UNIVERSIDADE FEDERAL FLUMINENSE

JANIO CARLOS NASCIMENTO SILVA

Algorithmic Aspects of Problems Related to Optimization, Circuits, and Parameterized Complexity

NITERÓI 2021

UNIVERSIDADE FEDERAL FLUMINENSE

JANIO CARLOS NASCIMENTO SILVA

Algorithmic Aspects of Problems Related to Optimization, Circuits, and Parameterized Complexity

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Ciência da Computação.

Advisors: Uéverton dos Santos Souza Luiz Satoru Ochi

> NITERÓI 2021

Ficha catalográfica automática - SDC/BEE Gerada com informações fornecidas pelo autor

S586a Silva, Janio Carlos Nascimento Algorithmic Aspects of Problems Related to Optimization, Circuits, and Parameterized Complexity / Janio Carlos Nascimento Silva ; Uéverton dos Santos Souza, orientador ; Luiz Satoru Ochi, coorientador. Niterói, 2021. 104 f. : il. Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2021. DOI: http://dx.doi.org/10.22409/PGC.2021.d.03234594392 1. Otimização. 2. Complexidade de Circuitos. 3. Complexidade Parametrizada. 4. W-hierarquia. 5. Produção intelectual. I. Souza, Uéverton dos Santos, orientador. II. Ochi, Luiz Satoru, coorientador. III. Universidade Federal Fluminense. Instituto de Computação. IV. Título. CDD -

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

JANIO CARLOS NASCIMENTO SILVA

Algorithmic Aspects of Problems Related to Optimization, Circuits, and Parameterized Complexity

> Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Ciência da Computação.

Aprovada em 17 de Novembro de 2021.

BANCA EXAMINADORA Verturton des Somtes Sun Prof. D.Sc. Uéverton dos Santos Souza, Orientador - UFF Prof. D.Sc. Luiz Satoru Ochi, Orientador - UFF Joon Machado Coelho Prof. D.Sc. Igor Machado Coelho - UFF Burkohs Prof. D.Sc. Fábio Protti - UFF uerpertand

Prof. D.Sc. Luerbio Faria - UERJ

EDROHENRIGUEGONZALERSILVA

Prof. D.Sc. Pedro Henrique González Silva - CEFET-RJ

NITERÓI 2021

À Vanessa, minha companheira, com todo amor.

Agradecimentos

Sou grato...

Por cada noite em claro, Por cada centavo suado, conquistado com extrema dificuldade e investido em caderno, livro e caneta. Por cada longa caminhada pela Vila Vitória, Por cada jornada de ônibus pelas ruas de Imperatriz, Por cada pedalada sob o sol de Palmas, Por cada prato de comida que me ofereceram, Por cada colchão em que pude deitar e por cada teto sob o qual pude morar.

Esse doutorado começou em 1994, quando entrei na escola pela primeira vez. Sou grato a muitas pessoas que me ajudaram nesse caminho. Família, amigos, professores e colegas de trabalho. Aqui nesses agradecimentos, quero destacar alguns personagens fundamentais.

Agradeço aos meus pais, Alzenir e José, por tudo. Pela vida, pelo cuidado e pelo suporte.

Sou grato aos meus orientadores, Prof. Uéverton dos Santos Souza, pela confiança, pelo direcionamento e pelo entusiasmo em cada tarefa; e Prof. Luiz Satoru Ochi, pelos conhecimentos passados e pelo valioso incentivo neste projeto.

Sou grato a todos meus professores, em especial, aos inestimáveis Warley Gramacho da Silva, Ary Henrique Morais de Oliveira e Glêndara Aparecida de Souza Martins, da UFT; e aos incríveis José Gilson Sales e Silva, Maria Aparecida Alves Rocha Coelho (Professora Cida) e Raimundo Pinho Godinho, do IFMA.

Agradeço aos professores do IC/UFF, Prof. Fábio Protti, Prof. Igor Machado Coelho e Prof. Maria Cristina da Silva Boeres, pelos conhecimentos que adquiri com eles.

Agradeço a minha esposa, Vanessa Batista Nogueira, pelo amor, pelo entusiasmo em meus projetos e por cuidar dos nossos filhos nos momentos de minha ausência. E também por esse cuidado, agradeço aos meus irmãos Allan, Janielly e Taluany e aos meus amigos Elvis Nascimento, Rafael Miranda, Ranyere Vanderlei e Luciano Franco. Por fim, sou grato aos amigos Alan Diêgo, Heder Dorneles, Gelson Schneider, Cristiane Tavares, Vanessa Lima, Kíllya de Paiva, Renatha Capua, Thibaut Vidal, Edcarllos Santos, João Thompson e Marcos Guerine, por serem a minha família em Niterói; e também aos amigos William Girão, Bruno Barros, Emerson Souza e Leandro Botelho, pelos momentos de companheirismo e colaboração.

A todos vocês, muito, muito, muito obrigado!

Resumo

Nas últimas décadas, a resolução de problemas de otimização combinatória considerados difíceis tornou-se um dos campos mais prolíficos da ciência da computação. Diversos estudos tratam destes problemas tanto em uma perspectiva teórica quanto prática. Para além de resolver os problemas, a comunidade científica tem-se dedicado em compreender os diferentes níveis de dificuldade de problemas e como esses problemas comportam-se em termos de tratabilidade e estrutura. Nesta tese, apresentamos descobertas relacionadas a teoria da complexidade a partir de três diferentes perspectivas.

Em um primeiro momento, dedicamos atenção a duas medidas de complexidade de circuitos: A largura de certificação (*certification-width*) e a Complexidade de Energia. Ao tratar de circuitos monótonos, provamos a NP-completude de CERTIFICAÇÃO SUCINTA DE UM CIRCUITO MONÓTONO (SMCC) e de COMPLEXIDADE DE ENERGIA DE MELHOR-CASO EM ATRIBUIÇÕES SATISFACTÍVEIS (MINEC⁺_M), mesmo em casos onde os circuitos de entrada são planares. Provamos que ambos os problemas são W[1]-hard, mas que SMCC pertence a W[P], enquanto fornecemos um algoritmo XP para MINEC⁺_M. Em seguida, para ambos os problemas foram elaboradas estratégias de pré-processamento dos grafos de entrada (baseado na estratégia *win-win*), onde foi possível limitar a treewidth em circuitos com *genus* limitado; e a partir dessa estrutura resultante, apresentamos algoritmos FPT de programação dinâmica baseado em decomposição em árvore destes grafos pré-processados.

Numa segunda perspectiva, apresentamos uma hierarquia de problemas parametrizados baseada na satisfabilidade de Circuitos *Threshold* – A Th-hierarquia. O estudo da Th-hierarquia buscou compreender possíveis lacunas existentes na W-hierarquia e perceber como as classes dessas hierarquias interagem. Provou-se que tais hierarquias colapsam nos níveis mais altos (i.e. W[P] = Th[P]). Na sequência, ao estudar uma forma de converter circuitos threshold em circuitos booleanos convencionais com profundidade ótima, provou-se a possibilidade de construir uma rede de ordenação AKS em tempo polinomial. Isso deu base para mostrar que $Th[t] \subseteq W[SAT]$, para todo $t \in \mathbb{N}$.

Por fim, numa frente experimental, formalizamos um conceito de exploração de vizinhança chamado *Multi Improvement* (alternativa aos tradicionais *First Improvement* and *Best Improvement*) e construímos algoritmos de Programação dinâmica para o PRO-BLEMA DO MULTI IMPROVEMENT MÁXIMO, modelado como etapa interna de uma busca local sobre instâncias do TSP. Experimentos mostraram que esta abordagem provê uma possibilidade de realizar buscas locais em vizinhança com rapidez e alta estabilidade.

Palavras-chave: Otimização, Complexidade de Circuitos, Complexidade Parametrizada, W-hierarquia.

Abstract

In the last few decades, the resolution of hard combinatorial optimization problems has become one of the most prolific fields in Computer Science. Several studies address these problems from both theoretical and practical perspectives. In addition, the scientific community has dedicated efforts to understand the different levels of hardness and how problems behave in terms of tractability and structure. In this thesis, we present discoveries related to complexity theory from three different perspectives.

First, we address two measures in circuit complexity: The *certification-width* and the *Energy Complexity*. When dealing with monotone circuits, we prove the NP-Completeness of SUCCINCT MONOTONE CIRCUIT CERTIFICATION (SMCC) and BEST-CASE ENERGY COMPLEXITY IN SATISFYING ASSIGNMENTS (MINEC⁺_M) even for planar circuits. We also prove that both problems are W[1]-hard, but SMCC belongs to W[P], while an XP-algorithm for MINEC⁺_M is provided. After all, for both problems, we develop a preprocessing of input circuit (inspired by *win-win approach* strategy) where it was possible to prune graphs with bounded genus; this results in a structure whose treewidth is bounded. Hence, dynamic programming algorithms on tree decompositions were provided, solving the problems in FPT-time.

In a second perspective, we present a hierarchy of classes of parameterized problems based on threshold circuit satisfiability – the *Th-hierarchy*. The study of Th-hierarchy aims to understand possible gaps in W-hierarchy and the interaction Th-hierarchy versus W-hierarchy levelwise. We show that the hierarchies collapse in high levels (i.e., W[P] = Th[P]). Next, we study ways to convert threshold circuits into Boolean circuits with optimal depth; thus, we demonstrate that it is possible to construct an AKS sorting network in polynomial-time. This supports to prove that $Th[t] \subseteq W[SAT]$, for every $t \in \mathbb{N}$.

Additionally, in an experimental front, we formalize the concept of neighborhood exploration called *Multi Improvement* (alternative to the traditional *First Improvement* and *Best Improvement*) and we build dynamic programming algorithms to solve the Maximum Multi Improvement Problem (MMIP) modeled as an inner step of a local search for TSP instances. Experiments have shown that this approach provides a possibility to perform fast neighborhood searches with high stability.

Keywords: Optimization, Circuit Complexity, Parameterized Complexity, W-hierarchy.

List of Figures

2.1	Graph representation of a circuit and a satisfying assignment for it	11
2.2	Preprocessing step of the reduction from PVC to SMCC	13
2.3	Reduction from PVC to SMCC on planar circuits.	15
2.4	Consequences of a minimum vertex cover of H in the circuit C	17
2.5	Relation between genus and geometrical shapes	20
2.6	Graph Γ_7	24
3.1	Anomalous behavior in certificates for $MINEC_M^+$	33
3.2	An instance Q for MULTICOLORED CLIQUE	38
3.3	Circuit C obtained from Q (Fig. 3.2) after reduction $\ldots \ldots \ldots \ldots$	39
4.1	Relationship between $W[t]$ and $Th[t]$ classes	54
4.2	Comparison elements	55
4.3	A bipartite $(3, 1/4)$ -expander graph G	58
4.4	A perfect matching in a (k, ϵ) -expander determining swap stages	58
4.5	A (1/4)-halver constructed from G with three swap stages	58
4.6	A $(1/8, 1/18, 1/72)$ -separator	63
4.7	Relationship between W-hierarchy and Th-hierarchy.	64
5.1	Traditional neighborhood exploration strategies	66
5.2	Multi Improvement	67
5.3	Spanning subgraphs for swap operator	70
5.4	Effect of $2Opt(2,5)$ in a graph $\ldots \ldots \ldots$	71
5.5	3-Opt possibilities	74
5.6	OrOpt-K scheme	75

5.7	Convergence of 2-Opt for berlin52.	79
5.8	Convergence of MI-2Opt in five runs on berlin52 and lin318	80
5.9	Curve fittings for berlin52 and lin318.	81

List of Tables

5.1	Comparison between BI, FI and MI for 20pt neighborhood	78
5.2	Comparison between MI-2Opt and optimal solutions	78
5.3	Performance of local searches.	79
5.4	Curve Fitting about MI-2Opt convergence.	81

Contents

1	Intr	troduction 1		
	1.1	Background	3	
		1.1.1 Parameterized Complexity	4	
		1.1.2 Optimization of measures in circuits	5	
	1.2	Organization of this Thesis	6	
2	Suco	cinct Certification of Monotone Circuits	8	
	2.1	Preliminaries	9	
	2.2	NP-completeness on planar circuits	13	
	2.3	Parameterized Complexity	18	
		2.3.1 Bounding the treewidth of monotone circuits with bounded genus .	20	
		2.3.2 Dynamic programming on tree decomposition	24	
	2.4	Final considerations	29	
3	Ene	rgy Complexity of Satisfying Assignments	31	
	3.1	Preliminaries	32	
	3.2	Computational complexity analysis	34	
		3.2.1 On monotone circuits with bounded genus	40	
	3.3	Dynamic programming on bounded treewidth circuits	43	
4	Para	ameterized complexity classes defined by threshold circuits	48	
	4.1	Satisfiability of threshold circuits	51	
		4.1.1 Preliminaries	52	

Re	eferen	ces		86
6	Con	clusion		84
	5.5	Remar	ks	82
	5.4	Experi	iments and convergence	77
		5.3.3	OrOpt-k Neighborhood	75
		5.3.2	3-Opt Neighborhood	72
		5.3.1	2-Opt Neighborhood	70
	5.3	Dynan	nic Programming Multi Improvement	69
	5.2	The M	laximum Multi Improvement Problem	67
	5.1	Prelim	inaries	66
5	Max	imum I	Multi Improvement on Neighborhood Exploration	65
		4.3.2	$Th[t] \subseteq [SAT] \dots \dots \dots \dots \dots \dots \dots \dots \dots $	60
		4.3.1	AKS sorting networks	57
	4.3	On the	e classes Th[t], Th[SAT], and W[SAT] $\ldots \ldots \ldots \ldots \ldots \ldots$	55
		4.2.1	Sorting networks	54
	4.2	W-hier	rarchy versus Th-hierarchy	53
		4.1.2	The Th-hierarchy	52

Chapter 1

Introduction

Combinatorial optimization problems have been handled by a large diversity of tools and approaches (e.g., exact algorithms, approximation algorithms, and heuristics/metaheuristics). Depending on the scenario, each approach has its benefits and disadvantages. For example, if we need a fast implementation for solving large instances of an NP-hard problem, then exact algorithms do not seem suitable. In this context, metaheuristics have gained considerable relevance. Even without optimality assurance, these methods can quickly provide sub-optimal solutions, which can be satisfactory in many activities. In other words, if an optimization problem has strong practical appeal, indeed, there is an effort of practitioners in designing efficient metaheuristic strategies.

In the late 20th Century, a profusion of creative techniques was developed. Such an eflort produced the most famous heuristics and metaheuristics used until nowadays. In contrast, in [26], we have a recent and thorough survey that reveals which even novel heuristics has been created, the major focuses still on quantitative comparations over benchmark instances. This phenomenon is historically defined by [72] as *framework-centered period* where the researchers have dedicated attention in combining and adapting methods with a focus on performance. Besides that, the authors in [72] are confident in the advent of a *scientific-centered period*, which is the moment when the study of metaheuristics will be focused on observing the behavior of the methods, identifying why a heuristic works well or discussing which are the best and worst scenarios for applying a method (e.g., [85]). The same concern is approached in another recent survey [41] where the authors (after analyzing 1222 meaningful works across 31 years of research in metaheuristics) address three gaps in the metaheuristic field: (i) lack of accuracy in performance validation criteria, (ii) lack of mathematical and theoretical foundations and other discussions about convergence and minor details about results of the implementations, and (iii) lack of scalable metaheuristics with self-adaptable capability to face complex scenarios.

However, what other possibilities can we observe besides metaheuristics when dealing with optimization problems? Since the beginning of this doctorate process, we have studied a diversity of modern tools used to solve optimization problems. We found inspiring contributions from Computational Complexity by searching for creative ideas in producing tools with significant impact in optimization problem resolutions. An interesting topic involves the study of instances from NP-hard problems that may be solved by polynomial (or sub-exponential) time algorithms. A famous example is HAMILTONIAN PATH in tournament graphs. The HAMILTONIAN PATH is proven to be NP-hard [34]. However, for a Tournament graph, which is a directed complete graph, there is an algorithm that performs $O(n \cdot \log n)$ [73]. The discovery of these subclasses of problems gives an essential insight into problems where the instance characteristics are known. Polynomial and subexponential algorithms for instances of NP-hard problems may provide optimal solutions for specific situations in an acceptable time, and the examples like HAMILTONIAN PATH IN TOURNAMENT GRAPHS motivate the search for such tractable cases.

Also, in the Computational Complexity area, creative strategies with the real potential of contribution to state-of-the-art of optimization are in rising when we observe the *Parameterized Complexity*. FPT algorithms, kernelization of problems, and the diversity of frameworks for graphs are examples of powerful theoretical features which can be explored even in implementations with a focus on performance. The parameterized complexity theory encourages a broad comprehension of the problem to be studied. Writing parameterized algorithms is the art of discovering what turns a problem hard. Precisely for that, analyses based on parameterized complexity theory are widely adopted in graph problems. Graph problems are very sensitive to structure, and this offers an open field to explore structural parameters, kernels, subclasses of graphs, and other characteristics. Therefore, modeling optimization problems using graph theory notation has become quite common; for instance, we have optimization problems in Boolean circuits, which are traditionally modeled as graphs.

Optimization in circuits has a wide spectrum of applications. One can optimizing either in logic, architecture, devices or systems levels ([17],[24], [47]). The abstraction based on optimization in circuits is even useful in biological studies ([18], [80]). Most of these approaches take advantage of manipulating circuits as graphs due to the 'arsenal' of existing algorithms suitable for graph structures. This intrinsic relation between graphs, parameterized complexity and circuits is well-materialized in WEIGHTED CIRCUIT SATISFIABILITY – a computational problem in Boolean circuits whose characterization defines classes in W-hierarchy. Deeply looking at Circuit Complexity, we can find more examples of measures in Boolean circuits with detailed structural analysis, but without discussing the time complexity in computing them.

Gathering all these thoughts, we decide to investigate measures of circuit complexity from the classical and parameterized complexity perspective. In summary, our efforts were pointed in the following directions:

- Classical and Parameterized Complexity analysis for two optimization problems in monotone Boolean circuits (Chapters 2 and 3).
- A discussion about Parameterized Complexity Classes defined by Threshold Circuits with a complexity analysis in a Sorting Network Construction (Chapter 4).

Additionally, as a side project, we propose an optimization problem based on the optimal resolution of Multi Improvement in internal stages of a local search (Chapter 5).

Throughout the text, we present the results developed during the doctorate. In common, all these projects include complexity analysis of exact algorithms (even in the study with background on metaheuristics). Each chapter is self-contained, with its own preliminaries and results. Even so, we present some principles in Section 1.1

1.1 Background

A computational problem can be seen as a set of variables and restrictions together with a question or a goal to be reached. An instance of a computational problem is a data structure that fits values to some variables of this problem. In this work, a problem can be seen as a *decision problem* or an *optimization problem*. Given an instance to a problem, if the question of this problem only admits YES or NO as an answer, usually based on a data structure called *certificate*; then it is a decision problem.

Regarding decision problems, we have some important classes of problems. The class P of computational problems refers to the problems which can be solved by a deterministic algorithm with polynomial time in function the input's size. The class NP includes decisions problems whose the certificate for any positive instance can be verified by a deterministic algorithm in polynomial time. The possibility of solving any NP problem by a deterministic algorithm in polynomial time implies the most important open question in

computer science (P = NP?) and without this answer, we need to deal with the problems that are as 'hard' as the most difficult problem in NP – the NP-hard problems. A problem Π is said NP-hard if and only if each problem $L \in NP$ is reducible to Π in polynomial time $(L \propto \Pi)$. If an NP-hard problem is in NP then this problem is called NP-complete.

On the other hand, the answer for an optimization problem is not YES/NO, but a value (cost associated to a certificate/solution). If a problem aims minimization (resp. maximization), then a certificate with minimum (resp. maximum) cost represents an *optimal solution*.

An exact algorithm is an algorithm that solves an optimization problem (find the optimal solution) for any instance. Assuming $P \neq NP$, the known exact algorithms for optimization problems associated to NP-hard decision problems are infeasible in practical projects with large instances. Alternatively, approximation algorithms and heuristics/metaheuristics are very popular for practical purposes even without optimality assurance. However, how can we improve the time of exact algorithms without losing the assurance of optimality? One of the answers is *Parameterized Algorithms*.

1.1.1 Parameterized Complexity

A parameterized problem [28] is a decision problem whose instances are pairs $(x, k) \in \Sigma^* \times \mathbb{N}$, where k is called the parameter. A parameterized problem is fixed-parameter tractable (FPT) if there exists an algorithm \mathcal{A} , a computable function f, and a constant c such that given an instance I = (x, k), \mathcal{A} (called an FPT algorithm) correctly decides whether I is a yes- or no-instance in time bounded by $f(k) \cdot |I|^c$. The discovery of FPT algorithms fulfills the objective of solving problems more efficiently with exact algorithms spending less time than a brute force algorithm, whenever k is considered small.

A parameterized problem is *slice-wise polynomial* (XP) if there exists an algorithm \mathcal{A} and two computable functions f, g such that given an instance I = (x, k), \mathcal{A} (called an XP *algorithm*) correctly decides whether I is a yes- or no-instance in time bounded by $f(k) \cdot |I|^{g(k)}$. Within parameterized problems, the class W[1] may be seen as the parameterized equivalent to the class NP of classical optimization problems. Without entering into details (see [22, 28] for the formal definitions), a parameterized problem being W[1]-hard can be seen as a strong evidence that this problem is not FPT. The canonical example of W[1]-hard problem is CLIQUE parameterized by the size of the solution. To transfer W[1]-hardness from one problem to another, one uses a parameterized reduction, which given an input I = (x, k) of the source problem, computes in time $f(k) \cdot |I|^c$, for some computable function f and a constant c, an equivalent instance I' = (x', k') of the target problem, such that k' is bounded by a function depending only on k. Equivalently, a problem is W[1]-hard if there is a parameterized reduction from CLIQUE parameterized by the size of the solution.

In addition to the W[1] class, some classes of parameterized problems are defined according to their parameterized intractability level. These classes are organized in the so-called W-hierarchy FPT \subseteq W[1] \subseteq W[2] \subseteq ... \subseteq W[P] \subseteq XP, and it is conjectured that each of the containments is proper [28]. If P = NP, then the hierarchy collapses [28].

1.1.2 Optimization of measures in circuits

Circuit Complexity is a research field that aims to study bounds for measures (such as size and depth) of circuits that compute Boolean functions. The *size* of a circuit is its number of logic gates, and *depth* is the largest path from any input to the output gate. A circuit complexity analysis can provide precise lower/upper bounds on circuits classes that represent classic decision problems besides the possibility to design efficient Boolean circuits according to specific properties (see [40, 61, 86]).

In addition, some important bounds are described to deal with different definitions of size [2]. From a combinatorial point of view, several optimization problems address the minimization of measures in some circuits classes, such as the notorious WEIGHTED CIRCUIT SATISFIABILITY problem, where the weight measures the amount of **true** values assigned to the input variables.

Despite this 'zoo of measures', optimizing properties like size and depth does not always guarantee an 'efficient' design of a specific circuit class. Depending on the purpose, a circuit with a small size (either considering gates or wires) or depth can be inappropriate; such a situation was identified in two measures: *Certification-width* and *Energy Complexity. Certification-width* is based on the number of the enabled edges in a satisfied circuit and *Energy Complexity* is about the number of gates outputting true in a circuit. Considering a sparse activation of edges and gates, both measures can represent natural parameters, and optimizing them using parameterized algorithms is fully justified in this context.

1.2 Organization of this Thesis

After this introduction, we present four self-contained chapters with each project developed during this doctorate. The sequence of this work is organized as follows.

Chapter 2 shows an analysis of an optimization problem based on a novel measure of circuit complexity called *certification-width*. After the analisys of SUCCINT CERTIFICA-TION OF MONOTONE CIRCUIT (SMCC) (the problem in find a positive certificate with minimum certification-width), we prove:

- the NP-completeness of SMCC for planar circuits;
- the W[P]-membership and W[1]-hardness of SMCC;
- that there is a polynomial-time algorithm capable of bounding the treewidth of the underlying graph of a monotone planar circuit by a function of the input circuit's genus and the size of the solution. This approach is known as *win-win approach* (See [22]).

Lastly, after obtaining a bounded treewidth instance, we construct a dynamic programming to solve the problem in FPT-time, i.e., the problem is fixed-parameter tractable. The results presented in Chapter 2 are published in proceedings of *The 26th International Computing and Combinatorics Conference* (COCOON'2020) [4] and *Theoretical Computer Science* [5].

In Chapter 3, we deal with another circuit measure closely related to certificationwidth: The Energy Complexity. The concept of Energy Complexity was introduced in [80]. When SMCC focuses on edges activated in the succinct certificate, here, our focus turns to the number of gates that outputs **true** in a Boolean circuit. In that sense, we define BEST-CASE ENERGY COMPLEXITY OF SATISFYING ASSIGNMENT IN MONOTONE CIRCUITS (MINEC⁺_M). Besides the similarities between SMCC and MINEC⁺_M, early analysis shows that some of our initial conclusions for SMCC should be revisited. Due to that, we achieve a proof of W[1]-hardness for MINEC⁺_M; an XP algorithm for MINEC⁺_M; and polynomial pre-processing based on win-win approach. After all, we present a novel dynamic programming based on tree decomposition. The results of Chapter 3 were published in proceedings of The 15th International Conference on Algorithmic Aspects in Information and Management (AAIM'2021) [70]. In Chapter 4, we introduce the Th-hierarchy, a hierarchy of parameterized problems based on the satisfiability of threshold circuits. In order to compare Th-hierarchy and W-hierarchy, we discuss ways to transform threshold circuits into Boolean circuits in an efficient manner. Thus, Sorting Networks appear as a candidate for these transformations. More precisely, we explore the complexity in constructing an AKS Sorting Network (a type of network with logarithmic depth). With that, we can conclude that the higher level of the hierarchies collapses, and we also prove $Th[t] \subseteq W[SAT]$, for every $t \in \mathbb{N}$. The results of Chapter 4 were published in proceedings of The 15th Annual International Conference on Combinatorial Optimization and Applications (COCOA'2021) [58].

Chapter 5 shows a study about the effect of the optimal Multi-improvement innersolvability in an outer problem. In other words, we construct three dynamic programmings that solve the MAXIMUM MULTI IMPROVEMENT PROBLEM in a single stage of the local search of a neighborhood exploration on TSP modeling. Our findings include:

- Definition of the MAXIMUM MULTI IMPROVEMENT PROBLEM and its characterization in relation with MAXIMUM WEIGHTED CLIQUE PROBLEMS.
- Development of three polynomial dynamic programming for 2-Opt, 3-Opt, and Or-Opt neighborhood operators.
- Experimental stability analysis of the local search descent based on dynamic programming multi improvement *versus* First and Best Improvement local searches.

These achievements are published on paper for *Optimization Letters* [69].

Each chapter presents its own conclusions, remarks, and open questions. Hence, in Chapter 6 we finalize this work with a succinct global conclusion.

Chapter 2

Succinct Certification of Monotone Circuits

Boolean circuits are one of the earliest combinatorial formalisms for the representation of Boolean functions. Besides being a fundamental object of study in classical complexity theory, Boolean circuits also play a central role in the field of parameterized complexity [28]. More specifically, while the satisfiability problem for general Boolean circuits can be used to define the class NP, the satisfiability problem for Boolean circuits of bounded weft can be used to define the levels of the W-hierarchy [28]. An important, and well-studied, subclass of Boolean circuits is the class of monotone Boolean circuits, i.e., circuits where only AND and OR gates are allowed. While the standard satisfiability problem for monotone Boolean circuits is trivial, since the all-ones vector is always a satisfying assignment, some weighted versions of satisfiability problems are still interesting in this setting. One of these problems is the WEIGHTED MONOTONE CIRCUIT SATISFIABILITY (WMCS) problem, where we are given a monotone Boolean circuit C as input, and the goal is to find a minimum-weight satisfying assignment for the inputs of C [21, 43, 52]. The WMCS is particularly relevant in the field of circuit design, since the minimum number of inputs necessary to make a monotone circuit evaluate to true is a parameter that is often taken into consideration [48].

In this chapter, we deal with the notion of succinct certificates for monotone Boolean circuits. Given a monotone circuit C, a succinct certificate for C is a connected sub-circuit of C with a minimal set of edges that is sufficient to ensure that C is satisfiable. Just like circuit size and circuit depth, the minimum size of a succinct certificate is an interesting complexity measure. Additionally, a succinct certificate may be seen as a minimal map to be followed by a satisfying assignment. This map may find applications in the field of circuit design and may be used as a way of representing solutions to problems modeled

through monotone circuits. In the literature similar structures are known as *accepting* subtree [82], positive proof [27], and proving circuit [54].

We study the complexity of computing the size of a minimum succinct certificate of a given monotone circuit C. We call this invariant the *certification-width* of C, and name the problem of computing the value of this invariant as the SUCCINCT MONOTONE CIRCUIT CERTIFICATION (SMCC) problem. The problem under consideration is both of theoretical and practical relevance. From a theoretic perspective, the minimum size of a succinct certificate naturally gives information about the complexity of a circuit. Therefore, determining the underlying structure that makes SMCC (fixed-parameter) tractable is interesting from the perspective of complexity theory. From a practical perspective, SMCC can be applied in many problem-reduction representations [56], since monotone circuits can be seen as unweighted And/Or graphs [74, 75].

The notion of planarity is well-explored in graph theory and has significant relevance in the field of circuit analysis. In particular, VLSI (Very Large-Scale Integration) circuits, which are widely applied in electronics and engineering, are typically modeled by planar graphs. In addition, there are several studies on circuits and satisfiability problems defined on certain structures that are planar or that satisfy certain structural properties (see [10, 15, 43, 44, 48, 52, 68, 76, 79]).

We show that SMCC is NP-hard when the input monotone circuit is planar, and it is W[1]-hard, but in W[P], when parameterized by solution size. Subsequently, we present a polynomial-time algorithm that takes a monotone circuit as input and either solves the instance or bounds the diameter of the input; then using the notion of contraction obstructions for treewidth we are able to conclude that the treewidth of the resulting circuit is bounded by k + g, where k is the solution size and g is the genus of the input circuit. Thus, by using such a win/win approach and applying a dynamic programming algorithm we solve SMCC in FPT time when parameterized by k + g. This result also implies that SMCC can be solved in time $2^{O(k)} \cdot n + m$ on planar circuits.

2.1 Preliminaries

We use standard graph-theoretic and parameterized complexity notation, and we refer the reader to [22, 28] for any undefined notation.

A Boolean circuit is a combinatorial model for the representation of Boolean functions. We formalize the notion of a Boolean circuit according to Definition 1. In general, a circuit can have multiples outputs. Nevertheless, for convenience, we will adopt the following definition.

Definition 1. A Boolean circuit is a directed acyclic graph C(V, E) having only one sink, where the set of vertices V is partitioned into $(I, G, \{v_{out}\})$: (i) a set of inputs $I = \{i_1, i_2, ...\}$ composed of the vertices of in-degree 0; (ii) a set of gates $G = \{g_1, g_2, ...\}$, which are vertices labeled with Boolean operators; (iii) and the single output (sink) vertex v_{out} with out-degree equal to 0 and also labeled with a Boolean operator. The input vertices represent Boolean variables that can take values from $\{0, 1\}$ ({false, true}, depending on the conventions), and the label/operator of a gate or output vertex w is given by f(w).

Note that we are considering general circuits with no restrictions on the number of inneighbors and out-neighbors. Additionally, in this chapter, we only deal with monotone circuits. Also, for a Boolean circuit C(V, E), in this chapter, we let n = |V(C)| and m = |E(C)|.

Definition 2. A monotone circuit is a Boolean circuit where the Boolean operators allowed are in {AND, OR}.

Definition 3. An assignment of C is a vector $X = [x_1, x_2, ..., x_{|I|}]$ of values for the set of inputs I, where for each $j, x_j \in \{\texttt{false}, \texttt{true}\}\)$ is the value assigned to input i_j . We say that X is a satisfying assignment if the circuit C evaluates to true when given x as input.

In Fig. 2.1a, we have an example of a circuit C with six inputs i_1, i_2, \ldots, i_6 , four gates g_1, g_2, g_3, g_4 and the output vertex v_{out} . Fig. 2.1b shows an example of the results of an assignment X to the circuit presented in Fig. 2.1a. In this example, the function AND of v_{out} returns true, thus, X is a satisfying assignment according to Definition 3.

We denote by $X \to C$ the adapted directed graph in which the values of X were assigned to I, and the label of the gates are replaced by the returned values of their respective functions (see Fig. 2.1b).

The directions of the edges represent inputs to functions of the gates. When all inedges of a gate g_j have values assigned to them, then g_j will be evaluated according to the operator $f(g_j)$ under these input values. We note that the value of an input may not reach v_{out} , for example, in Fig. 2.1b, the assignment sets i_5 to **true**. However, this value cannot reach v_{out} because $f(g_2)$ was not satisfied. This situation brings us another important definition: the *critical edges*.



Figure 2.1: Graph representation of a circuit and a satisfying assignment for it.

Definition 4. Given a monotone circuit C with a satisfying assignment X, an edge (v_j, v_k) is considered critical to $X \to C$ if v_j evaluates to true and there is a path P from v_k to v_{out} in which all gates along P (including v_{out}) also evaluate to true.

According to Definition 4, in Fig. 2.1 the edge (i_5, g_2) is not critical while (i_2, g_1) , $(i_3, g_3), (i_4, g_4), (g_1, g_3), (g_3, v_{out})$ and (g_4, v_{out}) are critical edges in $X \to C$. This motivates the notion of *positive certificate* stated in Definition 5.

Definition 5. Given a monotone circuit C, and a satisfying assignment X of C, a positive certificate for $X \to C$ is a connected subgraph of C formed by the critical edges and their respective vertices.

Since a positive certificate may have redundant edges, next, we present the notion of succinct certificate; and certification-width of C.

Definition 6. Given a monotone circuit C, and a satisfying assignment X of C, a succinct certificate for $X \to C$, is a connected subgraph $SC_{X\to C}$ of its positive certificate such that:

- v_{out} is a vertex of $SC_{X\to C}$; and
- for every vertex v of $SC_{X\to C}$ holds that

- if f(v) = AND, then every in-edge of v is in $SC_{X \to C}$;

- if f(v) = OR, then exactly one in-edge of v is in $SC_{X \to C}$.

The size of a succinct certificate $SC_{X\to C}$ is the number of edges of $SC_{X\to C}$.

In the context of counting complexity and arithmetic circuits, the succinct certificates are also known as proving circuits [54].

Definition 7. The certification-width of a monotone circuit C is the minimum size among all possible succinct certificates on all satisfying assignments of C.

We remark that there are similarities between the notions of certification-width and the *energy complexity* of a circuit C, which is defined as the number of gates evaluating to true among all assignments to C (see [80]).

Now, we have all elements to describe our main problem.

SUCCINCT MONOTONE CIRCUIT CERTIFICATION (SMCC)
Instance : A monotone circuit C ; a positive integer k .
Goal : Determine whether the certification-width of C is at most k .

We denote by k-SMCC the parameterized version of SUCCINCT MONOTONE CIRCUIT CERTIFICATION where k is the parameter.

Similar notions of succinct certificate and certification-width were introduced in [67], in that context, defined as *accepting computation tree* and *tree-size* of a Alternating Turing Machines (ATM). According to W. L. Ruzzo [67], the main motivation of these notions was defining a novel complexity measure as an abstraction that would provide a spectrum of complexity classes intermediate between non-determinism and full alternation In 1989, H. Venkateswara [83] defined the concept of *accepting subtree* of a Boolean circuit as an analogy to the notion of accepting computation tree for ATMs. As can be seen in [82], the notion of accepting subtree of a Boolean circuit is similar to what we propose to call a succinct certificate. Also, there are some works which refers to such structures as (positive) proof tree (see [27, 53]). Besides, succinct certificates are also known as proving (sub-)circuits (see [16, 54]). Note that this kind of "witness for acceptance" gives an intuitive and natural idea of important features in different frameworks from complexity theory (see [27, 53, 67, 82, 83]). Nevertheless, these works typically address the relation between the size of the certificate and the characterization of complexity classes. In this chapter, we address another direction, we deal with the time complexity of computing such structures with minimum size for (monotone) Boolean circuits.



Figure 2.2: Green vertices are those a' and b' inserted in G after the preprocessing step.

2.2 NP-completeness on planar circuits

Now, we dedicate our attention to SMCC restricted to planar circuits. Clearly, SMCC is in NP. Next, we show its NP-hardness. For that, we will use a reduction from PLANAR VERTEX COVER.

PLANAR VERTEX COVER (PVC)		
Instance : A planar graph G ; a positive integer c .		
Question : Is there a set S of size at most c, such that for each edge $(u, v) \in E(G)$		
either $u \in S$ or $v \in S$?		

Theorem 1. SMCC is NP-complete even restricted to planar circuits.

Proof. Given a circuit C and an integer k, forming an instance of SMCC, a connected subgraph of C having at most k edges and satisfying Definition 6 can be seen as a certificate for the "yes" answer of this instance. Since it is easy to verify in polynomial time the conditions described in Definition 6, it holds that SMCC is in NP.

In order to prove its NP-hardness, we present a reduction from PLANAR VERTEX COVER. First, consider the following preprocessing: Let (H, c') be an instance of PLANAR VERTEX COVER. By subdividing twice each edge of H, we obtain a graph G where each edge e = (ab) of H is replaced by a P_4 ab'a'b, where a' and b' are new vertices. Notice that G is planar; H has a vertex cover of size c' if and only if G has a vertex cover of size c = c' + |E(H)|; and given a planar embedding of G, the common boundary of any pair of adjacent faces of G contains at least three edges. From a fixed planar embedding of the instance (G, c) of PLANAR VERTEX COVER, we proceed with the reduction. We will construct an instance (C, k) of SMCC where Cis a planar monotone circuit, and k is the target size of the solution. From the original structure of G, we apply the following:

- 1. Firstly, set V(C) = V(G);
- 2. for each vertex $v_i \in V(G)$, create an input vertex v_i^{in} , assign $f(v_i) = \text{AND}$, and add a directed edge (v_i^{in}, v_i) ;
- 3. for each edge $e_i = (u, v) \in E(G)$, create a vertex $v_{e_i}^{cover}$ such that $f(v_{e_i}^{cover}) = \mathsf{OR}$, and create the directed edges $(v, v_{e_i}^{cover})$ and $(u, v_{e_i}^{cover})$. This step guarantees that if $v_{e_i}^{cover}$ is in the succinct certification, then either v or u will also be on the certificate.

Notice that C is still planar. Now, preserving the planarity, we will ensure that every $v_{e_i}^{cover}$ is in any succinct certification of C as follows:

- 4. create an output vertex v_{out} such that $f(v_{out}) = AND$;
- 5. for each vertex $v_{e_i}^{cover}$ which are in the external face of G, create one directed edge from $v_{e_i}^{cover}$ to v_{out} ;

Let D_G be the dual graph of G, and denote by f_1 the vertex representing the external face of G. Let \mathcal{T}_{D_G} be the spanning tree of D_G obtained from a breadth-first search of D_G rooted at f_1 . In a top-down manner, according to a level-order traversal of \mathcal{T}_{D_G} , we visit each edge $e = (f_i, f_j)$ of \mathcal{T}_{D_G} , applying the following:

6. Let f_j be a child of f_i in \mathcal{T}_{D_G} ;

By construction of G, it follows that the boundary between f_i and f_j contains at least three edges, at least one of which being between vertices a' and b' that do not exist in H;

Thus, create a vertex v_{f_j} , add edges from v_{f_j} to such a' and b'; and for each $v_{e_\ell}^{cover}$ in the face f_j that does not reach v_{out} , yet, add an edge from $v_{e_\ell}^{cover}$ to v_{f_j} ; after that, if v_{f_j} has in-degree greater than 0, then set $f(v_{f_j}) = \text{AND}$, otherwise v_{f_j} is an input vertex;

7. Finally, set $k = c + 2 \cdot |E(G)| + |V(\mathcal{T}_{D_G})| - 1$.



Figure 2.3: Reduction from PVC to SMCC on planar circuits. Green vertices are those a' and b' inserted in G after the preprocessing step. Blue vertices refer to $v_{e_i}^{cover}$ created in step 3. Edges outgoing from $v_{e_i}^{cover}$ are highlighted in red. Each yellow vertex refers to v_{f_j} inserted on step 6; Edges outgoing from v_{f_j} are highlighted in green. For simplicity, vertices created in step 2 were omitted.

Fig. 2.2 illustrates graphs H, G, while Fig. 2.3 shows its \mathcal{T}_{D_G} , and the resulting C.

Given a vertex cover S of G with c vertices, without loss of generality, we can assume that S does not contain pairs of adjacent vertices that do not belong to V(H). By setting 1 (true) to the corresponding inputs of S in C, in exactly c edges flows **true** from vertices v_i^{in} to its out-neighbor v_i ; from each v_i assigned with **true** flow positive values to each v_e^{cover} such that e is an out-edge of v_i . Since S is a vertex cover, each v_e^{cover} receives at least one positive value, which implies that every vertex v_{f_j} evaluates to **true**, and v_{out} also evaluate to **true**. Thus, C has a succinct certificate SC where all in-edges of v_{out} are in SC; each in-neighbor of v_{out} has as in-neighbor one vertex of S in SC; since the vertices representing faces and vertices of G have the label AND, by construction, every v_e^{cover} is in SC and has exactly one out-edge in SC; Given that S does not contain pairs of adjacent vertices that do not belong to V(H), each vertex v_{f_j} also has exactly one out-neighbor in SC; and as every v_e^{cover} is labeled OR, one can construct SC in such a way that v_e^{cover} has as in-neighbor exactly one vertex in S. Thus, SC has size equal to $k = c + 2 \cdot |E(G)| + |V(\mathcal{T}_{D_G})| - 1$. (Namely, c in-edges of v_i^{in} vertices; one in-edge and one out-edge for each v_e^{cover} ; one out-edge for each vertex v_{f_i} .)

Fig. 2.4 illustrates vertex covers of graphs G and H shown in Fig 2.2, and the succinct certificate SC of the correspondent circuit C.

Conversely, suppose that C has a succinct certificate SC with at most k edges. By construction, all |E(G)| vertices v_e^{cover} are in SC, and each v_e^{cover} in SC demands exactly two edges in SC (one in-going and another out-going), which form a set of $2 \cdot |E(G)|$ edges. Also, by construction, we know that all v_{f_i} vertices (created in step 6) must be in SC(recall that these vertices are responsible to connect every inner v_e^{cover} in a path to v_{out}). The vertices v_{f_i} demand at least $|V(\mathcal{T}_{D_G})| - 1$ other edges in SC, at least one out-edge by vertex. Note that the in-edges of a vertex v_{f_i} have already been considered as out-edges of v_e^{cover} vertices. Thus, since SC has size at most $k = c + 2 \cdot |E(G)| + |V(\mathcal{T}_{D_G})| - 1$, remains at most c edges to be considered in SC. Therefore, there are at most c input vertices v_i^{in} in SC. At this point, one can construct a vertex set S of G with cardinality at most c by adding v_i in S if its corresponding input v_i^{in} is in SC. Finally, as each v_e^{cover} vertex is in SC, at least one of its out-neighbors (endpoints of e) and corresponding input are also in SC, which implies that S is a vertex cover of size at most c of G.



(c) Succinct certificate obtained from a minimum vertex cover of G.

(d) Bottom-up representation of the succinct certificate.

Figure 2.4: Consequences of a minimum vertex cover of H in the circuit C. Recall that each blue vertex is a OR-vertex, v_e^{cover} , representing an edge e of G, while the other vertices are AND-vertices.

2.3 Parameterized Complexity

In this section, we analyze the hardness of SMCC regarding the W-hierarchy. First, we present the WEIGHTED CIRCUIT SATISFIABILITY problem, a well-known W[P]-complete problem [28].

WEIGHTED CIRCUIT SATISFIABILITY
Instance: A decision circuit C.
Parameter: A positive integer k.
Question: Does C have a satisfying assignment of weight k?

The W[P]-membership of k-SMCC follows from a reduction to WEIGHTED CIRCUIT SATISFIABILITY.

Lemma 1. k-SMCC is in W/P/.

Proof. Let C be a instance of k-SMCC, we construct a decision circuit D as follows:

- 1. For each edge e of C, create (in D) an input i_e ;
- 2. for each non-input AND-gate (resp. OR-gate) v of C, create an AND-gate (resp. OR-gate) g_v in D;
- 3. if there is an edge e from an input-vertex v_i to v_j , add an edge from i_e to g_{v_j} ;
- 4. if there is an edge e to a vertex v_i from a non-input vertex v_j , create an AND-gate g_{e^*} , add edges to g_{e^*} from i_e and g_{v_j} , and add an edge to g_{v_i} from g_{e^*} .

In summary, the inputs of D represent edges outgoing from inputs/gates of C and the gates g_v created in step 2, a priori, mimic the same structure of C. In addition, the AND-gates g_{e^*} and its respective edges added in step 4, force the evaluation to true of the input i_e if the path corresponding to the edge e of C needs to be used in D. In other words, an edge e from a non-input vertex v_j to a vertex v_i in C is represented by a gadget formed by the path $g_{v_j}, g_{e^*}, g_{v_i}$ plus the edge i_e, g_{e^*} . Thus, the evaluation to true of g_{e^*} depends of the previous evaluation to true of g_{v_j} (preserving the structure of the succinct certificate) and also of the evaluation to true of i_e , which produces the correspondence between using e in a succinct certificate of C with the evaluation to true of i_e in D. Hence, it holds that: if C has a succinct certificate with k edges, then by assigning exactly the kcorresponding inputs of D as true, one can obtain a satisfying assignment with weight k for D; conversely, if D has a satisfying assignment A with weight k then C has a succinct certificate induced by the set of edges $\{e \in E(C) : i_e \text{ evaluates to true in } A\}$.

Recall that $W[P] \subseteq XP$. Therefore, the membership shown in Lemma 1 is a result stronger than an XP algorithm.

Next, we prove the W[1]-hardness of k-SMCC using a reduction from MULTICOLORED CLIQUE, a well-known W[1]-complete problem [31].

Multicolored Clique

Instance: A graph Q with a vertex-coloring $\ell : V(G) \to \{1, 2, ..., c\}$. **Parameter**: A positive integer c. **Question**: Does Q have a clique including of all c colors?

Theorem 2. k-SMCC is W[1]-hard.

Proof. Let (Q, c) be an instance of MULTICOLORED CLIQUE and let V_1, V_2, \ldots, V_c be the color classes of Q. Without loss of generality, we consider that each vertex in V_i has at least one neighbor in $V_j (i \neq j)$. We construct an instance (C, k) of k-SMCC as follows:

- 1. create an output gate v_{out} in C and set $f(v_{out}) = AND$;
- 2. for each color c_i of Q, create a gate w_i with $f(w_i) = OR$ and add an edge from w_i to v_{out} ;
- 3. for each color class V_i of Q, create copies V_i^1, V_i^2, V_i^3 and V_i^4 in C;
- 4. add edges from each vertex in V_i^4 to w_i ;
- 5. let v^1, v^2, v^3 and v^4 be the copies of a vertex $v \in V(Q)$; add edges $(v^1, v^2), (v^2, v^3)$ and (v^3, v^4) to G; set V_i^1 as the input set; and assign $f(v^2) = f(v^3) = OR$ and $f(v^4) = AND$;
- 6. for each vertex $v^4 \in V_i^4 (1 \le i \le c)$), create c-1 new or-in-neighbors $a_{v^4}^j (1 \le j \le c)$ and $i \ne j$), and add an edge to $a_{v^4}^j$ from a vertex $u^2 \in V_j^2$ if and only if v and u are neighbors in Q;
- 7. finally, set $k = 2c^2 + 3c$.

If Q contains a multicolored clique K such that |K| = c, then it is possible to construct a succinct certificate SC of C with k edges as follows: (a) v_{out} and all of its in-edges belong to SC; (b) for each OR-gate $w_i \in V(SC)$, include in SC an edge (v^4, w_i) of C such that $v \in K$; (c) for each $v^4 \in V(SC)$, add all of its in-edges to SC; (d) for each $v^3 \in V(SC)$ add its in-edge to SC; (e) for each $v^2 \in V(SC)$ add its in-edge to SC; (f) finally, for each $a_{v^4}^j$ in SC, choose an in-edge $(u^2, a_{v^4}^j)$ to be added to SC such that $u \in K$. Thus, SC has exactly $k = 2c^2 + 3c$.

Conversely, if C has a succinct certificate SC with $k = 2c^2 + 3c$ edges, then it is possible to obtain a multicolored clique K of Q as follows: a vertex v of Q belongs to K if and only if v^2 belongs to V(SC). Since, by construction, any succinct certificate of C has at least k edges, if SC has $k = 2c^2 + 3c$ edges, then SC has only c vertices of type v^2 , and for each pair v^2 , $a_{u^4}^j (u \neq v \text{ and } v^2 \in V_j^2)$ in SC there is an edge between them in Q, implying that K is a multicolored clique of size c of Q.

2.3.1 Bounding the treewidth of monotone circuits with bounded genus

In this section, we bound the treewidth of bounded genus circuits. The strategy adopted in this section is based on the grid minor theorems proposed by Robertson and Seymour [65, 66], see also [37].

First consider the following definitions.

A graph G has genus g if it can be drawn without crossings on a surface of genus g (a sphere with g handles, see Fig. 2.5), but not on a surface of genus g - 1. We refer the reader to [36] for more information on the genus of a graph. We consider the genus of a circuit as the genus of its underlying undirected graph.



Figure 2.5: Relation between genus and geometrical shapes.

Definition 8. Let G be an undirected graph. A tree decomposition of G is a pair $\mathcal{T} =$

 $(T, \{X_t\}_{t \in V(T)})$ such that T is a tree where each node t is assigned to a set of vertices $X_t \subseteq V(G)$, called bags, according to the following conditions:

- $\bigcup_{t \in V(T)} X_t = V(G)$, i.e. every vertex must be in at least one bag;
- for each $(u, v) \in E(G)$, there exist a node t such that $\{u, v\} \subseteq X_t$;
- for each $v \in V(G)$, the set $T_v = \{t \in V(T) : v \in X_t\}$ spans a connected subtree of T.

The width $\operatorname{tw}(\mathcal{T})$ of a tree decomposition \mathcal{T} is the size of the largest bag of T minus one. The treewidth of G is the minimum treewidth among all its possible tree decompositions.

Definition 9. A graph H is a minor of a graph G if H can be constructed from G by deleting vertices or edges, and contracting edges.

Definition 10. A grid $p \times q$, denoted by $\boxplus_{p \times q}$ or \boxplus_p when p = q, is a graph whose set of vertices is $V(G) = \{v_{ij} | (i,j) \in \{1,2,\ldots,p\} \times \{1,2,\ldots,q\}\}$ and there is an edge $(v_{ij}, v_{i'j'}) \in E(G)$ exactly if |i' - i| = 1 or |j' - j| = 1, but not both.

Theorem 3 (Excluded Grid Theorem [65]). Let t be a non-negative integer. Then every planar graph G of treewidth at least 9t/2 contains \boxplus_t as a minor.

Definition 11 ([66]). For every face F of a planar embedding M, we define d(F) to be the minimum value of r such that there is a sequence F_0, F_1, \ldots, F_r of faces of M, where F_0 is the external face, $F = F_r$, and for $1 \le j \le r$ there is a vertex v incident with both F_{j-1} and F_j . The radius $\rho(M)$ of M is the minimum value r such that $d(F) \le r$ for all faces F of M. The radius of a planar graph is the minimum across all radii of its planar embeddings.

From the Excluded Grid Theorem, it is easy to see that there is a connection between the radius of a planar graph and its treewidth. In [66], Robertson and Seymour presented a bound for the treewidth of a planar graph with respect to its radius.

Theorem 4 (Radius Theorem [66]). If G is planar and has radius at most r, then its treewidth is at most 3r + 1.

Using Theorem 4 we are able to either solve k-SMCC on planar circuits or output an equivalent instance C' with treewidth bounded by a function of k. First consider the following. **Lemma 2.** Let (C, k) be an instance of SMCC with n = |V(C)| and m = |E(C)|. There is an algorithm that in O(n + m) time either solves (C, k) or outputs an instance (C', k) of SMCC such that:

- (C', k) is an yes-instance of SMCC if and only if (C, k) is also an yes-instance;
- C' is an induced subcircuit of a circuit C* that has diameter at most 2k and it is also an induced subcircuit of C.

Proof. Let (C, k) be an instance of SMCC. Firstly, we apply to C the following preprocessing steps to generate a graph C':

- 1. For each vertex v_j such that $f(v_j) = \text{AND}$, if $|N_{v_j}^-| > k$, then v_j is deleted;
- 2. delete every vertex which is at a distance greater than k from v_{out} ;
- 3. delete all vertices whose in-degree became equal to 0 (the original inputs are not affected by this step);
- 4. delete all AND-vertices that lost one of its in-neighbors;
- 5. repeat steps 3 and 4 as long as possible;
- 6. delete all vertices that have become unreachable from v_{out} ;
- 7. if $C' = \emptyset$, then we conclude that (C, k) is a *no*-instance of SMCC.

It is easy to see that the rules described above are safe. Thus, after this preprocessing, if $C' \neq \emptyset$, then C' is an induced subgraph of the circuit C^* , obtained after applying Step 2, which has only vertices at a distance at most k from v_{out} . Now, observe that steps 1-4 and 6-7 can be performed in O(V(C) + E(C)) time (one can use breadth-first search from v_{out} to calculate the distances and degrees); also, note that although Step 5 is described as a loop of previous steps, since the circuit is acyclic, the vertices to be removed by this "Repeat...until" procedure can be identified efficiently by traversing the vertices of C in reverse sequence to a topological order, which takes O(V(C) + E(C)) time.

Regarding Lemma 2, notice that, after Step 2, the underlying undirected graph of the current circuit has diameter at most 2k, which implies that if it is planar, then it also has a radius at most 2k. Thus, by Theorem 4, it follows that the underlying undirected graph of such a circuit has treewidth at most 6k + 1. Since the property of having bounded
treewidth is hereditary, it holds that the output circuit C' also has treewidth at most 6k + 1.

Note that the property of having bounded diameter is not hereditary. However, if it is desired that C' also has bounded diameter, one can modify Step 5 to repeat steps 2-4, so the following holds.

Lemma 3. Let (C, k) be an instance of SMCC. There is an algorithm that in $O(n^2 + nm)$ time either solves (C, k) or outputs an instance (C', k) of SMCC having depth at most k, such that (C', k) is an yes-instance of SMCC if and only if (C, k) is also an yes-instance.

Next, we extend the previous reasoning for bounded genus graphs.

According to [43], for an edge e = uv of a graph G, contracting e means removing the two vertices u and v from G, replacing them with a new vertex w, and for every vertex y in the neighborhood of v or u in G, adding in the new graph an edge wy whose multiplicity is the sum of the multiplicities of the edges of G between v and y and between u and y. If in the above definition we do not sum up multiplicities, and if the initial graph G is a simple graph, then we call the operation simple contraction, or for short s-contraction. Given a vertex-set $S \subseteq V(G)$ such that the subgraph of G induced by S, denoted G[S], is connected, contracting S means contracting the edges between the vertices in S to obtain a single vertex at the end. We say that a graph H is an s-contraction of a graph G if H can be obtained after applying to G a (possibly empty) sequence of edge s-contractions.

The following is a construction presented in [33] and [43]. Consider an $(r \times r)$ -grid. A corner vertex of the grid is a vertex of the grid of degree 2. By Γ_r we denote the graph obtained from the $(r \times r)$ -grid as follows: construct first a graph called Γ'_r by triangulating all internal faces of the $(r \times r)$ -grid such that all internal vertices of the grid are of degree 6, and all non-corner external vertices of the grid are of degree 4 (Γ'_r is unique up to isomorphism). Two of the corners of the initial grid have degree 2 in Γ'_r ; let x be one of them. Now Γ_r is obtained from Γ'_r by adding all the edges having x as an endpoint and a vertex of the external face of the grid that is not already a neighbor of x as the other endpoint (see Fig. 2.6 for an illustration of Γ_7). Observe again that Γ'_r is unique up to isomorphism. The following is a lemma from [43] implied from Lemma 6 in [33].

Lemma 4 (Lemma 4.5 in [43]). Let G be a graph of genus g, and let r be any positive integer. If G excludes Γ_r as an s-contraction, then the treewidth of G is at most $(2r+4) \cdot (g+1)^{3/2}$.



Figure 2.6: Graph Γ_7 .

Lemma 5. Let C' be the circuit obtained from Lemma 3. It holds that C' has treewidth at most $(4k + 10) \cdot (g + 1)^{3/2}$, where g is the genus of C'.

Proof. First, notice that for each vertex u of a Γ_{2k+3} there is another vertex v such that the distance between u and v is at least k+1. Now, suppose that C^* , the circuit obtained after Step 2, has Γ_{2k+3} as an s-contraction, and let u be a vertex of a Γ_{2k+3} such that u is either v_{out} or a vertex obtained by contracting S containing v_{out} . Since there is a vertex v such that the distance between u and v is at least k+1, it holds that C^* does not have depth greater than k, which is a contradiction. Thus, by Lemma 4 we have that the treewidth of C^* is at most $(4k + 10) \cdot (g + 1)^{3/2}$. Therefore, as the properties of having bounded genus, as well as bounded treewidth, are hereditary, then C' also has treewidth at most $(4k + 10) \cdot (g + 1)^{3/2}$.

2.3.2 Dynamic programming on tree decomposition

From Lemma 5, in order to solve k-SMCC on bounded genus graphs, it is enough to present an FPT algorithm parameterized by the treewidth of C.

In general, for a tree decomposition $(T, \{X_t\}_{t \in V(T)})$ it is common to distinguish one vertex r of T which will be the root of T. This introduces natural parent-child and ancestor-descendant relations in the tree T [22]. To design dynamic programmings based on tree decompositions, it is useful to obtain rooted tree decompositions that satisfy some auxiliary conditions. Such decompositions are so-called *nice tree decompositions*, and they are defined as follows:

Definition 12. A rooted tree decomposition $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ is nice if the following

conditions are satisfied:

- The root bag X_r and the leaf bags are empty;
- Every non-leaf bag of T is of one of the following types:
 - Introduce node a node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$; we say that v is introduced at t.
 - Forget node a node t with exactly one child t' such that $X_t = X_{t'} \setminus \{v\}$ for some $v \in X_{t'}$; we say that v is forgotten at t.
 - Join node a node t with two children t_1 and t_2 such that $X_t = X_{t_1} = X_{t_2}$.

Additionally, an *extended* nice tree decomposition is an extended version of a nice tree decomposition where we also have introduce edge nodes. An *Introduce edge node* is a node t, with exactly one child t' such that $X_t = X_{t'}$, and labeled with an edge $(u, v) \in E(G)$ such that $u, v \in X_t$; we say that (u, v) is introduced at t. Besides, we assume that each edge of a graph G is introduced precisely once in an extended nice tree decomposition of G.

Based on the following results, we can assume, without loss of generality, that we are given a nice tree decomposition of G.

Theorem 5. [14] There exists an algorithm that, given an n-vertex graph G and an integer k, runs in time $2^{O(k)} \times n$ and either outputs that the treewidth of G is larger than k, or constructs a tree decomposition of G of width at most 5k + 4.

Lemma 6. [22] Given a tree decomposition $(T, \{X_t\}_{t \in V(T)})$ of G of width at most k, one can in time $O(k^2 \cdot \max(|V(T)|, |V(G)|))$ compute a nice tree decomposition of G with at most $O(k \cdot |V(G)|)$ nodes and width at most k.

Theorem 6. k-SMCC can be solved in time $2^{O(tw)} \cdot n$, where tw is the treewidth of the input.

Proof. Let C be a planar monotone circuit and $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be an extended nice tree decomposition of C. For convenience, we add the vertex v_{out} to every bag of T; thus, the treewidth of \mathcal{T} is increased by 1. The root bag X_r and the leaves are equal to $\{v_{out}\}$. This change ensures that for every bag, there exists at least one possible subsolution. It is worth to remember that all succinct certification necessarily contains v_{out} . Another preprocessing must be made: The introduce edge nodes will be labeled according to Boolean functions of the head of the directed edge, thus, we have "*introduce edge of an* AND-gate" and "*introduce edge of an* OR-gate". The same alteration is applied to forget nodes, i.e., we have "*forget node of* AND-gate" and "*forget node of* OR-gate". This separation of nodes help us to organize the subproblems according to the previous subsolutions already computed.

Let \mathcal{T}_{X_t} be the subtree of \mathcal{T} rooted by X_t and G_{X_t} be the graph/circuit having \mathcal{T}_{X_t} as tree decomposition, and formed by edges already introduced in \mathcal{T}_{X_t} .

Each subproblem of the dynamic programming is represented by $c[t, X, \mathcal{B}]$, which denotes the minimum number of edges of a succinct subcircuit of G_{X_t} , where $X \subseteq X_t$ is the set of vertices of X_t in such a subcircuit, and \mathcal{B} is a Boolean vector of size at most |X| such that for each $v \in X$ with $f(v) = \mathsf{OR}$ it holds that if $\mathcal{B}[v] = 1$, then the OR -gate vhas an in-edge in the subcircuit and $\mathcal{B}[v] = 0$ means that v does not yet have an out-edge in the subcircuit.

Note that since v_{out} belongs to every solution, we do not need to handle the connectivity issue, as this is a guaranteed property of any minimal solution. Therefore, the optimal solution can be found at $c[t, \{v_{out}\}, \mathcal{B}]$ where $\mathcal{B} = \emptyset$ if v_{out} is an AND-gate; otherwise $\mathcal{B}[v_{out}] = 1$. The recurrences are presented below.

Leaf node – Let t' be a leaf node, then $X_t = \{v_{out}\}$ which gives us two possibilities:

$$c[t, X, \mathcal{B}] = \begin{cases} +\infty, & \text{if } (v_{out} \notin X) \text{ or } (f(v_{out}) = \mathsf{OR} \text{ and } \mathcal{B}[v_{out}] = 1) \\ 0, & \text{otherwise} \end{cases}$$
(2.1)

Introduce vertex node – Let t be an introduce vertex node with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$. Since no edge of v was introduced yet, v is isolated in G_{X_t} . The recurrence in Eq. 2.2 resumes the subproblems.

$$c[t, X, \mathcal{B}] = \begin{cases} c[t', X \setminus \{v\}, \mathcal{B}], & \text{if } v \in X \\ c[t', X, \mathcal{B}], & \text{if } v \notin X \end{cases}$$
(2.2)

Introduce edge of an AND-gate – Let t be a node that introduces the directed edge (u, v), where f(v) = AND and let t' be the child of t. For each possible tuple (t, X, \mathcal{B}) , we have three situations (See Eq. 2.3):

1. If $v \in X$ and $u \in X$, then the edge (u, v) must be in the solution (increase the

recurrence by 1);

- 2. if $v \notin X$, then a solution for t' is recovered;
- 3. lastly, if $v \in X$ and $u \notin X$, then the edge (u, v) cannot be used, thus, this solution is invalid because v is an AND-gate.

$$c[t, X, \mathcal{B}] = \begin{cases} c[t', X, \mathcal{B}] + 1, & \text{if } v \in X \text{ and } u \in X \\ c[t', X, \mathcal{B}], & \text{if } v \notin X \\ +\infty & \text{if } v \in X \text{ and } u \notin X. \end{cases}$$
(2.3)

Introduce edge of an OR-gate – Let t be a node that introduces the edge (u, v) where f(v) = OR and let t' be the child of t. For each possible tuple (t, X, \mathcal{B}) , we need to check the following situations:

- 1. If $u \in X$, $v \in X$ and $\mathcal{B}[v] = 1$, then the edge (u, v) can be included in the solution depending on the most advantageous conditions in t':
 - for a tuple of t' with Boolean vector \mathcal{B}' such that $\mathcal{B}'[v] = 0$ and $\mathcal{B}'[w] = \mathcal{B}[w] \quad \forall w \neq v$, the OR-gate v was not satisfied, so we can sum 1 to its corresponding result since we may consider the use of (u, v) in t from this stage;
 - for a tuple t' with Boolean vector equals $\mathcal{B}(\mathcal{B}[v] = 1)$, the OR-gate v has already been satisfied, i.e., it uses another edge. Thus, it is enough to consider the result of this stage.
- 2. Case $u \notin X$, $v \notin X$ or $\mathcal{B}[v] = 0$, the edge (u, v) can not be utilized in current solution; here we copy the solution of t' with the same conditions.

$$c[t, X, \mathcal{B}] = \begin{cases} \min_{\forall \mathcal{B}'} \{c[t', X, \mathcal{B}'] + 1, c[t', X, \mathcal{B}]\}, & \text{if } \{u, v\} \in X \text{ and } \mathcal{B}[v] = 1\\ c[t', X, \mathcal{B}], & \text{if } u \notin X, v \notin X \text{ or } \mathcal{B}[v] = 0 \end{cases}$$
(2.4)

Forget node of an AND-gate – Let t be a forget node and t' be its child such that $X_t = X_{t'} \setminus v$ and f(v) = AND. In this case, we need to choose the best of two possibilities: v is part of the solution; v is not a part of the current solution. These two situations are represented in Eq. 2.5.

$$c[t, X, \mathcal{B}] = \min\left\{c[t', X, \mathcal{B}], c[t', X \cup \{v\}, \mathcal{B}]\right\}$$

$$(2.5)$$

Forget node of an OR-gate – Let t be a forget node and t' be its child such that $X_t = X_{t'} \setminus v$ and f(v) = OR. We need the best of two possibilities:

- 1. v is part of the solution in this case $\mathcal{B}'[v] = 1$ and $\mathcal{B}'[w] = \mathcal{B}[w] \quad \forall w \neq v$; if $\mathcal{B}'[v] = 0$ the solution would be unfeasible.
- 2. If v is not a part of the current solution in this case we recover the solution from t' with the similar conditions.

$$c[t, X, \mathcal{B}] = \min\left\{c[t', X, \mathcal{B}], c[t', X \cup \{v\}, \mathcal{B}']\right\}$$
(2.6)

Join node – Let t be a join node with two children t_1 and t_2 . For tabulation of the join nodes, we need to encode the merging of two partial solutions: one originating from $G_{X_{t_1}}$ and another from $G_{X_{t_2}}$. When merging two partial solutions, we need to check if some OR-gate has more than one in-edge. This can be done through a simple strategy: when we merge two solutions, regarding vectors \mathcal{B}_1 and \mathcal{B}_2 from t_1 and t_2 , respectively, to form \mathcal{B} , we may assume that $\mathcal{B}[i] = \mathcal{B}_1[i] + \mathcal{B}_2[i]$ for each i. Thus, for each possible tuple (t, X, \mathcal{B}) , we have:

$$c[t, X, \mathcal{B}] = \min_{\mathcal{B}_1, \mathcal{B}_2} \left\{ c[t_1, X, \mathcal{B}_1] + c[t_2, X, \mathcal{B}_2] \right\}$$
(2.7)

Recall that every bag of \mathcal{T} has at most tw + 1 vertices, each bag has at most 2^{tw+1} possible subsets X, and at most 2^{tw+1} possible Boolean vectors \mathcal{B} . Since each entry of the table can be computed in $2^{O(tw)}$ time, and the correctness of each entry is straightforward from its description, it holds that SMCC can be solved in time $2^{O(tw)} \cdot n$.

Corollary 1. k-SMCC can be solved in time $2^{O(k \cdot (g+1)^{3/2})} \cdot n + m$, where g is the genus of the input.

Proof. It follows immediately from Lemma 3, Lemma 5 and Theorem 6.

At this point, it is worth investigating how much Corollary 1 can be generalized to encompass larger classes of circuits. Towards this goal, it is convenient to define the notion of the *diameter-treewidth* property as introduced in [29].

A graph class \mathcal{F} is said to have the *diameter-treewidth* property if there is a function $\alpha_{\mathcal{F}} : \mathbb{N} \to \mathbb{N}$ such that every graph in \mathcal{F} of diameter at most d has treewidth at most $\alpha_{\mathcal{F}}(d)$. By combining Lemma 3 with Theorem 6, we have Corollary 2.

Corollary 2. Let \mathcal{F} be a hereditary class of graphs with the diameter-treewidth property. Then k-SMCC on monotone Boolean circuits whose underlying graph belongs to \mathcal{F} can be solved in time $\alpha_{\mathcal{F}}(k) \cdot n + m$, where $\alpha_{\mathcal{F}}$ is a function that depends only on the class \mathcal{F} .

We say that a graph H is an apex graph if there is a vertex in V(H) such that $H \setminus \{v\}$ is a planar graph. Classes of graphs that do not contain every apex graph play an important role in algorithmic theory. Indeed, NP-hard combinatorial problems can be solved much more efficiently on graphs belonging to such classes than on general graphs. Classes of bounded genus are an important example of a graph class that fall into this framework. Indeed, for each $g \in \mathbb{N}$, there is some apex graph that does not belongs to the class of graphs of genus at most g. A celebrated theorem due to Eppstein [29] (see also [23]) states that for each minor-closed class of graphs \mathcal{F} , \mathcal{F} has the diameter-treewidth property if and only if \mathcal{F} does not contain every apex graph. By combining this theorem with Lemma 3 and Theorem 6, we have that Corollary 3 holds.

Corollary 3. Let \mathcal{F} be a class of graphs that does not contain every apex graph. Then k-SMCC on monotone Boolean circuits whose underlying graph belong to \mathcal{F} can be solved in time $\alpha_{\mathcal{F}}(k) \cdot n + m$, where $\alpha_{\mathcal{F}}$ is a function that depends only on the class \mathcal{F} .

Recall that planar graphs have genus 0. Finally, we remark the infeasibility of polynomial kernels for k-SMCC on planar graphs.

Theorem 7. k-SMCC on planar circuits does not admit a polynomial kernel, unless NP \subseteq coNP/poly.

Proof. An or-composition for k-SMCC on planar circuits can be easily obtained by first making a disjoint union of instances and then adding a new OR-gate v_{out} which its inneighbors are exactly the output gates of the instances provided. Thus, we can build an instance with certification-width at most k+1 if and only if one of the input instances has certification-width at most k. Therefore, by the OR-composition framework (see [28]), it holds that k-SMCC on planar graphs does not admit a polynomial kernel, unless NP \subseteq coNP/poly.

2.4 Final considerations

Although several works deal with complexity measures closely related to the notion of a succinct certificate, most of the literature results focus on discovering lower and upper bounds for these measures and characterizing related complexity classes. In this chapter, we address another direction, we introduce the SUCCINCT MONOTONE CIRCUIT CERTIFICATION problem and investigate its time complexity. We show that SMCC is NP-complete even when the input monotone circuit is planar. Regarding Parameterized Complexity, we show the W[P]-membership of k-SMCC and its W[1]-hardness. In addition, from an approach based on bounding the treewidth of the input, we present an FPT algorithm for k-SMCC on graphs with bounded genus.

In this chapter, we deal only with monotone circuits. Therefore, it is interesting to investigate the behavior of succinct certificates on general Boolean circuits and consider the number of negations as an aggregate parameter. Also, it seems interesting to analyze the time complexity of SMCC on other circuit classes. Such studies may provide a robust complexity framework for a measure potentially helpful in designing circuits with sparse activation.

Chapter 3

Energy Complexity of Satisfying Assignments

When faced with threshold circuits used as an artificial neural network, it is possible to observe a contrast with neurons of the human brain. The authors in [80] (based in neuroscience literature) argue that the activation of neurons in a human brain happens sparsely. It was shown in [46] that the metabolic cost of a single spike in cortical computation is very high in a way that approximately 1% of the neurons can be activated simultaneously. This phenomenon happens due to the asymmetric energy cost between neurons activated and non-activated in natural cases. From the other side, digital circuits, when satisfied (outputting true), on average activate 50% of the gates. Under different perspectives, 'energy' (or 'power') of a circuit is a measure that has a lot of attention in the literature. Due to multiple models (from biology, electronics, or purely theoretical), several works address different ways of analyzing the energy of a circuit. In [45], the energy consumption of a circuit considers the switching energy consumed by wires (edges) and gates of VLSI circuits. In [6] and [9], it is analyzed the voltage-to-energy consumed by the gates, taking into consideration the *failure-to-energy*. Other different models are explored in [71] and [12], such works try to explore concepts of energy too intrinsic to the design of practical circuits on electronics.

In this chapter, we deal with a circuit complexity measure called *energy complexity* (EC). The idea behind this measure is to evaluate the number of gates in a circuit that returns **true** for an assignment. A similar concept called 'power of circuit' was studied by [81]. The term *energy complexity* was introduced in [80] as an alternative to the dilemma *artificial vs natural* described above. In [80], the authors prove that the minimization of circuit energy complexity obtains a different structure from the minimization of previously considered circuit complexity measures and potentially closer to the structure of

neural networks in the human brain. The authors proved initial lower bounds for energy complexity and other circuit complexity measure called *entropy*.

With a different perspective, this chapter dedicates attention to optimization and decision problems related to energy complexity. More precisely, we consider the problem of determining the satisfying assignment with minimum energy consumption in monotone circuits, i.e., the best case energy complexity of a satisfying assignment in the class of monotone circuits – $MINEC_M^+$. The minimization of energy complexity potentially can help the design of circuits with sparse activation, hence, more similar to biological models. Our focus is on time (parameterized) complexity of the henceforth defined $MINEC_M^+$.

In [4], a measure called *certification-width* was described, which is the size of a minimum subset of edges that are enough to certificate a satisfying assignment. Such edges form a structure called *succinct certificate* that can be seen as a minimal map of edges to be followed in order to activate the output gate. Note that there are similarities between certification-width and energy complexity. Both measures indicate saturation levels of circuits, but while certification-width focuses on edges, energy complexity is about the activation of gates. However, energy complexity presents two additional challenges: (i) EC ignores the 'firing' of input gates; (ii) EC counts activated gates even if its signal does not reach the output gate (due to unsatisfied gates – see Fig. 3.1). These two issues forbid rushed conclusions about EC based on what we know about certification-width. Nevertheless, the study in [4] also motivates the study of the complexity of computing the best case energy complexity of satisfying assignments in monotone circuits.

Note that in energy complexity problems in addition to working with the gates needed to satisfy the circuit, it is still necessary to handle gates that assignments may collaterally activate (see Fig. 3.1). Such behavior makes working with energy complexity problems more challenging than typical satisfying problems where the focus is only on the minimal set of inputs, gates, or wires/edges sufficient to satisfy the circuit.

3.1 Preliminaries

A Boolean circuit is a model that computes a Boolean function $g : \{0,1\}^n \to \{0,1\}$ over a basis of operators (e.g. {AND, OR, NOT, ...}). In terms of Graph Theory, a Boolean circuit is a directed acyclic graph C(V, E) where the set of vertices $V = (I, G, \{v_{out}\})$ is partitioned into: (i) a set of inputs $I = \{i_1, i_2, ...\}$ composed by the vertices of in-degree 0; (ii) a set of gates $G = \{g_1, g_2, ...\}$, which are vertices labeled with Boolean operators;



Figure 3.1: Anomalous behavior in certificates for $MINEC_M^+$. Note that the edge (i_4, g_2) produces a 'leak' of the assignment, i.e. g_2 output true (increasing the energy complexity), but its signal is not relevant to satisfy v_{out} .

(iii) and the single output (sink) vertex v_{out} with out-degree equal to 0 and also labeled with a Boolean operator (see Fig. 2.1). The input vertices represent Boolean variables that can take values from $\{0, 1\}$ ({true, false}, depending on the conventions), and the label/operator of a gate or output vertex w is given by f(w). A monotone circuit is a Boolean circuit where the Boolean operators allowed are in {AND, OR}. An assignment of C is a vector $X = [x_1, x_2, \ldots, x_{|I|}]$ of values for the set of inputs I, where for each j, $x_j \in \{0, 1\}$ is the value assigned to input i_j . We say that X is a satisfying assignment if the circuit C evaluates to 1 (true) when X is given as input.

Given a Boolean circuit C and an assignment X, the Energy Complexity of X into C, EC(C, X), is defined as the number of gates that output true in C according to the assignment X. The (Worst-Case) Energy Complexity of C (denoted by EC(C)) is the maximum EC(C, X) among all possible assignments X (See [25]). Analogously, the Best-Case Energy Complexity of C (denoted by MinEC(C)) is the minimum EC(C, X) among all possible assignments X.

While computing the worst-case energy complexity of satisfying assignments in monotone circuits is trivial (activate all inputs), the problem of computing the best-case energy complexity among all satisfying assignments in monotone circuits seems a challenge. Therefore, in this chapter, we address this particular case where the circuit is monotone, focusing on the following decision problem: BEST-CASE ENERGY COMPLEXITY OF SATISFYING ASSIGNMENTS IN MONOTONE CIRCUITS – $MINEC_M^+$ Instance: A monotone Boolean circuit C and a positive integer k. Question: Is there a satisfying assignment X for C such that $EC(C, X) \leq k$?

Besides, we denote by k-MINEC⁺_M the parameterized version of MINEC⁺_M where k is taking as the parameter.

3.2 Computational complexity analysis

In this section, we present our (parameterized) complexity results regarding MINEC_{M}^{+} . Using a reduction from PLANAR VERTEX COVER we are able to show that MINEC_{M}^{+} is NP-complete even when restrict to planar circuits. Our NP-completeness proof follows from a polynomial-time reduction from PLANAR VERTEX COVER similar to that employed in 2. For convenience, we repeat even similar steps in this chapter. Here, it is not necessarily take care with possible 'leak' (like that described in Introduction – See Fig. 3.1) since all OR-vertices activated in the reduction below are needed to satisfy the output gate.

Theorem 8. MINEC⁺_M is NP-complete even when restricted to planar circuits.

Proof. Given a circuit C and an integer k, forming an instance of MINEC_M^+ , an assignment of Boolean values can be seen as a certificate for this instance. Since it is easy to count the number of gates outputting **true** according to an assignment, clearly, MINEC_M^+ is in NP.

Now, we show the NP-hardness of MINEC_M^+ by a reduction from PLANAR VERTEX COVER (PVC), a well-known NP-complete problem.

PLANAR VERTEX COVER (PVC) **Instance**: A planar graph G; a positive integer c. **Question**: Is there a set S of size at most c, such that for each edge $(u, v) \in E(G)$ either $u \in S$ or $v \in S$?

First, consider the following preprocessing: let (H, c') be an instance of PLANAR VERTEX COVER, by subdividing twice each edge of H, we obtain a graph G where each edge e = (ab) of H is replaced by a P_4 ab'a'b, where a' and b' are new vertices (Recall Fig. 2.2 in Chapter 2). Notice that G is also planar; H has a vertex cover of size c' if and only if G has a vertex cover of size c = c' + |E(H)|; and given a planar embedding of G, the boundary of any pair of adjacent faces of G contains at least three edges.

From a fixed planar embedding of the instance (G, c) of PLANAR VERTEX COVER, we proceed with the reduction. We will construct an instance (C, k) of MINEC⁺_M where C is a planar monotone circuit, and k is the target size of the energy complexity.

From the structure of G, we apply the following (see Fig. 2.3 and Fig. 2.4):

- 1. first, set V(C) = V(G);
- 2. for each vertex $v_i \in V(G)$, create an input vertex v_i^{in} , assign $f(v_i) = \text{AND}$, and add a directed edge (v_i^{in}, v_i) ;
- 3. for each edge $e_i = (u, v) \in E(G)$, create a vertex $v_{e_i}^{cover}$ such that $f(v_{e_i}^{cover}) = \mathsf{OR}$, and create the directed edges $(v, v_{e_i}^{cover})$ and $(u, v_{e_i}^{cover})$.

Notice that C is still planar. Now, preserving the planarity, we will ensure that every $v_{e_i}^{cover}$ outputs true for any assignment of C as follows:

- 4. create an output vertex v_{out} such that $f(v_{out}) = AND$;
- 5. for each vertex $v_{e_i}^{cover}$ which are in the external face of G, create one directed edge from $v_{e_i}^{cover}$ to v_{out} ;

Let D_G be the dual graph of G, where f_1 represents the external face of G. Let \mathcal{T}_{D_G} be the spanning tree of D_G obtained from a breadth-first search of D_G rooted at f_1 . In a top-down manner, according to a level-order traversal of \mathcal{T}_{D_G} , we visit each edge $e = (f_i, f_j)$ of \mathcal{T}_{D_G} applying the following:

6. let f_j be a child of f_i in \mathcal{T}_{D_G} ; by construction of G, it follows that the boundary between f_i and f_j contains at least three edges, being at least one of which between vertices a' and b' that do not exist in H; thus, create a vertex v_{f_j} , add edges from v_{f_j} to such a' and b'; and for each $v_{e_\ell}^{cover}$ in the face f_j that, yet, doesn't reach v_{out} , add an edge from $v_{e_\ell}^{cover}$ to v_{f_j} ; after that, if v_{f_j} has in-degree greater than 0 then set $f(v_{f_j}) = \text{AND}$, otherwise create an input vertex $v_{f_j}^{in}$ add an edge from $v_{f_j}^{in}$ to v_{f_j} From Step 6 holds that if v_{out} outputs true then every vertex $v_{e_i}^{cover}$ and v_{f_j} also output true. Besides, as each v_{f_j} can be added in the planar embedding inside its respective face, then the resulting graph still planar, and by using \mathcal{T}_{D_G} it holds that the added edges preserve the graph acyclic.

7. finally, set $k = c + |E(G)| + |V(\mathcal{D}_{\mathcal{G}})|$.

If G has a vertex cover S such that |S| = c then we obtain a satisfying assignment A to C having energy complexity at most k as follows: for each $v_i \in V(G)$ such that $v \notin S$ we set **false** to its corresponding input v_i^{in} (created in step 2 of the reduction); and we assign **true** to the other inputs. Thus, in exactly c edges flows **true** from vertices v_i^{in} to its out-neighbor v_i (so, at most c vertices v_i will output **true**); and from each v_i set as **true** flows positive values to each v_e^{cover} such that e is an out-edge of v_i . Therefore, since S is a vertex cover, in a bottom-up manner according to \mathcal{T}_{D_G} , we can observe that each face vertex v_{f_j} , each vertex $v_{e_i}^{cover}$, as well as v_{out} will be set as **true**. Thus, A is a satisfying assignment to C. At this point, it remains to analyze the energy of A. Note that C has exactly |E(G)| vertices $v_{e_i}^{cover}$, $|V(D_G)| - 1$ face vertices v_{f_j} , and all of them is set as **true** by A, which with the addition of v_{out} implies energy consumption $|E(G)| + |V(D_G)|$, since at most c vertices v_i will output **true**, it follows that A has energy $|E(G)| + |V(D_G)| + c$.

Conversely, let C be a circuit with k gates outputting true. By construction, if C is satisfiable, then all vertices $v_{e_i}^{cover}$ and v_{f_j} also are satisfiable and, consequently, there is at most c vertices v_i outputting true, representing, in this way, a vertex cover with c vertices in G.

Next, we investigate the parameterized complexity of $MINEC_M^+$.

Theorem 9. k-MINEC⁺_M is in XP.

Proof.

Let C = (V, E) be a circuit with $V = I \cup G \cup \{v_{out}\}$, where I is the set of inputs of C, G is the set of gates and v_{out} is the output gate. If C has a satisfying assignment X such that $EC(C, X) \leq k$ then we can find X as follows:

1. Suppose that X is the satisfying assignment with $EC(C, X) \leq k$ having minimum weight (i.e., minimum number of inputs assigned as true);

- 2. First, we "guess" the set T of gates that should be activated by X, that is, in $n^{O(k)}$ time, we enumerate each subset T of gates such that $|T| \leq k$ and check each one in a new branch;
- 3. For each T we can check in polynomial time whether it is consistent, that is:
 - $v_{out} \in T;$
 - For each OR-gate v in T either it has an in-neighbor in T or it has an in-neighbor in I, and for each AND-gate v in T its in-neighborhood is contained in $T \cup I$;
 - Conversely, each OR-gate $w \notin T$ has no in-neighbor in T, and for each AND-gate $w \notin T$ has at least one in-neighbor that is not in T;
 - Also, no input is mutually in-neighbor of an AND-gate in T and an OR-gate not in T;
 - Note that, if T is the set of gates activated by X then it holds that: any input i that is in-neighbor of an AND-gate in T should be set as true in X; any input i that is in-neighbor of an OR-gate not in T should be set as false in X. Let X' be such a partial assignment;
 - Therefore, each OR-gate v in T having no in-neighbor in T has at least one in-neighbor in I that is not set as false by X', and each AND-gate $w \notin T$ having no in-neighbor in $G \setminus T$ has at least one in-neighbor in I that is not set as true in X'.
- 4. Since X has minimum weight, from a given consistent set T, in order to extend X' into a satisfying assignment X with $EC(C, X) \leq k$ (if any), it is enough to "guess" the minimal set of inputs that should be set as **true** to activate the OR-gates in T having no in-neighbor in T. As $|T| \leq k$, such subset of inputs is also bounded by k, thus, in $n^{O(k)}$ time, we can enumerate (if any) each assignment X" extending X' by setting at most k additional inputs as **true** in such a way that each OR-gates in T has at least one in-neighbor activated. At this point, from the guessed set T we obtain the assignment X if there is some X" for which each AND-gate $w \notin T$ having no in-neighbor in $G \setminus T$ has at least one in-neighbor in I that is set as **false**.

Note that for any satisfying assignment X of C the set of activated gates must satisfy the properties described in step 3. Since steps 2 and 4 check in $n^{O(k)}$ time all possibilities, it holds that MINEC_M^+ is XP-time solvable. Now, we show the W[1]-hardness of k-MINEC⁺_M using a reduction from MULTICOL-ORED CLIQUE, a well-known W[1]-complete problem [31].

Multicolored Clique

Instance: A graph Q with a vertex-coloring $\ell : V(G) \to \{1, 2, \dots, c\}$.

Parameter: A positive integer c.

Question: Does Q have a c-clique containing all c colors?



Figure 3.2: An instance Q for MULTICOLORED CLIQUE

Proof. Let (Q, c) be an instance of MULTICOLORED CLIQUE and let V_1, V_2, \ldots, V_c be the color classes of Q. Without loss of generality, we consider that each vertex in V_i has at least one neighbor in $V_j (i \neq j)$. We construct an instance (C, k) of MINEC⁺_M(k) as follows (Recall Fig. 3.3 in Chapter 2):

- 1. create an output gate v_{out} in C and set $f(v_{out}) = AND$;
- 2. for each color c_i of Q, create a gate w_i with $f(w_i) = OR$ and add an edge from w_i to v_{out} ;
- 3. for each color class V_i of Q, create copies V_i^1, V_i^2, V_i^3 and V_i^4 in C;
- 4. add edges from each vertex in V_i^4 to w_i ;
- 5. let v^1, v^2, v^3 and v^4 be the copies of a vertex $v \in V(Q)$; add edges $(v^1, v^2), (v^2, v^3)$ and (v^3, v^4) to G; set V_i^1 as the input set; and assign $f(v^2) = f(v^3) = \mathsf{OR}$ and $f(v^4) = \mathsf{AND}$;
- 6. for each vertex $v^4 \in V_i^4 (1 \le i \le c)$, create c-1 new OR-in-neighbors $a_{v^4}^j (1 \le j \le c)$ and $i \ne j$, and for each $u^2 \in V_j^2$ such that $vu \in E(Q)$ create an AND-vertex b_{vu}^j and the following edges: $(b_{vu}^j, a_{v^4}^j)$, (u^2, b_{vu}^j) and (v^2, b_{vu}^j) ;
- 7. finally, set $k = 2c^2 + 2c + 1$.



Figure 3.3: Circuit C obtained from Q (Fig. 3.2) after reduction. The vertices represented as rhombuses are AND-gates; the other vertices are OR-gates except those with in-degree 0 (inputs).

Theorem 10. k-MINEC⁺_M is W[1]-hard.

If Q contains a multicolored clique K such that |K| = c, then it is possible to find a satisfying assignment of C that consumes k energy by mapping the set S of gates/vertices that must be activated (outputs true) as follows: (a) v_{out} and all of its in-neighbors must belong to S; (b) for each OR-gate $w_i \in S$, we want include in S exactly the in-neighbor $v^4 \in V_i^4$ such that $v \in K$, therefore, we set $f(v^1)$ =true if and only if $v \in K$ (At this point, by construction, for each $v \notin K$ holds that every vertex between v^1 and v^4 will be inactivated); (c) for each $v^4 \in S$, all of its in-neighbors must be in S, and for each $a_{v^4}^j$ in S, its unique in-neighbor in S must be the AND-gate b_{vu}^j such that $f(v^1) = f(u^1)$ =true (recall that K has exactly one vertex per color). (d) finally, a vertex $v^2 \in V_2$ belongs to S if and only if its in-neighbor v^1 outputs true. Through a simple count one can conclude that $|S| = 2c^2 + 2c + 1$. Thus, the defined assignment satisfies C by consuming k energy as required.

Conversely, if C has a satisfying assignment X with energy complexity at most $k = 2c^2 + 2c + 1$ then it is possible to obtain a multicolored clique K of Q as follows: a vertex v of Q belongs to K if and only if v^2 outputs **true**. Since, by construction, any satisfying assignment of C activates at least $2c^2 + 2c + 1$ gates in $V(C) \setminus (V_2 \cup V_1)$, the assignment X activates at most c gates in V_2 . Besides, the construction also implies that at least one input per color must activated in order to satisfy C. So, X activates exactly c gates in V_2 (one per color). Therefore, K has exactly one vertex per color. Now, to show that

K induces a clique is enough to observe the if v_2 and u_2 are activated in X into C and $vu \notin E(G)$, then for the color j of u holds that b_{vr}^j is inactivated by X into C for any neighbor r of v with color j. Thus, v_4 and w_i are also inactivated, where i is the color of v, which contradicts the fact that X satisfies C. Therefore, K induces a clique.

3.2.1 On monotone circuits with bounded genus

A graph G has genus at most g if it can be drawn on a surface of genus g (a sphere with g handles) without edge intersections (Recall Fig. 2.5 in Chapter 2). We refer the reader to [36] for more information on the genus of a graph. We consider the genus of a circuit as the genus of its underlying undirected graph.

In this section, we show that k-MINEC⁺_M on bounded genus circuits can be reduced to k-MINEC⁺_M on bounded treewidth circuits.

Definition 13. Let G be an undirected graph G. A tree decomposition G is a pair $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$, such that T is a tree where each node t is assigned to a set of vertices $X_t \subseteq V(G)$, called bags, according to the following conditions:

- $\bigcup_{t \in V(T)} X_t = V(G);$
- For each $uv \in E(G)$ there is a node t such that $\{u, v\} \subseteq X_t$;
- For each $v \in V(G)$, the set $T_v = \{t \in V(T) : v \in X_t\}$ spans a subtree of T.

The width of a tree decomposition \mathcal{T} is the size of its largest bag minus one. The treewidth of G is the minimum width among all tree decompositions of G.

Definition 14. A graph H is a minor of a graph G if H can be constructed from G by deleting vertices or edges, and contracting edges.

Theorem 11 (Excluded Grid Theorem [65]). Let t be a non-negative integer. Then every planar graph G of treewidth at least 9t/2 contains a grid $t \times t$ as a minor.

From the Excluded Grid Theorem, it is easy to see that there is a connection between the diameter of a planar graph and its treewidth. In [66], Robertson and Seymour presented a bound for the treewidth of a planar graph with respect to its radius, which also implies a bound regarding the diameter. **Definition 15.** For every face F of a planar embedding M, we define d(F) to be the minimum value of r such that there is a sequence F_0, F_1, \ldots, F_r of faces of M, where F_0 is the external face, $F = F_r$, and for $1 \le j \le r$ there is a vertex v incident with both F_{j-1} and F_j . The radius $\rho(M)$ of M is the minimum value r such that $d(F) \le r$ for all faces F of M. The radius of a planar graph is the minimum of the radius of its planar embeddings.

Theorem 12 (Radius Theorem [66]). If G is planar and has radius at most r then its treewidth is at most 3r + 1.

Using Theorem 12 we are able to either solve MINEC_M^+ on planar circuits or outputs an equivalent instance C' with treewidth bounded by a function of k.

Lemma 7. Let (C, k) be an instance of MINEC_M^+ . There is an algorithm that in polynomial time either solves (C, k) or outputs an equivalent instance (C', k) of MINEC_M^+ where each vertex is at distance at most 2k + 1 from v_{out} in the underlying undirected graph of C'.

Proof. From an instance (C, k) of MINEC⁺_M, we apply the following reduction rules to obtain C':

- 1. Delete every input vertex which is at a distance greater than k to v_{out} ;
- 2. Delete every vertex which is at a distance greater than k + 1 from its nearest input vertex;
- 3. Delete any AND-vertex which lost one of its in-neighbors;
- 4. Delete any OR-vertex in which its in-degree became equal to 0;
- 5. Repeat steps 1 to 4 as long as possible;
- 6. If $C' = \emptyset$ then we conclude that (C, k) is a *no*-instance of MINEC⁺_M.

We now discuss the safety of the previous reduction rules: if an input vertex v is at a distance greater than k from v_{out} , since C is monotone, then v is not useful to satisfy v_{out} in any assignment X with $EC(C, X) \leq k$, thus we can assume that v outputs false and given the monotonicity of C we can safely remove v (Rule 1). Similarly, gates that are at a distance greater than k from its nearest input vertex must output false in an assignment X; otherwise, X consumes energy greater than k. Note that vertices at a distance exactly k + 1 from its nearest input vertex can be useful to show that a given assignment consumes energy greater than k. However, gates at a distance of at least k + 2from its nearest input vertex can be removed once its neighbors are sufficient to certify the negative answer (Rule 2). Besides, if for any assignment X with $EC(C, X) \leq k$ holds that some (resp. every) in-neighbor of an AND(resp. OR)-vertex v must output false, then v must output false as well. Thus, Rule 3 and Rule 4 are safe. From the safety of rules 1-4, it follows that Rule 5 and Rule 6 are safe. Finally, if $C' \neq \emptyset$ then C' has only vertices at a distance at most 2k + 1 from v_{out} in the underlying undirected graph of C'.

Note that the underlying undirected graph of the circuits obtained from Lemma 7 have diameter bounded by 4k + 2. Therefore, contrasting with the W[1]-hardness for the general case, Corollary 4 holds.

Corollary 4. MINEC⁺_M is fixed-parameter tractable when restricted to monotone circuits having bounded maximum in-degree.

Also, notice that a gate with large in-degree can always be replaced by a binary tree using only binary gates, but for or-gates it makes a relevant difference in the energy complexity. Therefore, replacing large in-degree gates is not a useful strategy for dealing with k-MINEC⁺_M. On the other hand, Lemma 7 also implies that if C' is planar then it also has bounded radius, thus, by Theorem 12, it follows that the underlying undirected graph of C' has treewidth bounded by a function of k. We extend the previous reasoning for bounded genus circuits.

Given a vertex-set $S \subseteq V(G)$ of a simple graph G such that the subgraph of G induced by S, denoted G[S], is connected, contracting S means contracting the edges between the vertices in S to obtain a single vertex at the end. We say that a graph H is an *s*-contraction of a graph G if H can be obtained after applying to G a (possibly empty) sequence of edge contractions.

The following is a construction presented in [33] and [43]. Consider an $(r \times r)$ -grid. A corner vertex of the grid is a vertex of the grid of degree 2. By Γ_r we denote the graph obtained from the $(r \times r)$ -grid as follows (Fig.2.6 illustrates Γ_7): construct first the Γ'_r by triangulating all internal faces of the $(r \times r)$ -grid such that all internal vertices of the grid are of degree 6, and all non-corner external vertices of the grid are of degree 4 (Γ'_r is unique up to isomorphism). Two of the corners of the initial grid have degree 2 in Γ'_r ; let x be one of them. Γ_r obtained from Γ'_r by adding all the edges having x as an endpoint and a vertex of the external face of the grid that is not already a neighbor of x as the other endpoint. Observe again that Γ'_r is unique up to isomorphism. The following is a lemma from [43] implied from Lemma 6 in [33].

Lemma 8 (Lemma 4.5 in [43]). Let G be a graph of genus g, and let r be any positive integer. If G excludes Γ_r as an s-contraction, then the treewidth of G is at most $(2r+4) \cdot (g+1)^{3/2}$.

Lemma 9. Let C' be the circuit obtained from Lemma 7. It holds that C' has treewidth at most $(8k + 14) \cdot (g + 1)^{3/2}$, where g is the genus of C'.

Proof. First, notice that for each vertex u of a Γ_{4k+5} there is another vertex v such that the distance between u and v is at least 2k + 2. Now, suppose that C' has Γ_{4k+5} as an s-contraction, and let u be a vertex of a Γ_{4k+5} such that u is either v_{out} or a vertex obtained by contracting S containing v_{out} . Since there is a vertex v such that the distance between u and v is at least 2k + 2, it holds that C' has a vertex at distance greater than 2k + 1 from v_{out} , which is a contradiction (see Lemma 7). Thus, by Lemma 8 we have that the treewidth of C' is at most $(8k + 14) \cdot (g + 1)^{3/2}$.

3.3 Dynamic programming on bounded treewidth circuits

From Lemma 9, in order to solve k-MINEC⁺_M in FPT-time on bounded genus instances, it is enough to present an FPT algorithm parameterized by the treewidth of the input. To design a dynamic programming on tree decompositions, without loss of generality, we may consider that we are given a tree decomposition that is a rooted *extended nice tree decomposition* (see [22] for details).

Theorem 13. MINEC⁺_M can be solved in time $2^{O(tw)} \cdot n$, where tw is the treewidth of the underlying undirected graph of the input.

Proof. Let $C = (I, G, v_{out})$ be a monotone circuit where I is the set of inputs of C, G is the set of gates with out-degree greater than 0 and v_{out} is a single output vertex. Let $\mathcal{T} = (T, \{X_t\}_{t \in V(T)})$ be a rooted extended nice tree decomposition of C. Consider also T_t as the subtree of T rooted by node t (bag X_t) and C_t be the graph/circuit having $\mathcal{T}_t = (T_t, \{X_i\}_{i \in V(T_t)})$ as tree decomposition. For convenience, we add the vertex v_{out} to every bag of T; thus, the width of \mathcal{T} is increased by at most one.

Now, note that an assignment X satisfies a monotone circuit C if and only if it induces an activation set S_X such that: 1. $v_{out} \in \mathcal{S}_X;$

- 2. for each $v \in \mathcal{S}_X$ holds that:
 - if f(v) = AND then every in-neighbor of v is in S_X ;
 - if f(v) = OR then at least one among the in-neighbors of v is in S_X ;
- 3. for each $v \notin S_X$ holds that:
 - if f(v) = AND then at least one among the in-neighbor of v is not in S_X ;
 - if f(v) = OR then every in-neighbor of v is not in S_X ;

Properties 1 and 2 describe the necessary and sufficient conditions for a set S_X of activated gates to certify a satisfying assignment. Property 3 ensures that S_X is maximal regarding the property of having been activated by X.

Therefore, finding a satisfying assignment X which minimizes EC(C, X) can be seen as the problem of finding a satisfying assignment X which minimizes $|\mathcal{S}_X \setminus I|$. Thus, we define $c[t, S, \mathcal{B}^{OR}, \mathcal{B}^{AND}]$ as the cardinality of a minimum set of gates \mathcal{S}_t (if any) of C_t such that:

- $v_{out} \in S$ and $S = X_t \cap S_t$; (we say that $X_t \setminus S = \overline{S}$)
- for each $v \in V(C_t) \setminus X_t$ properties 2 and 3 holds with respect to \mathcal{S}_t ;
- for each $v \in S$ such that f(v) = AND, all in-neighbors of v in C_t are in S_t ;
- The set \mathcal{B}^{OR} is the subset of OR-gates in S already having in-neighbors in \mathcal{S}_t ;
- for each $v \in \overline{S}$ such that $f(v) = \mathbf{OR}$, all in-neighbors of v in C_t are not in \mathcal{S}_t ;
- The set \mathcal{B}^{AND} is the subset of AND-gates in \overline{S} already having in-neighbors that are not in \mathcal{S}_t ;

Furthermore, the optimal solution of the main problem can be found:

- either at $c[r, \{v_{out}\}, \{v_{out}\}, \{\}]$, if $f(v_{out}) = OR$,
- or at $c[r, \{v_{out}\}, \{\}, \{\}]$, if $f(v_{out}) = AND$

where r is the root of the tree decomposition \mathcal{T} .

Recall that, for any node t we assume that $v_{out} \in S$.

In order to solve MINEC_{M}^{+} , the counting of gates that output true (in the solution) are made in introduce vertex nodes. Note that in the introduce node of an OR-vertex v, it can not be simultaneously in S and \mathcal{B}^{OR} because when a vertex is introduced then it is isolated in C_t (we are considering an extended nice tree decomposition, i.e. the edges are introduced in introduce edge nodes).

Leaf node Let t be a leaf node, then $X_t = \{v_{out}\}$. Since v_{out} must be in S, then $\mathcal{B}^{AND} = \emptyset$. Thus, we have three subproblems in Equation (3.1).

$$c[t, \{v_{out}\}, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] = \begin{cases} 1, & \text{if } f(v_{out}) = \mathsf{AND} \\ 1, & \text{if } f(v_{out}) = \mathsf{OR} \text{ and } v_{out} \notin \mathcal{B}^{\mathsf{OR}} \\ \infty, & \text{if } f(v_{out}) = \mathsf{OR} \text{ and } v_{out} \in \mathcal{B}^{\mathsf{OR}} \end{cases}$$
(3.1)

Introduce vertex node Let t be an introduce vertex node with exactly one child t'such that $X_t = X_{t'} \cup \{v\}$. In the graph C_t , v is an isolated vertex; consequently, as in the leaf nodes, there is infeasibility whenever v belongs to $\mathcal{B}^{\mathsf{OR}}$ or $\mathcal{B}^{\mathsf{AND}}$. Besides, we have the possibility of v be an input vertex $(f(v) \notin \{\mathsf{AND}, \mathsf{OR}\})$ or $v \notin S$, such situations only rescue previous subproblems without increment the current subsolution. On the other hand, we increment the subsolution by 1 whenever $v \in S$. All possibilities are covered in Equations (3.2), (3.3) and (3.4).

• If $f(v) \notin \{\text{AND}, \text{OR}\}$ then

$$c[t, S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] = c[t', S \setminus \{v\}, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}]$$
(3.2)

• If f(v) = OR then

$$c[t, S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] = \begin{cases} c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}], & \text{if } v \notin S \\ c[t', S \setminus \{v\}, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] + 1, & \text{if } v \in S \text{ and } v \notin \mathcal{B}^{\mathsf{OR}} \\ \infty, & \text{if } v \in S \text{ and } v \in \mathcal{B}^{\mathsf{OR}} \end{cases}$$
(3.3)

• If f(v) = AND then

$$c[t, S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] = \begin{cases} c[t', S \setminus \{v\}, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] + 1, & \text{if } v \in S \\ c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}], & \text{if } v \notin S \text{ and } v \notin \mathcal{B}^{\mathsf{AND}} \\ \infty, & \text{if } v \notin S \text{ and } v \in \mathcal{B}^{\mathsf{AND}} \end{cases}$$
(3.4)

Introduce edge node Let t be an introduce edge node and t' its child such that $X_t = X_{t'}$, which introduces the directed edge uv such that $\{u, v\} \subseteq X_t$. Now, by including an edge, we can evaluate each subproblem concerning the sets \mathcal{B}^{OR} and \mathcal{B}^{AND} ; so, for each OR-gate $v \in S$, at least one in-neighbor also must be in S; so, either uv attend this demand or another already introduced edge satisfied that. We apply the same reasoning for AND-gates: considering an AND-gate $v \in \overline{S}$, then at least one in-edge of v need comes to another vertex in \overline{S} ; if uv do not attend this requirement, the current subproblem is assigned to a previous subproblem where $v \in \mathcal{B}^{AND}$. All these conditions are handled in Equations (3.5) and (3.6). Recall that we are introducing the directed edge uv.

• If f(v) = OR then $c[t, S, \mathcal{B}^{OR}, \mathcal{B}^{AND}]$ is equal to

$$\begin{cases} c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}], & \text{if } u \notin S \\ \infty, & \text{if } u \in S \text{ and } v \notin S \cap \mathcal{B}^{\mathsf{OR}} \\ \min \left\{ c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}], c[t', S, \mathcal{B}^{\mathsf{OR}} \setminus \{v\}, \mathcal{B}^{\mathsf{AND}}] \right\}, & \text{if } u \in S \text{ and } v \in S \cap \mathcal{B}^{\mathsf{OR}} \end{cases}$$
(3.5)

• If f(v) = AND then $c[t, S, \mathcal{B}^{OR}, \mathcal{B}^{AND}]$ is equal to

$$\begin{cases} c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}], & \text{if } u \in S \\ \infty, & \text{if } u \notin S \text{ and } v \in S \\ \infty, & \text{if } \{u, v\} \subseteq \overline{S} \text{ and } v \notin \mathcal{B}^{\mathsf{AND}} \\ \min \{c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}], c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}} \setminus \{v\}]\}, & \text{if } \{u, v\} \subseteq \overline{S} \text{ and } v \in \mathcal{B}^{\mathsf{AND}} \end{cases}$$
(3.6)

Forget node Let t be a forget node and t' be its child such that $X_t = X_{t'} \setminus v$. In this case, we verify the best among either selecting or not v in current subproblem. If v is an input vertex, then this verification is trivial (it is enough to rescue the minimum subsolution varying only the membership of v in S). For OR-gates and AND-gates, the same verification are made but considering the feasibility of v through its membership in \mathcal{B}^{OR} and \mathcal{B}^{AND} . Equations (3.7), (3.8) and (3.9) summarize these three scenarios. • If $f(v) \neq \{\text{AND}, \text{OR}\}$ then

$$c[t, S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] = \min\left\{c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}], c[t', S \cup \{v\}, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}]\right\}$$
(3.7)

• If f(v) = OR then

$$c[t, S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] = \min\left\{c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}], c[t', S \cup \{v\}, \mathcal{B}^{\mathsf{OR}} \cup \{v\}, \mathcal{B}^{\mathsf{AND}}]\right\}$$
(3.8)

• If f(v) = AND then

$$c[t, S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] = \min\left\{c[t', S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}} \cup \{v\}], c[t', S \cup \{v\}, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}]\right\}$$
(3.9)

Join node Let t be a join node with two children t_1 and t_2 . For tabulation of the join nodes, we need to combine two partial solutions – one originating from C_{t_1} and another from C_{t_2} – in such a way that the merging is a feasible solution. Recall that G is acyclic so we don't need to care about cycles. Also, if a gate is activated in C_t it must be activated in both children, so we must subtract duplicity. However, since each edge of C_t is in either C_{t_1} or C_{t_2} , the feasibility of merging children's solutions is guaranteed assuming that whether $v \in \mathcal{B}^{OR}/\mathcal{B}^{AND}$ then it is also in the respective set of one of the children, as described in Equation 3.10.

$$c[t, S, \mathcal{B}^{\mathsf{OR}}, \mathcal{B}^{\mathsf{AND}}] = \min_{\mathcal{B}_1^{\mathsf{OR}}, \mathcal{B}_2^{\mathsf{AND}}, \mathcal{B}_2^{\mathsf{OR}}, \mathcal{B}_2^{\mathsf{AND}}} \left\{ c[t_1, S, \mathcal{B}_1^{\mathsf{OR}}, \mathcal{B}_1^{\mathsf{AND}}] + c[t_2, S, \mathcal{B}_2^{\mathsf{OR}}, \mathcal{B}_2^{\mathsf{AND}}] \right\} - |S \setminus I| \quad (3.10)$$

where $\mathcal{B}^{\mathsf{OR}} = \mathcal{B}_1^{\mathsf{OR}} \cup \mathcal{B}_2^{\mathsf{OR}}$ and $\mathcal{B}^{\mathsf{AND}} = \mathcal{B}_1^{\mathsf{AND}} \cup \mathcal{B}_2^{\mathsf{AND}}$.

Every bag of \mathcal{T} has at most tw + 2 vertices (including v_{out}) and v_{out} is fixed in the solution, thus each bag has at most 2^{tw+1} possible subsets S, there are at most 2^{tw+2} possible sets \mathcal{B}^{OR} , and there are at most 2^{tw+1} sets \mathcal{B}^{AND} . Therefore, the entire matrix has size $2^{O(tw)} \cdot n$. As each entry of the table can be computed in $2^{O(tw)}$ time, it holds that the algorithm performs in time $2^{O(tw)} \cdot n$.

Corollary 5. MINEC⁺_M can be solved in time $2^{O(k \cdot (g+1)^{3/2})} \cdot n^{O(1)}$, where g is the genus of the input.

Chapter 4

Parameterized complexity classes defined by threshold circuits

A decidable problem is a *parameterized problem* when coupled to its instance, some additional information representing particular aspects of the input (that constitute the parameter) are also given. Typically, the parameters measure structural properties of the input or the solution size to be found. In general, the parameters are aspects of the input and/or question to be answered that are isolated for further specialized analysis, where it is assumed that the size of these parameters is much smaller than the size of the input.

Considering an instance I of a problem Π parameterized by k, we say that Π is fixedparameter tractable (it belongs to the class FPT) if Π can be solved by an algorithm (called FPT algorithm) in $f(k) \cdot poly(|I|)$ time. Alternatively, a parameterized problem Π is slice-wise polynomial ($\Pi \in XP$) if there is an algorithm that solves any instance I of Π in $f(k) \cdot |I|^{g(k)}$ time.

The main goal in a parameterized complexity analysis is to design FPT algorithms for the target problem. However, some problems have a higher level of intractability that brings to us the concepts related to the *W*-hierarchy.

A decision Boolean circuit is a Boolean circuit consisting of small and large gates¹ with a single output line, and no restriction on the fan-out of gates. For such a circuit, the depth is the maximum number of gates on any path from the input variables to the output line, and the weft is the maximum number of large gates on any path from the input variables to the output line.

The W-hierarchy was originally motivated by considering the "circuit representation"

¹A gate is called large if its fan-in exceeds some bound, which is typically considered to be two.

of parameterized problems, and terms the union of parameterized complexity classes defined by the weft of decision Boolean circuits.

Before defining the W-hierarchy classes, we consider the following definitions.

Definition 16 (fixed-parameter reduction). Let $A, B \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized problems. A fixed-parameter (or parameterized) reduction from A to B is an algorithm that, given an instance (x, k) of A, outputs an instance (x', k') of B such that

- (x,k) is a yes-instance of A if and only if (x',k') is a yes-instance of B,
- $k' \leq g(k)$ for some computable function g, and
- the running time is $f(k) \cdot |x|^{O(1)}$ for some computable function f.

WEIGHTED WEFT t DEPTH h CIRCUIT SATISFIABILITY – WCS(t, h)Instance: A Boolean decision circuit C with weft t and depth h. Parameter: A positive integer k. Question: Does C have a weight k satisfying assignment?

Definition 17. A parameterized problem Π belongs to the class W[t] if and only if Π is fixed-parameter reducible to WCS(t, h) for some constant h.

For instance, k-INDEPENDENT SET (parameterized by k) is fixed-parameter reducible (FPT-reducible) to WCS(1, 2), thus it belongs to W[1]. Alternatively, k-DOMINATING SET (parameterized by k) is FPT-reducible to WCS(2, 2), which implies that it is in W[2]. In addition, it is conjectured that k-DOMINATING SET cannot be fixed-parameter reducible to WCS(1, h) for some h, since it is complete for the class W[2]. Therefore, it is assumed that k-DOMINANTING SET has higher parameterized complexity than k-INDEPENDENT SET, since it seems to admit only more complex circuit representations (i.e., circuit representations of bounded depth with greater weft).

Based on this, several parameterized problems are classified according to their parameterized complexity level. Recall that $FPT \subseteq W[1] \subseteq W[2] \subseteq \ldots \subseteq XP$, and it is conjectured that each of the containment is proper [28].

The W[t] classes are defined by satisfiability problems of circuits with bounded depth. Additionally, they are also considered parameterized complexity classes defined by circuits having no bound on the depth, so-called W[P] and W[SAT]. These classes are generated by the following problems. WEIGHTED CIRCUIT SATISFIABILITY – WCS
Instance: A decision Boolean Circuit C.
Parameter: A positive integer k.
Question: Does C has a satisfying assignment of weight k?

Weighted Satisfiability – WSAT

Instance: A decision circuit C corresponding to a Boolean formula

(or alternatively, just a Boolean formula C).

Parameter: A positive integer k.

Question: Does C has a satisfying assignment of weight k?

Definition 18. The class W[P] is the class of parameterized problems that are fixedparameter reducible to WEIGHTED CIRCUIT SATISFIABILITY.

Definition 19. The class W[SAT] is the class of parameterized problems that are fixedparameter reducible to WEIGHTED SATISFIABILITY.

Although W[P] and W[SAT] are both defined by unbounded depth circuits, it is worth mentioning that circuits corresponding to Boolean formulas are treelike circuits, i.e., circuits whose graph induced by its gates is isomorphic to a tree. Besides, general decision Boolean circuits can be transformed into treelike circuits; however, the time complexity for such a transformation typically takes exponential time on their number of gates and depth. Thus, it is conjectured that W[SAT] \subset W[P], i.e., the containment is proper.

Thus, the W-hierarchy is organized as follow:

$$W[1] \subseteq W[2] \subseteq \ldots \subseteq W[SAT] \subseteq W[P].$$

Besides, the W-hierarchy classes are defined by circuits restricted to conventional Boolean operators (AND, OR and NOT). By allowing another kind of circuits, potentially, it is possible to represent more decision parameterized intractable problems. In Circuit Complexity, the characterization of complexity classes concerning threshold circuits is well-known. In 1987, Hajnal et al. [38] defined TC^0 , the class of all languages which are decided by threshold circuits with constant depth and polynomial size. Analogously, we can use similar reasoning to establish a hierarchy of parameterized complexity classes generated by weighted satisfiability problems on non-conventional circuits. In [30], the authors constructed a hierarchy of classes called W(C) as an alternative to W-hierarchy classes. However, in [30], the main discussion is restricted to bounded connectives gates (including threshold gates with bounded threshold). In addition, they also presented some results regarding majority gates.

In this chapter, we focus on general threshold gates and define the *Th-hierarchy* as an analogue of the W-hierarchy by replacing decision Boolean circuits by decision *threshold* circuits. The primary tool used in this chapter is *sorting networks*, which are used to transform threshold gates in Boolean circuits efficiently.

In Section 4.1, we present some preliminaries about threshold circuits and sorting networks. Also, in Section 4.1, the Th-hierarchy is formally defined. In Section 4.2, we made the first comparisons between W- and Th-hierarchies, especially about W[P] and Th[P]; here, we use sorting networks to support our conclusions. In Section 4.3, we face a challenge in relating the Th-hierarchy classes with the W[SAT] class. As the W[SAT] class deals with treelike circuits, trivial conversions using sorting networks are not helpful because converting a circuit C(V, E) with depth h into a equivalent treelike circuit takes exponential time with respect to h. Finally, by analyzing the time complexity to construct a particular sorting network with $O(\log n)$ depth, called AKS, we show that Th[t] \subseteq W[SAT] for every $t \in \mathbb{N}$.

4.1 Satisfiability of threshold circuits

The characterization of decision problems as WCS(t, h) in standard Boolean circuits has widespread attention, especially considering the enormous advance in Parameterized Complexity Theory. When classifying a problem in the W-hierarchy, in short, we are encapsulating the parameterized intractability of this problem in terms of satisfiability of a corresponding circuit based on Boolean functions (AND, OR and NOT). Thus, some questions emerge. Is W-hierarchy comprehensive enough? Are there problems complete considering a more general basis of functions?

Due to these questions, naturally, our curiosity turns into the *threshold circuits*. Threshold circuits are circuits that admit threshold gates, i.e., gates that emulate threshold functions (See Definition 20). In Section 4.1.1, we provide some notations about threshold circuits.

4.1.1 Preliminaries

First, we present some conventions and preliminaries that are important for the sequence of this chapter.

Definition 20 (Threshold function). Given a set $A = \{a_1, a_2, \ldots, a_n\}$ of inputs (with $a_i \in \{0, 1\}$, for any $1 \le i \le n$), a set $W = \{w_1, w_2, \ldots, w_n\}$ of weights (with $w_i \in \mathbb{Z}$, for any $i \le n$) and an integer value t called threshold, then a threshold function $T_t^n(A, W)$ holds as follows:

$$T_t^n(A, W) = \texttt{true} \iff \sum_i^n (a_i \times w_i) \ge t, \text{ otherwise } T_t^n(A, W) = \texttt{false}.$$

We can specialize the Definition 20 for functions where every $w_i \in W$ is equal to 1, such functions are called unweighted threshold functions. In practice, an unweighted function evaluates **true** when exactly t inputs $a_i \in A$ are set to be 1. A particular unweighted threshold function is the *majority* function, which has threshold t equals to n/2.

Definition 21 (Decision threshold circuits). A decision threshold circuit is a decision circuit containing AND gates, OR gates, NOT gates, and unweighted threshold gates, where every unweighted threshold gate computes an unweighted threshold function.

Note that one can consider circuits having weighted threshold gates; however, in this chapter we are dealing only with unweighted threshold gates.

4.1.2 The Th-hierarchy

For convenience, we present a generalization of WCS(t, h) by considering decision threshold circuits.

WEIGHTED WEFT t DEPTH h THRESHOLD CIRCUIT SATISFIABILITY – WTCS(t, h)Instance: A decision threshold circuit C with weft t and depth h. Parameter: A positive integer k.

Question: Does C has a satisfying assignment of weight k?

Similarly, we present the complexity classes Th[t].

Definition 22. A parameterized problem Π belongs to the class Th[t] if and only if Π is fixed-parameter reducible to WTCS(t, h) for some constant h.

Analogously, we define WEIGHTED THRESHOLD CIRCUIT SATISFIABILITY (WTCS) and Th[P] as a generalization of WCS and W[P] by considering decision threshold circuits instead of decision Boolean circuits. In addition, we define the generalization of WSAT as follows.

WEIGHTED TREELIKE THRESHOLD CIRCUIT SATISFIABILITY – WTTSAT Instance: A decision threshold circuit C whose graph induced by its gates is isomorphic to a tree (treelike circuit).

Parameter: A positive integer k.

Question: Does C has a satisfying assignment of weight k?

Hence, the *Th*-hierarchy is as follow:

 $Th[1] \subseteq Th[2] \subseteq \cdots \subseteq Th[SAT] \subseteq Th[P].$

By definition, it holds that $W[t] \subseteq Th[t]$ (for every $t \in \mathbb{N}$) as well as $W[SAT] \subseteq Th[SAT]$ and $W[P] \subseteq Th[P]$.

At this point, some questions emerge such as "W[t] = Th[t], for each t?", "W[P] = Th[P]?", "W[SAT] = Th[SAT]?", and "W[1] = Th[1]?". In Sections 4.2 and 4.3, we explore some of these issues.

To understand the relationship between these classes (at the highest levels), we revisit the Sorting Network field and present a time complexity analysis for the construction of AKS sorting networks [59].

4.2 W-hierarchy versus Th-hierarchy

Although the W-hierarchy is a set of infinite classes of parameterized problems, it may possible that the W-hierarchy is not complete in the sense that may exist a parameterized problem Π such that $\Pi \in W[t+1]$; $\Pi \notin W[t]$; Π is hard for W[t]; but, Π is not complete for W[t+1], for some t. Then, it is possible that there are problems between W[t] and W[t+1], or between W[SAT] and W[P].

Therefore, one of the motivation of the work in this chapter is consider classes based on threshold circuits to identify potential gaps in the W-hierarchy.

By definition, the following proposition is clear.

Lemma 10. $W[t] \subseteq Th[t]$, for every $t \in \mathbb{N}$.



Figure 4.1: Relationship between W[t] and Th[t] classes.

In contrast, it is not clear if $Th[t] \subseteq W[t]$ for some t. Figure 4.1 depicts the current snapshot of these classes.

We begin our discussion by disregarding structural constraints on circuits, which leads us to the W/P versus Th/P dilemma.

To show that $\text{Th}[P] \subseteq W[P]$ it is enough to present a fixed-parameter reduction from WEIGHTED THRESHOLD CIRCUIT SATISFIABILITY to WEIGHTED CIRCUIT SATISFIA-BILITY. In this case, it suffices to provide a way to locally replace each unweighted threshold gate for an equivalent Boolean circuit. For that, we consider the *Sorting Networks*.

4.2.1 Sorting networks

A sorting network is a comparison network circuit with n inputs and n outputs, where the outputs are monotonically ordered (using AND and OR gates only). Such circuits are represented by directed graphs having n inputs (bits to be ordered) and n outputs (ordered bits).

The first implementation of a sorting network was proposed by Daniel G. O'Connor and Raymond J. Nelson in 1954, patented three years later [57]. In 1968, Kenneth E. Batcher [11] presented some fundamental concepts about Sorting Networks. One of these concepts is the *comparison element* idea (See Figure 4.2a), which consists in a node of the network that receives two inputs A and B and it returns the outputs L and H, such that L = min(A, B) and H = max(A, B). For Boolean values, a comparison elements can be constructed with two Boolean gates as shown in Figure 4.2b.

Therefore, we can define sorting networks as circuits with n inputs that flow by multiple comparison elements, resulting in ordering those n values in a deterministic sequence of steps.



Figure 4.2: Comparison elements.

It is possible to construct a sorting network that simulates famous sort algorithms, e.g., Bubble Sort. In [11], for instance, was described the *Bitonic Sorter* inspired by the Merge Sort algorithm.

Depending on the strategy adopted to organize a sorting network, we can have circuits with different depths. While a sorting network based on bubble sort has O(n) depth, a bitonic sorter has $O(\log^2 n)$ depth. Furthermore, the most popular sorting networks can be constructed in polynomial time. As example, Bitonic Mergesort can be constructed in $O(n \times \log^2 n)$ time [11]. Thus, Lemma 11 is supported by a vast literature.

Lemma 11. Given an unweighted threshold gate T with n inputs and threshold t, in polynomial time with respect to n, one can construct an decision Boolean circuit C_T such that C_T computes the same function of T.

Proof. Let I be the set of inputs of T. It is enough to construct in polynomial time a sorting network for the set I of inputs (sorting from highest to lowest value) and then connect the network outputs in such a way that the output gate returns **true** if and only if t-th output of the sorting network is **true**.

Theorem 14. It holds that Th[P] = W[P].

Proof. By definition, it is clear that $W[P] \subseteq Th[P]$. To show that $Th[P] \subseteq W[P]$, it is enough to present a fixed-parameter reduction from WEIGHTED THRESHOLD CIR-CUIT SATISFIABILITY to WEIGHTED CIRCUIT SATISFIABILITY by locally replacing each unweighted threshold gate for an equivalent Boolean decision circuit. Therefore, by Lemma 14 the claim holds.

4.3 On the classes Th[t], Th[SAT], and W[SAT]

Due to the depth constraints, Lemma 11 does not implies that W[t] = Th[t], for any $t \in \mathbb{N}$. In addition, correlated with the structural issues of each class, we also have to

worry about the algorithmic time needed for converting a threshold circuit into a Boolean one, respecting such restrictions.

Also, the discussion between W[SAT] and Th[SAT] seems challenging. The WSAT problem does not have restrictions on weft or depth, but it considers only treelike circuits. This constraint brings us an alert about the depth when using sorting networks to convert threshold gates into a Boolean circuit, because after this conversion, it is still necessary to convert the resulting sub-circuit into a treelike one.

Since treelike decision circuits are exactly the decision circuits where each gate has fan-out equal to one (each gate has a single parent), by duplicating gates, it is easy to see that from a decision circuit G one can obtain an equivalent treelike decision circuit in $|V(G)|^{O(h)}$ time, where h is the depth of G and V(G) is the set containing gates and input variables of G. However, when each gate has fan-out bounded by a constant c, by duplicating the gates in top-down manner according to the gates' depth (the length of the longest path from each gate to the output line), one can duplicate each gate g_i at most c^{h_i} times, where h_i is the length of the longest path from g_i to the output line. Therefore, in this setting one can construct an equivalent treelike decision circuit in $c^h \cdot |V(G)|^{O(1)}$ time.

Note that sorting networks tend to have only bounded fan-out gates, since its construction is typically based on Boolean comparators. This motivates us to revisit an in-depth discussion in Theory of Computation about the existence and construction of sorting networks with $O(\log n)$ depth.

However, even if we have an algorithm that converts a threshold gate in Boolean circuits with $O(\log n)$ depth, we still have trouble with the cascade effect of the local gate-replacement/tree-conversion process, i.e., by replacing a threshold gate by an equivalent treelike decision circuit, one can increase the fan-out of gates that were seen as inputs by this threshold gate, which implies duplications in the level inter such sorting networks. Note that such a process may take $|V(G)|^{O(h)}$ time, since after the replacement of threshold gates the fan-out of some gates may be unbounded. Since instances of WSAT may have unbounded depth, we left open the question "W[SAT] = Th[SAT]?". However, our sorting network framework is able to show that Th[t] \subseteq W[SAT] (for every $t \in \mathbb{N}$), if one can construct in polynomial time sorting networks with depth $O(\log n)$ and bounded maximum fan-out.

Note that in Lemma 11 we do not ensure the existence of sorting networks with logarithmic depth. Besides, the only known sorting network that satisfies such a condition is the AKS Sorting Networks. However, to the best of our knowledge, there are no time complexity analysis of an explicit construction of an AKS sorting network. In Section 4.3.1, we detail this particular type of sorting network and we address the possibility of using AKS sorting networks to prove that $\text{Th}[t] \subseteq W[\text{SAT}]$, for every $t \in \mathbb{N}$.

4.3.1 AKS sorting networks

Proposed by M. Ajtai, J. Komlos, and E. SzemerAl'di [1], AKS sorting networks are originally based on a probabilistic error-recovery structure called *separator*. A separator is a network composed of structures called ϵ -halvers. A separator, in short, partitions its inputs into four parts semi-ordered where each partition has a tolerance for *strangers*. Hence, constructing an AKS sorting network is an arrangement of separators in an efficient manner that the values flow in it with a low depth. The core of AKS sorting networks is a kind of graphs called *expander graphs*, which are graphs endowed with good connectivity properties. We refer to the following definitions to better describe AKS sorting networks.

In this work, for the construction of an AKS network we consider a particular type of expander: a Bipartite balanced k-regular graph based on a Ramanujan graphs. Thus, for convenience, we define only one specific configuration of an expander in Definition 23.

Let G be a bipartite graph with a bipartition of V(G) into V_1 and V_2 . We denote by $\Gamma(W)$ the neighborhood of a subset $W \subset V(G)$.

Definition 23 (Bipartite (k, ϵ) -expander graphs). A bipartite graph $G = ([V_1, V_2], E)$ with n vertices is (k, ϵ) -expander, if and only if

- the sets V_1 and V_2 contain exactly n/2 each one;
- every vertex has degree k, and
- for every subset $W \neq \emptyset$ such that either $W \subseteq V_1$, or $W \subseteq V_2$ it holds that

 $|\Gamma(W)| \times \epsilon \ge \min(\epsilon/2, |W|) \times (1 - \epsilon)$

where $0 < \epsilon < 0.5$.

Figure 4.3 illustrates a bipartite (3, 1/4)-expander graph.

One of the AKS networks "tricks" is the tolerance with elements out of position (*strangers*) after a comparison stage. Therefore, AKS sorting networks uses ϵ -halvers instead of perfect halvers (See Definition 24).



Figure 4.3: A bipartite (3, 1/4)-expander graph G.

Definition 24 (Halvers). A circuit C with n inputs (where n is even) is a perfect halver if C return the input values in two output sets: One with exactly n/2 larger inputs; and other with n/2 smaller inputs. We say that a circuit is an ϵ -halver when these two output sets has at most $\epsilon \times n$ strangers (inputs which were directed to wrong output set).

Due to the expansion properties of a (k, ϵ) -expander, it is possible to extract several perfect matchings. Each perfect matching divides the *n* inputs into two sets with the same size. Thus, it is possible to perform n/2 swaps for each perfect matching and the pairs in matchings are natural comparators (see Fig. 4.4). From (k, ϵ) -expanders we design ϵ -halvers with k swap stages (see Fig. 4.5).



(a) A perfect matching M in G.



(b) Swap based in M.

Figure 4.4: A perfect matching in a (k, ϵ) -expander determining swap stages. Each vertical wire in (b) represents a comparator between its endpoints.

Considering that an ϵ -halver H outputs the sets V_1 and V_2 such that V_1 has the smallest values and V_2 has the largest values, with an ϵ -tolerance to *strangers*. Using these semi-ordered values outputted by H to construct another ϵ -halver H', we have a new round of swaps. Here, it is expected that H' outputs the sets V'_1 and V'_2 with less *strangers*. It is easy to see that a well-structured web of ϵ -halvers, without doubt, can



Figure 4.5: A (1/4)-halver constructed from G with three swap stages.
order n inputs in few stages. To this web, we call the notion of *separator*.

Definition 25 (Separator). A circuit C with n inputs is a $(\lambda, \sigma, \epsilon)$ -separator if C returns four output sets G_1, G_2, G_3 and G_4 (a partition of the inputs) of sizes $\lambda \times (n/2), (1 - \lambda) \times$ $(n/2), (1 - \lambda) \times (n/2)$ and $\lambda \times (n/2)$, respectively. In addition, G_1 and G_4 have at most $\sigma \times \lambda \times (n/2)$ strangers; and G_2 and G_3 have at most $\epsilon \times (n/2)$ strangers.

Applying the same reasoning of the ϵ -halvers to separators, it is easy to see that flowing values through a network of separators, in few stages, the partitioning of G_1, G_2, G_3 and G4 will dissipate the presence of its *strangers*, and consequently, this order the input values. That is a summary of the idea behind the AKS sorting network: a chain of separators, which is, in its turn, a chain of halvers. Note that we are not providing all details about AKS sorting networks. A survey about this topic can be found in [8].

For a while, the existence of a $O(\log n)$ -depth sorting network was an open question. The first version of an AKS sorting network [1] solved this question. However, behind the asymptotic expression $O(\log n)$ there is a constant factor of approximately 2^{100} . This huge constant factor inhibits the practical implementation of AKS sorting networks, and there are just a few explicit algorithms describing the construction of an AKS network. In 1990, Paterson [59] presented an improved construction of AKS networks with a depth substantially smaller, but still impracticable. After all, several works address some slight improvements in the AKS construction and Paterson algorithm. However, none of those works achieve a substantial decrease in the constant factor of AKS networks' depth.

In [87], Hang Xie presented an explicit description of the Paterson's strategy. However, although Hang Xie presented a constructive proof to obtain AKS networks, to the best of our knowledge, there is no work presenting a time complexity analysis of the construction of this sorting networks. The main reason for the absence of this analysis is due to the "galactic" size of these networks, which makes their practical implementation impossible. However, in this chapter, our interest in AKS networks is different, since our focus is on complexity classes.

Nevertheless, from the explicit algorithm detailed in [87], we have all tools to address the time complexity of such a construction. As we remark in Section 4.3.2 that the construction presented in [87] can be performed in polynomial time, we can conclude the proper containment of $\text{Th}[t] \subseteq W[\text{SAT}]$, for every $t \in \mathbb{N}$.

4.3.2 Th[t] \subseteq [SAT]

Now, we present a description of the construction of an AKS network detailed in [87]. We focused only on algorithmic time to perform each step of the construction. For more details on the construction see [87].

Given a value ϵ and n (number of keys to be ordered), we can construct a ϵ -halver according to Algorithm 1.

Algorithm 1 (ϵ -Halver construction [87]).

1. Let

$$K = \frac{2(1-\epsilon)(1-\epsilon+\sqrt{1-2\epsilon})}{\epsilon^2}$$

- 2. Pick the minimum prime p congruent to $1 \mod 4 (\geq K 1)$. Let k = p + 1.
- 3. Find the minimum prime q congruent to 1 mod 4, such that $q \ge n/2$.
- 4. Construct a k-regular Ramanujan graph with q vertices.
- 5. Construct a balanced bipartite k-regular graph G_B with 2q vertices.
- 6. Find a perfect matching of the graph using Hungarian algorithm and transform each pair in a sequence of comparators; remove this matching from the graph to get a (k-1)-regular graph;
- 7. Repeat 6 until G_B has no perfect matchings.

In this construction, ϵ -halvers are organized as balanced bipartite k-regular graphs generated from k-regular Ramanujan graphs of q vertices, where $q \ge n/2$. Additionally, the method used to construct Ramanujan graphs was based on [50, 51] (also known as LPS graphs). Thus, the first three steps in Algorithm 1 were dedicated to finding k and q based on the desirable K.

In order to find an appropriated value of K, Hang Xie was based on [78] to calculate the depth of an associated ϵ -halver providing the Lemma 12.

Lemma 12. (From [87]) Let

$$K = \frac{2(1-\epsilon)(1-\epsilon+\sqrt{1-2\epsilon})}{\epsilon^2}$$

For every k-regular LPS graph, if $k \ge K$ then every subset X of size ϵn has at least $(1 - \epsilon)n$ neighbors.

The choice of ϵ impacts the depth and size of this ϵ -halver. For example, if we decide to construct a (1/72)-halver, we have $K \approx 20162.99$. By steps 2 and 3, we can find p = 20173, which indicates the creation of a 20174-regular Ramanujan graph in step 4.

For a smaller (or larger) Ramanujan graph, we need to exploit the equation in Lemma 12 to discover another ϵ . This choice can be guided by the size of the input to be ordered. If we already know the desirable ϵ , then the first step costs O(1). Also, pand q are values used to construct Ramanujan graphs called LPS graphs (more details in [50, 87]). Be prime and congruent to 1 mod 4 are specific conditions for this construction. By number theory, we know that for a value $m \geq 13$ then there is a prime number xcongruent to 1 mod 4 such that $m \leq x \leq 2m$. Hence, since p need to be larger than Kthen in steps 2 find p by iterating from K to 2K until find the first number that fit the conditions. Hence, steps 2 takes polynomial time on K, since for each candidate value we have to check if it is prime. In addition, K depends only on ϵ . For q we need to find a prime number congruent 1 mod 4 and greater than n/2. By iterating q from n/2 to n, and verifying for each q if it is prime, we polynomial time on n.

For Step 4, it is well-known that a graph with a small second eigenvalue of its adjacency matrix is a good expander. By Alon-Boppana theorem [3], we know that $2\sqrt{k-1}$ is the lower bound for the second eigenvalue of a k-regular graph adjacency matrix. By Definition 26, we can observe that the Ramanujan graph is a family of k-regular graphs with the best possible second eigenvalue of its adjacency matrix, which guarantees Ramanujan graphs as good expanders.

Definition 26. A Ramanujan graph is a connected k-regular graph whose eigenvalues are at most $2\sqrt{k-1}$ in absolute value.

The explicit construction of a k-regular Ramanujan graph was presented in [50, 51] and takes polynomial time on k.

For Step 5, in order to create a balanced bipartite k-regular graph G_b from the Ramanujan graph G created in step 4, we present the Algorithm 2, which takes O(|E(G)|)time.

Algorithm 2.

- 1. Given a G(V, E) with q vertices $V = \{v_1, v_2, \dots, v_q\}$, create a graph $G_b([U, W], E_b)$ with 2q vertices divided in $U = \{u_1, u_2, \dots, u_q\}$ and $W = \{w_1, w_2, \dots, w_q\}$;
- 2. For each pair of positive integers (i, j), with $i \leq n$ and $j \leq n$, create an edge from u_i to w_j in E_b iff $(v_i, v_j) \in E$.

Finally, for Step 6 and 7, we perform k executions of the well-known Hungarian algorithm, which also can be done in polynomial time.

After analyzing each step of Algorithm 1, we conclude that this explicit construction of an ϵ -halver can be done in polynomial-time.

Now, it remains to verify the time complexity in arranging a separator.

Here, we present a simplified construction of a $(\lambda, \sigma, \epsilon)$ -separator. In [59, 87], there is more sophisticated constructions. But, our purpose is only to show it is possible to perform it with a polynomial construction.

Algorithm 3. $((\lambda, \sigma, \epsilon)$ -separator construction) [8].

- 1. Given an input with m keys to be ordered, create an array S with m positions in the bottom level d.
- 2. Construct a ϵ -halver in the first level (level 0) of the separator. Apply the *m* keys to this ϵ -halver and send the output sets L_0 and R_0 with m/2 (each) to level 1.
- 3. For each level 0 < i < d, construct two ϵ -halver h_i^L and h_i^R and then:
 - Apply the $m/2^i$ keys in L_{i-1} to h_i^L and send the output left half to level i+1 as L_i .
 - Send the output right half of h_i^L to the positions $S[(m/2^{i+1})+1]$ to $S[m/2^i]$ in bottom level.
 - Apply the $m/2^i$ keys in R_{i-1} to h_i^R and send the right half of the output to level i + 1 as R_i .
 - Send the output left half of h_i^R to the positions $S[(m/2^i) + 1]$ to $S[(m/2^i) + (m/2^{i+1})]$ in bottom level.

Note that by this construction $\lambda = 2^{(1-d)}$ and $\sigma = d \times \epsilon$ (the presence of $\epsilon \times m$ strangers dissipates across the levels resulting in σn). For a depth d = 4 (See Figure 4.6), we can construct a (1/8, 1/18, 1/72)-separator using (1/72)-halvers.



Figure 4.6: A (1/8, 1/18, 1/72)-separator. The array S is formed by $\{G1, G2, G3, G4\}$.

The sorting network has $O(\log n)$ layers with a complete binary tree on each layer with O(n) nodes. Each node has a $(\lambda, \sigma, \epsilon)$ -separator, and the layers are interconnected so that the complete circuit has depth $O(\log n)$ (See a detailed construction in [8], Section 11.4). Then, Proposition 1 holds.

Proposition 1. One can construct an AKS sorting network in polynomial time.

Lemma 13. For each $t \in \mathbb{N}$, it holds that $Th[t] \subseteq W[SAT]$.

Proof. To show the W[SAT]-membership of Th[t], for every $t \in \mathbb{N}$, we first ensure the existence of a polynomial-time algorithm that converts a threshold gate (with fan-in n) in Boolean circuit with $O(\log n)$ depth. We know that the AKS sorting networks have logarithmic depth, and by construction its gates has bounded fan-out. In addition, AKS sorting networks can be constructed in polynomial time on the number of inputs as described in this section.

Therefore, we can replace each unweighted threshold gate for an equivalent decision Boolean circuit by first construct an AKS network and then converting it into a treelike sub-circuit. After that, to obtain a complete treelike circuit, it is enough to handle with the gates inter sorting networks, which can be done since instances of WCS(t, h) have depth bounded by h, which is a constant.

Hence, in polynomial time one can take an instance of WTCS(t, h) and outputs an equivalent treelike Boolean circuit C, i.e., a decision circuit corresponding to a Boolean formula. Thus, WTCS(t, h) is fixed-parameter reducible to WSAT and $Th[t] \subseteq W[SAT]$, for each $t \in \mathbb{N}$.



Figure 4.7: Relationship between W-hierarchy and Th-hierarchy.

Figure 4.7 shows the inclusion relationships we know between classes of the W-hierarchy and Th-hierarchy.

Chapter 5

Maximum Multi Improvement on Neighborhood Exploration

Heuristics for combinatorial optimization problems often perform local search as an internal procedure, that usually fit inside global search frameworks called metaheuristics. Some metaheuristics use a concept of local search attached to the exploration of an explicit set of *neighbor* solutions (achievable by a single *move* operation) called neighborhood, e.g., Variable Neighborhood Search [55], Iterated Local Search [49], Tabu Search [35]. Traditionally, these methods include a Best Improvement (BI) or First Improvement (FI) strategies, respectively selecting the best or first improving solution, in a given neighborhood [77].

Recently, it was proposed the Multi Improvement (MI) neighborhood exploration strategy [63], in order to efficiently select independent moves in a neighborhood via heuristic or exact approaches. The need for performing local search on parallel computing architectures, previously promoted by multicore Central Processing Units (CPUs), and lately by Graphics Processing Units (GPUs), has driven the development of similar approaches (see [7, 19, 63, 84]). However, the lack of a theoretical formulation for the methodology and a well-defined concept of *independence on neighborhood operators* have inspired the development of this study.

The MI can be seen as an alternative to known BI and FI with a heavier computational load, but seeking to achieve better quality solutions in fewer iterations. It was recently applied to the MINIMUM LATENCY PROBLEM [13] and the analysis indicated that the gain with MI local searches depends on our capacity to quickly select a large set of independent moves in a neighborhoods, which also includes the best [64]. However, the problem in finding optimal set of independent moves, given a solution of a optimization problem,



Figure 5.1: Traditional neighborhood exploration strategies.

may be very difficult (see Section 5.2).

In this chapter, we consider the problem of finding a maximum set of independent moves from a given neighborhood structure, which is defined as the MAXIMUM MULTI IMPROVEMENT PROBLEM (MMIP).

MAXIMUM MULTI IMPROVEMENT PROBLEM **Instance**: An instance I of an optimization problem Π ; a feasible solution P of I; an evaluation function f according to Π ; and a neighborhood operator ψ . **Goal**: Determine a set S of independent moves according to ψ to be applied to P in order to obtain a feasible solution P' that maximizes f(P').

5.1 Preliminaries

Given a neighborhood operator ψ and a feasible solution X, we denote $\psi(X) = \{X'_1 = m_1(X), X'_2 = m_2(X), ...\}$ as the neighborhood¹ of X, where each m_i is a move. The First Improvement strategy selects the first neighbor X'_i (minimum value of i) that offers improvement over current solution X (see Figure 5.1a). On the other hand, the Best Improvement verifies all neighbors $\psi(X)$ and selects the one that promotes the greatest gain (see Figure 5.1b). Intuitively, FI strategy is faster (ignoring worst-case where the only 'improving neighbor' is last), but FI iterated descent converges 'slowly' (smaller improvements).

Nevertheless, each BI strategy tends to achieve a faster convergence (bigger improvements), but it is necessary to consider the cost to search all neighbors. This hunch was confronted in literature [39] when BI and FI were tested inside a descent local search, for two scenarios: (1) starting with a greedy initial solution, where BI (Steepest Descent Hill

¹Neighborhood is a set of solutions (neighbors) that can be reached in one move (2-Opt, 3-Opt, OrOpt, Swap, etc.) on actual solution.

Climbing) has more successful; and (2) starting with a random solution, where FI iterated descent was victorious.

In previous studies, concepts of independence and selection of moves based on dynamic programming (dynasearch) for iterative improvement were discussed (see [20, 60]). In the present chapter, we consider a broader concept of independence between moves: Two moves m_i and m_j are said to be *independent* (or, $m_i || m_j$) if the effect of these moves, simultaneously (in parallel), over the current solution have the same effect as these moves sequentially.

In other words, we say that m_i and m_j are independent, iff one does not interfere in the calculation of the gain of each other. This is a fundamental difference from previous works in literature, since this enables the development of much more efficient MI algorithms, by using concurrent data structures to handle the application of independent moves.



Figure 5.2: Multi Improvement: two (or more) improving moves are used simultaneously (if both are independent)

Implementations of the BI strategy usually verify all possible moves to discover the best of all. If all the information about these moves are already known, why do not use them to apply several good moves instead use only one move? This question inspired the adoption of Multi Improvement strategies, consisting on selecting some independent moves simultaneously (Figure 5.2).

5.2 The Maximum Multi Improvement Problem

For convenience, MAXIMUM MULTI IMPROVEMENT PROBLEM will be expressed as following:

• Based on an instance I for an optimization problem Π , an objective function f, consider a given feasible solution P;

- Let $M = \{m_1, m_2, ...\}$ be the set of moves given by a neighborhood operator ψ , such that neighborhood $\psi(P)$ is defined as $\psi(P) = \{P'_1 = m_1(P), P'_2 = m_2(P), ...\};$
- Let g(m', P), a function that returns the gain/loss when m' ∈ M is applied on P, i.e. g(m', P) = f(m'(P)) − f(P). The function g will be redefined for each operator ψ in Section 5.3 for the purpose of simplifying the algorithms;
- Let $\sigma(m_i, m_j)$, a function that returns 1 if $\{m_i, m_j\} \in M$ are independent of each other and returns 0, otherwise. This can be defined formally as $\sigma(m_i, m_j) = 1 \iff g(m_i, P) = g(m_i, m_j(P)) \land g(m_j, P) = g(m_j, m_i(P));$
- Consider also a decision variable x_i ∈ {0,1} that represents if a move m_i ∈ M will be selected (1, case positive)

The solution for the MMIP is a set of moves that maximizes Eq. (5.1).

$$maximize \sum_{i \in 1..|\psi(P)|} x_i \cdot g(m_i, P)$$
(5.1)

Subject to:

$$x_i + x_j \le 1$$
, if $\sigma(m_i, m_j) = 0$ (5.2)

Now, we enunciate MAXIMUM WEIGHT CLIQUE PROBLEM: Given an undirected weighted-vertex graph G = (V, E) where V is a vertex set and E is an edge set. Consider a function $w(v_i)$ that returns the weight associated to vertex $v_i \in V$. The MWCP is to find a clique C where the sum of the weights of vertex $v_i \in C$ is maximum. The MAXIMUM CLIQUE PROBLEM (proven NP-hard [34]) can be reduced to MWCP when considering graphs with unitary weights. Therefore, MWCP is also NP-hard and there is a vast literature about techniques for solving famous instances, such as DIMACS [42] and BHOSLIB [88].

Theorem 15. $MMIP \propto MWCP$

Proof. Considering instances for decision problem versions for MWCP and MMIP as described, a polynomial reduction can be defined according to following steps:

1. Each vertex $v_i \in V(G)$ is associated to move $m_i \in M$;

- 2. weight $w(v_i)$ is associated to $g(m_i, P)$;
- 3. if $(v_i, v_j) \in E(G)$, then $\sigma(m_i, m_j) = 1$, and 0 otherwise.

If an instance for MMIP is YES for any value k, the associated instance for MWCP will be YES too:

(\Rightarrow) Lets suppose a graph G = (V, E) as input for MWCP, which does not have a clique with weight k; and a derivated instance for MMIP, which has a set m^* whose accumulated gain reach k. In this case, there must exist two moves $\{m_i, m_j\} \in m^*$ such that $\sigma(m_i, m_j) = 1$, while the edge $(v_i, v_j) \notin E(G)$. That is a contradiction, because if this edge did not exist, the reduction proposed would necessarily cause $\sigma(m_i, m_j) = 0$. (\Leftarrow) Now, lets suppose a instance for MMIP does not have a set m^* with gain k and the related graph G contains a clique with weight k. So, there is at least an edge $(v_i, v_j) \in E(G)$ where $\sigma(m_i, m_j) = 0$, which is a contradiction.

Figure 5.3 shows two spanning subgraphs for a particular MMIP instance ($\Pi = \text{TSP}$ and $\psi = \text{Swap}$). In the context of a TSP route P, swap(i, j) consists in exchanging two vertex at positions i and j from the tour, i.e., elements P[i] and P[j]. When some algorithm decides to perform a swap between two cities in a tour, some others swaps moves are forbidden (due to conflicts); for example, swap(1, 2) forbids swap(2, 3) or swap(3, 7)because these moves conflict in the calculation of gain with the initial swap. On the other hand, swap(1, 2) is said to be in harmony with move swap(4, 5), since no conflict happens when applying both moves simultaneously (both are independent). Figures 5.3a and 5.3b depict a TSP tour with 8 cities, where the vertices of both spanning subgraphs represent all possible swap moves. The explicit edges show a neighborhood for swap(1, 2)(see Figure 5.3a) and swap(1, 8) (see Figure 5.3b). Thus, a multi improvement method cannot perform these moves in the same stage of the local search. The maximum weight clique in this graph consists of the optimal multi improvement with swap moves.

In the next section, we propose efficient exploration techniques based on dynamic programming for several neighborhood operators on a MI.

5.3 Dynamic Programming Multi Improvement

In this section, we will analyze 2-Opt, 3-Opt, Or-Opt-k neighborhood operators. Each operator yields several moves, which also have particular properties that forbids other moves. All neighborhood operators and examples below refers to the classic TSP.



Figure 5.3: Spanning subgraphs for swap operator. Edge represents independent moves (harmonic)

The TRAVELING SALESMAN PROBLEM [32] can be defined as follows: Given a graph G = (V, E), where each vertex $v \in V(G)$ represents a city and, for each edge $(v_i, v_j) \in E(G)$, $d(v_i, v_j)$ represents the cost to travel from v_i to v_j . A salesman needs to visit all cities only one time and to return to initial city in minimum cost. Along this chapter, we will deal/treat only Symmetric TSP (where $d(v_i, v_j) = d(v_j, v_i)$). Typically, a solution for the TSP is a permutation of n elements $v_1, v_2, ..., v_{n-1}, v_n$ that represents a single feasible tour for this problem. Thus, the cost of a solution is a sum of cost of the edges between each element and its successor on the permutation.

Before discussing the neighborhood operators, firstly, we define the k-Opt neighborhood of a given route P as all routes consisting of the removal of k edges and the insertion of k new edges, such that:

$$\forall i \in 1 \dots k - 1, i \operatorname{Opt}(P) \cap k \operatorname{Opt}(P) = \emptyset$$

We also define 1-Opt as an empty neighborhood (remove/add single edge).

5.3.1 2-Opt Neighborhood

The first neighborhood structure analyzed was based on the classic 2-Opt move. In the 2-Opt, two edges are removed from solution and another two edges are included. Figure 5.4 shows a simple example for a 2-Opt between 2 and 5.



Figure 5.4: Effect of 2Opt(2,5) in a graph

In other words, 2-Opt moves for TSP can be treated through the inversion in a range inside vector that represents a solution for this problem. The idea behind this inversion inspired a dynamic programming for 2-Opt neighborhood as follows.

Dynamic Programming strategy:

- Consider neighborhood operator $\psi_1 = 2$ -Opt and the function $g_{\psi_1}(i, j)$, which returns gain (or loss) in a 2-opt move between v_i and v_j in a TSP solution: $g_{\psi_1}(i, j) = -d(v_{i-1}, v_i) - d(v_j, v_{j+1}) + d(v_{i-1}, v_j) + d(v_i, v_{j+1})$.
- We define L(a, b) as the best possible gain with a 2-opt move in the range between v_a and v_b (of route P). When two cities v_i and v_j are selected for move 2Opt(i, j), a subproblem arise: L(a, i-2) (optimal solution for the range between a and selected 2-opt move);
- The base case is L(a, b) = 0, if $a \ge b$ and the optimal Multi Improvement is given by the $max\{L(1, n - 1), L(2, n)\}$.
- Therefore, we present the recurrence in Eq. (5.3).

$$L(a,b) = \max \left\{ \begin{array}{l} L(a,b-1), \\ \max_{a \le i < b} g_{\psi_1}(i,b) + L(a,i-2) \end{array} \right\}, \text{ if } a < b.$$
(5.3)

• Recurrence in Eq. (5.3) provides a dynamic programming algorithm that costs $O(n^3)$, according to Algorithm 1 and its complexity analysis on Eq. (5.4).

The inner loop in Algorithm 1 realizes a number of steps based in a sum of aritmethic progressions always started in 1 and finished in d. It performs 1 step for d = 1, 1 + 2

Alg	gorithm 1: Memoization for 2-Opt
1 fc	or $d \leftarrow 1$ to n do
2	for $a \leftarrow 1$ to $n - d$ do
3	$b \leftarrow a + d$
4	$L[a][b] \leftarrow L[a][b-1]$
5	for $i \leftarrow a \ to \ b \ do$
6	if $(g_{\psi_1}(i,b) + L[a][i-2]) > (L[a][b])$ then
7	$L[a][b] \leftarrow g_{\psi_1}(i,b) + L[a][i-2]$

steps for d = 2, 1 + 2 + 3 steps for d = 3, etc. For *n* cities, $(1 \cdot n) + (2 \cdot (n - 1) + (3 \cdot (n - 2)) + \dots + (n - 1) \cdot 2 + n \cdot 1$, that represents a sum in 5.4.

$$\sum_{d=1}^{n} d \cdot (n - (d - 1)) = -\frac{1}{6} n \cdot (n + 1) \cdot (2n - 3n - 2) = O(n^3)$$
(5.4)

When 2Opt(i, j) move is selected, the adjacency of cities between i+2 and j-2 remains. The internal range (i+2, j-2) can provide more one subproblem (including independent moves), just being careful to manage eventual concurrent accesses on P data structure.

This leads to a more complete version of dynamic programming exploration for 2-Opt, as described by Eq. (5.5).

$$L(a,b) = \max \left\{ \begin{array}{l} L(a,b-1), \\ \max_{a \le i < b} g_{\psi_1}(i,b) + L(a,i-2) + L(i+2,b-2) \end{array} \right\}, \text{ if } a < b.$$
(5.5)

This version has the same asymptotic complexity $O(n^3)$, while considering a much larger combination of 2-Opt moves.

5.3.2 3-Opt Neighborhood

The 2-Opt move deletes two edges and inserts two new links in a tour. Naturally, a 3-Opt move removes three edges and re-connects the cycle with three new edges. However, there exists eight different forms to reconnect a cycle (for a general undirected graph). One of these forms is to reinsert the same edges that was removed, however this generates the same original solution. Other three forms can be reduced into 2-Opt moves. Thus, there are four different possible re-connections to be analyzed. In Figure 5.5a, we have a representation of a solution from TSP (tour starts in v_1 and finishes in v_n). In Figure 5.5b, three edges were chosen to be removed. In a sequence, we present (Figures 5.5c, 5.5d, 5.5e and 5.5f) four types of 3-Opt moves. Both subtours between deleted edges are called **range 1** and **range 2** in these figures. The different ways to manipulate this two ranges determines the 3-Opt types.

Dynamic Programming strategy:

- Consider the neighborhood operator $\psi_2 = 3$ -Opt, and let $g_{\psi_2}(i, j, k, t)$ be a function that calculates the gain in removing edges highlighted in red on Figure 5.4b and adding the edges highlighted in green on Figures 5.5c, 5.5d, 5.5e and 5.5f, according to specified type t.
- L(a, b) represents a optimal set of 3-Opt moves between $v_a \in v_b$. In this case, k will be fixed in function g_{ψ_2} with the value of b when i, j and t vary, thus, only one subproblem will emerge: L(a, i-2).
- The base case is L(a,b) = 0, if $a \ge b$ and the optimal Multi Improvement is given by the $max\{L(1, n-1), L(2, n)\}$.

$$L(a,b) = \max_{a \le i < j \le b} \left\{ \begin{array}{l} L(a,b-1), \\ \max_{1 \le t \le 4} g_{\psi_2}(i,j,b,t) + L(a,i-2) \end{array} \right\} \text{ if } a < b.$$
(5.6)

• Eq. 5.6 provides the memoization process present in the Algorithm 2:

```
Algorithm 2: Memoization for 3-Opt
1 for d \leftarrow 1 to n do
       for a \leftarrow 1 to n - d do
 2
           b \leftarrow = a + d
 3
           L[a][b] \leftarrow L[a][b-1]
 4
           for i \leftarrow a \ to \ b-1 \ do
 5
               for j \leftarrow i \ to \ b-1 \ do
 6
                   best = 0
 7
                   for t \leftarrow 1 to 4 do
 8
                      9
10
           if best > L[a][b] then
11
               L[a][b] \leftarrow best
12
```

The inner loop in Algorithm 2 perform a constant number of steps which do not depends on size of input. The second inner loop performs a number of steps that represents



Figure 5.5: 3-Opt possibilities

a sequence similar to Algorithm 1, but it is a sum of arithmetic progressions. For d = 1, only 1 step; for d = 2, 1 + (1 + 2); for d = 3, 1 + (1 + 2) + (1 + 2 + 3) steps; etc. For d = n, the same number of steps in Eq. (5.4) will be performed. Therefore, we have the final sum in Eq. (5.7).

$$\sum_{m=n}^{1} \sum_{d=1}^{m} d \cdot (m - (d - 1)) = O(n^4)$$
(5.7)

5.3.3 OrOpt-k Neighborhood

OrOpt-k neighborhood structure consists in a selection of k cities in the tour that will be shifted to another position in a actual solution. It is interesting to note that OrOpt-k is a specific case of 3-Opt movement, i.e., type 1 on Figure 5.5c. In OrOpt-k movement, there are no range to be reversed. Thus, in case of shift move in OrOpt-k is to right, we assume a equivalence between the range of k elements and shift size with range 2 and range 1 from 3-Opt, respectively (see Figure 5.6b). If the shift move is to left then the range of k elements corresponding to range 1 and shift size refers to range 2. In Figure 5.6 is represented a OrOpt-k shift to right.





(b) Shifting k elements to right v_{i+s}



(d) Solution after move

Figure 5.6: OrOpt-K scheme

Dynamic Programming strategy:

- Consider neighborhood operator $\psi_3 = \text{OrOpt-k.}$ According to Figure 5.6b, OrOpt-k can be reduced to type 1 from 3-Opt. In this case, the initial vertex inside the range to be shifted and shift size provide all information to a function g_{ψ_3} .
- Let $g_{\psi_3}(i, k, s)$ a function that returns a gain in a move of a range between v_i and v_{i+k} shift after v_{i+k+s} , we have a conversion from g_{ψ_3} to g_{ψ_2} in Eq (5.8).
- Let L(a, b) be the best possible gain with a Or-Opt-k move in the range between v_a and v_b (of route P).
- The base case is L(a,b) = 0, if $a \ge b$ and the optimal Multi Improvement is given by the $max\{L(1, n-1), L(2, n)\}$.

$$g_{\psi_3}(i,k,s) = g_{\psi_2}(i,i+k,i+k+s,1)$$
(5.8)

• A minor adjust provide a recurrence in Eq. (5.9).

$$L(a,b) = \max_{a \le i < b} \left\{ \begin{array}{l} L(a,b-1), \\ \max\{g_{\psi_3}(i,k,s), g_{\psi_3}(i,s,k)\} + L(a,i-2) \end{array} \right\} \text{ if } a < b.$$
(5.9)

Note that the line **6** in Algorithm 3 describe a double verification of g_{ψ_3} function. The comparison $g_{\psi_3}(i, k, s)$ and $g_{\psi_3}(i, s, k)$ guarantees that the algorithm verify as left shifts as right shifts.

Algorithm 3: Memoization for OrOpt-k
1 for $d \leftarrow 1$ to n do
2 for $a \leftarrow 1$ to $n - d$ do
$3 b \leftarrow a + d$
$4 \qquad \qquad L[a][b] \leftarrow L[a][b-1]$
5 for $i \leftarrow a \ to \ b-1 \ do$
6 if $g_{\psi_3}(i,k,s) > g_{\psi_3}(i,s,k)$ then
7 $aux \leftarrow g_{\psi_3}(i,k,s) + L[a][i-2]$
8 else
9 $ $ $aux \leftarrow g_{\psi_3}(i,s,k) + L[a][i-2]$
10 if $aux > L[a][b]$ then
11 $L[a][b] \leftarrow aux$

The asymptotic number of operations for memoization in Algorithm 3 is $O(n^3)$, the same as Algorithm 1.

5.4 Experiments and convergence

Now, we report the results of the experimental analysis of those three dynamic programmings performance against the traditional implementations of Best Improvement and First Improvement. The main focus in this analysis is to verify the impact of MMIPsolvability in a major problem like TSP and how the convergence of MI-inspired techniques can be explored in practical solutions.

In our tests, we write classic versions of BI and FI along with the dynamic programming techniques. Both were implemented on C++11 language. A set of ten instances of different sizes from TSPLIB [62] was selected. The performance of all local searches was evaluated for FI-2Opt, BI-2Opt and MI-2Opt.

The experiments were conducted in an environment equipped with Ubuntu 18.04 LTS 64-bit on a CPU Intel O CoreTMi5-7300HQ CPU at 2.50GHz with 8 GB RAM. All local searches were run 1000 times for every ten instances. Each run only stopped when local optimum were found. In Table 5.1, the values for **avg**, **best** and **#wins** refers to (i) average of final solution costs, (ii) the best solution cost founded and (iii) the number of times when the local search has reached the best solution of a run, respectively. Note that the sum of wins of the three local searches can be higher than 1000 because ties increased the number of wins of all local searches that achieved the best cost of the solution. On Table 5.2, the column **Opt.** indicates optimal value for the instance; columns **gapAvg** and **gap** present a distance between solutions found by MI-2Opt and optimal solutions for selected instances. Table 5.3, the column **nIter** shows the range for minimum and maximum number of iterations; column **AvgIter** refers to average time of all runs in milliseconds.

On Table 5.1, we can see that while the FI local search achieves most of the best results, MI also gets a reasonable amount of wins. Asymptotically, a traditional implementation of best and first improvement local searches, clearly spend less time than multi improvement sequential implementation on CPU. We remember that is a comparison of $O(n^2)$ technique versus $O(n^3)$ dynamic programming (Algorithm 1). As expected, Table 5.3 shows that the average time to perform the dynamic programings was bigger than BI and FI, however,

					DIAO	1				
Instances		F1-20pt	,		BI-20p	C	MI-2Opt			
mstances	avg	\mathbf{best}	$\# \mathbf{wins}$	avg	\mathbf{best}	$\#\mathbf{wins}$	avg	\mathbf{best}	$\# \mathbf{wins}$	
berlin52	8156	7542	320	8123	7542	360	8155	7542	321	
eil76	580	548	487	585	555	305	587	552	244	
eil101	675	643	612	685	648	209	686	651	211	
bier127	126066	119256	377	126032	119774	383	127140	119813	241	
kroA150	28235	26903	585	28601	27275	272	28902	27245	143	
rat195	2512	2396	864	2600	2456	71	2603	2463	67	
ts225	133092	127749	611	136406	127147	164	135640	127488	226	
gil262	2569	2449	645	2605	2482	234	2625	2492	126	
a280	2823	2648	691	2891	2708	153	2894	2696	164	
lin318	45032	43275	533	45372	43493	313	45723	43530	154	

Table 5.1: Comparison between BI, FI and MI for 2Opt neighborhood.

Instances	Ont	MI-2Opt								
Instances	Opt.	avg	\mathbf{best}	$\mathbf{gapAvg}(\%)$	$\mathbf{gap}(\%)$					
berlin52	7542	8155	7542	7.52	0					
eil76	538	587	552	8.35	2.54					
eil101	629	686	651	8.31	3.38					
bier127	118282	127140	119813	6.97	1.28					
kroA150	26524	28902	27245	8.23	2.65					
rat195	2323	2603	2463	10.76	5.68					
ts225	126643	135640	127488	6.63	0.66					
gil262	2378	2625	2492	9.41	4.57					
a280	2579	2894	2696	10.88	4.34					
lin318	42029	45723	43530	8.08	3.45					

Table 5.2: Comparison between MI-2Opt and optimal solutions.

the dynamic programmings has reached the local optimum in fewer search iterations and with slight differences between the minimum and the maximum number of iterations.

Figure 5.7 shows the behavior of descent of BI and FI in our first run. Note that First Improvement has a chaotic convergence, while BI descends more neatly, but without necessarily a pattern to follow. It is worth noting that BI local search reach some improvement-peaks at the last iterations, meanwhile, FI have some of the bests peaks precisely at the end. That affects the possibility to predict the status of descent in a certain stage. After several runs, we realize some standards about the convergence of MI. Such thoughts concern the stability of these local searches. For example, in large instances for TSP, the choice of stable methods is a very powerful tool. In practical situations, where time to find the local optimum may be prohibitive, selecting an partial solution may be the most praticable option.

Instances	Ι	FI-2Opt		BI-2Opt		MI-2Opt			
Instances	nIter	AvgIter	time	nIter	AvgIter	time	nIter	AvgIter	time
berlin52	99-226	160.7	8.2	36-60	46.1	8.8	7-17	10.5	41.4
eil76	189-316	240.1	27.5	55-79	67.5	27.9	8-17	11.0	135.3
eil101	269-434	346.4	68.5	77 - 105	90.5	65.6	9-20	12.2	342.9
bier127	426-738	566.1	166.5	111 - 150	128.9	148.0	12 - 24	15.4	840.6
kroA150	563 - 789	675.0	326.2	181 - 272	162.8	249.1	12-22	14.9	1287.4
rat195	754 - 1005	876.2	725.5	183 - 228	201.8	528.2	12-22	16.1	3096.5
ts225	882-1140	1010.3	1011.5	136-288	250.2	966.9	16-34	20.5	6272.6
gil262	1163 - 1468	1304.8	1766.4	267 - 317	292.2	1382.1	15 - 26	18.7	8613.4
a280	1219 - 1553	1379.9	2297.9	285 - 345	308.0	1743.0	15 - 28	19.4	11327.3
lin318	1580-2019	1784.6	3776.7	355 - 409	381.2	2709.0	16-28	19.7	16643.2

Table 5.3: Performance of local searches.



Figure 5.7: Convergence of 2-Opt for berlin52.

A stable method, supposedly, must guarantee a certain estimated gain in function of a number of steps. From Figure 5.8, we can see that the curve of the descent of MI in the analyzed instance obtained a desired behavior. Figure 5.8 plots the behavior of five random descent of MI with 2-Opt applied to berlin52 and lin318 instances. However, a thorough analysis should be done to verify if this behavior is intrinsic to MI or if it is just an isolated phenomenon. Using the data from the 1,000 runs, Table 5.4 have strong evidence that is attesting to the robust stability of the use of the search for maximum multi improvement at each local search step as strategy (i.e., solving the MMIP at each iteration). Based on the plots of Figure 5.8, we can see that the evolution of the gain over gap by iteration has a very homogeneous behavior, even with the MI search starting with random initial solution. Thus, the idea of making curve fitting that determines the best model function for the observed behavior arises.



Figure 5.8: Convergence of MI-2Opt in five runs on berlin52 and lin318.

Curve Fitting is a roll of methods which consists in finding a curve that has the best fit with a data sequence (points, coordinates, measures). Through a curve fit approach, it is possible to infer the behavior of the descent of MI using the data about gain over the possible gap by iteration of local searches. The plots in Figure 5.8 suggest the adoption of some exponential model functions to find the best fit. With the help of Optim package for GNU Octave 4.2.2, we submit the collected data to four model functions of descending exponential: (i) Generic exponential curve $f(x) = ab^{-x}$, (ii) basic exponential curve $f(x) = ae^{-bx}$, (iii) power curve $f(x) = ax^{-b}$, and (iv) half-life curve $f(x) = a \cdot 2^{\frac{-x}{b}}$.

Table 5.4 brings the average of coefficients a and b from described models (founded by Least-Square Methods). The table also presents the *root-mean-square error* (rmse)² and *R-squared*³. The table do not contains the fit returned by $f(x) = ab^{-x}$ because the R-squared for the curve fit in this model function was less than 0.6 (It is much worse than the curves).

 $^{^{2}}$ Root Mean Square is helpful measure mainly used in analysis of alternating electrical current. In predictions and estimation, RMS error represents the imperfection of the fit. The fit is more 'perfect' when RMS error is closer to zero.

 $^{{}^{3} \}mathbf{R}^{2}$ or **R-squared** provides a measure of how reliable is the fit. The value of R-squared is inside the range between 0 and 1. Bests fits returns R-squared more closer to 1.

Instances	(basic exponential) $f(x) = ae^{-bx}$				(power curve) $f(x) = ax^{-b}$				(half life) $f(x) = a \cdot 2^{\frac{-x}{b}}$			
	a	b	rmse	R-squared	a	b	rmse	R-squared	a	b	rmse	R-squared
berlin52	0.799	0.597	0.0120	0.990	0.456	1.283	0.022	0.969	0.800	1.171	0.012	0.990
pr76	0.757	0.580	0.011	0.990	0.442	1.261	0.187	0.976	0.757	1.201	0.011	0.990
eil101	0.745	0.577	0.0108	0.991	0.438	1.267	0.0172	0.978	0.745	1.205	0.109	0.990
bier127	0.647	0.520	0.009	0.991	0.407	1.216	0.016	0.973	0.646	1.335	0.009	0.989
kroA150	0.832	0.645	0.012	0.988	0.455	1.353	0.0120	0.988	0.833	1.076	0.012	0.988
rat195	0.855	0.664	0.011	0.988	0.457	1.373	0.010	0.990	0.855	1.046	0.011	0.988
ts225	0.652	0.540	0.009	0.989	0.403	1.247	0.012	0.981	0.652	1.285	0.009	0.989
gil262	0.678	0.553	0.009	0.990	0.412	1.259	0.012	0.982	0.678	1.252	0.092	0.990
a280	0.780	0.622	0.0102	0.989	0.438	1.335	0.010	0.988	0.780	1.114	0.010	0.989
lin318	0.685	0.559	0.009	0.990	0.413	1.268	0.012	0.983	0.685	1.239	0.009	0.990

Table 5.4: Curve Fitting about MI-2Opt convergence.

The data from Table 5.4 confirms our guess. It is possible to notice that the three selected functions have a great fit with the behavior of Multi Improvement, as the little variation of the coefficients between different instances. In other words, the curve that adjusts the descent in the smaller instances is pretty much the same as the one that adjusts the bigger instances. Besides, the 1000 curves generated for each instance showed little variation among themselves and few outliers.



Figure 5.9: Curve fittings for berlin52 and lin318.

Plots in Figure 5.9 exemplify two curves drawn with the average of coefficients present in Table 5.4. The point marks belongs to first run of MI-2Opt over the such instances, even so the 'average curve' get a good fit with the points.

5.5 Remarks

This chapter presented a formal model of the MAXIMUM MULTI IMPROVEMENT PROBLEM, which gives the optimal combination of moves in a Multi Improvement neighborhood exploration. Although the First and Best Improvement strategies have been widely studied on literature, little work has been devoted to exact move composition. Modern parallel computing architectures (such as GPUs) have motivated the design of novel neighborhood exploration techniques, in order to build faster and wider local searches, and evidence has suggested that MI strategies may converge faster than other classical techniques. Most of the existing algorithms based on MI use heuristic strategies for selecting independent moves, leaving space the development for novel exact approaches. This also increases the interest on developing ways to perform the exploration at maximum $O(n^2)$ (case of BI and FI for Swap and 2-Opt neighborhoods) or $O(n^3)$ (in the case of 3-Opt). In this sense, our results contributes to the literature by providing three different Dynamic Programming implementations for MI local searches on the Traveling Salesman Problem.

The MMIP was modeled as a graph class to be solved as a MWCP, what does not discourage the study of MI, as it further establishes a solid base to create novel efficient algorithms. Additionally, through neighborhood structure analysis on TSP it was possible to discover some instance classes for the MMIP that can be solved in polynomial time. Now, we have an open research field for future works involving to find algorithms less complex for 2-Opt, 3-Opt and OrOpt-k as well as to analyze the behavior of MMIP in other permutation structures (like the Swap).

Experiments attested that, unlike FI and BI, MI has good stability, a fact that guarantees the knowledge/estimation of the cost of the partial solution at a given step of the search; some advantages emerge from this perspective:

- MI have a massive improvement at beggining of local search;
- The curves of MI tends to have not improvement-peaks at the ending of the search;
- With basic exponential and half-life exponential model functions, it was possible to find a fit with R-squared of 0.99.

Moreover, the similarity between MMIP and MWCP allow us to adapt fast techniques already tested on MWCP with the goal of performing sub-optimal multi improvements in a short time. The development of MI-inspired techniques may allow the creation of faster local searches, since the MI is highly suited for parallel computing architectures. On the other hand, the exact exploration of many neighborhood operators may not be feasible, as we suspect that some of them may not have exact polynomial solutions. Finally, future researches can address the need to creating general frameworks for MI exploration of neighborhood structures.

Chapter 6

Conclusion

In this thesis, we investigate several aspects of computational problems related to circuits and neighborhood exploration. This work presents (i) polynomial and parameterized reductions, (ii) NP-completeness proofs, (iii) classical and parameterized complexity analysis, (iv) implementations of exact algorithms and metaheuristics, and even (v) experiments. Supported by a vast literature, we explore notable trends in algorithms, optimization, and complexity; and we provide some results for each topic discussed. Next, we present a short summary of our achievements:

- Succinct Certification We introduce the notion of *certification-width* of a monotone Boolean circuit, a complexity measure that intuitively quantifies the minimum number of edges that need to be traversed by a minimal set of positive weight inputs in order to certify that a given circuit is satisfied. We call the problem of computing this invariant as SUCCINCT MONOTONE CIRCUIT CERTIFICATION (SMCC). We prove that SMCC is NP-complete even when the input monotone circuit is planar. Subsequently, we show that k-SMCC, the problem parameterized by the size of the solution, is W[1]-hard, but still in W[P]. In contrast, we show that k-SMCC is fixed-parameter tractable when restricted to monotone circuits of bounded genus.
- Energy Complexity We discuss the time complexity needed to compute the best case energy complexity among satisfying assignments of a monotone Boolean circuit, and we call such a problem as MINEC_M^+ . We prove that MINEC_M^+ is NP-complete for planar circuits. In addition, we show that the problem is W[1]-hard but in XP when parameterized by the size of the solution. Afterall, we show that MINEC_M^+ on bounded genus circuits is FPT.
- Th-hierarchy We introduce the *Th-hierarchy*, a natural generalization of the W-

hierarchy defined by weighted *threshold* circuit satisfiability problems. Investigating the relationship between Th-hierarchy and W-hierarchy, we discuss the complexity of transforming Threshold circuits into Boolean circuits, and observe that sorting networks are powerful tools to handle such transformations. First, we show that these hierarchies collapse at the last level (W[P]=Th[P]). After that, we present a time complexity analysis of an AKS sorting network construction, which supports some of our results. Finally, we prove that Th[t] \subseteq W[SAT] for every $t \in \mathbb{N}$.

• Multi Improvement - We propose a formal description for the MAXIMUM MULTI IMPROVEMENT PROBLEM (MMIP), as a theoretical background for the MI. Moreover, we develop three dynamic programming algorithms for solving the MMIP, given a solution tour for a TRAVELING SALESMAN PROBLEM and neighborhood operators 2-Opt, 3-Opt and OrOpt-k. The analysis suggest the rise of a new open topic focused on developing novel efficient neighborhood searches.

During the doctorate process, many possibilities emerged from each seminar, class, or lecture. Since the beginning of this thesis writing, one idea has been clear: To avoid trivialities. With this purpose, our four projects detailed in previous chapters aimed at the proposition of unexplored concepts. The problems SUCCINCT MONOTONE CIRCUIT CERTIFICATION, BEST-CASE ENERGY COMPLEXITY and MAXIMUM MULTI IMPROVE-MENT are unprecedented (likewise the concept of Th-hierarchy). It is always risky to describe novel problems, measures, and concepts. It is impossible to predict the reception of the scientific community when it is faced with non-addressed problems. However, all concerns about the novelty of our projects were outdated. Besides, all those works receive positive answers from relevant journals and conferences.

References

- [1] AJTAI, M., KOMLÓS, J., SZEMERÉDI, E. An 0 (n log n) sorting network. In Proceedings of the fifteenth annual ACM symposium on Theory of computing (1983), p. 1–9.
- [2] ALLENDER, E., KOUCKY, M. Amplifying lower bounds by means of self-reducibility. Journal of the ACM (JACM) 57, 3 (2010), 1–36.
- [3] ALON, N. Eigenvalues and expanders. Combinatorica 6, 2 (1986), 83–96.
- [4] ALVES, M. R., DE OLIVEIRA OLIVEIRA, M., SILVA, J. C. N., DOS SANTOS SOUZA, U. Succinct monotone circuit certification: Planarity and parameterized complexity. In *International Computing and Combinatorics Conference* (2020), Springer, p. 496– 507.
- [5] ALVES, M. R., DE OLIVEIRA OLIVEIRA, M., SILVA, J. C. N., DOS SANTOS SOUZA, U. Succinct certification of monotone circuits. *Theoretical Computer Science 889* (2021), 1–13.
- [6] ANTONIADIS, A., BARCELO, N., NUGENT, M., PRUHS, K., SCQUIZZATO, M. Energy-efficient circuit design. In Proceedings of the 5th conference on Innovations in theoretical computer science (2014), p. 303–312.
- [7] ARAUJO, R. P. Strategies for neighborhood exploration with gpu for optimization problems (in portuguese). Master Thesis (University of the State of Rio de Janeiro), 2018.
- [8] BADDAR, S. W. A.-H., BATCHER, K. E. The aks sorting network. In *Designing Sorting Networks*. Springer, 2011, p. 73–80.
- [9] BARCELO, N., NUGENT, M., PRUHS, K., SCQUIZZATO, M. Almost all functions require exponential energy. In *International Symposium on Mathematical Foundations* of Computer Science (2015), Springer, p. 90–101.
- [10] BARRINGTON, D. A. M., CHI-JEN LU, MILTERSEN, P. B., SKYUM, S. On monotone planar circuits. In Proceedings. Fourteenth Annual IEEE Conference on Computational Complexity (1999), p. 24–31.
- [11] BATCHER, K. E. Sorting networks and their applications. In Proceedings of the April 30-May 2, 1968, spring joint computer conference (1968), p. 307-314.
- [12] BLAKE, C. G., KSCHISCHANG, F. R. On the vlsi energy complexity of ldpc decoder circuits. *IEEE Transactions on Information Theory* 63, 5 (2017), 2781–2795.

- [13] BLUM, A., CHALASANI, P., COPPERSMITH, D., PULLEYBLANK, B., RAGHAVAN, P., SUDAN, M. The minimum latency problem. In *Proceedings of the twenty-sixth* annual ACM symposium on Theory of computing (1994), p. 163–171.
- [14] BODLAENDER, H. L., DRANGE, P. G., DREGI, M. S., FOMIN, F. V., LOKSH-TANOV, D., PILIPCZUK, M. A c^kn 5-approximation algorithm for treewidth. SIAM Journal on Computing 45, 2 (2016), 317–378.
- [15] CAI, L., FELLOWS, M., JUEDES, D., ROSAMOND, F. The complexity of polynomialtime approximation. *Theory of Computing Systems* 41, 3 (2007), 459–477.
- [16] CAUSSINUS, H., MCKENZIE, P., THÉRIEN, D., VOLLMER, H. Nondeterministicnc1computation. Journal of Computer and System Sciences 57, 2 (1998), 200–212.
- [17] CHANG, S.-C., VAN GINNEKEN, L. P., MAREK-SADOWSKA, M. Fast boolean optimization by rewiring. In *Proceedings of International Conference on Computer Aided Design* (1996), IEEE, p. 262–269.
- [18] CHKLOVSKII, D. B., SCHIKORSKI, T., STEVENS, C. F. Wiring optimization in cortical circuits. *Neuron* 34, 3 (2002), 341–347.
- [19] CIEZA, E., TEYLO, L., FROTA, Y., BENTES, C., DRUMMOND, L. M. A. A gpu-based metaheuristic for workflow scheduling on clouds. In *High Performance Computing for Computational Science – VECPAR 2018* (Cham, 2019), H. Senger, O. Marques, R. Garcia, T. Pinheiro de Brito, R. Iope, S. Stanzani, and V. Gil-Costa, Eds., Springer International Publishing, p. 62–76.
- [20] CONGRAM, R. K., POTTS, C. N., VAN DE VELDE, S. L. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *IN-FORMS Journal on Computing* 14, 1 (2002), 52–67.
- [21] CREIGNOU, N., VOLLMER, H. Parameterized complexity of weighted satisfiability problems: Decision, enumeration, counting. *Fundamenta Informaticae* 136, 4 (2015), 297–316.
- [22] CYGAN, M., FOMIN, F. V., KOWALIK, L., LOKSHTANOV, D., MARX, D., PILIPCZUK, M., PILIPCZUK, M., SAURABH, S. Parameterized Algorithms. Springer, 2015.
- [23] DEMAINE, E. D., HAJIAGHAYI, M. T. Diameter and treewidth in minor-closed graph families, revisited. Algorithmica 40, 3 (2004), 211–215.
- [24] DEVADAS, S., MALIK, S. A survey of optimization techniques targeting low power vlsi circuits. In Proceedings of the 32nd annual ACM/IEEE Design Automation Conference (1995), p. 242–247.
- [25] DINESH, K., OTIV, S., SARMA, J. New bounds for energy complexity of boolean functions. *Theoretical Computer Science* (2020).
- [26] DOKEROGLU, T., SEVINC, E., KUCUKYILMAZ, T., COSAR, A. A survey on new generation metaheuristic algorithms. *Computers & Industrial Engineering* 137 (2019), 106040.

- [27] DORZWEILER, O., FLAMM, T., KREBS, A., LUDWIG, M. Positive and negative proofs for circuits and branching programs. *Theoretical Computer Science* 610 (2016), 24–36.
- [28] DOWNEY, R. G., FELLOWS, M. R. *Parameterized complexity*. Springer Science & Business Media, 2012.
- [29] EPPSTEIN, D. Diameter and treewidth in minor-closed graph families. Algorithmica 27, 3-4 (2000), 275–291.
- [30] FELLOWS, M., FLUM, J., HERMELIN, D., MÜLLER, M., ROSAMOND, F. Whierarchies defined by symmetric gates. *Theory of Computing Systems* 46, 2 (2010), 311–339.
- [31] FELLOWS, M. R., HERMELIN, D., ROSAMOND, F. A., VIALETTE, S. On the parameterized complexity of multiple-interval graph problems. *Theoretical Computer Science* 410, 1 (2009), 53–61.
- [32] FLOOD, M. M. The traveling-salesman problem. Operations research 4, 1 (1956), 61–75.
- [33] FOMIN, F. V., GOLOVACH, P., THILIKOS, D. M. Contraction obstructions for treewidth. Journal of Combinatorial Theory, Series B 101, 5 (2011), 302–314.
- [34] GAREY, M. R., JOHNSON, D. S. Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA, 1979.
- [35] GLOVER, F., LAGUNA, M. Tabu search. In Handbook of combinatorial optimization. Springer, 1998, p. 2093–2229.
- [36] GROSS, J. L., TUCKER, T. W. Topological graph theory. Courier Corporation, 2001.
- [37] GU, Q.-P., TAMAKI, H. Improved bounds on the planar branchwidth with respect to the largest grid minor size. *Algorithmica* 64, 3 (2012), 416–453.
- [38] HAJNAL, A., MAASS, W., PUDLÁK, P., SZEGEDY, M., TURÁN, G. Threshold circuits of bounded depth. *Journal of Computer and System Sciences* 46, 2 (1993), 129–154.
- [39] HANSEN, P., MLADENOVIĆ, N. First vs. best improvement: An empirical study. Discrete Applied Mathematics 154, 5 (2006), 802–817.
- [40] HASTAD, J. Almost optimal lower bounds for small depth circuits. In *Proceedings* of the eighteenth annual ACM symposium on Theory of computing (1986), p. 6–20.
- [41] HUSSAIN, K., SALLEH, M. N. M., CHENG, S., SHI, Y. Metaheuristic research: a comprehensive survey. Artificial Intelligence Review 52, 4 (2019), 2191–2233.
- [42] JOHNSON, D. S., TRICK, M. A. Cliques, coloring, and satisfiability: second DI-MACS implementation challenge, October 11-13, 1993, vol. 26. American Mathematical Soc., 1996.

- [43] KANJ, I., THILIKOS, D. M., XIA, G. On the parameterized complexity of monotone and antimonotone weighted circuit satisfiability. *Information and Computation* 257 (2017), 139–156.
- [44] KHANNA, S., MOTWANI, R. Towards a syntactic characterization of PTAS. In Proceedings of the twenty-eighth annual ACM symposium on Theory of computing (1996), p. 329–337.
- [45] KISSIN, G. Measuring energy consumption in vlsi circuits: A foundation. In Proceedings of the fourteenth annual ACM symposium on Theory of computing (1982), p. 99–104.
- [46] LENNIE, P. The cost of cortical computation. *Current biology* 13, 6 (2003), 493–497.
- [47] LIGHTNER, M., DIRECTOR, S. Multiple criterion optimization for the design of electronic circuits. *IEEE Transactions on Circuits and Systems 28*, 3 (1981), 169– 179.
- [48] LIMAYE, N., MAHAJAN, M., SARMA, J. M. Upper bounds for monotone planar circuit value and variants. *Computational Complexity* 18, 3 (2009), 377.
- [49] LOURENÇO, H. R., MARTIN, O. C., STÜTZLE, T. Iterated local search. In Handbook of metaheuristics. Springer, 2003, p. 320–353.
- [50] LUBOTZKY, A., PHILLIPS, R., SARNAK, P. Explicit expanders and the ramanujan conjectures. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing* (1986), p. 240–246.
- [51] LUBOTZKY, A., PHILLIPS, R., SARNAK, P. Ramanujan graphs. Combinatorica 8, 3 (1988), 261–277.
- [52] MARX, D. Completely inapproximable monotone and antimonotone parameterized problems. Journal of Computer and System Sciences 79, 1 (2013), 144–151.
- [53] MCKENZIE, P., REINHARDT, K., VINAY, V. Circuits and context-free languages. In International Computing and Combinatorics Conference (1999), Springer, p. 194–203.
- [54] MCKENZIE, P., VOLLMER, H., WAGNER, K. W. Arithmetic circuits and polynomial replacement systems. SIAM Journal on Computing 33, 6 (2004), 1513–1531.
- [55] MLADENOVIĆ, N., HANSEN, P. Variable neighborhood search. Computers & operations research 24, 11 (1997), 1097–1100.
- [56] NILSSON, N. J. Problem-solving methods in artificial intelligence. McGraw-Hill computer science series. McGraw-Hill, 1971.
- [57] O'CONNOR, D. G., NELSON, R. J. Sorting system with nu-line sorting switch, abril de 10 1962. US Patent 3,029,413.

- [58] PARANHOS, R. M., NASCIMENTO SILVA, J. C., SOUZA, U. S., OCHI, L. S. Parameterized complexity classes defined by threshold circuits: Using sorting networks to show collapses with w-hierarchy classes. In *Combinatorial Optimization and Applications* (Cham, 2021), D.-Z. Du, D. Du, C. Wu, and D. Xu, Eds., Springer International Publishing, p. 348–363.
- [59] PATERSON, M. S. Improved sorting networks with o (log n) depth. Algorithmica 5, 1 (1990), 75–92.
- [60] POTTS, C. N. Dynasearch-iterative local improvement by dynamic programming. Internal technical Report LPOM-95-11, 1995.
- [61] RAZBOROV, A. A. Lower bounds on the size of bounded depth circuits over a complete basis with logical addition. *Mathematical Notes of the Academy of Sciences* of the USSR 41, 4 (1987), 333–338.
- [62] REINELT, G. Tsplib a traveling salesman problem library. ORSA journal on computing 3, 4 (1991), 376–384.
- [63] RIOS, E., COELHO, I. M., OCHI, L. S., BOERES, C., FARIAS, R. A benchmark on multi improvement neighborhood search strategies in cpu/gpu systems. In 2016 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW) (2016), IEEE, p. 49–54.
- [64] RIOS, E., OCHI, L. S., BOERES, C., COELHO, V. N., COELHO, I. M., FARIAS, R. Exploring parallel multi-gpu local search strategies in a metaheuristic framework. *Journal of Parallel and Distributed Computing* 111 (2018), 39–55.
- [65] ROBERTSON, N., SEYMOUR, P., THOMAS, R. Quickly excluding a planar graph. Journal of Combinatorial Theory, Series B 62, 2 (1994), 323–348.
- [66] ROBERTSON, N., SEYMOUR, P. D. Graph minors. III. Planar tree-width. Journal of Combinatorial Theory, Series B 36, 1 (1984), 49–64.
- [67] RUZZO, W. L. Tree-size bounded alternation. Journal of Computer and System Sciences 21, 2 (1980), 218–235.
- [68] SAVAGE, J. E. Planar circuit complexity and the performance of VLSI algorithms+. In VLSI Systems and Computations. Springer, 1981, p. 61–68.
- [69] SILVA, J. C. N., COELHO, I. M., SOUZA, U. S., OCHI, L. S., COELHO, V. N. Finding the maximum multi improvement on neighborhood exploration. *Optimiza*tion Letters (2020), 1–19.
- [70] SILVA, J. C. N., SOUZA, U. S., OCHI, L. S. Energy complexity of satisfying assignments in monotone circuits: On the complexity of computing the best case. In *Algorithmic Aspects in Information and Management* (Cham, 2021), W. Wu and H. Du, Eds., Springer International Publishing, p. 380–391.
- [71] SÍMA, J. Energy complexity of recurrent neural networks. Neural Computation 26, 5 (2014), 953–973.

- [72] SÖRENSEN, K., SEVAUX, M., GLOVER, F. A History of Metaheuristics. Springer International Publishing, Cham, 2018, p. 1–18.
- [73] SOROKER, D. Fast parallel algorithms for finding hamiltonian paths and cycles in a tournament. Journal of Algorithms 9, 2 (1988), 276–286.
- [74] SOUZA, U. S., PROTTI, F. Tractability, hardness, and kernelization lower bound for and/or graph solution. Discrete Applied Mathematics 232 (2017), 125 – 133.
- [75] SOUZA, U. S., PROTTI, F., DA SILVA, M. D. Revisiting the complexity of and/or graph solution. Journal of Computer and System Sciences 79, 7 (2013), 1156 – 1163.
- [76] SZEIDER, S. On fixed-parameter tractable parameterizations of SAT. In International Conference on Theory and Applications of Satisfiability Testing (2003), Springer, p. 188–202.
- [77] TALBI, E.-G. Metaheuristics: from design to implementation, vol. 74. John Wiley & Sons, 2009.
- [78] TANNER, R. M. Explicit concentrators from generalized n-gons. SIAM Journal on Algebraic Discrete Methods 5, 3 (1984), 287–293.
- [79] TURÁN, G. On the complexity of planar boolean circuits. Computational Complexity 5, 1 (1995), 24–42.
- [80] UCHIZAWA, K., DOUGLAS, R., MAASS, W. On the computational power of threshold circuits with sparse activity. *Neural Computation* 18, 12 (2006), 2994–3008.
- [81] VAINTSVAIG, M. N. On the power of networks of functional elements. In *Doklady Akademii Nauk* (1961), vol. 139-2, Russian Academy of Sciences, p. 320–323.
- [82] VENKATESWARAN, H. Circuit definitions of nondeterministic complexity classes. SIAM Journal on Computing 21, 4 (1992), 655–670.
- [83] VENKATESWARAN, H., TOMPA, M. A new pebble game that characterizes parallel complexity classes. SIAM Journal on Computing 18, 3 (1989), 533–549.
- [84] VERHOEVEN, M., SEVERENS, M. Parallel local search for steiner trees in graphs. Annals of Operations Research 90, 0 (Jan 1999), 185–202.
- [85] WATSON, J.-P., BECK, J. C., HOWE, A. E., WHITLEY, L. D. Problem difficulty for tabu search in job-shop scheduling. *Artificial intelligence* 143, 2 (2003), 189–217.
- [86] WILLIAMS, R. Nonuniform acc circuit lower bounds. Journal of the ACM (JACM) 61, 1 (2014), 1–32.
- [87] XIE, H. Studies on sorting networks and expanders. Tese de Doutorado, Ohio University, 1998.
- [88] XU, K. Bhoslib: Benchmarks with hidden optimum solutions for graph problems (maximum clique, maximum independent set, minimum vertex cover and vertex coloring)-hiding exact solutions in random graphs. web site. Web site, http://www. nlsde. buaa. edu. en/~ kexu/benchmarks/graphbenchmarks. htm (2004).