indicate that the root node is parallel, the red edges indicate that the root is series, and the black edges indicate that the root is a prime graph.

Parallel tree contraction is a "bottom-up" technique for constructing parallel algorithms on trees. There are two basic operations called *rake* and *compress*. During each contraction, processors are assigned to leaves of the tree and perform local modifications by removing these leaves, hence creating new leaves that are processed at the next round. This operation is called rake. Clearly, removing leaves is not sufficient for a tree that is thin and tall, like a linked list, which would take a linear number of rounds to reduce the tree to a point. Thus, a complementary operation called compress that reduces a chain of vertices, each with a single child to a chain of half the length is introduced. Ideally, rake and compress work on different parts of the tree simultaneously. During the run of the algorithm, the rake operation tends to produce chains that are then reduced by the compress operation. Thus, the whole tree can be evaluated in $\mathcal{O}(\log n)$ time using $\mathcal{O}(n)$ parallel processors.

4.3 Applications Parameterized by Modular-width

In this section, we show how modular-width can be used to derive efficient parallel algorithms for the weighted maximum clique problem and the maximum matching problem. Our results imply that these two problems are in FPPT.

The input to our algorithms is assumed to be a graph of modular-width at most k, and we shall represent the input graph as an algebraic expression consisting of the following operations:

1. G has only one vertex. This corresponds to a leaf node in the modular decomposition tree.

2. G is a *disjoint union* of two graphs G_1 and G_2 of modular-width at most k. The disjoint union of G_1 and G_2 defined as a graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$. This corresponds to a parallel node in the modular decomposition tree.

3. G is a *complete join* of two graphs G_1 and G_2 of modular-width at most k. The complete join of G_1 and G_2 is defined as a graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2 \cup \{\{u, v\} : u \in V_1 \text{ and } v \in V_2\}$. This corresponds to a series node in the modular decomposition tree.

4. The *substitution* operation with respect to a graph G is the reverse of the quotient operation and defined as replacing a vertex of G by $G_i = (V_i, E_i)$ of modular-width at most k while preserving the neighborhood,

$$G_{x \to G_i} = (V \setminus \{x\} \cup V_i,$$
$$(E \setminus \{(x, y) \in E\} \cup E_i \cup \{(y, z) : (x, y) \in E, z \in V_i\})).$$

   This is corresponding to the maximal modular partition of G, and G_i is one of the maximal strong module of G.

Throughout the rest of this chapter, we may assume that a graph $G = (V, E)$ and the modular decomposition of G, of modular-width at most k, are already given. Otherwise, we can apply the algorithm presented in [19] to obtain one. Under this assumption, it is easy to note that the modular decomposition tree of G can be constructed in constant time using $\mathcal{O}(n)$ parallel processors. Moreover, the number of maximal strong modules in the decomposition tree of G is at most n, which is equal to the cardinality of the maximal modular partition of G.

The central idea of our algorithm is a bottom-up dynamic programming approach along the modular decomposition tree of the algebraic expression as defined above. For each node of the modular decomposition tree, we compute a record for the graph represented by the subtree of the modular decomposition below that node. That is, given the optimal solutions for the children of each node in the modular decomposition tree, we can compute an optimal solution for the node itself. In order to explore the decomposition tree efficiently, we take the parallel tree contraction technique due to Miller and Reif [51].

4.3.1 The Weighted Maximum Clique Problem

Let us consider the weighted maximum clique problem which is known to be NP-complete for general graphs. Given a graph $G = (V, E)$ and weights on each vertex, is there a clique with maximum weight ω?

Theorem 10. *The weighted maximum clique problem parameterized by the modular-width k can be solved in $\mathcal{O}(2^k \cdot \log n)$ time using $\mathcal{O}(n)$ parallel processors. Thus it is in FPPT.*

Proof. Clearly, graph G with bounded modular-width k can be represented by the four operations mentioned above according to Theorem 9. We only need to show that each operation can be done efficiently, and the whole decomposition tree can be evaluated by the parallel tree contraction technique.

First, each leaf node in the modular decomposition tree is an isolated vertex, which can be represented by the first operation of the algebraic expression. Thus, the maximum clique weight of each vertex is trivially its own weight. Obviously, this can be done in constant time with a linear number of parallel processors.

Next, we consider other operations on combining two modules to form a larger module:

- If G is the disjoint union of G_1 and G_2, then the maximum clique weight of G would be:
$$\omega(G) = \max\{\omega(G_1), \omega(G_2)\},$$
since the disjoint union operation corresponds to a parallel node, which implies two modules are non-adjacent.

- If G is the complete join of G_1 and G_2, then the maximum clique weight of G would be:
$$\omega(G) = \omega(G_1) + \omega(G_2),$$
since complete join corresponding to a series node, which implies any vertex in G_1 is adjacent to all the vertices in G_2.

The last case is for G is a substitution of G_i for $1 \leq i \leq k$, which means G is neither obtained by disjoint union operation nor complete join operation. In other words, the quotient graph of G is prime, and the vertices are in one-to-one correspondence with G_i for $1 \leq i \leq k$. In this scenario, graph G can be treated as a graph with at most k vertices, and the weight of each vertex is equal to the maximum clique weight of the corresponding module G_i for $1 \leq i \leq k$. Since each G_i looks like a black box to the other G_j in the maximal modular decomposition of G for $1 \leq i, j \leq k$ and $i \neq j$, and there has no efficient algorithm for the maximum weighted clique problem, we have no choice but take a brute-force strategy to evaluate G if the maximum clique weight of each G_i for $1 \leq i \leq k$ are given, and this can be done in $\mathcal{O}(2^k)$ time.

Now we show how to parallelize the algorithm by the parallel tree contraction technique. We construct a computation tree corresponding to the modular decomposition tree of G, such that each tree node corresponds to a maximal strong module of size at most k and has at most k children. Suppose v is an internal node in the computation tree, we call v is *half-evaluated* when all but one of its children has been evaluated. With the parallel tree contraction technique, it can be compressed later. Suppose the unevaluated child is v_1 and its maximum clique weight is ω', the maximum clique weight of v without v_1 evaluated to a, and the maximum clique weight among evaluated children $v_2, \ldots, v_k$ of v is b, a and b are known values. Then the maximum clique weight of v is

$$\omega = \max\{a, \omega' + b\}.$$

During each contraction progress, we can take the above function recursively and have

$$\omega'' = \max\{c, \omega + d\},$$

where c and d are two known values for next round, then

$$\omega'' = \max\{\max\{b + c, d\}, \omega' + (a + c)\}.$$

Thus, the running time is $\mathcal{O}(2^k \cdot \log n)$ using $\mathcal{O}(n)$ parallel processors, because $\mathcal{O}(2^k)$ time is required to compute a maximum weight clique for a prime graph with at most k vertices, the parallel tree contraction takes $\mathcal{O}(\log n)$ time using a linear number of parallel processors, and half-evaluating a node requires $\mathcal{O}(\log k)$ time.

□

4.3.2 The Maximum Matching Problem

A matching in a graph is a set of edges such that no two edges share a common vertex. We now consider the maximum matching problem which seeks a matching of maximum size (i.e., the largest number of edges). The existence of an NC algorithm for this problem has been open for several decades, even if the graph is planar. By considering the modular-width as the parameter, we prove the following:

Theorem 11. *The maximum matching problem parameterized by the modular-width k can be solved in $\mathcal{O}(2^k \cdot \log n)$ time using $\mathcal{O}(n)$ parallel processors. Therefore the problem is in FPPT.*

Proof. We follow the same strategy as Theorem 10 and evaluate different operations on combining modules. Let n_1, n_2 denote the number of vertices and u_1, u_2 denote the number of unmatched vertices of graphs G_1 and G_2. We use the pair $< n_i, u_i >$ to track the maximum matching of graph G_i.

First, each leaf node i in the modular decomposition tree is an isolated vertex, thus

$$n_i = 1 \text{ and } u_i = 1.$$

Next, we consider various operations on combining two modules to form a larger module:

- If G is the disjoint union of G_1 and G_2, the values of G would be:

  $$n = n_1 + n_2 \text{ and } u = u_1 + u_2,$$

  since disjoint union corresponds to a parallel node, which implies there is no edge between G_1 and G_2.

- If G is the complete join of G_1 and G_2, the values of G would depend on the values of n_1, n_2, u_1 and u_2. We have to consider different cases for this scenario. In fact, no matter for which cases,

  $$n = n_1 + n_2$$

  always valid, we only need to consider u.

  1. If $u_1 > n_2$, then the unmatched vertices in G_1 are more than the vertices of G_2, then the values of G would be:

     $$u = u_1 - n_2,$$

because we could match all vertices of G_2 through the edges between unmatched vertices in G_1 and all vertices of G_2.

2. Symmetrically, if $u_2 > n_1$, the values of G would be:

$$u = u_2 - n_1.$$

3. The last case comes to $u_1 < n_2$ and $u_2 < n_1$. In this circumstances, we are able to match almost all vertices in G_1 and G_2, and only have one unmatched vertex left over if there is an odd number of vertices in G, the values of G would be

$$u = n_1 + n_2 \pmod 2.$$

Without loss of generality, suppose $u_1 - u_2 \geq 2$, we can further match the u_2 unmatched vertices in G_2 by the additional edges of complete join operation. After that, all vertices in G_2 are matched, and only $u_1 - u_2 \geq 2$ vertices left in G_1 still unmatched. Suppose x, y are two unmatched vertices in G_1; we know that x, y are also adjacent to all vertices in G_2 because of the complete join operation, and all vertices of G_2 are matched. Then there must be at least one edge (a, b) in the matching of G_2, such that $x - a - b - y$ is an augmenting path and the matching can be extended by 1. Thus the number of unmatched vertices in G only depends on the parity of $u_1 + u_2$, which is equal to the parity of $n_1 + n_2$.

Thus for the complete join of G_1 and G_2, we have

$$n = n_1 + n_2,$$

and

$$u = \max\{u_1 - n_2, u_2 - n_1, n_1 + n_2 \pmod 2\}.$$

Obviously, this can be done in a constant time given the values of G_1 and G_2.

- Finally, we consider the case where G is a prime graph, which is obtained by the substitution operation on modules $G_1, \ldots, G_k$. As claimed in the proof of Theorem 10, G can be treated as a prime graph with at most k vertices, and each vertex corresponding to a module G_i for $1 \leq i \leq k$ in this case.

  It is well-known that the maximum matching problem can be formulated as integer linear programming. Once again, let u_i denote the number of unmatched vertices in G_i, E denotes the edge set among $G_1, \ldots, G_k$, and $e_{i,j}$ denote the number of matched

edges between G_i and G_j for $1 \leq i, j \leq k$. Then finding a maximum matching in G is equivalent to solving the following problem:

$$\begin{aligned}
\text{Maximize } &\left\{ \sum_{(i,j)\in E} e_{i,j} + \sum_{i} e_{i,i} \right\} \text{ subject to} \\
&2e_{i,i} + \sum_{(i,j)\in E} e_{i,j} \leq n_i \text{ for } i = 1, \ldots, k \\
&e_{i,i} \leq \frac{(n_i - u_i)}{2} \text{ for } i = 1, \ldots, k \\
&e_{i,j} \in [1, k] \text{ for } 1 \leq i, j \leq k.
\end{aligned}$$

For each prime graph, we can compute the matching by taking the maximum of our objective function at every feasible solution. This can be done in $\mathcal{O}(2^k)$ time since the graph has a bounded modular-width k.

Now, we show how to parallelize the algorithm using the parallel tree contraction technique.

Let $n_1, \ldots, n_k$ denote the number of vertices and $u_1, \ldots, u_k$ denote the number of unmatched vertices in $G_1, \ldots, G_k$. Suppose $n_1, \ldots, n_k$ and $u_2, \ldots, u_k$ are known, but u_1 is not. Then the number of unmatched vertices of the graph G can be represented as a function u of u_1 of the form $\max\{p, u_1 - q\}$ for a proper choice of constants p and q, such that $p = u(n_1 \pmod 2)$ and $q = n_1 - u(n_1)$.

As argued in the complete join operation of two graphs, u_1 must have the same parity as n_1. For any x of the same parity as n_1 between 2 and n_1, it is clear that $u(x-2) \leq u(x) \leq u(x-2)+2$. We will show that $u(u_1)$ is a piecewise linear function, consisting of a constant portion for low values of u_1 followed by a portion with slope 1 for high values of u_1 in Lemma 10. Thus $u(u_1)$ has the form $\max\{p, u_1 - q\}$. We choose $p = u(n_1 \pmod 2)$ so the formula is correct at the low end. For the high end, we choose $q = n_1 - u(n_1)$.

We now use the parallel tree contraction technique. Composing functions of the form $u(x) = \max\{p, x - q\}$ leaves another function of the same form which can be computed in constant time. Therefore, the tree contraction can be done in $\mathcal{O}(\log n)$ time with $\mathcal{O}(n)$ parallel processors.

□

Lemma 10. *If x has the same parity as n_1 and also $4 \leq x \leq n_1$, then it cannot satisfy:* $u(x) = u(x-2) = u(x-4) + 2$.

Proof. Let M' be the matching used to calculate $u(x-4)$ and M be the matching used to calculate $u(x)$. Then we have

$$n_1 = 2 * |M'| + (x-4) = 2 * |M| + x;$$

thus,

$$|M'| = |M| + 2;$$

It follows that M' contains at least two edges between vertices in G_1.

Let M'' be the matching M' without an edge e between two of the vertices in G_1, then M'' will be a matching used to calculate $u(x-2)$. If we choose $u_1 = x-2$, both matchings M and M'' have the same cardinality and both are also maximum. Let G' be the resultant graph from taking the symmetric difference of M and M''; i.e. $(M - M'') \cup (M'' - M)$. Every connected component of G' must be either an even cycle whose edges alternate between M and M'' or an even length path whose edges alternate between M and M'' with distinct endpoints. Now add the edge e that is in $M' \setminus M''$, its connected component must be an odd path. If there is another matched edge e' between two of the vertices in G_1, and e' is not in the same connected component, then we can take the edges from M in the component of e', add them to the rest of M', and have a larger matching for the case $u_1 = x-2$. Alternatively, if there does not exist a matching edge between vertices in G_1, that is also in a different component from e, we can still modify the matching to obtain a larger one when $u_1 = x-2$. Let e' be another edge from M' that is contained in G_1. Any vertex outside of G_1 that is adjacent to the vertices of e is also adjacent to the vertices of e'. We can add an edge between one of these vertices and a vertex of e' so that an even cycle is created. We use this even cycle to take a different set of edges. This new set has the property that it no longer includes edge e'. We have found a larger matching for the case $u_1 = x-2$, contradicting our assumption. In both cases, $u(x-2) = u(x) + 2$ follows from our premises.

□

Chapter 5

Concluding Remarks and Future Work

In this thesis, we extend the "parameterized" idea to parallel settings and rename a class called FPPT to characterize the problems that have efficient fixed parameter parallel algorithms. We firs propose an efficient parallel algorithm for the general monotone circuit value problem and conclude that for some (not all) P-complete problems, it is possible to find an algorithm that makes the problem fall into NC by fixing one or more non-trivial parameters. Meanwhile, an interesting analogy would be: *FPPT is with respect to P-complete what FPT is with respect to NP-complete*.

Then we show an parallel approach for computing the crown decomposition, which directly implies a parallel algorithm for the parameterized vertex cover problem that runs in time $\mathcal{O}(k \cdot \log^3 n)$ using $\mathcal{O}(m)$ parallel processors, where m is the number of edges and n is the number of vertices of the input graph, and k is the size of the sought vertex cover. Since the crown structure admits a kernel with size at most $3k$. Thus, the parallel crown decomposition and the parameterized vertex cover problem are in FPPT. We believe this will lead to proving other problems are also in FPPT, where a kernel is obtained via the crown structure.

In the end, we explore the modular-width parameter that covers a significantly large class of graphs under the framework of parameterized parallel complexity and show that the WEIGHTED MAXIMUM CLIQUE PROBLEM and the MAXIMUM MATCHING PROBLEM are FPPT when parameterized by modular-width. These results are of interests for several reasons. First, not only for P-complete and NP-complete problems but also for those that are still open for P-complete or NC, there are parameterized parallel algorithms with non-trivial parameters for them. Thus, FPPT is orthogonal to P-complete, NP-complete and probably some unknown classes between NC and P-complete (if P is not equal to NC). Second, there exist some parameters that make a large number of problems in FPPT which are in different complexity classes in the traditional hierarchy.

We further raise some open questions.

At this stage, the first obvious question is which other problems belong to FPPT.

Numerous NP-complete problems fall into the class FPT, but clearly not all of them do. It was shown that there is a $W[*]$ hierarchy in the NP class where FPT $= W[0]$. With the introduction of parameterization into the field of parallel computing, some NP-complete and P-complete problems were shown to have a fixed-parameter parallel algorithm by fixing one (or more) parameter(s), such as the vertex cover problem, the graph genus problem [25], and the monotone circuit value problem. In other words, these problems are in FPPT. Now the question is what happens if we restrict our attention to P-complete problems: could it be that the P class also has a hierarchy, say $Z[*]$, analogous to $W[*]$ in the class NP, where FPPT $= Z[0]$? In fact, we believe such a hierarchy should exist because the following is true. The MCVP the NAND circuit value problem are both P-complete. However, taking the graph genus as a parameter, the first is in FPPT while the second is not. Another conceivable example is the lexicographically first maximal subgraph problem (LFMIS). Miyano showed that LFMIS is P-complete even for bipartite graphs with bounded degree at most 3 [53]. It would be interesting, and probably not too difficult, to obtain hardness results showing that a problem cannot belong to FPPT unless some new parameterized parallel-complexity hierarchy collapses.

We showed that the weighted maximum clique problem and the maximum matching problem are in FPPT when parameterized by modular-width. It would be interesting to find out whether other problems are in FPPT when parameterized by modular-width. It was shown that the maximum network flow problem is in FPPT with respect to treewidth as parameter [39]. We know that the maximum network flow problem is not easier than the maximum matching problem from a parallel complexity standpoint, being P-Complete. However, we believe that the maximum network flow problem would also fall in FPPT when parameterized by modular-width. Moreover, it was shown that the chromatic number problem, the hamiltonian cycle problem, the maximum cut problem, and the edge dominating set problem are in FPT when parameterized by treewidth but become intractable when parameterized by clique-width. Also, when parameterized by modular-width, the chromatic number problem and the hamiltonian cycle problem are in FPT, while the other two are still open. We conjecture that the chromatic number problem parameterized by modular-width is not FPPT, mainly because Miyano [53] showed that most of the lexicographically first maximal subgraph problems are still P-complete even if the instances are restricted to graphs with bounded degree 3.

Furthermore, there has been an interest in the so-called "gradually intractable problems" for the class NP. The question here is whether "gradually *unparallelizable* problems" can

be analogously investigated in the class of P-complete. We suggest involving one or more parameters to characterize the "gradually" procedure. This would be helpful to understand the intrinsic difficulty of P-complete problems and to answer the question of whether P = NC, which is one of the main motivations behind this thesis.

References

[1] Faisal N. Abu-Khzam, Shouwei Li, Christine Markarian, Friedhelm Meyer auf der Heide, and Pavel Podlipyan. The monotone circuit value problem with bounded genus is in NC. In *The International Computing and Combinatorics Conference, COCOON 2016, Ho Chi Minh City, Vietnam, August 2-4, Proceedings*, pages 92–102, 2016.

[2] Faisal N. Abu-Khzam, Shouwei Li, Christine Markarian, Friedhelm Meyer auf der Heide, and Pavel Podlipyan. On the parameterized parallel complexity and the vertex cover problem. In *The International Conference on Combinatorial Optimization and Applications, COCOA 2016, Hong Kong, China, December 16-18, Proceedings*, pages 477–488, 2016.

[3] Faisal N. Abu-Khzam, Shouwei Li, Christine Markarian, Friedhelm Meyer auf der Heide, and Pavel Podlipyan. Modular-width: An auxiliary parameter for parameterized parallel complexity. In *The International Frontiers of Algorithmics Workshop, FAW 2017, Chengdu, China, June 23-25, Proceedings*, pages 139–150, 2017.

[4] Manindra Agrawal, Thanh Minh Hoang, and Thomas Thierauf. The polynomially bounded perfect matching problem is in NC ^{2}. In *The Annual Symposium on Theoretical Aspects of Computer Science, STACS 2007, Aachen, Germany, February 22-24, Proceedings*, pages 489–499, 2007.

[5] Vladimir E. Alekseev. Polynomial algorithm for finding the largest independent sets in graphs without forks. *Discrete Applied Mathematics*, 135(1-3):3–16, 2004.

[6] Hans L. Bodlaender. Treewidth: Characterizations, applications, and computations. In *The International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2006, Bergen, Norway, June 22-24, Proceedings*, pages 1–14, 2006.

[7] Hans L. Bodlaender, Rodney G. Downey, and Michael R. Fellows. Applications of parameterized complexity to problems of parallel and distributed computation. In *Unpublished extended abstract*, 1994.

[8] Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[9] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

[10] Allan Borodin. On relating time and space to size and depth. *SIAM Journal on Computing*, 6(4):733–744, 1977.

[11] Binh-Minh Bui-Xuan, Jan Arne Telle, and Martin Vatshelle. Boolean-width of graphs. *Theoretical Computer Science*, 412(39):5187–5204, 2011.

[12] Marco Cesati and Miriam Di Ianni. Parameterized parallel complexity. *Electronic Colloquium on Computational Complexity (ECCC)*, 4(6), 1997.

[13] M. Chein, Michel Habib, and M. C. Maurer. Partitive hypergraphs. *Journal of Discrete Mathematics*, 37(1):35–50, 1981.

[14] Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved upper bounds for vertex cover. *Theoretical Computer Science*, 411(40-42):3736–3756, 2010.

[15] Marek Chrobak, Joseph Naor, and Mark B. Novick. Using bounded degree spanning trees in the design of efficient algorithms on claw-free graphs. In *The Workshop on Algorithms and Data Structures, WADS 1989, Ottawa, Canada, August 17-19, Proceedings*, pages 147–162, 1989.

[16] Stephen A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Computation*, 64(1-3):2–21, 1985.

[17] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12–75, 1990.

[18] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[19] Elias Dahlhaus. Efficient parallel modular decomposition. In *The International Workshop on Graph-Theoretic Concepts in Computer Science, WG 1995, Aachen, Germany, June 20-22, Proceedings*, pages 290–302, 1995.

[20] Elias Dahlhaus, Péter Hajnal, and Marek Karpinski. On the parallel complexity of hamiltonian cycle and matching problem on dense graphs. *Journal of Algorithms*, 15(3):367–384, 1993.

[21] Giuseppe Di Battista and Enrico Nardelli. An algorithm for testing planarity of hierarchical graphs. In *The International Workshop on Graph-Theoretic Concepts in Computer Science, WG 1986, Bernried, Germany, June 17-19, Proceedings*, pages 277–289, 1986.

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

[23] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[24] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17(3):449–467, 1965.

[25] Michael Elberfeld and Ken-ichi Kawarabayashi. Embedding and canonizing graphs of bounded genus in logspace. In *The Annual Symposium on Theory of Computing, STOC 2014, New York, USA, May 31 - June 03, Proceedings*, pages 383–392, 2014.

[26] Shimon Even and Robert Endre Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2(3):339–344, 1976.

[27] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Clique-width: on the price of generality. In *The Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, USA, January 4-6, Proceedings*, pages 825–834, 2009.

[28] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Algorithmic lower bounds for problems parameterized by clique-width. In *The Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, USA, January 17-19, Proceedings*, pages 493–502, 2010.

[29] Fedor V. Fomin, Petr A. Golovach, Daniel Lokshtanov, and Saket Saurabh. Intractability of clique-width parameterizations. *SIAM Journal on Computing*, 39(5):1941–1956, 2010.

[30] Steven Fortune and James Wyllie. Parallelism in random access machines. In *The Annual Symposium on Theory of Computing, STOC 1987, San Diego, USA, May 1-3, Proceedings*, pages 114–118, 1978.

[31] Jakub Gajarský, Michael Lampis, and Sebastian Ordyniak. Parameterized algorithms for modular-width. In *The International Symposium on Parameterized and Exact Computation, IPEC 2013, Sophia Antipolis, France, September 4-6, Proceedings*, pages 163–176, 2013.

[32] Robert Ganian, Petr Hlinený, Jaroslav Nesetril, Jan Obdrzálek, Patrice Ossona de Mendez, and Reshma Ramadurai. When trees grow low: Shrubs and fast MSO1. In *The International Symposium on Mathematical Foundations of Computer Science, MFCS 2012, Bratislava, Slovakia, August 27-31, Proceedings*, pages 419–430, 2012.

[33] Ratan K. Ghosh and G. P. Bhattacharjee. Parallel breadth-first search algorithms for trees and graphs. *International Journal of Computer Mathematics*, 15(1-4):255–268, 1984.

[34] Leslie M. Goldschlager. The monotone and planar circuit value problems are log space complete for p. *SIGACT News*, 9(2):25–29, 1977.

[35] Leslie M. Goldschlager. A universal interconnection pattern for parallel computers. *Journal of the ACM (JACM)*, 29(4):1073–1086, 1982.

[36] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo. *Limits to parallel computation: P-completeness theory*. Oxford University Press, 1995.

[37] Dima Grigoriev and Marek Karpinski. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. In *The Annual Symposium on Foundations of Computer Science, FOCS 1987, Los Angeles, USA, October 27-29, Proceedings*, pages 166–172, 1987.

[38] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

[39] Torben Hagerup, Jyrki Katajainen, Naomi Nishimura, and Prabhakar Ragde. Characterizing multiterminal flow networks and computing flows in networks of small treewidth. *Journal of Computer and System Sciences*, 57(3):366–375, 1998.

[40] Amos Israeli and Yossi Shiloach. An improved parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):57–60, 1986.

[41] Joseph JáJá and Janos Simon. Parallel algorithms in graph theory: Planarity testing. *SIAM Journal on Computing*, 11(2):314–328, 1982.

[42] Philip N. Klein and John H. Reif. An efficient parallel algorithm for planarity. In *The Annual Symposium on Foundations of Computer Science, FOCS 1986, Toronto, Canada, October 27-29, Proceedings*, pages 465–477, 1986.

[43] Richard E. Ladner. The circuit value problem is log space complete for p. *SIGACT News*, 7(1):18–20, 1975.

[44] Jens Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *Journal of Algorithms*, 20(1):20–44, 1996.

[45] James O. Lee, Ralph G. Stanton, and Cowan D. Donald. Graph decomposition for undirected graphs. In *The Southeastern International Conference on Combinatorics, Graph Theory, and Computing, SEICCGTC 1972, Boca Raton, USA, February 28 - March 2, Proceedings*, pages 281–290, 1972.

[46] Gavriela Freund Lev, Nicholas Pippenger, and Leslie G. Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE Transactions on Computers*, 30(2):93–100, 1981.

[47] Keqin Li and Victor Y. Pan. Parallel matrix multiplication on a linear array with a reconfigurable pipelined bus system. *IEEE Transactions on Computers*, 50(5):519–525, 2001.

[48] Nutan Limaye, Meena Mahajan, and Jayalal Sarma. Upper bounds for monotone planar circuit value and variants. *Computational Complexity*, 18(3):377–412, 2009.

[49] Meena Mahajan and Kasturi R. Varadarajan. A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs. In *The Annual Symposium on Theory of Computing, STOC 2000, Oregon, Portland, May 21-23, Proceedings*, pages 351–357, 2000.

[50] Gary L. Miller. An additivity theorem for the genus of a graph. *Journal of Combinatorial Theory, Series B*, 43(1):25–47, 1987.

[51] Gary L. Miller and John H. Reif. Parallel tree contraction part 1: Fundamentals. *Advances in Computing Research*, 5:47–72, 1989.

[52] George J. Minty. On maximal independent sets of vertices in claw-free graphs. *Journal of Combinatorial Theory, Series B*, 28(3):284–304, 1980.

[53] Satoru Miyano. The lexicographically first maximal subgraph problems: P-completeness and NC algorithms. In *The International Colloquium on Automata, Languages, and Programming, ICALP 1987, Karlsruhe, Germany, July 13-17, Proceedings*, pages 425–434, 1987.

[54] Sang-il Oum. Rank-width and vertex-minors. *Journal of Combinatorial Theory, Series B*, 95(1):79–100, 2005.

[55] Vijaya Ramachandran and Honghua Yang. An efficient parallel algorithm for the general planar monotone circuit value problem. *SIAM Journal on Computing*, 25(2):312–339, 1996.

[56] Vijaya Ramachandran and Honghua Yang. An efficient parallel algorithm for the layered planar monotone circuit value problem. *Algorithmica*, 18(3):384–404, 1997.

[57] Robert Endre Tarjan and Uzi Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14(4):862–874, 1985.

[58] Honghua Yang. An NC algorithm for the general planar monotone circuit value problem. In *The Symposium on Parallel and Distributed Processing, SPDP 1991, Dallas, USA, December 2-5, Proceedings*, pages 196–203, 1991.