Positive and Negative Results in Approximation and Parameterized Complexity

Edouard Bonnet

UNIVERSITÉ PARIS-DAUPHINE — ÉCOLE DOCTORALE DE DAUPHINE
LAMSADE

RÉSULTATS POSITIFS ET NÉGATIFS EN APPROXIMATION ET COMPLEXITÉ PARAMÉTRÉE

ÉDOUARD BONNET

MANUSCRIT POUR L'OBTENTION D'UN DOCTORAT ÈS SCIENCES

SPÉCIALITÉ INFORMATIQUE

20 novembre 2014

Henning FERNAU ..... Rapporteur

Klaus JANSEN ..... Rapporteur

Christoph DÜRR ..... Membre du jury

Bruno ESCOFFIER ..... Membre du jury

Dieter KRATSCH ..... Membre du jury

Claire MATHIEU ..... Membre du jury

Vangelis Th. PASCHOS ..... Directeur de thèse

Université Paris-Dauphine
Place du Maréchal de Lattre de Tassigny
75116 Paris, France

Abstract

Several real-life problems are **NP**-hard. This means that, as it is widely believed that $\mathbf{P} \neq \mathbf{NP}$, they cannot be solved in polynomial time. The two main options to overcome this issue are: *approximation* and *parameterized complexity*. The approximation paradigm consists of producing solutions which are not so close, in terms of performance, from an optimal solution. The goal of parameterized complexity is to solve problems in polynomial time, in the size of the input, times a function, which can be exponential or even superexponential, of a parameter of the problem. The idea hidden behind, is that if you know that your instances have a *small* parameter, the complexity you get is roughly polynomial. An emerging and interesting research area is to combine both paradigms. It can be called *parameterized approximation*.

In this thesis, we present a new technique called *greediness-for-parameterization* and we use it to improve the parameterized complexity of many problems.

We revisit a notion introduced in parameterized complexity called *necessary sets* within the scope of approximation. We define a very close notion that we call *intersectivity*, and which enables us to establish parameterized inapproximability for a large class of problems that we call *subset problems*. We also use this notion to obtain parameterized algorithms for some problems in bipartite graphs.

Aiming at establishing negative results on the approximability in subexponential time and in parameterized time, we introduce new methods of *sparsification* that preserves approximation. We combine those *sparsifiers* with known or new reductions to achieve our goal.

Finally, we present some hardness results of games such as Bridge and Havannah.

Résumé

De nombreux problèmes de la vie réelle sont **NP**-difficiles. Si, comme il est très fortement conjecturé $\mathbf{P} \neq \mathbf{NP}$, ils ne peuvent pas être résolus en temps polynomial. Deux paradigmes notables pour les résoudre quand même sont : l'*approximation* et la *complexité paramétrée*. L'approximation consiste à produire des solutions relativement proches de la performance d'une solution optimale. La complexité paramétrée vise à résoudre en temps polynomial en la taille de la donnée multiplié par une fonction pouvant être exponentiel ou même surexponentiel d'un paramètre du problème. L'idée est que pour des instances où ce paramètre est connu pour être *petit*, la complexité obtenue est bonne. *L'approximation en temps paramétrée*, combinaison des deux paradigmes, est un domaine de recherche récent et intéressant.

Dans cette thèse, on présente une nouvelle technique appelée *greediness-for-parameterization* (gloutonnerie-pour-la-paramétrisation). On l'utilise pour établir ou améliorer la complexité paramétrée de nombreux problèmes.

On revisite une notion en complexité paramétrée appelée *necessary sets* (ensembles nécessaires) dans le contexte de l'approximation. On définit alors une notion voisine qu'on appelle *intersectivité*. Celle-ci nous permet d'établir des résultats d'inapproximabilité en temps paramétré d'une large classe de problèmes qu'on nomme ici *subset problems* (problèmes de sous-ensemble). On utilise également cette notion pour obtenir des algorithmes paramétrés pour des problèmes à cardinalité contrainte sur des graphes bipartis.

En vue d'établir des résultats négatifs sur l'approximabilité en temps sous-exponentiel et en temps paramétré, on introduit différentes méthodes de *sparsification* d'instances préservant l'approximation. On combine ces *sparsifieurs* à des réductions nouvelles ou déjà connues pour parvenir à nos fins.

En guise de digestif, on présente des résultats de complexité de jeux comme le Bridge et Havannah.

Acknowledgements/Remerciements

I warmly thank Christoph Dürr, Henning Fernau, Klaus Jansen, Dieter Kratsch, and Claire Mathieu for accepting to be members of my jury. An additional thank to Henning Fernau and Klaus Jansen for accepting to review the present manuscript, and to Claire Mathieu for attending a pre-defense and giving me many useful suggestions and pieces of advice.

Vangelis, je mesure à quel point j'ai eu de la chance que tu encadres ma thèse, tant sur le plan professionel que sur le plan humain. Je te remercie infiniment pour t'être montré si disponible envers moi, pour m'avoir beaucoup appris, pour ta bonne humeur et ton humour naturels.

Bruno, tu as réussi à me co-encadrer malgré tout ce qui t'arrivait: médaille de bronze du CNRS, nouvel enfant, poste de professeur et son lot de nouvelles responsabilités, déménagement. Je te suis très reconnaissant d'avoir su me garder du temps dans ce contexte.

J'ai une pensée pour ma famille, mes amis et mes collègues qui animent ma vie et ont été un précieux soutien pendant ces trois ans (et parfois bien avant). À commencer par ma soeur Louise, qui m'a devancé de quelques semaines dans l'épreuve de la soutenance, ma mère, mon père, si prévenants, mes grands-parents Papé et Alice, Annie et Jacques, sans oublier mes cousins, cousines, oncles et tantes, et Lilith le chat.

Faisal Abu-Khzam, Mohamed Ali Aloulou, Marin Bougeret, Nicolas Bourgeois, Denis Cornaz, Federico Della Croce, Bruno Escoffier, Florent Foucaud, Aristotelis Giannakos, Florian Jamain, Eunjung Kim, Michael Lampis, Jérôme Monnot, Abdallah Saffidine, Florian Sikora, Georgios Stamoulis, Vangelis Paschos (obviously), Émeric Tourniaire, and Rémi Watrigant it has been a great pleasure formally working with you or informally discussing research.

Abdallah, Bernard, Denis, Lucie, Meltem, Virginie, nous avons été amenés à collaborer dans notre mission d'enseignement. Sachez que cette collaboration m'a été bénéfique et agréable.

Je remercie aussi toute l'équipe administrative: Eleni, Hawa, Katerina, Mireille, Nathalie, Valérie pour leur serviabilité et leur efficacité que ce soit pour les missions de recherche, les réservations de salles ou les préparatifs de soutenance. Olivier, véritable mascotte du laboratoire, quel que soit le point auquel je faisais crasher mon système, tu savais le restaurer.

Cristina et Jérôme, merci à vous deux pour ce que vous faites pour la vie doctorale

au LAMSADE. Florian, ton bureau m'était toujours ouvert, pour le *travail* comme pour les *pauses*. J'ai appris récemment que tu n'étais plus en thèse donc tu dois être en postdoc. Émeric, unique frère de thèse, même si tu as soutenu il y a plus d'un an maintenant, je n'oublie pas les discussions politiques animées, ta tentative de conversion universelle au bubble tea et ton séminaire transversal.

J'ai passé d'innombrables bons moments grâce à mes collègues doctorants passés, présents et futurs: Abdallah, Amel, Amine, Anaëlle, Émeric, Florian, Liangliang, Lydia, Lyes, Marek, Nathanaël, Nicolas, Raouia, Renaud, Satya, Sébastien, Tom, Yann...

Maxime et la famille Martin, PE et mes anciens camarades de Lycée bisontins, Youssef et la famille Ouchene, Arthur, Ambroise et la famille Brody vous avez égayé mes vacances dans des locations exotiques comme Rabat, Vensac, Besançon ou encore Chapelle-des-Bois.

Pendant ma thèse, j'ai eu trois activités extérieures prédominantes: le bridge, les échecs et le théâtre. Je salue au passage mes petits camarades dans ces trois sphères.

Mes principaux partenaires de bridge Mickaël, Sylvain, PH et JB ainsi que mes autres co-équipiers de 4: notre capitaine Éric, JF et Nicolas.

Mes amis du club d'échecs de Fontainebleau, cercle florissant, où l'ineffable Florian Jamain m'a *trainé* il y a près de 3 ans pour aller jouer la montée de nationale 4, et qui se targue[1] aujourd'hui de trois équipes dont deux en nationale 2 et nationale 3.

Les membres de l'association de théâtre de l'université. Monter deux spectacles avec vous a été une expérience amusante et enrichissante. Je suis reconnaissant de ce cher Abdallah qui, il y a près de deux ans, m'a donné l'opportunité de vous rencontrer.

Enfin, je te remercie Tatiana pour partager ma vie, être merveilleuse et drôle; Я тебя люблю.

[1] Plus l'entier succédant *nationale* est petit, mieux c'est.

Contents

Notes to the reader

The problems mentioned in this thesis are all defined in the appendix, and are listed there in alphabetical order.

Within the pages, you will find two kinds of boxes: one with a light gray background, the other with a darker gray background.

In this kind of box, you will find important pieces of information, key ideas, or summary of results. They might be worth reading since they clarify important points.

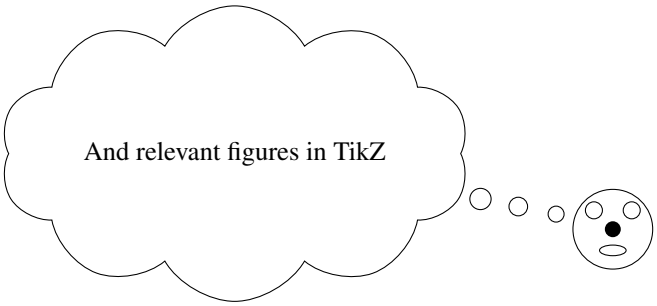
In that kind of box, you will find anecdotes, non decisive remarks, or clarification of minor points[a]. If you do not like anecdotes, you can skip the darker gray boxes, and that might be the first and last you read until the end.

[a] This is like a long footnote, in a way.

For the sake of readability, any PROBLEM, **CLASS**, and `ALGORITHM` have always this very typography. What about words in *italics*? Mostly, words are in *italics* because they are not defined yet, so the reader do not have to worry too much (in principle, they will be explain later). Once explained those words will not be in italics anymore. *Additionally*, a word can be in italics because it is crucial in its sentence. You may encounter the description of an algorithm in pseudo-code.

Algorithm 1: Here is the standard procedure.

Input: A reader.
Output: An informed reader.
for *each line* L **do**
    read L;
end



We wish you a pleasant reading!

1 Introduction

Each scientific field has its deep philosophical questions.

In physics, you have plenty: *when/how/why did all start?*, *what is universe made of?*, *will all end? if so, when/how/why?*, etc. And basically, every time you look through a window/telescope/microscope and you see something you cannot explain, you ask a new question. As universe is oddly complicated, you receive too many answers to be fully satisfied and you ask: *can we explain everything in a more unified way?*. In mathematics, your inquiries position in an idealized world: *what is a proof?*, *can we prove every true statement?*, *is it odd that the same number appears so often?*, etc. And basically, every time you think of a statement which might be true, you ask yourself if you can prove it. Physics and mathematics (at least practicing by humans) are thousands years old, so the most crucial issues have been at least partially answered[1]. Thus, our minds are sufficiently appeased to process the challenges of the new born (only sixty years old) computer science.

The theoretical computer scientist is a special mathematician who likes to solve problems constructively. *What can I solve?*, or equivalently, *what can I compute?* is his main concern. But wait, what is it to *solve* or to *compute*? And, what is a *problem*?

1.1 Problems, Instances, and Algorithms

A *problem* is a question, defined in some formal system, that you can ask for an infinite family of inputs or *instances*. The answer to this question is called *output*. For example, *give the prime factorization of your input* is a problem, and its interesting instances are the natural numbers. We may see problems as functions. If given the input 12, a desired

[1] Okay, this is truly debatable.

output would be $(2, 2, 3)$. Our example is very much alike the function $f : n \mapsto$ the prime factorization of n. In fact, this most general class of problems that we have just defined is called *function problems*. Basically, given an input, we have to compute its image by a function which is the core of the problem. Another example might be the *sorting problem*. You are given a list of, say, integers and you are asked to order this list by non decreasing values and to output the result.

A central subclass of problems is the class of *decision problems*. In a *decision problem*, your answer is either YES or NO. *Is the input an even number?* is a decision problem, though not particularly challenging. *Is the input a prime number?* is a more interesting decision problem. Decision problems can be defined formally (contrary to function problems) as the set of the instances for which the answer is YES, called for short YES-instances. If Π is a decision problem, we denote by $\mathcal{L}(\Pi)$ the sets of its YES-instances, and we will sometimes identify Π to $\mathcal{L}(\Pi)$.

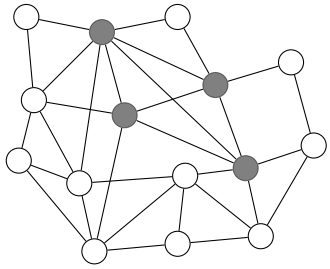
Another crucial subclass of problems is the class of *optimization problems*. In an *optimization problem*, given an input x, you have to find the output y optimizing (minimizing or maximizing) the value $m(x, y)$ where m is a function specified by the problem. If the value of the solution y can be computed independently of the input x, we will write $\mathrm{val}(y)$ instead of $m(x, y)$.

> Each function problem can be artificially translated in a maximization (or minimization) problem. Assuming f is the function one wants to compute, we set $m(x, y) = 1$ if $y = f(x)$, and $m(x, y) = 0$ otherwise. On the contrary, a decision problem cannot be seen as a function problem.

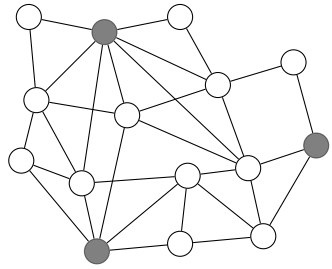
MAX CLIQUE is an optimization problem. Let us describe this problem[2]. Given a graph[3] G, you want to find the maximum number of vertices such that there is an edge between each pair of vertices you have taken. Such a set of vertices is called a *clique*. This is a maximization problem for which the function m is defined by $m(G, S) = |S|$ if G corresponds to the description of a graph, and S is a clique of G, and $m(G, S) = \infty$ otherwise. MIN DOMINATING SET is an example of a minimization problem where, given a graph $G = (V, E)$, you want to find the minimum number

[2] Additionally, we recall that all the problems introduced are defined in the appendix.

[3] If you do not know what a graph is, you may read first Section 1.4.



(a) A maximum clique of a graph. Vertices in the clique are filled.



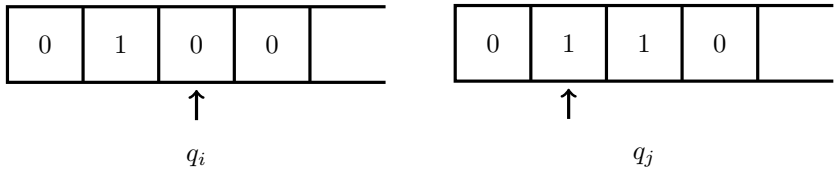
(b) A minimum dominating set in the same graph.

Figure 1.1: Solutions for MAX CLIQUE and MIN DOMINATING SET on an instance.

of vertices whose closed neighborhood is V. There is a canonical way of projecting optimization problems into decision problems, by fixing a threshold value. For instance, CLIQUE consists of determining if there is a clique of size at least k. It is the *decision version* of MAX CLIQUE.

An *algorithm* is a finite sequence of instructions specified in some formalism. This definition is vague but the Church-Turing thesis states that all the realistic models of computation that we could think of, are equivalent, in the sense that they define the same set of *computable functions*. For instance, the *λ-calculus* of Alonzo Church, the *Turing Machine* of Alan Turing, and the *recursive functions* of Kurt Gödel and Jacques Herbrand are all equivalent. As far as we are concerned, we will define an algorithm as a Turing Machine.

In 1928, David Hilbert asks in a conference if there was an algorithm which, given a first-order formula, tells whether or not it is *universally valid* (or equivalently *provable*, since first-order logic is complete). The first step was to define properly what an *algorithm* was. Alonzo Church [42] and Alan Turing [123] gives a negative answer to Hilbert's question provided we accept their (equivalent) definition of what is *algorithmically computable*. This started an uninterrupted discussion around the Church-Turing Thesis.



(a) The configuration before the transition. (b) The configuration after the transition.

Figure 1.2: Applying the transition $(0, q_i, 1, q_j, \leftarrow)$.

Definition 1. A *Turing Machine* is a tuple $\langle Q, q_1, q_a, q_r, \Delta \rangle$ where Q is a finite set of *states*; $q_1 \in Q$ is a distinguished *initial* state; $q_a \in Q$ is a distinguished *accepting* state; $q_r \in Q$ is a distinguished *rejecting* state; $\Delta \subseteq Q \times \{0,1\} \times Q \times \{0,1\} \times \{\leftarrow, \uparrow, \rightarrow\}$ is a set of *transition rules*.

A Turing Machine can be seen as a semi-infinite tape composed of cells filled with 0 or 1, together with a *head* positioned upon one of the cell.

Definition 2. A *configuration* is a triple (w_1, q, w_2) where q is the current state, $w_1 w_2$ is the content of the tape, and the head is upon the first letter of w_2.

When in a configuration (u, q, av) ($a \in \{0,1\}$), the Turing Machine can apply the transition $(q, a, q', b, d) \in \Delta$ and the following happens. Bit a is replaced by bit b, the state changes to q', and the head moves to the right if $d = \rightarrow$, to the left if $d = \leftarrow$, or stays still if $d = \uparrow$. A Turing Machine is said deterministic if given a state q and a bit a there is at most one transition of the form $(q, a, \cdot, \cdot, \cdot)$ in Δ (at most one transition is doable). Therefore, given an input, a deterministic Turing Machine has a unique run. A configuration (w_1, q_a, w_2) is *accepting*, while a configuration (w_1, q_r, w_2) is *rejecting*. When the Turing Machine reaches an accepting or a rejecting configuration, it halts.

A Turing Machine *accepts* an input w, if *there is* a finite sequence of transitions from the configuration (ε, q_1, w) (where ε is the empty word) to an accepting configuration. A Turing Machine *rejects* an input w, otherwise. The set of accepted inputs is the *language recognized* by the Turing Machine. That definition fits the solving of decision problems. If one wants to solve function problems in general, one can look at the content of the tape (output) when the computation ends.

A decision problem Π is *decidable* if there is a Turing Machine stopping on every input and whose language is exactly $\mathcal{L}(\Pi)$. Otherwise, Π is said undecidable. CLIQUE

is an example of a decidable problem. Indeed, we can enumerate all the sets of k vertices of the graph, and check if one of them constitutes a clique. Describing an algorithm with the transition rules of a Turing Machine is fastidious. Instead, we will give some Turing-complete pseudo-code. The problem of deciding if a Turing Machine halts on the empty input is an example of an undecidable problem.

In this thesis, we focus on decidable problems. Our main concern will not be *can I solve?* but *how fast can I solve?*. But before we introduce the main classes of efficiency, we present a central notion known as *reductions*.

1.2 Reductions

In computer science, a *reduction* is a translation of a problem into another.

Formally, a reduction from a decision problem Π_A to another decision problem Π_B is a function $\rho : \mathcal{I}_A \to \mathcal{I}_B$, where $\mathcal{I}_x$ is the set of instances of Π_x, such that $\forall I \in \mathcal{I}_A$, $I \in \mathcal{L}(\Pi_A) \Leftrightarrow \rho(I) \in \mathcal{L}(\Pi_B)$.

Why could it be interesting to translate Π_A into Π_B? What comes in mind naturally is that if you know how to solve Π_B, you now can solve Π_A. In this case, we use a reduction to get a *positive* result.

In fact, we more often use the contrapositive, that is, assuming that we cannot solve efficiently Π_A, we derive that Π_B also cannot be efficiently solved. In that case, we use a reduction to get a *negative* result.

Reductions may happen in real-life. Let us assume that the standard procedure `P` to make pasta is to take an empty pan, fill it with water, turn on the cooktop, put the saucepan upon it, wait for the water to boil, add the pasta, wait for them to be cooked. But now, you encounter a difficulty: the only saucepan you find is filled with water. You still want to make pasta. What do you do? Well, this is easy: you throw the water away and then you apply `P`. Throwing the water away is a (quite silly) real-life reduction from the problem MAKING PASTA WITH A FILLED SAUCEPAN to the problem MAKING PASTA WITH AN EMPTY SAUCEPAN which gives a positive result (pasta).

We now get back the theoretical world, and we present an example of a reduction from SAT to INDEPENDENT SET which gives, as we will see in the next section, a negative result. Again, the reader is referred to Section 1.4 and to the appendix for the definitions of both problems. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ be any instance of SAT with n variables. We explain how we construct a graph $G = \rho(\mathcal{C})$ such that G has an independent set of size $n + m$ if and only if $\mathcal{C}$ is satisfiable. For each variable x_i, there

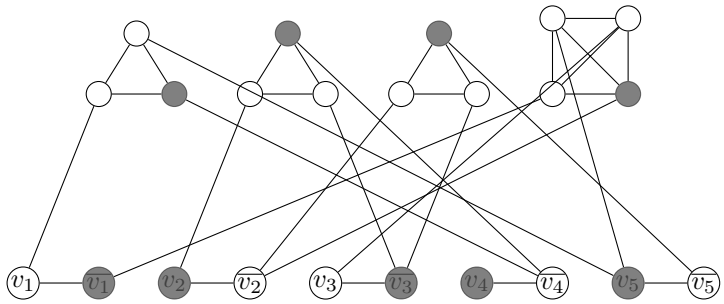


Figure 1.3: The graph $\rho(\mathcal{C})$ for the instance $\mathcal{C} = \{x_1 \vee \neg x_4 \vee x_5, x_2 \vee \neg x_3 \vee \neg x_4, \neg x_2 \vee \neg x_3 \vee \neg x_5, \neg x_1 \vee \neg x_2 \vee x_3 \vee x_5\}$ and an independent set corresponding to a truth assignment.

are two vertices v_i and $\overline{v}_i$ linked by an edge in G. A piece of structure which encodes a building block of the initial problem is called a *gadget*. Thus, a pair of vertices linked by an edge is a fairly simple *variable gadget*. For each clause $C_j = l_j^1 \vee l_j^2 \vee \ldots \vee l_j^k$, we add a clique of size k in G. Each vertex of this clique represents one literal of C_j (in a one-to-one correspondence) and is linked to the vertex encoding the same literal in the variable gadgets. This defines the clause gadgets.

As each of the n variable gadgets and each of the m clause gadgets is a clique, an independent set in G has size at most $n + m$ (one vertex per clique).

If $\mathcal{C}$ is satisfiable, then taking the vertex corresponding to the opposite literal in one fixed truth assignment, can be completed by taking one vertex in each clause gadget (since in each clause of $\mathcal{C}$ there is a literal set to true). This defines an independent set of size $n + m$. Reciprocally, if there is an independent set of size $n + m$, then it takes exactly one vertex in each gadget. Setting x_i to true if $\overline{v_i}$ is in the independent set, and to false if v_i is, is a truth assignment. In each clause, this assignment satisfies at least the literal whose corresponding vertex is in the independent set. Hence, $\mathcal{C}$ is satisfiable.

We observe that this reduction can be computed in polynomial time. More formally, a reduction ρ is a polynomial time reduction, if ρ can be computed in polynomial time, and in that case, we notice that $|\rho(I)|$ is necessarily polynomial in $|I|$, where $|\cdot|$ maps an instance to its size. This means that if we assume that SAT cannot be solved in polynomial time, then neither can INDEPENDENT SET. We will discuss further this phenomenon in the following section.

1.3 Computational Complexity Classes

Computational complexity classes measure how the running time (or the space required) of an algorithm scales with respect to the size of the input. There are hundreds of complexity classes. Here, we define only the most standard ones.

P is the class of problems that can be solved by a deterministic Turing Machine in polynomial time, that is in time $O(n^c)$ for some constant c, where n is the size of the input. For instance, PALINDROME, the problem of determining if a word reads the same forwards and backwards like *kayak*, is in **P**. Indeed, in quadratic time $O(n^2)$, we can check if the first letter matches the last one, the second matches the next to last, and so forth, up to the middle of the word. With a Turing Machine, we lose some time in meaningless comings and goings of the head within the input. In fact, we may think algorithms as reading a bit of the input in constant time $O(1)$, as in the RAM (random-access machine) model, or equivalently as a piece of pseudo-code. It turns out that all the reasonable models are *polynomially* equivalent, in the sense that the blow-up to solve a same problem with different models of computation is always polynomially bounded. Thus, the model does not alterate the class **P** and the upcoming classes.

Algorithm 2: A simple algorithm to solve PALINDROME

Input: A word $w = w[1]w[2]\dots w[n]$ of size n.
Output: YES if w is a palindrome, NO otherwise.
$i \leftarrow 1$;
while $w[i] = w[n-i+1] \wedge i \leqslant \lfloor n/2 \rfloor$ **do**
    $i \leftarrow i + 1$
end
return $i > \lfloor n/2 \rfloor$

Obviously, this is a basic example. Finding a polynomial time algorithm for a problem can be harder (2SAT [8]) or even harder than that (MAXIMUM MATCHING [55]) or even way harder than that (PRIME [2]).

NP is the class of problems which can be solved in polynomial time by a (non deterministic) Turing Machine. Algorithm 3 shows that SAT is in **NP**.

We may observe that in Algorithm 3, all the non-deterministic steps are grouped together at the beginning. It consists of setting non-deterministically each variable to either True or False. Then, the rest of the algorithm is purely deterministic: evaluating the formula for the given assignment. In fact, with no loss of expressiveness, every

Algorithm 3: A non-deterministic algorithm to solve SAT in polynomial time. $a|b$ is the non-deterministic transition which does either a or b.

Input: A CNF formula ϕ.
Output: YES if ϕ is a satisfiable, NO otherwise.
for *each variable* x_i *in* ϕ **do**
    $x_i \leftarrow$True | $x_i \leftarrow$False ;
end
return $\phi(\vec{x})$;

problems in **NP** can be solved by performing first all the non-deterministic transitions and then by doing only deterministic transitions. Thus, we can formulate an alternative definition of **NP**. A problem Π is in **NP** if there are a constant c and a deterministic polynomial time Turing Machine M such that:

- given any YES-instance $I \in \mathcal{L}(\Pi)$, there exists a word y of size $|I|^c$ called *certificate* such that $M(I, y)$ outputs YES.
- given any NO-instance $I \notin \mathcal{L}(\Pi)$, for any word y of size $|I|^c$ such that $M(I, y)$ outputs NO.

The most natural certificate that we can think of is a solution of the instance. Provided the description of a solution can always be done with only a polynomial blow-up in the size of the instance, if we can check in polynomial time that something is a solution, then the problem is in **NP**.

A problem Π is **NP**-*hard* if there is a polynomial time reduction from the problem of simulating a non-deterministic Turing Machine which halts in polynomial time to Π. The first problem shown to be **NP**-hard is SAT in a paper of Stephen Cook in 1971 [43], and independently by Leonid Levin. This result is usually called the Cook Theorem or the Cook-Levin Theorem. A problem which is simultaneously in **NP** and **NP**-hard is said **NP**-*complete*. This class is denoted by **NP**-c. With the previous remarks, SAT is **NP**-complete. The relation $\Pi_A \underset{\mathbf{P}}{\rightarrow} \Pi_B$ defined by *there exists a polynomial time reduction from* Π_A *to* Π_B, is transitive. Thus, to show that a problem Π is **NP**-hard, we can show that SAT[4] $\underset{\mathbf{P}}{\rightarrow} \Pi$. In Section 1.2, we did show that INDEPENDENT SET is **NP**-hard. The reader is referred to the seminal paper of Richard Karp [87] and to the

[4] In fact, we can reduce our problem from any **NP**-hard problem.

book of Michael Garey and David Johnson [71] for further reading on the theory of **NP**-hardness.

The polynomial time reduction also permits to define the classes of the hardest problems within the two following classes **PSPACE** and **EXP**. They are similarly denoted by **PSPACE**-c and **EXP**-c.

PSPACE is the class of problems that can be solved with a deterministic (or non deterministic by Savitch's Theorem [115]) Turing Machine using polynomial space, that is space $O(n^c)$ for some constant c, where n is the size of the input. QBF is in **PSPACE** and by a construction similar to the proof of the Cook-Levin Theorem it can be shown **PSPACE**-complete (in **PSPACE**-c).

EXP is the class of problems that can be solved with a deterministic Turing Machine in exponential time, that is, in time $O(2^{n^c})$ for some constant c. The problem of deciding if a generalized[5] chess position is winning for White is in **EXP** and it is even **EXP**-complete (in **EXP**-c) [65].

R is the class of problems that can be solved by a Turing Machine. All the problems introduced so far in this section are in **R**, whereas, as said above, the problem of deciding if a Turing Machine halts on the empty input is not in **R**. As the following holds **P** $\subseteq$ **NP** $\subseteq$ **PSPACE** $\subseteq$ **EXP** $\subsetneq$ **R** and **P** $\subsetneq$ **EXP**, at least one of the three inclusions **P** $\subseteq$ **NP**, **NP** $\subseteq$ **PSPACE** and **PSPACE** $\subseteq$ **EXP** is strict. It is widely believed that **P** $\subsetneq$ **NP**, but it has not been proven yet though such a proof would be rewarded by a generous tip of one million dollars from the Clay Mathematics Institute. Anyway, it is likely that the **NP**-complete problems are not all solvable in polynomial time, which is equivalent of saying that none **NP**-complete problem is solvable in polynomial time. It is even conjectured that solving SAT requires exponential time $O^*(\lambda^n)$ for some $\lambda > 1$, where n is the number of variables. This constitutes the Exponential Time Hypothesis (ETH, for short) [81]. The Strong Exponential Time Hypothesis (SETH) is slightly more debatable and states that solving SAT requires time $O^*(2^n)$. Thus, the brute-force algorithm, checking all the possible assignments, would be asymptotically optimal. In general, the theoretical assumption that polynomial time is *tractable* while exponential time is not, must be put into perspectives by practical considerations: an algorithm performing in exponential time 1.0000001^n is far better than a polynomial time algorithm working in time $n^{100000000}$ for instances of a reasonable size.

In Section 1.3.1 and Section 1.3.2, we introduce two fields coping with **NP**-

[5] on a $n \times n$ board with an unbounded number of pieces, and without the fifty-moves rule.

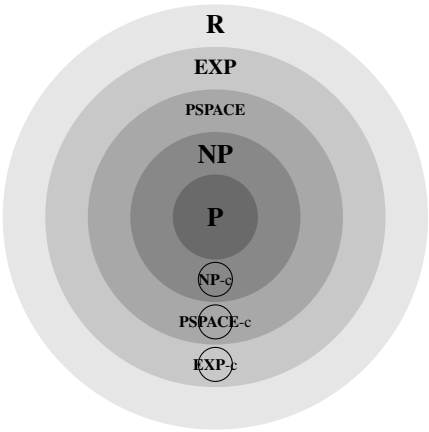


Figure 1.4: The dartboard of classical computational complexity. The difficulty of a problem is the throwing distance. The more accurate you are, the better your algorithm performs. Though, everything is not possible. For instance, at the distance of an **EXP**-complete problem, the **P** area is *physically* untouchable.

hardness, namely *approximation* and *parameterized complexity*.

1.3.1 Approximation

The field of *approximation* can be traced back to the early seventies with the seminal paper of David Johnson [84]. The crux of approximation is to produce in polynomial time a solution whose performance is as close as possible to optimality. Therefore and by essence, approximation algorithms only handle optimization problems. In fact, this is not such a restriction since many decision problems can be seen as optimization problems. For instance, MAX SAT is a natural maximization version of SAT, where instead of satisfying all the clauses, we *just* need to satisfy as many clauses as we can.

Let $\mathrm{opt}(I)$ be the value of an optimal solution for a specified problem Π on instance I. If there is no ambiguity on the instance considered, we may shorten $\mathrm{opt}(I)$ to opt. An approximation algorithm $\mathtt{A}$ with *ratio* $\rho : \mathbb{N} \to \mathbb{R}$ is an algorithm such that for all instance I, $\mathrm{val}(\mathtt{A}(I)) \leqslant \rho(|I|)\,\mathrm{opt}(I)$ for a minimization problem (and, $\mathrm{val}(\mathtt{A}(I)) \geqslant \rho(|I|)\,\mathrm{opt}(I)$ for a maximization problem). For maximization problems, it is sometimes more convenient to have ratios greater than 1 and we will explicitly

redefine the ratio by $\mathrm{val}(\mathtt{A}(I))\rho(I) \geqslant \mathrm{opt}(I)$. $\mathtt{A}$ is a $\rho(n)$-*approximation* and the problem is said $\rho(n)$-*approximable*. A problem Π is *constant approximable* if an approximation algorithm with a ratio ρ which is a constant function. The class of such problems is called **APX**.

`APP-VC`, detailed in Algorithm 4, is a 2-approximation algorithm for MIN VERTEX COVER, hence MIN VERTEX COVER is in **APX**. The ratio 2 is obtained since if you consider the set $S = \mathtt{APP\text{-}VC}(G)$ of the edges e whose two endpoints u and v have been added to S, a feasible solution takes at least one vertex in $\{u, v\}$, so $\mathrm{val}(S) \leqslant 2\,\mathrm{opt}$.

Algorithm 4: A 2-approximation algorithm for MIN VERTEX COVER

Input: A graph $G = (V, E)$.
Output: A vertex cover of G of size at most $2\,\mathrm{opt}$.
$S \leftarrow \emptyset$;
`APP-VC`(G):
while G *is not empty* **do**
    pick any edge $e = (u, v)$ in G ;
    $S \leftarrow S \cup \{u, v\}$;
    delete u and v from G ;
end

As MIN VERTEX COVER is **NP**-complete it is unlikely to solve it exactly in polynomial time, but we might ask for a better ratio $r < 2$. We could be even more optimistic and ask for a family of approximation algorithms achieving any ratio strictly greater than 1. This is called a *polynomial-time approximation scheme* (PTAS for short). In fact, a PTAS for MIN VERTEX COVER is ruled out by a *gap-introducing* reduction showing that finding a 1.36-approximation is already **NP**-complete [50]. A *gap-introducing* reduction is a reduction ρ from a decision problem Π_A to a minimization (respectively, maximization) problem Π_B such that:

- If $I \in \mathcal{L}(\Pi_A)$ then $\mathrm{opt}(\rho(I)) \leqslant a$ (respectively, $\mathrm{opt}(\rho(I)) \geqslant a$).
- If $I \notin \mathcal{L}(\Pi_A)$ then $\mathrm{opt}(\rho(I)) > ra$ (respectively, $\mathrm{opt}(\rho(I)) < ra$).

An approximation algorithm with ratio r or better would solve Π_A to the optimum, and r is called the *gap* of the reduction. Thus, if Π_A is **NP**-hard and ρ can be computed in polynomial time, then Π_B cannot be r-approximated unless $\mathbf{P} = \mathbf{NP}$.

We give an example of how to obtain an inapproximability result with a gap-introducing reduction. We recall a simple gap-introducing reduction from SAT to MIN INDEPENDENT DOMINATING SET devised in [82] and then in [75]. Given an instance ϕ of SAT, each variable x_i in the instance of SAT is encoded by two vertices v_i (representing literal x_i) and $\overline{v_i}$ (representing literal $\neg x_i$) linked by an edge. Each clause C_j is encoded by an independent set of size rn, where n is the number of variables in ϕ. If literal x_i (resp., $\neg x_i$) appears in clause C_j, then vertex v_i, (resp. $\overline{v_i}$), is linked to all the rn vertices encoding clause C_j. If ϕ is satisfiable, then there exists an independent dominating set of size n which consists of the vertices representing satisfied literals. If ϕ is not satisfiable, then any independent dominating set contains more than the rn vertices representing a clause. Thus, the reduction builds a gap r and therefore MIN INDEPENDENT DOMINATING SET is not in **APX**. We may even show that MIN INDEPENDENT DOMINATING SET is not $n^{1-\varepsilon}$-approximable, for any $\varepsilon > 0$, assuming $\mathbf{P} \neq \mathbf{NP}$.

Now, we want to define the notion of *gap* for a problem itself. A *promise* problem is a decision problem where it is guaranteed that the instances either come from $\mathcal{L}_{\text{YES}}$ a subset of YES-instances or from $\mathcal{L}_{\text{NO}}$ a subset of NO-instances. Most combinatorial optimization problems can be expressed as a *max constraint satisfaction problem* (max CSP for short). In max CSP problems, one wants to find an assignment of the variables in a specific domain, that satisfies the greatest number of constraints. The constraints are relations over the variables. For instance, MAX SAT is a max CSP problem, where the domain is $\{\bot, \top\}$ and the constraints are the clauses. For $0 \leqslant s < c \leqslant 1$, the *gap* problem (c, s)-GAP Π is a promise max CSP problem such that the instances either come from $\mathcal{L}_{\text{YES}}$ and in that case, at least cm constraints are satisfiable, or from $\mathcal{L}_{\text{NO}}$ and in that case, none assignment satisfies more that sm constraints, where m is the total number of constraints. Assuming $\mathbf{P} \neq \mathbf{NP}$, showing that (c, s)-GAP Π is **NP**-hard, proves that Π cannot be $\frac{s}{c}$-approximated. *Gap-preserving* reductions and *gap-amplifying* reductions are two other kinds of reductions in the inapproximability toolkit. The former preserves a function of the gap while the latter increases the gap.

Coming back to MIN VERTEX COVER, it is quite possible that the 2-approximation does not admit any significant improvement. Indeed, if the Unique Games Conjecture (conjecture, whose odds of being true is substantial) holds then, for every $\varepsilon > 0$, there is no $2 - \varepsilon$-approximation for MIN VERTEX COVER [88].

In the eighties, the analysis of Interactive Proof systems lead to a new characterization of the class **NP** by the PCP (*Probabilistically Checkable Proofs*) Theorem [7]. This

In the UNIQUE LABEL COVER problem, we are given a graph $G = (V, E)$, an integer k, and a function $\pi : E \to S_k$ where S_k is the set of permutations on k elements. A k-coloring of G is a mapping $c : V \to \{1, \ldots, k\}$. An edge $\{u, v\} \in E$ is satisfied if $\pi(c(u)) = c(v)$. The goal is to find a k-coloring which satisfies the greatest number of edges. The Unique Games Conjecture (UGC) states that there exist constants $c < 1$, $s > 0$ and integer k such that (c, s)-GAP UNIQUE LABEL COVER is **NP**-hard.
Unlike **P** $\neq$ **NP**, there is no general consensus in the scientific community that UGC is true. Nevertheless, this conjecture has inspired many interesting works and the interested reader is referred to the survey of Subash Khot [89].

theorem opened the door to inapproximability results and most of the approximation algorithms of the seminal paper of David Johnson [84], as the greedy algorithm for MIN SET COVER achieving ratio $\log(n)$ and the observation that nothing seems to work for MAX CLIQUE, prove to be almost optimal. Irit Dinur gives a simpler and only combinatorial proof of the PCP Theorem whose main ingredients are gap-amplifying reductions and *expanders* [49]. At this point, we do not wish to bother the reader with PCPs and expanders. Those definitions are deferred to Chapter 3.

The reader can learn a lot more about approximation algorithms and inapproximability in a book edited by Dorit Hochbaum [78] and a book of Vijay Vazirani [125].

1.3.2 Parameterized Complexity

The field of *parameterized complexity* has been invented in the early nineties by Rodney Downey and Michael Fellows. All the definitions introduced in this section can also be found in their book [51]. The seminal idea of parameterized complexity is that, in practice, instances of (**NP**-)hard problems are not totally random. It is quite possible that structural parameters such as degree, treewidth, diameter of a graph, or the size of a solution, are small integers for a large class of instances for which we have to solve the problem. For those instances, if we are able to confine the superpolynomial blow-up to such a parameter and not anymore to the size of the input, then our algorithm performs in polynomial time.

FPT is the class of problems that can be solved in time $O(f(k)n^c)$, where k is the value of the parameter and n is the size of the input. From hereon, time $O(f(k)n^c)$ is called *FPT* time. **FPT** stands for *fixed parameter tractable* and corresponds to the easiest parameterized problems. Often, we take as parameter k the size of the solution, which is called the *natural* parameter. It is the implicit parameter. When not precised otherwise, the parameterized complexity is with respect to the natural parameter. Algorithm 5 shows that k-VERTEX COVER is in **FPT**.

Algorithm 5: A simple branching algorithm to solve k-VERTEX COVER

Input: A graph $G = (V, E)$ and an integer k.
Output: A vertex cover of G of size k, if existing.
$S \leftarrow \emptyset$;
$\texttt{VC}(G, k, S)$:
if G *is empty* **then**
    return S ;
else
    if $k = 0$ **then**
        return *Void* ;
    else
        pick any edge $e = (u, v)$ in G ;
        run $\texttt{VC}(G[V(G) \setminus \{u\}], k - 1, S \cup \{u\})$;
        run $\texttt{VC}(G[V(G) \setminus \{v\}], k - 1, S \cup \{v\})$;
    end
end

The fact that a vertex cover contains at least one of the endpoint of every edge of the graph ensures the soundness of VC. The running time of VC is $O^*(2^k)$ where $O^*(\cdot)$ suppresses the polynomial factors.

If I is an instance of a parameterized problem, we denote by $\kappa(I)$ the value of the parameter for the particular instance I. An *FPT reduction* from a parameterized problem Π_A to a parameterized problem Π_B is a reduction ρ from Π_A to Π_B, two computable functions f and g, and a constant c, such that for every instance I of Π_A, $\kappa(\rho(I)) \leqslant f(\kappa(I))$, ρ can be computed in FPT time $O(g(\kappa(I))|I|^c)$, and in particular, $|\rho(I)| \leqslant g(\kappa(I))|I|^c$. This reduction is defined so that if Π_B is in **FPT** then Π_A is in **FPT**, too. Indeed, if we have an FPT algorithm solving Π_B in time $O(h(\kappa(I))|I|^{c'})$, we can then solve Π_A in FPT time $O(h(f(\kappa(I)))(g(\kappa(I)))^c|I|^{cc'})$. As for polynomial

Sometimes, the easiest ideas are overlooked. In the paper [35], the authors propose a linear time algorithm to find vertex covers of size up to 5. In fact, the simple branching procedure of Algorithm 5 applied to graphs with bounded vertex cover has a constant time complexity. During the eighties, this algorithm has been discovered [105], forgotten [62], and rediscovered [61]. Currently, the best known algorithm works in time $O^*(1.274^k)$ [39]

time reduction, we more often use FPT reductions to prove that Π_B cannot be solved in FPT time (assuming that Π_A cannot be solved in FPT time).

A *logical circuit* is a directed acyclic graph[6] whose vertices called *gates* partition into *input* gates, *negation* gates, *OR* gates, *AND* gates, *large* gates, and one *output* gate. The negation gate has in-degree 1 and out-degree 1. It outputs the negation of its input. The OR gate (respectively, AND gate) has in-degree 2 and out-degree 1, and outputs the logical or (respectively, logical and) of its two inputs. The large gates are special OR gates or AND gates of arbitrary in-degree. The input gates have in-degree 0, and the output gate has out-degree 0. An assignment of the input gates to 0 or 1, can be seen as a boolean vector. The boolean vector *satisfies* the circuit if it outputs 1. A boolean vector is of weight k if the number of 1s is equal to k. The *height* of a circuit is the maximum length of a directed path from an input gate to the output gate. The *weft* of a circuit is the maximum number of large gates along any directed path in the logical circuit.

W[P] is the class of problems that can be FPT reduced to the problem of finding a satisfying boolean vector of weight k in circuits of constant height. For each positive integer t, **W[t]** is defined similarly to **W[P]** but is restricted to circuits of weft t. Thus, **W[1]** $\subseteq$ **W[2]** $\subseteq \ldots \subseteq$ **W[P]**. A problem Π is *hard* for one of those classes if all the problems of that class FPT reduce to Π, and is *complete* if it also belongs to the class. Again, the class of complete problems in the class **X** is denoted by **X**-c.

The classes **W[1]** and **W[2]** have a significant interest in practice since they classify many standard problems. For instance, k-INDEPENDENT SET and k-CLIQUE are **W[1]**-complete, while k-DOMINATING SET is **W[2]**-complete. Marco Cesati gives a characterization with Turing Machines of **W[1]**, **W[2]**, and **W[P]** [36]. This is particularly use-

[6] Again, the reader will find in Section 1.4 the basic definitions concerning graphs.

ful to show the membership in one of these three classes. Given a Turing Machine M, an input word w, and an integer k, the problem of deciding if there is an accepting run of M on w which takes at most k steps is **W[1]**-complete for single-tape machines, and **W[2]**-complete for multi-tape machines. The former problem is called SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION and the latter SHORT MULTI-TAPE NONDETERMINISTIC TURING MACHINE COMPUTATION. Given a Turing Machine M, an input word w, an integer k, and an integer n encoded in unary, the problem BOUNDED NONDETERMINISM TURING MACHINE COMPUTATION of deciding if there is an accepting run of M on w which takes at most n steps and at most k non-deterministic steps is **W[P]**-complete.

Let us use this characterization to show that k-INDEPENDENT SET is in **W[1]**. We need to construct an FPT reduction from k-INDEPENDENT SET to SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION. An important point is that the size of the alphabet and the number of states of the Turing Machine can depend polynomially in the number of vertices n of the instance of k-INDEPENDENT SET. So, let say that there is one symbol in the alphabet for each vertex in the graph. If k is the parameter of the starting problem, let $k + \alpha\binom{k}{2}$ be the parameter of SHORT NONDETERMINISTIC TURING MACHINE COMPUTATION, where α is a constant to be defined. Now, we informally describe the behavior of the Turing Machine. It guesses the k right vertices in the independent set (recall that it takes only k steps since the alphabet is of size n). For each of the $\binom{k}{2}$ pairs of vertices, the Turing Machine checks in constant time α that there is no edge linking the two vertices.

This reduction is quite general and shows that finding a special (induced) subgraph of size k is in **W[1]**. The only part which differs is checking in time $f(k)$ that you have the desired subgraph. Thus, k-INDEPENDENT SET, k-CLIQUE but also k-DENSEST, k-SPARSEST are in **W[1]**.

XP is the class of problems that can be solved in time $O(n^{f(k)})$, where k is the value of the parameter, f is any computable function, and n is the size of the input. k-INDEPENDENT SET is in **XP**. Indeed, one can enumerate in time n^k the subsets of size k of a graph, and check for each subset if it is an independent set. A more general observation to put k-INDEPENDENT SET in the class **XP** is that **W[P]** $\subseteq$ **XP** and we saw that k-INDEPENDENT SET is in **W[1]**. On the contrary, k-COLORING is not in **XP** since 3-COLORING is already **NP**-complete.

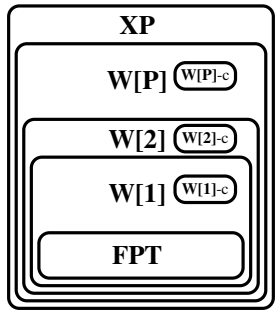


Figure 1.5: The parameterized complexity classes and their inclusions.

1.4 Graphs and Formulas

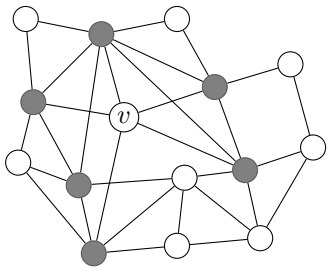
Graphs and formulas are simple and fundamental objects rich enough to formalize nicely most of the problems in algorithmic.

1.4.1 Graphs

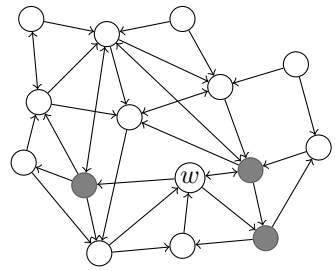
The definitions in this section constitute the very basics of graph theory and can be found, for instance, in the book of Claude Berge [10]. An *undirected graph* G is a pair (V, E) where V is a finite set of elements usually called *vertices*, and $E \subseteq \binom{V}{2}$ is a set of *edges*. A *directed graph* or *digraph* is a pair (V, A) where V is a finite set of vertices, and $E \subseteq V^2$ is a set of *arcs*. A *(directed) path* is a sequence of vertices $u_1, u_2, \ldots, u_r$ such that for each $i \in [1, r-1]$, $\{u_i, u_{i+1}\}$ is an edge ((u_i, u_{i+1}) is an arc). A *cycle* is a path such that $u_1 = u_r$. A *directed acyclic graph* (DAG) is a directed graph with no cycle. As we will deal almost exclusively with undirected graphs, *a graph* will implicitly be undirected.

In an undirected graph (V, E), vertices u and v are *neighbors* if $\{u, v\} \in E$. We also say that v is a neighbor of v. In a directed graph (V, A), v is a neighbor u if $(u, v) \in A$. The set of neighbors of v in a(n un)directed graph G is denoted by $N(v)$, or $N_G(v)$ in case of ambiguity, and is called the *neighborhood* of v. $N[v] = N(v) \cup \{v\}$ is the *closed neighborhood* of v. For any subset of vertices $U \subseteq V$, $N(U)$ is defined as $\bigcup_{u \in U} N[u]$ and $N(U) = N[U] \setminus U$.

In what follows, we only deal with undirected graphs. A *subgraph* of $G = (V, E)$



(a) An undirected graph. Vertices are represented by circles, and edges by line segments. The neighbors of vertex v are filled.



(b) A directed graph. An arc (s, t) is represented by an arrow from s to t. The neighbors of vertex w are filled.

Figure 1.6: The neighborhood in graphs.

is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. An *induced subgraph* of $G = (V, E)$ is a subgraph $G' = (V', E')$ of G such that $V' \subseteq V$ and each edge in E having both endpoints in V', is also in E'. The *subgraph induced by U* denoted by $G[U]$ is the induced subgraph of G of the form (U, E'). The *complementary* $\overline{G}$ of a graph $G = (V, E)$ is defined by $\overline{G} = (V, \overline{E})$ where $\overline{E} = \{\{u, v\} \mid \{u, v\} \notin E\}$. We denote by $V(G)$ the vertex set of the graph G and $E(G)$ its edge set. An *independent set* $S \subseteq V(G)$ is a subset of vertices such that $\forall u, v \in S$, $\{u, v\} \notin E(G)$. A *clique* $S \subseteq V(G)$ is a subset of vertices such that $\forall u, v \in S$, $\{u, v\} \in E(G)$. A *dominating set* $S \subseteq V(G)$ is a subset of vertices such that $\forall u \in V(G) \setminus S$, there is a vertex v satisfying $\{u, v\} \in E(G)$. A *vertex cover* $S \subseteq V(G)$ is a subset of vertices such that $\forall e = \{u, v\} \in E(G)$, $u \in S \vee v \in S$. A *forest* is an acyclic graph, that is a graph with no cycle, and a *tree* is a connected forest.

1.4.2 Formulas

A *boolean variable* is a variable that can get value True or False. A *boolean formula* is either a boolean variable, either the negation ($\neg$) of a boolean formula, either the conjunction ($\wedge$) of two boolean formulas, or the disjunction ($\vee$) of two boolean formulas. The semantics of $\neg$, $\vee$ and $\wedge$ is as expected. As $\wedge$ and $\vee$ are associative, we can extend these two operators to any arity. A *literal* is a variable or the negation of a variable, a *clause* is a disjunction of literals, and a conjunctive normal form

(CNF for short) formula, is a boolean formula which is the conjunction of *clauses*. Usually, the clauses of the conjunction are separated by commas. $\psi = x_1 \vee \neg x_2 \vee x_4 \vee x_5, x_2 \vee \neg x_3 \vee x_5, x_3 \vee \neg x_4 \vee \neg x_5$ is an example of a CNF formula. An *assignment* is a function mapping each variable to True or to False. A *truth assignment* is an assignment which makes the formula to be valued at True. The formula is said *satisfiable* if there is a truth assignment. SAT is the problem of deciding if a CNF formula is satisfiable. In Section 1.3, we mentioned that this problem is **NP**-complete. Given a CNF formula ϕ and a variable x appearing in ϕ, $\phi[x \leftarrow \top]$ (respectively, $\phi[x \leftarrow \bot]$) is the CNF formula obtained by setting x to True (respectively, to False) in ϕ and simplifying. The simplification consists of removing the clauses where the literal that contains x is satisfied, and removing the unsatisfied literals containing x (if it has the effect of emptying a clause, this clause is replaced by False). For instance, $\psi[x_2 \leftarrow \bot] = \neg x_3 \vee x_5, x_3 \vee \neg x_4 \vee \neg x_5$

A *quantified boolean formula* (QBF formula for short) is any formula of the form $Qx_1Qx_2 \dots Qx_r\phi$ where ϕ is a CNF formula, $\forall i \in [1, r]$, x_i appears in ϕ, $Q \in \{\exists, \forall\}$, and the semantics of the quantifier Q is as expected: $\exists x\phi$ is logically equivalent to $\phi[x \leftarrow \bot] \vee \phi[x \leftarrow \top]$ and $\forall x\phi$ is logically equivalent to $\phi[x \leftarrow \bot] \wedge \phi[x \leftarrow \top]$. QBF is the problem of deciding if a QBF formula is satisfiable. In Section 1.3, we mentioned that this problem is **PSPACE**-complete. It remains **PSPACE**-complete if we impose that the existential quantifiers $\exists$ and the universal quantifiers are alternated, and that all the variables in the formula are quantified. In that case, the problem is to decide if the formula is equivalent to True or to False.

1.5 Motivation and Organization

The next chapter is based on the following two papers and excerpt of a manuscript:

- [15] Édouard Bonnet and Vangelis Th. Paschos. Parameterized (in)approximability of subset problems. *Oper. Res. Lett.*, 42(3):222–225, 2014
- [17] Édouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Émeric Tourniaire. Multi-parameter complexity analysis for constrained size graph problems: Using greediness for parameterization. In *IPEC*, pages 66–77, 2013
- [19] Édouard Bonnet, Vangelis Th. Paschos, and Florian Sikora. Multiparameterizations for max k-set cover and related satisfiability problems. *CoRR*, abs/1309.4718, 2013

This chapter entirely revolves around a simple idea occasionally called *necessary sets* that knowing that in a local (bounded) area of elements, there exists an optimal solution taking at least one of this element, is a precious help, and especially so to design FPT algorithms. In Section 2.1, we define *subset problems* which are a bit more restrictive that what is sometimes informally called subset problems. We extend the notion of *safe approximation* [73] to the more general *intersective approximation*. In Section 2.2, we show that *subset problems* are unlikely to admit FPT intersective approximation. In Section 2.3, we present a technique that we call *greediness-for-parameterization*. In Section 2.4, we introduce a special subclass of *size-constrained* graph problems, the *local graph partitioning problems*. In Subsection 2.4.1, we solve the *degrading local partitioning graph problem* in time $O^*((\Delta+1)^k)$, improving on the *random separation* technique, and in Subsection 2.4.2, we solve the *local partitioning graph problem*, in general, in time $O^*((2k\sqrt{\Delta})^{2k})$, improving on the *random separation* technique whenever $k = 2^{o(\Delta)}$. In Section 2.5, we show some additional properties for two specific *local partitioning graph problems*, namely MAX $(k, n-k)$-CUT and MIN $(k, n-k)$-CUT, for both exact and approximate computations. In Section 2.6, we extend *greediness-for-parameterization* to *size-constrained* set and satisfiability problems. We mainly show that MAX SAT-k is in **FPT** when parameterized by the number of satisfied clauses. In Section 2.7, we show that the bipartiteness of a graph separates the parameterized complexity the minimization and the maximization versions of some main *size-constrained* problems. More precisely, in Subsection 2.7.1, we show that MAX k-VERTEX COVER and MAX $(k, n-k)$-CUT restricted to bipartite graphs are in **FPT**, and in Subsection 2.7.2, we show that MIN k-VERTEX COVER and MIN $(k, n-k)$-CUT restricted to bipartite graphs remain **W[1]**-complete. Whereas, those four problems are **W[1]**-complete in general graphs. In Section 2.8, we conclude Chapter 2 by relating recent advances on the topic and we give some open problems.

The third chapter is based on the following paper and manuscript:

- [16] Édouard Bonnet, Bruno Escoffier, Eun Jung Kim, and Vangelis Th. Paschos. On subexponential and fpt-time inapproximability. In *IPEC*, pages 54–65, 2013
- [14] Édouard Bonnet and Vangelis Th. Paschos. Sparsification and subexponential approximation. *CoRR*, abs/1402.2843, 2014

As we have mentioned in the previous sections of this introduction, there are some bad news for approximation (inapproximability) and for parameterized complexity

(W-hardness).

In this chapter, we try and combine those bad results, and show some inapproximability results in FPT time and in subexponential time.

In Section 3.1, we present a conjecture called LPC (for Linear PCP Conjecture), which is more an open question than a widely believed conjecture (as ETH is), and a novel notion: problems satisfying APETH, a generalization of ETH to constant-ratio approximations. Then, we overview the main results of the two next sections. In Section 3.2, we show some consequences of LPC combined with ETH to inapproximability in superpolynomial time. In Section 3.3, we formalize the notion of an *approximation preserving sparsifier* and show that many optimization problems are equivalent with respect to APETH (that is, if any of the problem satisfies APETH, then they all do). We finally link APETH to a credible hypothesis of paramaterized inapproximability.

In Section 3.4, we state a recent and almost tight inapproximability result in subexponential time for MAX INDEPENDENT SET [37]. Our objective is then to adapt the sparsifier of Section 3.3 to extend this inapproximability to other problems.

We construct such a sparsifier in Section 3.5. In Section 3.6, we present yet another sparsifier which runs in polynomial time. In Section 3.7, we obtain some results based on the sparsifiers of the two previous sections, the result [37], and the inapproximability in subexponential time of MIN VERTEX COVER of Proposition 10. We make further remarks concerning *sparsifiers* and *sparsification* in Section 3.8.

The fourth chapter is based on the following papers:

- [20] Édouard Bonnet, Florian Jamain, and Abdallah Saffidine. On the complexity of trick-taking card games. In Francesca Rossi, editor, *23rd International Joint Conference on Artificial Intelligence (IJCAI)*, Beijing, China, August 2013. AAAI Press
- [18] Édouard Bonnet, Florian Jamain, and Abdallah Saffidine. Havannah and TwixT are PSPACE-complete. In *Computers and Games*, pages 175–186, 2013

Whenever the reader likes to refresh his/her mind from reading the previous two chapters, he may find some results about the complexity of games in this last, and hopefully playful and interesting, chapter. There, we mainly show that the perfect information card play in Bridge and the board game Havannah are **PSPACE**-complete to play optimally.

We briefly motivate why we consider those two games. First, Bridge is a popular game played by 200 million people worldwide. Second, the purity of its card play[7] makes it an interesting case study for other card games. There are some programs that play Bridge quite well and some established polynomial fragments [126, 127], but no hardness result for the general problem was known. Besides, the complexity of the card play in Bridge was posed as a significant open problem in the thesis of Robert Hearn dedicated to games and puzzles [76]. Havannah is more confidential a game but there have been several papers on this game [122, 97, 120]. Yet its computational complexity was not known, while the complexity of some games of roughly the same notoriety, like Amazons, was [69].

In Section 4.1, we formalize the card play in Bridge as a decision problem and introduce some notation and definitions on *trick taking card games*. In Section 4.2, we show that when the number of hands is unbounded, the problem is **PSPACE**-complete, even if the number of cards per suit is 2. In Section 4.3, we show that, even if the number of hands is bounded (by 6), the problem remains **PSPACE**-complete. In Section 4.4, we introduce *connection games*. In Section 4.5, we show that HAVANNAH is **PSPACE**-complete and in Section 4.6 we show that TWIXT is **PSPACE**-complete. In Subsection 4.7.1, we give some perspectives to the analysis *trick-taking card games* and in Subsection , we do the same for *connection games*.

[7] There is basically just one rule: following a suit.

2 FPT Algorithms and Approximation

2.1 Subset Problems

In Section 2.1 and Section 2.2, we discuss approximability in FPT time for a large class of *subset problems* where a feasible solution S is a subset of the input data. The class handled encompasses many well-known graph, set, and satisfiability problems such as MIN VERTEX COVER, MIN SET COVER, MAX INDEPENDENT SET. We introduce the notion of *intersective* approximability that generalizes the one of *safe* approximability introduced in [73] and we show parameterized inapproximability results for many subset problems.

Parameterized approximation aims at bringing together two very active fields of theoretical computer science, polynomial approximation and parameterized complexity. We say that a minimization (respectively maximization) problem Π, together with a parameter k, is *parameterized r-approximable*, if there exists an FPT time algorithm which computes a solution of size at most (respectively at least) rk whenever the input instance has a solution of size at most (respectively at least) k, otherwise, it outputs an arbitrary solution. This line of research was initiated by three independent works [53, 32, 40]. Daniel Marx wrote an interesting overview on that topic [102].

Here, we define *subset problems* as follows:

Definition 3. A problem Π is called a *subset problem*, if the following conditions hold:

- feasible solutions for Π are subsets of elements encoding its instances that verify some specific property;
- Π is *decomposable*, that is, for any instance I and element e of the encoding of I, there exists an instance $I(e)$ computable in polynomial time and satisfying $|I(e)| \leqslant |I|$ such that if there is a solution S for I containing e, then $S \setminus \{e\}$ is a solution for $I(e)$.

The existence of one instance encoding for Π satisfying Definition 3 is sufficient for Π to be a subset problem; thus, Definition 3 does not depend on the encoding. In what follows, we give several examples of graph, set, and satisfiability subset problems in order to clarify the notion of decomposability of the second item in Definition 3. We recall (see Section 1.4) the standard notation on graphs: $G[S]$ is the subgraph of G induced by S, $N(v)$ is the set of neighbors of v, $N[v] = N(v) \cup \{v\}$, V usually denotes the set of vertices of a graph, E the set of its edges, $n = |V|$, and $m = |E|$.

- Let a graph be encoded by the set of its vertices and the set of its edges.
  - MAX INDEPENDENT SET is decomposable with $G(v) = G[V \setminus N[v]]$ since any independent set in G containing v, is an independent set in $G[V \setminus N[v]]$ combined with the vertex v.
  - MAX CLIQUE is decomposable with $G(v) = G[N(v)]$. Indeed, any clique containing v is entirely included in $N(v)$.
  - MIN VERTEX COVER is decomposable with $G(v) = G[V \setminus \{v\}]$.
  - A *generalized dominating set* is a subset of vertices which dominates an imposed subset of vertices $V' \subseteq V$. Given a subset $V' \subseteq V$, GENERALIZED MIN DOMINATING SET aims at finding a generalized dominating set of minimum size. MIN DOMINATING SET is a special case of GENERALIZED MIN DOMINATING SET with $V' = V$. GENERALIZED MIN DOMINATING SET is decomposable with $(G, V')(v) = (G[V \setminus \{v\}], V' \setminus N[v])$.
  - A *feedback vertex set* is a vertex-subset S such that $G[V \setminus S]$ is a forest. MIN FEEDBACK VERTEX SET is decomposable with $G(v) = G[V \setminus \{v\}]$.
  - In GENERALIZED k-DENSEST, given a graph G and a subset V' of V, one looks for a superset of V' with k vertices inducing a subgraph of G

with a maximum number of edges. k-DENSEST is a special case with $V' = \emptyset$. GENERALIZED k-DENSEST is decomposable with $(G, V')(v) = (G, V' \cup \{v\})$. The minimization version GENERALIZED k-SPARSEST is also a subset problem.

- In GENERALIZED MAX $(k, n-k)$-CUT, given a graph G and a set V' of vertices, one looks for a superset S of V' with k vertices with a maximum number of edges between S and $V \setminus S$. It is decomposable with $(G, V')(v) = (G, V' \cup \{v\})$, so is its minimization version GENERALIZED MIN $(k, n-k)$-CUT.

- Let a set system be encoded by a ground set X and a collection $\mathcal{S}$ of its subsets.
  - A *set cover* is subcollection of $\mathcal{S}$ that covers C. MIN SET COVER is decomposable with $(\mathcal{S}, X)(S) = ((\mathcal{S} \setminus \{S\})_{|X \setminus S}, X \setminus S)$, where $A_{|B}$ is the projection to B of all the subsets in A.
  - In MAX k-SET COVER, one looks for a set of k subsets that covers a maximum number of elements. It is decomposable in the same way as MIN SET COVER.

- Let a CNF formula ϕ be encoded by its variables X and its clauses C.
  - SAT asks for determining a set of variables (if any) whose assignment to *true* satisfies the formula (the other variables all being assigned to false). It is decomposable with $\phi(x) = C[x \leftarrow \top]$ (i.e., the set of clauses satisfied when setting variable x to *true*). Analogous formulations make that MAX SAT or MIN SAT problems are subset problems.
  - For the same reason, SAT-k, asking for determining a truth assignment setting at most k variables to *true*, and its optimization counterpart MAX SAT-k are subset problems.

In what follows, we will focus mainly on *optimization* subset problems.

2.2 Intersective Approximability of Subset Problems

In this section, we study parameterized approximability of subset problems. For that purpose, we introduce a new approximability framework called *intersective approximation*. This framework is quite natural and really fits the class of subset problems.

It is important to note that *intersectivity* generalizes the model of safe approximation, introduced in a paper by Guo et al [73]. An approximation is said to be *safe*, if it produces solutions containing an optimal solution. The safe approximation can be used to get strong inapproximability results. For instance, it is shown in [73] that a safe $c \log n$-approximation, for any $c > 0$, for GENERALIZED MIN DOMINATING SET, can be turned into an exact FPT algorithm, contradicting **FPT** $\neq$ **W[2]**. However, the safe approximation only captures minimization problems and is very restrictive. The intersective approximation just requires that an optimal solution intersects with the approximate solution, and therefore, also fits the maximization problems.

Definition 4. A ρ-approximation algorithm $\mathtt{A}$ is said to be *intersective* for a problem Π if, when running on any instance I of Π, it computes a ρ-approximate solution $\mathtt{A}(I)$ and there exists an optimal solution S of I such that $\mathtt{A}(I) \cap S \neq \emptyset$.

In what follows, we prove that intersective approximation in FPT time is very unlikely for $\mathbf{W}[\cdot]$-hard subset problems, since such an approximation can be transformed into an exact FPT algorithm. However, as we will see, there is an important difference between minimization and maximization problems since, for the former, this transformation can be done only if intersective FPT approximation ratio is under a certain approximation level, while, for the latter, such transformation is independent on the level of the ratio.

We first prove the following more general theorem where, given an instance I of a subset problem Π, we denote by k the optimal value of I.

Theorem 1. *Let Π be an optimization subset problem. Then:*

- *if Π is a minimization problem and admits an intersective r-approximation computed in time $O(f(n,k))$, for some $r > 1$ and some positive increasing function f, then Π can be optimally solved in time $O((rk)^k f(n,k))$;*
- *if Π is a maximization problem, any intersective approximation computed in time $O(f(n,k))$, for some positive increasing function f can be transformed into an exact algorithm running in time $O(k^k f(n,k))$.*

Proof. Consider some minimization problem Π, an intersective FPT approximation algorithm $\mathtt{A}$ for Π achieving approximation ratio r, and let I be any instance of Π. Compute an intersective approximation $S = \mathtt{A}(I) = \{e_1, \ldots, e_{|S|}\}$ for I. If $|S| > rk$, then answer that I is a NO-instance.

Otherwise, branch on the at most rk instances $I(e_1), \ldots, I(e_{|S|})$ (since Π is decomposable, all these instances are well-defined). For all these instances, compute an r-approximation and keep the recursion on. When k elements have been taken in the solution, stop the recursion.

We claim that the best solution found at a leaf of the branching tree is an optimal solution. Indeed, starting from the root one can, by definition of intersective approximation, move to a child which has taken an element e contained in an optimal solution.

The branching tree has depth k since, at each step, one element is added in the solution, and arity bounded by rk. Hence, the number of its nodes is bounded by $2(rk)^k$. On each node, some $O(f(n,k))$ computation is done. So, the overall complexity is $O((rk)^k f(n,k))$.

We now handle maximization problems. Consider some maximization problem Π, an intersective approximation algorithm $\mathtt{A}$ for Π (achieving any approximation ratio) and let I be any instance of Π. Compute an intersective approximation $S = \mathtt{A}(I)$ for I. If $|S| \geqslant k$, answer YES and output this solution. Otherwise $|S| < k$, and the exact branching algorithm of the previous paragraph runs in time $O(k^k f(n,k))$. If one of the leaves of the branching tree contains a feasible solution, answer YES and output this solution. Otherwise, anwser NO. □

Theorem 1 has the following important corollary.

Theorem 2. *Let Π be a subset optimization problem. Then:*

- *if Π is a minimization problem and admits an FPT intersective $(g(k)\log n)$-approximation for some function g, then Π admits an exact FPT algorithm;*
- *if Π is a maximization problem and admits an FPT intersective approximation, then Π admits an exact FPT algorithm.*

Proof. For minimization problems just observe that, when $r = g(k)\log n$, the number of nodes in the branching tree is bounded by $2(kg(k))^k(\log n)^k$ and, on each node, some FPT computation is done, bounded by, say, $f(k)p(n)$. So, the overall complexity is $2(kg(k))^k f(k)(\log n)^k p(n)$, which is FPT, considering that $(\log n)^k$ is FPT with respect to k [119].

For maximization problems, the proof comes directly from Theorem 1 and it can be easily seen that no specific approximation guarantee is required for them. □

Remark 1. *The result of Theorem 2 for the case of minimization problems works, in fact, even if we consider approximation ratios $O(g(k)(\log n)^{h(k)})$ for* any *(increasing) functions h and g. Indeed, $O((g(k)(\log n)^{h(k)}k)^k f(k))$ (where f is the complexity of the FPT intersective algorithm) is $O(F(k)p(n))$, for some function F and polynomial p [119].*

Based upon Theorem 2, the following holds for the intersective FPT approximability of $\mathbf{W}[\cdot]$-hard problems.

Corollary 1. *Unless the **W**-hierarchy collapses at some level:*

- *no FPT intersective $(g(k)\log n)$-approximation exists for $\mathbf{W}[\cdot]$-hard minimization problems, for any positive increasing function g;*
- *no FPT intersective r-approximation (for any r) exists for $\mathbf{W}[\cdot]$-hard maximization problems.*

In particular:

- *unless **FPT** = **W[2]**, no FPT intersective $(g(k)\log n)$-approximation exists for either* MIN SET COVER*, or* GENERALIZED MIN DOMINATING SET*, for any positive increasing function g;*
- *unless **FPT** = **W[1]**, no FPT intersective approximation exists either for* MAX INDEPENDENT SET*, or for* MAX CLIQUE*.*

Note that the negative result for MIN SET COVER above, transfers also to MIN DOMINATING SET through the classical reduction from MIN SET COVER to MIN DOMINATING SET [110] which preserves both the approximation and the parameter.

The *fixed-cardinality* (or *size-constrained* or *cardinality-constrained*) graph problems are also subset problems. Such problems are defined on some graph $G(V, E)$ and integer k and feasible solutions are subsets $V' \subseteq V$ of size exactly k.

Notable representatives of such problems are MAX k-VERTEX COVER, GENERALIZED k-DENSEST and its minimization version GENERALIZED k-SPARSEST, GENERALIZED MAX $(k, n-k)$-CUT and its minimization version GENERALIZED MIN $(k, n-k)$-CUT, and MAX k-SET COVER. An intersective approximation for fixed-cardinality problems implies that k' elements from the solution, $0 < k' \leqslant k$, are common to both the optimum and the approximate solution. Then, Theorem 2 derives the following.

Corollary 2. *Unless **W[1]** $=$ **FPT**, no intersective FPT approximation algorithm can exist for any of the problems* MAX k-VERTEX COVER, GENERALIZED k-DENSEST, GENERALIZED k-SPARSEST, GENERALIZED MAX $(k, n-k)$-CUT, GENERALIZED MIN $(k, n-k)$-CUT, MAX k-SET COVER. *Unless **W[2]** $=$ **FPT**, no intersective FPT approximation algorithm can exist for* MAX k-SET COVER.

The intersective approximation allowed us to prove negative results on the possibility of subset problems of being intersectively approximable. The intersective approximation importantly relaxes and generalizes the safe approximation of [73] since:

- it reflects the realistic behavior of an approximation algorithm.
- it encompasses maximization problems.

Finally, the intersective approximability can be extended to several problems that are not subset problems per se. We sketch such an extension to coloring problems. A solution for a k-coloring can be seen as k sets $S_1, \ldots, S_k$ where S_i is the set of vertices (or edges) receiving color i. A ρ-intersective approximation to a k-coloring problem can be defined as an h-coloring $S'_1, \ldots, S'_h$ such that there exists an optimal solution $S_1, \ldots, S_k$ with $k \geqslant h/\rho$ and two integers i, j satisfying $S_i = S'_j$. Under this extended definition of intersective approximability, the following can be proved similarly to Theorem 2.

Corollary 3. *If a k-coloring problem Π has an FPT intersective $(c \log n)$-approximation (as extended just above) for some constant $c > 0$, then Π admits an exact FPT algorithm.*

2.3 Greediness-for-Parameterization

Here, we present a technique for obtaining parameterized results for *local graph partitioning problems* defined in Section 2.4. This technique is suited to *size-constrained*

problems, where the solution should be a subset of exactly k elements[1]. We describe the general idea in the case of a graph. As usual, Δ is the degree of the graph.

> Perform a branching with respect to a vertex chosen upon some greedy criterion. This criterion could consist of choosing some vertex v which optimizes the number of edges added to the solution under construction. Without branching, such a greedy criterion is not optimal. However, if at each step, either vertex v, or some of its neighbors (more precisely, a vertex at bounded distance from v) are a good choice (they are in an optimal solution), then a branching rule on neighbors of v leads to a branching tree whose size is bounded by a function of k and Δ, and at least one leaf of which is an optimal solution.

In order to prove the soundness of our algorithms, we will hybridize the solutions that we compute with a given optimal solution. Therefore, we require the following vocabulary. A *partial* solution is a subset of a (*complete*) solution. A *branching* algorithm is a recursive algorithm. Its execution on an instance I can be seen as a tree, called *branching tree*. In this tree, each *node* is labeled with a subinstance of I together with a partial solution, or more generally with some data maintained by the algorithm. The *root* is labeled with I and a *leaf* is a subinstance that causes the branching algorithm to stop. At a leaf, a complete solution is computed and returned. When identifying a node v to its label (a subinstance), the *children* of a subinstance are the subinstances which label the children of v in the branching tree. A node v of the branching tree is *in accordance* with a solution S if its corresponding partial solution is included in S. A node v of the branching tree *deviates* from a solution S if it is in accordance with S but none of its children are in accordance with S.

2.4 Local Graph Partitioning Problems

In a *local graph partitioning problem* the input is a graph $G = (V, E)$ and two integers k and p. Feasible solutions are subsets $V' \subseteq V$ of size exactly k. The value of a solution is a linear combination of the sizes of edge subsets $E(V')$ and $E(V', V \subseteq V')$, and the objective is to determine whether there exists a solution of

[1] Here, *elements* is meant in the most general sense.

value at least (or, at most) p. MAX k-VERTEX COVER, k-DENSEST, k-SPARSEST, MAX $(k, n-k)$-CUT and MIN $(k, n-k)$-CUT are local graph partitioning problems. They constitute a subclass of problems known as *fixed-cardinality* (or *size-constrained* or *cardinality-constrained*) problems. The parameterized complexity of those problems have mainly been studied in [29, 52], with respect to parameter k, and have been shown **W[1]**-hard.

In Section 2.5 we study the fixed-cardinality problems MAX and MIN $(k, n-k)$-CUT, parameterized by p. We prove that the former is in **FPT**. The latter has been proven in **FPT** soon after this work has been published [46]. In order to handle MAX $(k, n-k)$-CUT we first show that when $p \leqslant k$ or $p \leqslant \Delta$, the problem can be solved in polynomial time. So, the only *non-trivial* case occurs when $p > k$ and $p > \Delta$. That case is handled by greediness-for-parameterization. Unfortunately, this method gives the inclusion of MIN $(k, n-k)$-CUT in **FPT** only for some particular cases. In a technical report [64], the following problem is considered: given a graph G and two integers k, p, determine whether there exists a set $V' \subset V$ of size *at most* k such that at most p edges have exactly one endpoint in V'. They prove that this problem is in **FPT** with respect to p. Looking for a set of size at most k is really different from looking for a set of size exactly k. For instance, when $k = n/2$, the former is MIN CUT which is in **P**, while the latter is MIN BISECTION which is **NP**-hard.

In Section 2.5.3, we revisit the parameterization by k with the point of view of approximation. We prove that, although **W[1]**-hard, MAX $(k, n-k)$-CUT has an FPT approximation schema with respect to k and MIN $(k, n-k)$-CUT a randomized FPT approximation schema. These results exhibit two problems which are W-hard with respect to a given parameter but which become FPT for a *good* approximation (which is not feasible in polynomial time). Another problem having a similar behavior is another fixed-cardinality problem: MAX k-VERTEX COVER, where one has to find the subset of k vertices covering the greatest number of edges [102]. The existence of problems having this behavior with respect to the value of the solution is an open question asked in [102]. A polynomial approximation of MIN $(k, n-k)$-CUT has been studied in [60] where it is proven that, if $k = O(\log n)$, then the problem admits a randomized polynomial time approximation schema, while, if $k = \Omega(\log n)$, then it admits an approximation ratio $(1 + \frac{\varepsilon k}{\log n})$, for any $\varepsilon > 0$. The approximation of MAX $(k, n-k)$-CUT has been studied in several papers and a ratio 1/2 is achieved in [1] (slightly improved with a randomized algorithm in [59]), for all k.

First, we formally define the class of local graph partitioning problems.

Definition 5. In a local graph partitioning problem, the input is a graph $G = (V, E)$ and two integers k and p. The goal is to find a subset $V' \subseteq V$ of size exactly k such that $\mathrm{val}(V') = \alpha_1|E(V')| + \alpha_2|E(V', V \setminus V')| \leqslant p$ for a minimization problem ($\geqslant p$ for a maximization problem) where $\alpha_1, \alpha_2 \in \mathbb{R}$.

In the optimization versions, the threshold p is not given, and one just wants to optimize (minimize or maximize) $\mathrm{val}(V')$. We observe that $\alpha_1 = 1, \alpha_2 = 0$ corresponds to k-DENSEST and k-SPARSEST, $\alpha_1 = 0, \alpha_2 = 1$ to MAX $(k, n-k)$-CUT or MIN $(k, n-k)$-CUT, and $\alpha_1 = \alpha_2 = 1$ to MAX k-VERTEX COVER or MIN k-VERTEX COVER.

As a local graph partitioning problem is entirely defined by α_1, α_2 and $\mathrm{goal} \in \{\min, \max\}$ we will unambiguously denote by $\mathcal{L}(\mathrm{goal}, \alpha_1, \alpha_2)$ the corresponding problem. In what follows, *local problem* is a shortened form of *local graph partitioning problem*, k denotes the size of a solution and p its value. A partition into k and $n-k$ vertices is completely defined by a subset V' of size k. We will therefore consider V' to be the solution. A *partial solution* T is a subset of V' with less than k vertices. Similarly to the value of a solution, we define the value of a partial solution, and denote it by $\mathrm{val}\,(T)$.

Informally, we devise algorithms for local problems that add vertices to an initially empty set T (for *taken* vertices) and stop when T is of size k. A vertex introduced in T is irrevocably introduced there and will be not removed later.

Definition 6. Given a local graph partitioning problem $\mathcal{L}(\mathrm{goal}, \alpha_1, \alpha_2)$, the *contribution* of a vertex v within a partial solution T (such that $v \in T$) is defined by $\delta(v, T) = \frac{1}{2}\alpha_1|E(\{v\}, T)| + \alpha_2|E(\{v\}, V \setminus T)|$.

Note that the value of any (partial) solution T satisfies $\mathrm{val}\,(T) = \Sigma_{v \in T}\delta(v, T)$. One can also remark that $\delta(v, T) = \delta(v, T \cap N(v))$. Function δ is called the *contribution function* or simply the *contribution* of the corresponding local problem.

Definition 7. Given a local graph partitioning problem $\mathcal{L}(\mathrm{goal}, \alpha_1, \alpha_2)$, a contribution function is said to be *degrading* if for every v, T and T' such that $v \in T \subseteq T'$, $\delta(v, T) \leqslant \delta(v, T')$ for $\mathrm{goal} = \min$ (respectively, $\delta(v, T) \geqslant \delta(v, T')$ for $\mathrm{goal} = \max$).

We observe that for a maximization problem, a contribution function is degrading if and only if $\alpha_2 \geqslant \alpha_1/2$ ($\alpha_2 \leqslant \alpha_1/2$ for a minimization problem). In particular,

MAX k-VERTEX COVER, k-SPARSEST, and MAX $(k, n-k)$-CUT have a degrading contribution function.

2.4.1 Degrading Local Graph Partitioning Problems

Theorem 3. *Every local partitioning problem having a degrading contribution function can be solved in* $O^*((\Delta+1)^k)$.

Proof. With no loss of generality, we carry out the proof for a minimization local problem $\mathcal{L}(\min, \alpha_1, \alpha_2)$. We recall that T is a partial solution that will eventually be a feasible solution. Consider the following algorithm DLGPP (T, k) branching upon the closed neighborhood $N[v]$ of a vertex v minimizing the greedy criterion $\mathrm{val}(T \cup \{v\})$:

Algorithm 6: A description of the algorithm DLGPP.

Input: A graph $G = (V, E)$, an integer k, and a triple $\mathrm{goal} \in \{\min, \max\}$, α_1, α_2 defining a degrading local graph partitioning problem $\mathcal{L}(\mathrm{goal}, \alpha_1, \alpha_2)$.
Output: A set of k vertices $T \subseteq V$ optimizing $\mathrm{val}(T)$.
set $T = \emptyset$;
DLGPP (T, k):
if $k > 0$ **then**
  pick the vertex $v \in V \setminus T$ minimizing $\mathrm{val}(T \cup \{v\})$;
  for *each vertex* $w \in N[v] \setminus T$ **do**
    run DLGPP $(T \cup \{w\}, k-1)$;
  end
end
else
  $(k = 0)$ store the feasible solution T;
end
return *the best among the solutions stored*;

The branching tree of DLGPP has depth k, since we add one vertex at each recursive call, and arity at most $\max_{v \in V} |N[v]| = \Delta + 1$, where $N[v]$ denotes the closed neighborhood of v. Thus, the algorithm runs in $O^*((\Delta+1)^k)$.

For the optimality proof, we use a classical hybridization technique between some optimal solution and the one solution computed by DLGPP.

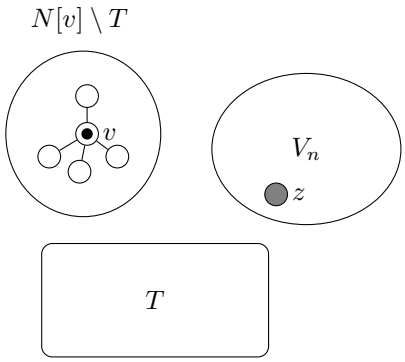


Figure 2.1: Situation of the input graph at a deviating node of the branching tree. Vertex v can replace z since $N[v] \setminus T$ and V_n are disjoint, and the contribution of a vertex can only decrease while the solution is being built.

Consider an optimal solution V'_{opt} different from the solution V' computed by DLGPP. A node s of the branching tree has two characteristics: the partial solution $T(s)$ at this node (denoted simply T if no ambiguity occurs) and the vertex chosen by the greedy criterion $v(s)$ (or simply v). We recall that a node s of the branching tree is said *in accordance* with the optimal solution V'_{opt} if $T(s) \subseteq V'_{\text{opt}}$ and that a node s *deviates* from the optimal solution V'_{opt} if none of its sons is in accordance with V'_{opt}.

We start from the root of the branching tree and, while it is possible, we move to a son in accordance with V'_{opt}. At some point we reach a node s which deviates from V'_{opt}. We set $T = T(s)$ and $v = v(s)$. Intuitively, T corresponds to the shared choices between the optimal solution and DLGPP made along the branch from the root to the node s of the branching tree. Setting $V_n = V'_{\text{opt}} \setminus T$, V_n does not intersect $N[v]$, otherwise s would not be deviating.

Choose any $z \in V_n$ and consider the solution induced by the set $V_e = V'_{\text{opt}} \cup \{v\} \setminus \{z\}$. As v has no neighbors in V_n, the contribution of v does not change when $V_n \setminus \{z\}$ is added to $T \cup \{v\}$. Moreover, as the problem is degrading, the contribution of z can only worsen. So, $\text{val}(V_e) - val(V'_{\text{opt}}) \leqslant \text{val}(T \cup \{v\}) - \text{val}(T \cup \{z\}) \leqslant 0$ (by choice of v). In fact, by optimality of V'_{opt}, $\text{val}(V_e) = val(V'_{\text{opt}})$.

Thus, by repeating this argument at most k times, going down in the branching tree, we conclude that the solution computed by DLGPP is as good as V'_{opt}. □

Corollary 4. MAX k-VERTEX COVER, k-SPARSEST, *and* MAX $(k, n-k)$-CUT *can be solved in* $O^*((\Delta+1)^k)$.

Indeed, those local graph partitioning problems have a degrading contribution.

2.4.2 General Local Graph Partitioning Problems

Theorem 4. *Every local partitioning problem can be solved in* $O^*((2k\sqrt{\Delta})^{2k})$.

Proof. Once again, with no loss of generality, we prove the theorem in the case of minimization, i.e., $\mathcal{L}(\min, \alpha_1, \alpha_2)$. The proof of Theorem 4 involves an algorithm fairly similar to `DLGPP` but instead of branching on a vertex chosen greedily and its neighborhood, we will branch on sets of vertices inducing connected components (also chosen greedily) and the neighborhood of those sets.

Let us first state the following straightforward lemma that bounds the number of induced connected components and the running time to enumerate them.

Lemma 1. *(Lemma 2 in [92]) One can enumerate the connected induced subgraphs of size up to* k *in time* $O^*((4\Delta)^k)$.

Consider now the following algorithm `LGPP`(T, k):

The branching tree of `LGPP` has size $O(k^{2k})$. Computing the S_i in each node takes time $O^*((4\Delta)^k)$ according to Lemma 1. Thus, the overall running-time of the algorithm is $O^*((2k\sqrt{\Delta})^{2k})$.

For the optimality of `LGPP`, we use the following lemma.

Lemma 2. *Let A,B,X,Y be pairwise disjoint sets of vertices such that* $\mathrm{val}\,(A \cup X) \leqslant \mathrm{val}\,(B \cup X)$, $N[A] \cap Y = \emptyset$ *and* $N[B] \cap Y = \emptyset$. *Then,* $\mathrm{val}\,(A \cup X \cup Y) \leqslant \mathrm{val}\,(B \cup X \cup Y)$.

Proof of Lemma 2. Simply observe that $\mathrm{val}\,(A \cup X \cup Y) = \mathrm{val}\,(Y) + \mathrm{val}\,(A \cup X) - 2\alpha_2|E(X,Y)| + \alpha_1|E(X,Y)| \leqslant \mathrm{val}\,(Y) + \mathrm{val}\,(B \cup X) - 2\alpha_2|E(X,Y)| + \alpha_1|E(X,Y)| = \mathrm{val}\,(B \cup X \cup Y)$. □

We now show that `LGPP` is sound, using again the hybridization of an optimal solution V'_{opt} and the one solution found by `LGPP`. We keep the same notation as in the proof of the soundness of `DLGPP`. Node s is a node of the branching tree which deviates from V'_{opt}, all nodes in the branch between the root and s is in accordance with V'_{opt}, the shared choices constitute the set of vertices $T = T(s)$ and, for each i, set $S_i = S_i(s)$

Algorithm 7: A description of the algorithm LGPP.

Input: A graph $G = (V, E)$, an integer k, and a triple $\text{goal} \in \{\min, \max\}$, α_1, α_2 defining any local graph partitioning problem $\mathcal{L}(\text{goal}, \alpha_1, \alpha_2)$.
Output: A set of k vertices $T \subseteq V$ optimizing $\text{val}(T)$.
set $T = \emptyset$;
LGPP (T, k):
if $k > 0$ **then**
  for $i \leftarrow 1$ ***to*** k **do**
    find $S_i \subseteq V \setminus T$ minimizing $val(T \cup S_i)$, with S_i inducing a connected component of size i;
    for *each* $v \in S_i$ **do**
      run LGPP $(T \cup \{v\}, k-1)$;
    end
  end
  else
    $(k = 0)$ store the feasible solution T;
  end
end
return *the best among the solutions stored*;

(analogously to $v(s)$ in the previous proof, s is now linked to the subsets S_i computed at this node). Set $V_n = V'_{\text{opt}} \setminus T$. Take a maximal connected (non empty) subset H of V_n. Set $S = S_{|H|}$ and consider $V_e = V'_{\text{opt}} \setminus H \cup S = (T \cup V_n) \setminus H \cup S = T \cup S \cup (V_n \setminus H)$. Note that, by hypothesis, $N[S] \cap V_n = \emptyset$ since s is a deviating node. By the choice of S at the node s, $\text{val}(T \cup S) \leqslant \text{val}(T \cup H)$. So, $\text{val}(V_e) = \text{val}(T \cup S \cup (V_n \setminus H)) = \text{val}(T \cup H \cup (V_n \setminus H)) = \text{val}(T \cup V_n) = \text{val}(V'_{\text{opt}})$ according to Lemma 2, since by construction neither $N[H]$ nor $N[S]$, do intersect $V_n \setminus H$. Iterating the argument at most k times we get to a leaf of the branching tree of LGPP which yields a solution as good as V'_{opt}. □

Corollary 5. MIN k-VERTEX COVER, k-DENSEST *and* MIN $(k, n-k)$-CUT *can be solved in* $O^*((2k\sqrt{\Delta})^{2k})$.

Indeed, both problems are local graph partitioning problems.

Theorem 3 improves the time complexity $O^*(2^{(\Delta+1)k}\ ((\Delta+1)k)^{\log((\Delta+1)k)})$ to solve local problems with the *random separation* technique introduced in [30]. Theorem 4 improves it whenever $k = 2^{o(\Delta)}$.

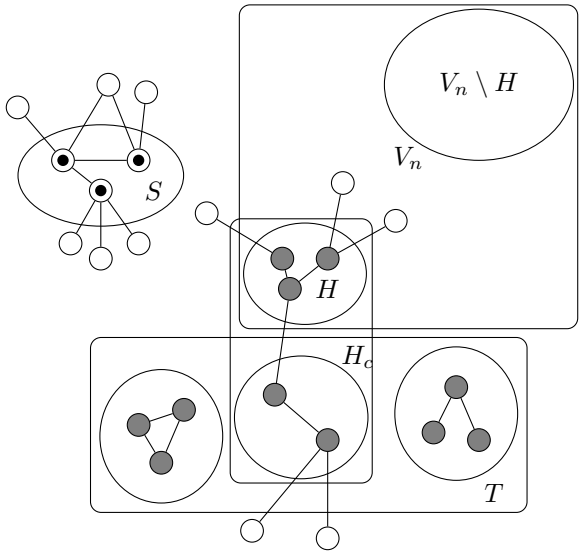
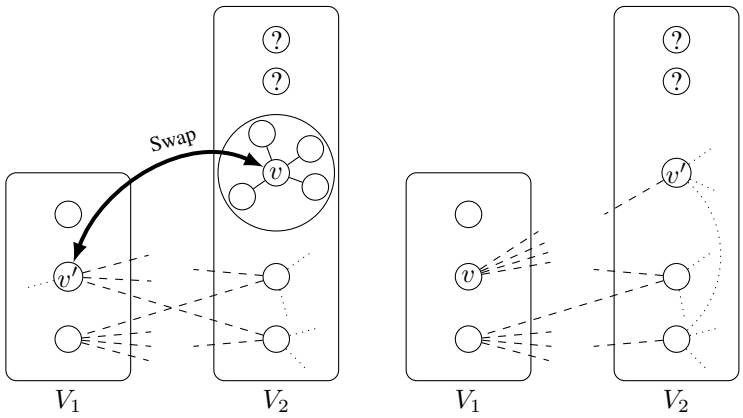


Figure 2.2: Illustration of the proof, with filled vertices representing the optimal solution V'_{opt} and dotted vertices representing the set $S = S_{|H|}$ computed by `LGPP` which can substitute H, since V_n does not interact with H_c nor with S.

Random separation is a special *color coding* [3] which consists of randomly guessing if a vertex is in an optimal subset V' of size k (white vertices) or if it is in the neighborhood $N(V') \setminus V'$ (black vertices). For every other vertices the guess is of no importance. As a right guess concerns at most only $k + k\Delta$ vertices, one can guess correctly with high probability if one iterates $\Theta(2^{(\Delta+1)k})$ random guesses. Given a random assignment $g : V \to \{\text{white,black}\}$, a solution can be computed in polynomial time by dynamic programming. So, the overall complexity is FPT with respect to $k + \Delta$. This can be derandomized with universal families.



(a) Vertices $v \in V_2$ and $v' \in V_1$ (that has at least one neighbor in V_1) will be swapped.

(b) The swap improves the value of the cut.

Figure 2.3: Illustration of a swap.

2.5 The Special Case of MAX and MIN $(\mathbf{k}, \mathbf{n-k})$-CUT

2.5.1 MAX $(\mathbf{k}, \mathbf{n-k})$-CUT

As $(k, n-k)$-CUT and $(n-k, k)$-CUT are equivalent for the classical complexity, and as the FPT algorithms in $\Delta + k$ of the previous sections are in fact FPT in $\Delta + \min(k, n-k)$, we assume now that $k \leqslant \frac{n}{2} \leqslant n-k$. In this section, we show that MAX $(k, n-k)$-CUT parameterized by the value p of the solution, is in **FPT**. By a swapping argument (see Figure 2.3), we show that the non-trivial case satisfies $p > k$.

We may also assume that $p > \Delta$. Indeed, if we put a vertex v with degree Δ in the part V' with k elements, and as many neighbors of v as we can in the other part $V \setminus V'$, the solution has at least value $\min(\Delta, n-k)$. If $p > n-k$, then, as by assumption $n - k \geqslant \frac{n}{2}$, the $O^*(2^n)$ brute-force algorithm is FPT in p. So, if $p < \Delta$ we can answer positively or brute-force.

Thus, the parameters range accordingly to Figure 2.4. The rest of the proof is an immediate application of Corollary 4.

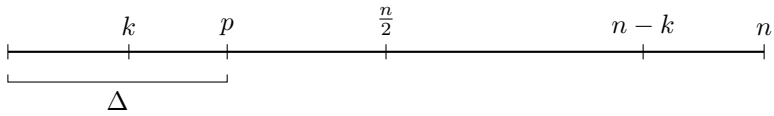


Figure 2.4: Location of parameter p, relatively to k and Δ.

Lemma 3. *In a graph with minimum degree r, the optimal value* opt *of a* MAX $(k, n-k)$-CUT *satisfies* $\text{opt} \geqslant \min\{n-k, rk\}$.

Proof. We divide arbitrarily the vertices of a graph $G = (V, E)$ into two subsets V_1 and V_2 of size k and $n-k$, respectively. Then, for every vertex $v \in V_2$, we check if v has a neighbor in V_1. If not, we try to swap v and a vertex $v' \in V_1$ which has strictly less than r neighbors in V_2 (see Figure 2.3). If there is no such vertex, then every vertex in V_1 has at least r neighbors in V_2, so determining a cut of value at least rk. When swapping is possible, as the minimum degree is r and the neighborhood of v is entirely contained in V_2, moving v from V_2 to V_1 will increase the value of the cut by at least r. On the other hand, moving v' from V_1 to V_2 will reduce the value of the cut by at most $r-1$. In this way, the value of the cut increases by at least 1.

Finally, either the process has reached a cut of value rk (if no more swap is possible), or every vertex in V_2 has increased the value of the cut by at least 1 (either immediately, or after a swap), which results in a cut of value at least $n-k$. □

Corollary 6. *In a graph with no isolated vertices, the optimal value for* MAX $(k, n-k)$-CUT *is at least* $\min\{n-k, k\}$.

Then, Corollary 4 suffices to conclude the proof of the the following theorem.

Theorem 5. *The* MAX $(k, n-k)$-CUT *problem parameterized by the parameter p is in **FPT**.*

2.5.2 MIN $(\mathbf{k}, \mathbf{n-k})$-CUT

We can prove that if $p \geqslant k$, then MIN $(k, n-k)$-CUT parameterized by the value p of the solution is in **FPT**. This is an immediate corollary of the following proposition.

Proposition 1. MIN $(k, n-k)$-CUT *parameterized by $p+k$ is in **FPT**.*

Proof. Each vertex v such that $|N(v)| \geqslant k+p$ has to be in $V \setminus V'$ (of size $n-k$). Indeed, if one puts v in V' (of size k), among its $k+p$ incident edges, at least $p+1$ leave from V'. So, it cannot yield a feasible solution. All the vertices v such that $|N(v)| \geqslant k+p$ are then rejected. Thus, one can adapt the FPT algorithm in $k+\Delta$ of Theorem 4 by considering the neighborhood of a vertex v not in the whole graph G, but in $G[T \cup U]$. In those subgraphs the degree is bounded by $p+k$, so we get an FPT algorithm in $p+k$. □

In [60], it is shown that, for any $\varepsilon > 0$, there exists a randomized $(1+\frac{\varepsilon k}{\log n})$-approximation for MIN $(k, n-k)$-CUT. From this result, we can easily derive that when $p < \frac{\log n}{k}$ then the problem is solvable in polynomial time (by a randomized algorithm). Indeed, fixing $\varepsilon = 1$, the algorithm in [60] is a $(1+\frac{k}{\log(n)})$-approximation. This approximation ratio is strictly better than $1+\frac{1}{p}$. This means that the algorithm outputs a solution of value lower than $p+1$, hence at most p, if there exists a solution of value at most p.

We now conclude this section by claiming that, when $p \leqslant k$, MIN $(k, n-k)$-CUT can be solved in time $O^*(n^p)$.

Proposition 2. *For $p \leqslant k$,* MIN $(k, n-k)$-CUT *can be solved in time $O^*(n^p)$.*

Proof. Since $p \leqslant k$, there exist in the optimal set V', $p' \leqslant p$ vertices incident to the p outgoing edges. So, the $k-p'$ remaining vertices of V' induce a subgraph that is disconnected from $G[V \setminus V']$.

Hence, one can enumerate all the $p' \leqslant p$ subsets of V. For each such subset $\widetilde{V}$, the graph $G[V \setminus \widetilde{V}]$ is disconnected. Denote by $C = (C_i)_{0 \leqslant i \leqslant |C|}$ the connected components of $G[V \setminus \widetilde{V}]$ and by α_i the number of edges between C_i and $\widetilde{V}$. We have to pick a subset $C' \subset C$ among these components such that $\sum_{C_i \in C'} |C_i| = k-p'$ and maximizing $\sum_{C_i \in C'} \alpha_i$. This can be done in polynomial time using standard dynamic programming techniques. □

2.5.3 Parameterized Approximation for MAX and MIN $(\mathbf{k}, \mathbf{n-k})$-CUT

Recall that both MAX and MIN $(k, n-k)$-CUT parameterized by k are W[1]-hard [52, 29]. In this section, we give some approximation algorithms working in FPT time with respect to parameter k.

Proposition 3. *When parameterized by k:*

- MAX $(k, n-k)$-CUT *admits an FPT approximation schema;*
- MIN $(k, n-k)$-CUT *parameterized by* k *has a randomized FPT approximation schema.*

Proof. We first handle MAX $(k, n-k)$-CUT. Fix some $\varepsilon > 0$. Given a graph $G = (V, E)$, let $d_1 \leqslant d_2 \leqslant \ldots \leqslant d_k$ be the degrees of the k largest-degree vertices $v_1, v_2, \ldots v_k$ in G. An optimal solution of value opt is obviously bounded from above by $B = \Sigma_{i=1}^{k} d_i$. Now, consider solution $V' = \{v_1, v_2, \ldots, v_k\}$. As there exist at most $k(k-1)/2 \leqslant k^2/2$ (when V' is a k-clique) inner edges, solution V' has a value sol at least $B - k^2$. Hence, the approximation ratio is at least $\frac{B-k^2}{B} = 1 - \frac{k^2}{B}$. Since, obviously, $B \geqslant d_1 = \Delta$, an approximation ratio at least $1 - \frac{k^2}{\Delta}$ is immediately derived.

If $\varepsilon \geqslant \frac{k^2}{\Delta}$ then V' is a $(1-\varepsilon)$-approximation. Otherwise, if $\varepsilon \leqslant \frac{k^2}{\Delta}$, then $\Delta \leqslant \frac{k^2}{\varepsilon}$. So, the branching algorithm of Theorem 5 whose running time is $O^*((\Delta+1)^k)$ gives here an FPT algorithm in time $O^*((\frac{k^2}{\varepsilon}+1)^k)$.

For MIN $(k, n-k)$-CUT, it is proven in [60] that, for $\varepsilon > 0$, if $k < \log n$, then there exists a randomized polynomial time $(1+\varepsilon)$-approximation. Else, if $k > \log n$, the exhaustive enumeration of the k-subsets takes time $O^*(n^k) = O^*((2^k)^k) = O^*(2^{k^2})$. □

We conclude this paragraph by showing that an approximation ratio $\frac{k^2}{f(k)} + 1$ for MIN $(k, n-k)$-CUT can be achieved in time $O^*(n^{f(k)})$. This, for instance, concludes a ratio $o(k^2)$ in time $O^*(n^{o(k)})$.

Proposition 4. *For every positive function* f, MIN $(k, n-k)$-CUT *is approximable within ratio* $\frac{k^2}{f(k)} + 1$ *in time* $O^*(n^{f(k)})$.

Proof. We distinguish three cases with respect to the parameter p. If $p \geqslant k$, then by the discussion just above, since any solution has size at most $k(k+p)$, an approximation ratio at most $2k$ is immediately derived.

Assume now $p \leqslant k$. Here, we distinguish two sub-cases, namely $p \leqslant f(k)$ and $k \geqslant p \geqslant f(k)$.

In the first of the sub-cases, using Proposition 2, an optimal solution for MIN $(k, n-k)$-CUT can be found in time at most $O^*(n^{f(k)})$.

For the second sub-case, consider a solution consisting of taking the set V' of the k vertices of G with lowest degrees, and denote by σ the sum of these degrees. Then, the value opt of an optimal solution is at least $\sigma - k^2$, i.e., $\sigma \leqslant \mathrm{opt} + k^2$. Hence, if

$p < \sigma - k^2$, the algorithm answers "no"; otherwise, some easy algebra leads to an approximation ratio bounded above by $\frac{k^2}{f(k)} + 1$. □

2.6 Set and Satisfiability Size-Constrained Problems

In the MAX k-SET COVER problem, we are given a family of subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$ over a set of elements $X = \{x_1, \ldots, x_n\}$, and an integer p. We say that a set *covers* an element if the element is contained in the set. The goal is to find a subcollection $\mathcal{T}$ of at most k subsets that covers at least p elements. MAX k-SET COVER is significant from a practical point of view, and arises frequently in several areas. For instance, in location problems when resources location is needed to perform a maximum coverage but the number of resources is restricted. In the MAX SAT-k problem, we are given a CNF on n variables and m clauses, one asks for setting to true at most k variables satisfying at least p clauses.

MAX k-SET COVER is known to be NP-hard (setting $p = n$, MAX k-SET COVER becomes the seminal MIN SET COVER problem). It is also known to be approximable within a factor $1 - 1/e$, but, for any $\epsilon > 0$, no polynomial algorithm can approximate it within ratio $1 - 1/e + \epsilon$ unless P = NP [58]. Concerning the parameterized complexity of the problem, MAX k-SET COVER is **W[2]**-hard for the parameter k, by setting $p = n$ since MIN SET COVER is **W[2]**-hard too [51]. An FPT algorithm with respect to the standard parameter p is given by Bläser in [13].

In this section, we use the greediness-for-parameterization technique to show that MAX k-SET COVER and MAX SAT-k are in **FPT** when parameterized by the value of the solution.

We now prove that MAX k-SET COVER is FPT with respect to $k + \Delta$. The useful vocabulary concerning branching algorithms is found in Section 2.3.

In fact, the following proposition can be obtained by the FPT algorithm for MAX k-SET COVER with respect to p [13]. Indeed, as $p \leqslant \Delta k$, this is also an FPT algorithm with respect to Δ and k combined. The alternative proof using greediness-for-parameterization may be read as an introduction to the FPT algorithm for MAX SAT-k.

Proposition 5. MAX k-SET COVER *parameterized by* $k + \Delta$ *is FPT.*

Proof. We present a branching algorithm (`k-SC`) whose particularity is to maintain, in addition to a partial solution T of subsets of $\mathcal{S}$, a subset $C \subseteq X$ of elements which

we commit to cover. The elements of C are therefore called the *imposed* elements. If R is a set of sets, $\bigcup R$ denotes the union of the elements of R. Now, a node v of the branching tree is *in accordance* with a solution T_0 if $T \subseteq T_0$ and $C \subseteq \bigcup T_0$. Let l be the labeling function of the nodes of the branching tree, such that for each node v, $l(v) = (T, C)$ where T and C are the sets described above. We can infer the subinstance I' at the node v from this labeling since $I' = \mathcal{S} \setminus T$.

To understand the bound over the number of imposed elements used in `k-SC`, note that if there exists a solution $\mathcal{T}$ covering more than p elements, then there exists a solution $\mathcal{T}' \subseteq \mathcal{T}$ that covers more than p elements and less than $p + \Delta - 1$ ones. Indeed, when no subset can be removed from a solution $\mathcal{T}_0$ (without going under p elements covered) then the number of covered elements can not exceed $p + \Delta - 1$, since removing a subset can uncover at most Δ elements. For a set of subsets R, we denote by $\mathcal{S} \downarrow R$ the set of subsets $\{S_i \setminus \bigcup R : S_i \in \mathcal{S}\}$. We set $T = \emptyset$ and $C = \emptyset$. The overall specification of `k-SC` is the following: Let us first establish the time

Algorithm 8: A description of the algorithm `k-SC`.

Input: A finite collection $\mathcal{S}$ of subsets of X.
Output: A set of k sets of $\mathcal{S}$ that covers the greatest number of elements of X.
set $T = \emptyset$, $C = \emptyset$;
`k-SC`(T, C):
if $|T| < k$ **and** $|C| < p$ **then**
  pick a set $S_i \in \mathcal{S} \setminus T$ that covers the largest number of elements in $X \setminus C$;
  run `k-SC`$(T \cup \{S_i\}, C \cup S_i)$;
  for *each element* $x \in S_i \setminus C$ **do**
    run `k-SC`$(T, C \cup \{x\})$;
  end
else
  if $|T| = k$ **then**
    store T;
  else
    ($p \leqslant |C| \leqslant p + \Delta - 1$) store a solution covering C, if possible;
  end
end
return *the best among the solutions stored*;

complexity of `k-SC`. The number of children of a node of the branching tree is at most $\Delta + 1$. At each step, we add either a subset or an element, so the depth of the

branching tree is, at most, $k+p+\Delta-1$. Note that $p \leqslant k\Delta$ on non-trivial MAX k-SET COVER-instances. So, the branching tree has size $O((\Delta+1)^{k+p})$. On an internal node of the branching tree, k−SC only does polynomial computations. In a leaf of this tree, we find in time $O^*(2^{p+\Delta-1})$ if at most $k-|T|$ subsets of $(\mathcal{S}\downarrow T)[C]$ can cover all the ground elements in C [63], where $\mathcal{S}[C]$ denotes the subsets in $\mathcal{S}$ entirely contained in C. So, k−SC works in time $O^*(2^{p+\Delta-1}(\Delta+1)^{k+p})$, i.e., it is fixed parameter with respect to $k+\Delta$.

We now show that k−SC is sound. Let $\mathcal{T}_0$ be a solution which covers between p and $p+\Delta-1$ elements. Recall that each node of the branching tree has one child adding a set to T and up to Δ children each adding one imposed element to C. Let $\mathcal{B}$ be a maximal branch in the branching tree from the root to a node v such that all the nodes of $\mathcal{B}$ are in accordance with $\mathcal{T}_0$. By the maximality of the branch, v deviates from $\mathcal{T}_0$. Let $(T,C)=l(v)$ and S_i the set chosen by our greedy criterion at the node v. We know that $S_i \notin \mathcal{T}_0$. Substituting in T, any subset of $\mathcal{T}_0 \setminus T$ by S_i, we cover at least as many elements as $\mathcal{T}_0$, since $\forall x \in S_i \setminus C$, x is not covered by the solution $\mathcal{T}_0$. From v, we consider again a maximal branch $\mathcal{B}'$ *in accordance* with $\mathcal{T}_0$, and we iterate the same hybridization trick at most $k+p$ times until we reach a leaf. At this leaf, k−SC computes an exact solution T_l containing at most $k-|T|$ subsets of $(\mathcal{S}\downarrow T)[C]$. So, $T \cup T_l$ is as good as $\mathcal{T}_0$, hence, an optimal solution. □

Let us now deal with MAX SAT-k. In what follows, C denotes the set of clauses of an instance and C' any subset of this set. We denote by $\mathrm{occ}^+(X_i,C')$ the number of positive occurrences of the variable X_i in the instance, and by $\mathrm{occ}^-(X_i,C')$ the number of its negative occurrences. We set $f(X_i)=\mathrm{occ}^+(X_i,C)+\mathrm{occ}^-(X_i,C)$; so, the frequency of the formula is $f=\max_i\{\mathrm{occ}^+(X_i,C)+\mathrm{occ}^-(X_i,C)\}$.

Lemma 4. MAX SAT-k *is solvable in* $O(2^m)$.

Proof. We take any variable X that appears positively *and* negatively. We do the standard branching: either set X to true (and decrease k by 1), either set X to false. This branching satisfies at least one more clause in each branch. Thus, it takes time bounded by $O(2^m)$. The branching stops when each variable appears only negatively, or only positively. At that point, the variables appearing only negatively can be set to false. This step is safe since we are constrained to put *at most* (not exactly) k variables to true. We end up with an instance containing only positive literals. This instance can be seen as an instance of k-HITTING SET which can be solved in time $O(2^m)$ [63]. Overall, it takes time $O(2^m)$. □

Proposition 6. MAX SAT-k *parameterized by* p *is FPT.*

Proof. We can assume that $p < \frac{m}{2}$. Indeed, if $p \geqslant \frac{m}{2}$, the algorithm of Lemma 4 is an FPT algorithm. We also assume that the number of clauses containing only negative literals is bounded above by $\frac{m}{2}$. Otherwise, setting all the variables to false, satisfies more than p clauses. We recall that we are not forced to set exactly k variables to true, but *at most* k. We observe that instances such that $p < \frac{f}{2}$ are always YES-instances, since one can set one variable X_i with frequency f to true if $\mathrm{occ}^+(X_i, C) \geqslant \mathrm{occ}^-(X_i, C)$, and to false otherwise. Note also that instances such that $p < k$ are all YES-instances, too. Indeed, one can iteratively set to true k variables such that at each step one satisfies at least one more clause. If, at some point this is no longer possible, then setting all the remaining variables to false will satisfy all the clauses which do not initially contain only negative literals, that is at least half of the clauses, so more than p clauses. We may now assume that $p \geqslant \frac{f}{2}$ and $p \geqslant k$, so our parameter might as well be $p + f + k$.

Once again, we construct a branching algorithm which operates accordingly to a greedy criterion. A solution, or *complete assignment*, is given by a set S of size up to k which contains all the variables set to true. Additionally, we maintain a list C_s of clauses that we satisfy or commit to satisfy. We set $C_u = C \setminus C_s$, $d_i(C') = \mathrm{occ}^+(X_i, C')$, and let $C^+(X_i, C')$ be the set of clauses in C' where X_i appears positively and $C^-(X_i, C')$ the set of clauses where X_i appears negatively. Set, finally, $C(X_i, C') = C^+(X_i, C') \cup C^-(X_i, C')$ and consider the following algorithm (`SAT-k`): The branching tree has depth at most $k + p$ and width at most $f + 1$, so the running time of `SAT-k` is $O^*(2^p(f + 1)^{k+p})$ that is FPT with respect to p, because completing a solution to satisfy all the clauses of C_s can be done in time $O^*(2^{|C_s|})$ since SAT-k can be solved in $O(2^m)$ by Lemma 4.

Let now S_0 be an optimal solution. From the root of the branching tree, follow a maximal branch where the variables set to true are all in S_0, and the clauses in C_s are satisfied by S_0. Let S_c be the set of variables set to true along this branch (by definition, $S_c \subseteq S_0$), and set $S_n = S_0 \setminus S_c$. By maximality of the branch, at its extremity v, `SAT-k` deviates from S_0, i.e., no child of v is in accordance with S_0. Let X_i be the variable chosen at this point by `SAT-k` and consider $C_d = C(X_i, C_u)$ that is the set of clauses not yet in C_s in which X_i appears positively or negatively. We know that no clause in C_d is satisfied by S_0. Let X_j be any variable in S_n. We claim that $S_h = (S_0 \setminus \{X_j\}) \cup \{X_i\}$ is also optimal and, by a straightforward induction, one

Algorithm 9: A description of the algorithm SAT−k.

Input: A set C of clauses on a set X of variables.
Output: A subset $S \subseteq X$ of size at most k such that setting all the variables in S to true and all the variables in $X \setminus S$ to false, satisfies the greatest number of clauses in C.

set $S = \emptyset$, $C_s = \emptyset$;
SAT−k(S, C_s):
if $|S| < k$ **and** $|C_s| < p$ **then**
  pick the variable X_i maximizing $d_i(C \setminus C_s)$;
  run SAT−k$(S \cup \{X_i\}, C_s \cup C^+(X_i, C \setminus C_s))$;
  for *each clause* $c \in C(X_i, C \setminus C_s)$ **do**
    run SAT−k$(S, C_s \cup \{c\})$;
  end
else
  if $|S| = k$ **then**
    store S;
  else
    ($|C_s| \geqslant p$) store a complete assignment satisfying C_s, if possible;
  end
end
return *the best among the solutions stored*;

solution at the leaves of the branching tree is as good as S_0. Indeed, setting X_j to false can lose at most $\mathrm{occ}^+(X_j, C_u) - \mathrm{occ}^-(X_j, C_u) \leqslant d_j(C_u)$ clauses and setting X_i to true gains $d_i(C_u)$ clauses and, by construction, $d_i(C_u) \geqslant d_j(C_u)$. □

The complexity of MAX SAT-k parameterized by $k + f$ remains open.

2.7 Size-Constrained Problems in Bipartite Graphs

In this section, we investigate the complexity of size-constrained problems in bipartite graphs, where one looks for a set of exactly k vertices which realizes a specific local property. This class of problems encompasses k-DENSEST, k-SPARSEST together with the maximization and minimization versions of k-VERTEX COVER, $(k, n-k)$-CUT, k-DOMINATING SET. In general graphs, we recall that the six first problems parameterized by k are **W[1]**-complete, while both versions of k-DOMINATING SET

are **W[2]**-complete [29]. We show the rather surprising result that MAX k-VERTEX COVER and MAX $(k, n-k)$-CUT are in **FPT** in bipartite graphs, whereas their minimization versions MIN k-VERTEX COVER and MIN $(k, n-k)$-CUT are **W[1]**-complete.

Again, the number of edges covered by a set S of vertices is denoted by $\mathrm{val}(S)$. We can notice that $\mathrm{val}(S) = |E(S)| + \delta(S)$, where $E(S)$ is the set of edges in $G[S]$, and $\delta(S)$ is the number of edges having exactly one endpoint in S. In MAX k-VERTEX COVER, given a graph $G = (V, E)$, one has to find a subset $S \subseteq V$ with k vertices which maximizes $\mathrm{val}(S)$. There are three independent works showing that MAX k-VERTEX COVER is **NP**-complete in bipartite graphs [85, 34, 5]. The easiest way to get this result is to follow [34].

2.7.1 Positive Results

Given a solution S, an edge which is covered by a single vertex v of S is called *private*, or a *private* edge of v. On the contrary, an edge covered by its two endpoints u and v is called a *shared* edge. A vertex can cover at most $k-1$ shared edges, in case its closed neighborhood contains the solution S.

Proposition 7. MAX k-VERTEX COVER *in bipartite regular graphs is trivial.*

Proof. Any regular bipartite graph $G = (V_1 \cup V_2, E)$ satisfies $|V_1| = |V_2| = \frac{n}{2}$. Let Δ be the degree of the regular graph G. Taking any $\min(k, \frac{n}{2})$ vertices in V_1 covers $\min(k\Delta, |E|)$ edges which is at least as good as any other solutions since one cannot cover more than Δ edges per vertex taken. □

We present an FPT algorithm for MAX k-VERTEX COVER which is inspired both from the easy proposition shown above and from the notion of *intersective algorithms* or *necessary sets*. Intuitively an intersective algorithm produces solutions which always intersect an optimal solution. This notion is presented in [15]. In fact, we will show that a generalization of MAX k-VERTEX COVER, called here GENERALIZED MAX k-VERTEX COVER, is in **FPT** in bipartite graphs. In GENERALIZED MAX k-VERTEX COVER, given a graph $G = (V, E)$ and a subset V', one has to find a subset $S \subseteq V'$ with k vertices which maximizes $\mathrm{val}(S)$. In this new and very similar problem, the only difference lies on the fact that one cannot take any vertices of $V \setminus V'$ in the solution. Then, MAX k-VERTEX COVER is the special case when $V' = V$.

Let $G = (V, E), V'$ be an instance of GENERALIZED MAX k-VERTEX COVER. For every $i \in \{1, \ldots, |V'|\}$, v_i is the i-th vertex of V' by non increasing degree (breaking ties arbitrarily). We show two lemmas to bound by $2k$ the difference among the vertices of V' between the highest degree and the lowest degree.

Lemma 5. *A vertex $v \in V'$ such that $d(v) \leqslant d(v_k) - k$ will never be part of an optimal solution.*

Proof. If S is a solution containing v, then at least one of the vertices $v_1, v_2, \ldots, v_k$ is not in the solution, otherwise the solution would contain strictly more than k vertices. So, let v_i be a vertex such that $i \in \{1, 2, \ldots, k\}$ and $v_i \notin S$. Then, $S \setminus \{v\} \cup \{v_i\}$ is a better solution than S. Indeed, removing v from the solution loses at most $d(v)$ edges (in case all the edges covered by v are private). Adding then v_i to the solution gains at least $d(v_i) - k + 1$ edges (in case v_i covers $k - 1$ shared edges). And, $d(v_i) - k + 1 \geqslant d(v_k) - k + 1 > d(v)$. □

Basically, Lemma 5 states that you can remove from V' all the vertices whose degree is less than $d(v_k) - k$ and preserve all the optimal solutions. From hereon, we can suppose that V' do not contain such vertices anymore.

Lemma 6. *If $d(v_k) \leqslant d(v_1) - k$, any optimal solution intersects $\{v_1, \ldots, v_{k-1}\}$, and the intersection can obviously be guessed in FPT time.*

Proof. Suppose S^* is an optimal solution which does not intersect $\{v_1, \ldots, v_{k-1}\}$ and v is a vertex in this solution. Then $S^* \setminus \{v\} \cup \{v_1\}$ is a better solution for the same reason as the one invoked to prove Lemma 5. One can exhaustively guess in FPT time 2^{k-1} the intersection between an optimal solution and $\{v_1, \ldots, v_{k-1}\}$. □

In the computation of Lemma 6, the intersection is non-empty. Thus, we will use it at most k times, since it adds at least one vertex to the solution. So, this first step takes at most FPT time $O^*(k^k)$. Combining Lemma 5 and Lemma 6, one can suppose that, after this step, the degree of all the vertices in V' is contained in the interval $[\Delta - 2k, \Delta]$ (even, $[\Delta - 2k + 2, \Delta]$ but it is somewhat irrelevant).

Theorem 6. MAX k-VERTEX COVER *parameterized by k, is in **FPT** in bipartite graphs.*

Proof. We prove the stronger statement that GENERALIZED MAX k-VERTEX COVER parameterized by k, is in **FPT** in bipartite graphs. Let $G = (V_1, V_2, E), V'$ be an instance of GENERALIZED MAX k-VERTEX COVER in bipartite graphs. The degrees of the vertices in V' goes from $\Delta - 2k + 1$ to Δ.

If there are more than $2k$ vertices with degree Δ in V', then there are more than k such vertices in one of the partite set, and taking them gives an optimal solution with value $k\Delta$. Otherwise, the number of vertices in V' with degree Δ strictly less than $2k$. Therefore, we can exhaustively guess the intersection between those vertices and an optimal solution. The vertices which are taken in the solution are removed from the graph, the vertices which are not, are just removed from V' but they stay in the graph (since their incident edges could still be covered by the other endpoints). Now, the intersection could be empty but the degree in V' decreases by at least 1.

Our algorithm branches on each of the at most $2k - 1$ vertices of maximum degree still in V'. There is an additional branch where one does not take any vertices in this subset. Therefore, a node of the branching tree has at most $2k - 1$ sons where a vertex is taken, and one son where only the maximum degree of vertices still in V' decreases. Now, the measure we consider is $l = k + d$ where d is the number of distinct degree of vertices in V'. We can observe that d is bounded by $3k$. By the previous lemmas, the distinct number of degrees in V' is initially bounded by $2k$. Though, vertices having a degree between $\Delta - 3k + 1$ and $\Delta - 2k$ in V' can be created by taking one or several of their neighbors in the solution.

As $l \leqslant 4k$, and the measure l decreases by at least 1 in each of the at most $2k$ sons of each node of the branching, our algorithm works in time $O^*((2k)^{4k})$. □

Theorem 7. MAX $(k, n-k)$-CUT *parameterized by k, is in **FPT** in bipartite graphs.*

Proof. Again, we show the stronger result that GENERALIZED MAX $(k, n-k)$-CUT parameterized by k, is in **FPT** in bipartite graphs. We show two analogous lemmas to Lemma 5 and Lemma 6 for MAX $(k, n-k)$-CUT. We identify a cut to the smallest set of the partition. And an instance $G = (V, E)$ is given with an additional set $V' \subseteq V$ such that only vertices in V' can be added to the solution.

Lemma 7. *A vertex $v \in V'$ such that $d(v) \leqslant d(v_k) - 2k$ will never be part of an optimal solution.*

Proof. If S is a solution containing v, then at least one of the vertices $v_1, v_2, \ldots, v_k$ is not in the solution, otherwise the solution would contain strictly more than k vertices.

So, let v_i be a vertex such that $i \in \{1, 2, \ldots, k\}$ and $v_i \notin S$. Then, $S \setminus \{v\} \cup \{v_i\}$ is a better solution than S. Indeed, removing v from the solution loses at most $d(v)$ edges. Adding then v_i to the solution gains at least $d(v_i) - 2k + 2$ edges. And, $d(v_i) - 2k + 2 \geqslant d(v_k) - 2k + 2 > d(v)$. □

By Lemma 7, you can remove from V' all the vertices whose degree is less than $d(v_k) - 2k$ and preserve all the optimal solutions.

Lemma 8. *If $d(v_k) \leqslant d(v_1) - 2k$, any optimal solution intersects $\{v_1, \ldots, v_{k-1}\}$, and the intersection can obviously be computed in FPT time.*

Proof. Suppose S^* is an optimal solution which does not intersect $\{v_1, \ldots, v_{k-1}\}$ and v is a vertex in this solution. Then $S^* \setminus \{v\} \cup \{v_1\}$ is a better solution for the same reason as the one invoked to prove Lemma 7. One can exhaustively guess in FPT time 2^{k-1} the intersection between an optimal solution and $\{v_1, \ldots, v_{k-1}\}$. □

Combining Lemma 7 and Lemma 8, we can assume that the degrees of the vertices in V' are contained in $[\Delta - 4k, \Delta]$. We make the same observation as Proposition 7: on regular bipartite graphs MAX $(k, n-k)$-CUT is trivially FPT. The algorithm is the same as for MAX k-VERTEX COVER and we get this time an overall time complexity in $O^*((2k)^{6k})$. □

2.7.2 Negative Results

Theorem 8. MIN k-VERTEX COVER *and* MIN $(k, n-k)$-CUT *parameterized by k, are both **W[1]**-complete even in bipartite graphs.*

Proof. Both problems belong to **W[1]** even on general graphs. We reduce from the problem k-CLIQUE which is **W[1]**-complete even on regular graphs. The reduction is the same for MIN k-VERTEX COVER and MIN $(k, n-k)$-CUT up to p the target value of the solution.

First, we describe the construction. Let $G = (V, E)$ be any regular instance of k-CLIQUE, Δ be its degree, $n = |V|$, and $m = |E|$. We build the graph $G' = (V', E')$ where $V' = V \cup V_E \cup D$. V_E consists of $|E|$ vertices in a one-to-one correspondence with the edges of G. $D = \{d_1, \ldots, d_{\Delta-2}\}$ is a set of $\Delta - 2$ dummy vertices. And, $E' = \{\{v, u_e\} \mid v \in V, u_e \in V_E, v \text{ is one endpoint of } e \text{ in } G\} \cup \{(u_e, d_i) \mid u_e \in V_E, 1 \leqslant i \leqslant \Delta - 2\}$. In other words, $G'[V \cup V_E]$ is the incidence graph of G, and

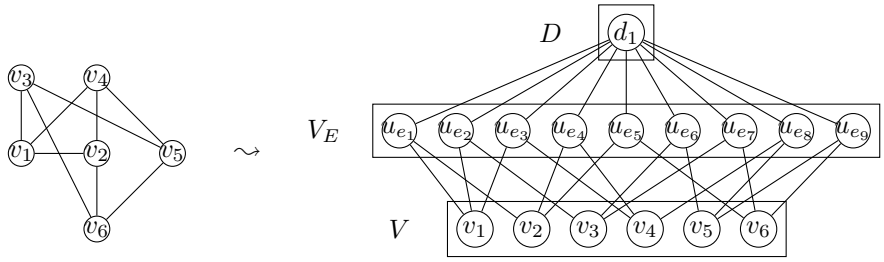


Figure 2.5: An example of a regular graph G and its corresponding G'. For MIN k'-VERTEX COVER, we ask for a solution covering $p = k\Delta - 2\binom{k}{2}$ edges. For MIN $(k', n-k')$-CUT, we ask for a solution with $p = k\Delta - 4\binom{k}{2}$ edges in the cut.

$G'[V_E \cup D]$ is a bipartite complete graph whose partite sets are V_E and D. We can observe that G' is bipartite since both V_E and $V \cup D$ are independent sets.

Vertices in V_E are called *edge* vertices, and vertices in D are called *dummy* vertices. In G', all the vertices in $V \cup V_E$ have degree Δ. Indeed, any vertex $v \in V$ is connected to Δ vertices in V_E corresponding to the Δ edges incident to v in G. Any edge vertex $u_e \in V_E$ is connected to two vertices in V which correspond to the endpoints of e in G, plus $\Delta - 2$ dummy vertices. Any dummy vertex has degree m. Figure 2.7.2 gives the above construction with an example of regular graph G and the reader might check the previous observations on the degree of vertices in G'.

For MIN k'-VERTEX COVER, we ask for a set of $k' = k + \binom{k}{2}$ vertices which covers at most $p = k\Delta - 2\binom{k}{2}$. First, we show a lemma in the same flavor as Lemma 5 and Lemma 6 but for the minimization version.

Lemma 9. *An optimal solution of* MIN k-VERTEX COVER *cannot take a vertex* v *and* not *a vertex* w *if* $d(v) \geqslant d(w) + k$.

Proof. Removing v from a solution S can reduce the number of covered edges by at least $d(v) - k + 1$, and taking w in the solution can increase the number of covered edges by at most $d(w)$. The resulting solution $S \setminus \{v\} \cup \{w\}$ covers at most $\mathrm{val}(S) + d(w) - d(v) + k - 1 < \mathrm{val}(S)$ which contradicts the optimality of S. □

We can assume that $m > \Delta + k' = \Delta + k + \binom{k}{2}$. Otherwise, as $m = \frac{\Delta n}{2}$, we would have $\Delta(\frac{n}{2} - 1) < k + \binom{k}{2}$ and a function of the parameter would bound from above the size of the input. In other words, instances of k-CLIQUE such that $m \leqslant \Delta + k + \binom{k}{2}$

are trivially FPT. For the same reason, we can assume that $n + m > k' = k + \binom{k}{2}$, i.e., one is not forced to take a vertex in D for space constraint. Thus, by Lemma 9, any optimal solution takes no vertex in D.

Therefore, an optimal solution consists of $\binom{k}{2} + r$ vertices of V_E and $k - r$ vertices of V, with $r \in [-\binom{k}{2}, k]$. As all the vertices in $V \cup V_E$ have degree Δ, the number of edges covered by a set S of k' vertices is $k\Delta - |E(S)|$. So, the problem is equivalent to finding a set $S \subseteq V \cup V_E$ such that $|E(S)| \geqslant 2\binom{k}{2}$. As all the vertices in V_E have exactly two neighbors in $V \cup V_E$, $|E(S)| \leqslant 2(\binom{k}{2} + r)$. Thus, for a solution covering p edges, r should be non negative, that is $r \in [0, k]$. The number of edges in S is bounded from above by $2\binom{k-r}{2} + \binom{k}{2} + r - \binom{k-r}{2} = \binom{k-r}{2} + \binom{k}{2} + r$. This corresponds to the best case where we take a set U of $k - r$ vertices in V which forms a clique in G and $\binom{k-r}{2}$ edge vertices corresponding to the edges of the clique, plus $\binom{k}{2} + r - \binom{k-r}{2}$ edge vertices corresponding to edges having one endpoint in U. If $r > 0$, $\binom{k-r}{2} + r < \binom{k}{2}$ for any $k \geqslant 3$. So, r has to be equal to 0, for S to cover p edges or less. We have proven that a solution S, if existing, takes a set S_v of exactly k vertices in V and a set S_e of exactly $\binom{k}{2}$ vertices in V_E, with $S = S_v \cup S_e$. The edges $2|S_e| = 2\binom{k}{2}$ edges going from S_e to V should all reach S_v. Thus, there is a set of $\binom{k}{2}$ edges whose endpoints are all in a set of k vertices. Therefore, S_v is a k-clique in G.

Reciprocally, if there is a k-clique C in G, then we can take the k corresponding vertices in G' and the $\binom{k}{2}$ edge vertices in V_E which corresponds to the inner edges of the clique C, and that solution covers exactly p edges in G'.

For MIN $(k', n - k')$-CUT, we ask for a set S of $k' = k + \binom{k}{2}$ vertices such that the cut $(S, V' \setminus S)$ contains at most $p = k\Delta - 4\binom{k}{2}$ edges.

Lemma 10. *An optimal solution of* MIN $(k, n - k)$-CUT *cannot take a vertex* v *and* not *a vertex* w *if* $d(v) \geqslant d(w) + 2k$.

Proof. Removing v from a solution S can reduce the number of edges in the cut by at least $d(v) - 2k + 2$, and taking w in the solution can increase the number of edges in the cut by at most $d(w)$. The resulting solution $(S' = S \setminus \{v\} \cup \{w\}, \overline{S'})$ has $d(w) - d(v) + 2k - 2 < 0$ edges more in the cut which contradicts the optimality of S. □

We can assume that $m > \Delta + 2k' = \Delta + 2k + 2\binom{k}{2}$, and $n + m > k' = k + \binom{k}{2}$, $k' < \frac{|V'|}{2}$, and identify a cut to the smallest set of the partition. Thus, by Lemma 10, any optimal solution takes no vertex in D.

	k-VERTEX COVER	$(k, n-k)$-CUT	k-DOMINATING SET
min	**W[1]-complete** (Th. 8)	**W[1]-complete** (Th. 8)	W[2]-complete (defined?)
max	**FPT** (Th. 6)	**FPT** (Th. 7)	W[2]-complete [111]

Figure 2.6: The parameterized complexity of size-constrained problems in bipartite graphs.

We can remark that a solution S induces a cut with $k'\Delta - 2|E(S)|$, so the problem is equivalent to finding a set $S \subseteq V \cup V_E$ with $2\binom{k}{2}$ inner edges. Henceforth, the rest of the proof is the same as for MIN k'-VERTEX COVER. □

We do not know if the minimization version of k-DOMINATING SET has already been defined in the literature. To complete the landscape concerning the parameterized complexity of size-constrained problems in bipartite graphs, k-SPARSEST is NP-complete but trivially **FPT** in bipartite graphs, since there is an independent set of size $\frac{n}{2}$, whereas k-DENSEST is already known to be **W[1]**-complete in bipartite graphs [44].

2.8 Recent Advances and Perspectives

We had two main open questions concerning local graph partitioning problems. The first was the parameterized complexity status of MIN $(k, n-k)$-CUT with respect to the value of the solution p. The second was whether or not we could improve the running time of solving non degrading local graph partitioning problems to $O^*((a\Delta)^{bk})$ for some constants a and b. Two recent articles answered both questions.

In [46] it is shown that MIN $(k, n-k)$-CUT, parameterized by p, is solvable in time $O(2^{O(p^3)}n^3 \log^3 n)$. The principal ingredient of the algorithm is a new tree-like decomposition of any graph in small separators. The algorithm is first conceived for MIN BISECTION and it is then generalized to MIN $(k, n-k)$-CUT and other variants.

In [117], it is shown that a time-complexity of $O^*(4^{k+o(k)}\Delta^k)$ can indeed be reached for general LGPPs. It is interesting to note that the algorithm presented in this article use a reduction to weighted exact cover which only works for non degrading LGPPs. So, our result of Theorem 3 is still needed to conclude that *all* LGPPs can be solved within time $O^*(O(\Delta)^k)$.

Figure 2.8 summarizes the parameterized complexity landscape of the main local graph partitioning problems.

	k-VERTEX COVER	$(k, n-k)$-CUT	SPARSEST/DENSEST
k, min	W[1]-c$^{\diamond}$, $O^*(4^{k+o(k)}\Delta^k)^{\S}$	W[1]-c$^{\diamond}$, $O^*(4^{k+o(k)}\Delta^k)^{\S}$	W[1]-c$^{\diamond}$, $\mathbf{O}^*((\mathbf{\Delta}+\mathbf{1})^{\mathbf{k}})$ (Th 3)
max	W[1]-c$^{\diamond}$, $\mathbf{O}^*((\mathbf{\Delta}+\mathbf{1})^{\mathbf{k}})$ (Th 3)	W[1]-c$^{\diamond}$, $\mathbf{O}^*((\mathbf{\Delta}+\mathbf{1})^{\mathbf{k}})$ (Th 3)	W[1]-c$^{\diamond}$, $O^*(4^{k+o(k)}\Delta^k)^{\S}$
p, min	?	FPT [46]	$\notin$ XP
max	FPT [13]	**FPT** (Th 5)	W[1]-h

Figure 2.7: Symbol $^{\diamond}$ refers to [29] and § refers to [117].

Concerning, now, what we have done for MAX $(k, n-k)$-CUT and MAX k-VERTEX COVER in bipartite graphs, Lemmas 5, 6, 7, and 8 bounding the range of the degrees, does not require the graph to be bipartite. They are still valid for general graphs. We ask then the following question: can we use those lemmas to prove that MAX $(k, n-k)$-CUT and MAX k-VERTEX COVER are in **FPT** in other classes of graphs? It seems possible to show that both problems are FPT in strongly chordal graphs which is a superset of proper interval graphs. Nonetheless, the parameterized complexity of MAX $(k, n-k)$-CUT and MAX k-VERTEX COVER in chordal graphs remains open. Finally, we observe that the lemmas bounding the range of the degrees can also be extended to the minimization versions MIN $(k, n-k)$-CUT and MIN k-VERTEX COVER for general graphs. It is possible that, though they are **W[1]**-hard in bipartite graphs, they might be FPT in other classes of graphs.

3 Inapproximability

3.1 Preliminaries

Fixed-parameter algorithms, approximation algorithms and moderately exponential/subexponential algorithms are major approaches for efficiently solving NP-hard problems. These three areas, each of them being very active in its own, have been considered as foreign to each other until recently. A polynomial-time approximation algorithm produces a solution whose quality is guaranteed to lie within a certain range from the optimum. One illustrative problem indicating the development of this area is MAX INDEPENDENT SET. The approximability of MAX INDEPENDENT SET within constant ratios had remained as one of the most important open problems for a long time in the field. It was only after the novel characterization of **NP** by PCP theorem [7] that such inapproximability was proven assuming $\mathbf{P} \neq \mathbf{NP}$. Subsequent improvements of the original PCP theorem led to much stronger result for MAX INDEPENDENT SET: it is inapproximable within ratios $\Omega(n^{\varepsilon-1})$ for any $\varepsilon > 0$, unless $\mathbf{P} = \mathbf{NP}$ [129].

Moderately exponential (subexponential, respectively) computation allows exponential (subexponential, respectively) running time for the sake of optimality. In this case, the endeavor lies in limiting the growth of the running time function as much as possible. Parameterized complexity provides an alternative framework to analyze the running time in a more refined way [51]. Given an instance with a parameter k, the aim is to get an $O(f(k) \cdot poly(n))$-time (or equivalently, FPT-time) algorithm, where $poly(n)$ is independent of k. As these two research programs offer a generous running time when compared to that of classic approximation algorithms, a growing amount of attention is paid to them as a way to cope with hardness in approximability. The first one yields *moderately exponential approximation*. In moderately exponential approxima-

tion, the core question is whether a problem is approximable in moderately exponential time while such approximation is impossible in polynomial time. Suppose a problem is solvable in time $\gamma^n poly(n)$, but it is NP-hard to approximate within ratio r. Then, we seek r-approximation algorithms of running time significantly faster than $\gamma^n poly(n)$. This issue has been considered for several problems such as SET PARTITIONING, MAX INDEPENDENT SET, COLORING, and BANDWIDTH [11, 23, 24, 45, 68].

The second research program handles approximation by fixed parameter algorithms. We say that a minimization (maximization, respectively) problem Π, together with a parameter k, is *parameterized r-approximable* if there exists an FPT-time algorithm which computes a solution of size at most (at least, respectively) rk whenever the input instance has a solution of size at most (at least, respectively) k. This line of research was initiated by three independent works [53, 32, 40]. For an excellent overview on early stages of the topic, see [102]. Since then very important research has been conducted on several aspects (both computational and structural) of parameterized approximation (see, for example, [24, 15, 41, 54, 73, 74]). In what follows, parameterization means "standard parameterization", i.e., where problems are parameterized by the cost of the optimal solution.

Several natural questions can be asked dealing with these two programs. In particular, the following ones have been asked several times [102, 53, 68, 24]:

Q1: can a problem, which is highly inapproximable in polynomial time, be well-approximated in subexponential time?

Q2: does a problem, which is highly inapproximable in polynomial time, become well-approximable in FPT-time?

Few answers have been obtained until now. Regarding **Q1**, negative results can be directly obtained by gap-reductions for certain problems. For instance, COLORING is not approximable in subexponential time within ratio $(4/3) - \epsilon$ since this would allow to determine whether a graph is 3-colorable or not in subexponential time. This contradicts the widely-acknowledged computational assumption ETH [81].

Regarding **Q2**, [53] shows that assuming FPT $\neq$ W[2], for any r the MIN INDEPENDENT DOMINATING SET problem is not r-approximable [1] in FPT-time.

Among interesting problems for which **Q1** and **Q2** are worth being asked are MAX INDEPENDENT SET, COLORING and MIN DOMINATING SET. They fit in the frame of

[1] Actually, the result is even stronger: it is impossible to obtain a ratio $r = g(k)$ for any function g.

both **Q1** and **Q2** above: they are hard to approximate in polynomial time while their approximability in subexponential or in parameterized time is still open.

Here, we study parameterized and subexponential (in)approximability of natural optimization problems. In particular, we follow two guidelines: (i) getting inapproximability results under some conjecture and (ii) establishing classes of uniformly inapproximable problems under approximation preserving reductions.

Following the first direction, we establish a link between a major conjecture in PCP theory and inapproximability in subexponential-time and in FPT-time, assuming ETH. Just below, we state this conjecture while the definition of PCP is deferred to the next section.

> *Linear PCP Conjecture* (LPC): 3SAT $\in \text{PCP}_{1,\beta}[\log|\phi| + D, E]$ for some $\beta \in (0,1)$, where $|\phi|$ is the size of the 3SAT instance (sum of lengths of clauses), D and E are constant.

Unlike ETH which is widely held to be a reasonable conjecture, LPC is a wide open question. We will claim in Lemma 12 that if LPC turns out to hold, it implies that one of the most interesting questions in subexponential and parameterized approximation is answered in the negative. In particular, the following holds for MAX INDEPENDENT SET on n vertices, for *any* constant $0 < r < 1$ assuming ETH:

(i) there is no r-approximation algorithm in time $O(2^{n^{1-\delta}})$ for any $\delta > 0$;

(ii) there is no r-approximation algorithm in time $O(2^{o(n)})$, if LPC holds;

(iii) there is no r-approximation algorithm in time $O(f(k)n^{O(1)})$, if LPC holds, where k is the size of a maximum independent set and f is any function.

We observe that (i) is not conditional upon LPC. In fact, this is an immediate consequence of the near-linear PCP construction achieved in [49]. Note that similar inapproximability results under ETH for MAX 3SAT and MAX 3LIN for some subexponential running time have been obtained in [107].

Following the second guideline, we show that a number of problems are equivalent with respect to approximability in subexponential time. Designing a family of equivalent problems is a common way to provide an evidence in favor of hardness of these problems. One prominent example is the family of problems complete under SERF-reducibility [81] which leads to equivalent formulations of ETH. More precisely, for a

given problem Π, let us formulate the following hypothesis, which can be seen as the approximate counterpart of ETH.

Hypothesis 1 (APETH(Π))**.** There exist two constants $\epsilon > 0$ and r ($r < 1$ if Π is a maximization problem, $r > 1$, otherwise), such that Π is not r-approximable in time $2^{\epsilon n}$.

We prove that several well-known problems are equivalent with respect to the APETH (APETH-equivalent). To this end, a notion called the *approximation preserving sparsification* is proposed. A recipe to prove that two problems A and B are APETH-equivalent consists of two steps. The first is to reduce an instance of A into a family of instances in "bounded" version (bounded degree for graph problems, bounded occurrence for satisfiability problems), which are equivalent with respect to approximability. This step is where approximation preserving sparsification comes into play. The second is to use standard approximation preserving reductions to derive equivalences between bounded versions of A and B. We consider L-reductions [109] for this purpose. Furthermore, we show that if APETH fails for one of these problems, then *any* problem in MaxSNP would be approximable for *any* constant ratio in subexponential FPT-time $2^{o(k)}$, which is also an evidence toward the validity of APETH. This result can be viewed as an extension of [33], which states that no MaxSNP-hard problem allows a algorithm in time $2^{o(k)}$ under ETH.

Results derived from PCP and LPC are given in Section 3.2. The second direction on equivalences between problems is described in Section 3.3. We now give some preliminaries and notation.

We will use in the sequel the well known *sparsification lemma* [81]. Intuitively, this lemma allows to work with 3SAT formula with linear lengths (the sum of the lengths of clauses is linearly bounded in the number of variables).

Lemma 11. *[81] For all $\epsilon > 0$, a* 3SAT *formula ϕ on n variables can be written as the disjunction of at most $2^{\epsilon n}$* 3SAT *formulæ ϕ_i on (at most) n variables such that ϕ_i contains each variable in at most c_ϵ clauses for some constant c_ϵ. Moreover, this reduction needs at most $poly(n)2^{\epsilon n}$-time.*

We denote by $\text{PCP}_{\alpha,\beta}[q,p]$ (see for instance [7] for more on PCP systems) the set of problems for which there exists a PCP verifier V which uses q random bits, reads at most p bits in the proof and is such that:

By Fagin's Theorem, NP is characterized as the class of graph problems expressible in existential second-order logic. In this logic, one can quantify existentially *and* universally over the vertices but one is restricted to existential quantification over sets of vertices. In SNP (for Strict NP), the quantification over vertices can only be universal.
We now introduce L-reductions which are *linear* reductions mostly preserving approximation schemata.
Let Π_A and Π_B be two optimization problems, v_A and v_B being the two corresponding functions mapping a solution to its value. An L-reduction is defined by two functions f and g computable in polynomial-time and two constants α and β such that:

- f maps instances of Π_A to instances of Π_B.
- g maps solutions of $f(I)$ to solutions of I.
- $\text{OPT}_B(f(I)) \leqslant \alpha \text{OPT}_A(I)$.
- for every solution S of $f(I)$, $|\text{OPT}_A(I) - v_A(g(S))| \leqslant \beta |\text{OPT}_B(f(I)) - v_B(S)|$.

MaxSNP is the class of problems that L-reduce to a maximization version of a SNP problems.
A problem Π is MaxSNP-hard if all the MaxSNP problems L-reduce to Π.
A problem is MaxSNP-complete if it is both MaxSNP-hard and in MaxSNP.

- if the instance is positive, then there exists a proof such that V accepts with probability at least α;
- if the instance is negative, then for any proof V accepts with probability at most β.

The following theorem is proved in [49] (see also Theorem 7 in [107]), presenting a further refinement of the characterization of NP.

Theorem 9. *[49] For every $\epsilon > 0$,*

$$3\text{SAT} \in \text{PCP}_{1,\epsilon}\left[(1 + o(1)) \log n + O\left(\log(1/\epsilon)\right), O\left(\log(1/\epsilon)\right)\right]$$

A recent improvement [107] of Theorem 9 (a PCP Theorem with two-query projection tests, sub-constant error and almost-linear size) has some important corollaries in subexponential-time approximation. In particular:

Corollary 7. *[107]* Under ETH, for every $\epsilon > 0$, and $\delta > 0$, it is impossible to distinguish between instances of MAX 3SAT with m clauses where at least $(1-\epsilon)m$ are satisfiable from instances where at most $((7/8)+\epsilon)m$ are satisfiable, in time $O(2^{m^{1-\delta}})$.

Under LPC, a stronger version of this result follows by a standard argument.

Lemma 12. *Under* LPC *and* ETH*, there exists $r < 1$ such that for every $\epsilon > 0$ it is impossible to distinguish between instances of* MAX 3SAT *with m clauses where at least $(1-\epsilon)m$ are satisfiable from instances where at most $(r+\epsilon)m$ are satisfiable, in time $2^{o(m)}$.*

Proof. Suppose that $3\text{SAT} \in \text{PCP}_{1,\beta}[\log|\phi| + D, E]$, where $\beta \in (0,1)$, $|\phi|$ is the sum of the lengths of clauses in the 3SAT instance, D and E are constants.

Given an $\epsilon > 0$, let ϵ' such that $0 < \epsilon' < \epsilon$. Given an instance ϕ of 3SAT on n variables, we apply the sparsification lemma [81] (with ϵ') to get $2^{\epsilon' n}$ instances ϕ_i on at most n variables. Since each variable appears at most $c_{\epsilon'}$ times in ϕ_i, the global size of ϕ_i is $|\phi_i| \leqslant c_{\epsilon'} n$.

Then for each formula ϕ_i we use the previous PCP assumption. The size of the proof is at most $E2^{|R|} = c'|\phi_i| \leqslant cn$ for some constants c', c that depend on ϵ' (where $|R| = \log n + D$ is the number of random bits) since $E2^{|R|}$ is the total number of bits that we read in the proof. Take one variable for each bit in the proof: $x_1, \cdots, x_{cn}$. For each random string R: take all the 2^E possibilities for the E variables read, and write a CNF formula which is satisfied if and only if the verifier accepts. This can be done with a formula with a constant number of clauses, say C_1, each clause having a constant number of variables, say C_2 (C_1 and C_2 depends only on E).

If we consider the CNF formula formed by all these CNF formulas for all the random clauses, we get a CNF formula with $C_1 2^{|R|}$ clauses on variables $x_1, \cdots, x_{cn}$. The clauses are on C_2 variables but by adding $\lceil C_2/4 \rceil$ variables we can replace a clause on C_2 variables by an equivalent set of 3-clauses. This way we get a 3-CNF formula and multiply the number of variables and the number of clauses by a constant, so they are still linear in n. For each R we have a set of say C_1' clauses.

Suppose that we start from a satisfiable formula ϕ_i. Then there exists a proof for which the verifier always accepts. By taking the corresponding values for the variables x_i, and extending it properly to the new variables y, all the clauses are satisfied.

Suppose that we start from a non satisfiable formula ϕ_i. Then for any proof (i.e., any truth values of variables), the verifier rejects for a proportion of at least $(1-\beta)$ of the random strings. If the verifier rejects for a random string R, then in the set of clauses corresponding to this variable at least one clause is not satisfied. It means that among the $C_1' 2^{|R|}$ clauses (total number of clauses), at least $(1-\beta)\cdot 2^{|R|}$ are not satisfied, i.e., a fraction $(1-\beta)/C_1'$ of the clauses.

Then either $m = C_1' 2^{|R|} = O(n)$ clauses are satisfiable, or at least $m(1-\beta)/C_1'$ clauses are not satisfied by each assignment. Distinguishing between these sets in time $2^{o(m)}$ would determine whether ϕ_i is satisfiable or not in $2^{o(n)}$. Doing this for each ϕ_i would solve 3SAT in time $poly(n)2^{\epsilon' n} + 2^{\epsilon' n} O(2^{o(n)}) = O(2^{\epsilon n})$. This is valid for any $\epsilon > 0$ so it would contradict ETH. □

The (conditional) hardness result of approximating MAX 3SAT stated in Lemma 12 will be the basis of the negative results of parameterized approximation in Section 3.2.1.

Let us now present two useful gap amplification results for MAX INDEPENDENT SET. First, the so-called self-improvement property [71] can also be proven for MAX INDEPENDENT SET in the case of parameterized approximation.

Lemma 13. *If there exists a parameterized r-approximation algorithm for some $r \in (0,1)$ for* MAX INDEPENDENT SET*, then this is true for any $r \in (0,1)$.*

Also, the very powerful tool of expander graphs allows us to derive a gap amplification for MAX INDEPENDENT SET, proved in Theorem 11. But first, in order to prove the theorem, let us recall some basics about gap amplification and expander graphs.

Definition 8. A graph G is a (n,d,α)-expander graph if:

(i) G has n vertices;

(ii) G is d-regular;

(iii) all the eigenvalues λ of G but the largest one is such that $|\lambda| \leqslant \alpha d$.

Lemma 14. *For any positive integer $k \in \mathbb{N}^*$ and any $\alpha > 0$ there exist d and a (k^2, d, α)-expander graph. Moreover, d depends only on α, and this graph can be computed in polynomial time for every fixed α.*

Proof. The lemma follows from the following two lemmata.

Lemma 15. *[70, 79] For every positive integer k, there exists a $(k^2, 8, {}^5\sqrt{2}/8)$-expander graph, computable in polynomial time.*

If G is a graph with adjacency matrix M, let us denote G^k the graph with adjacency matrix M^k. Then, the following lemma also holds.

Lemma 16. *[112] If G is an (n, d, α)-expander graph, then G^k is an (n, d^k, α^k)-expander graph.*

Lemma 16. G^k is obviously d^k regular, and the eigenvalues of G^k are the eigenvalues of G to the power of k. □

To complete the proof of the lemma, take $\alpha > 0$ and let p be the smallest integer such that $({}^5\sqrt{2}/8)^p \leqslant \alpha$. Graph G^p is as required and Lemma 14 is proved. □

Let G be a graph on n vertices and H be a (n, d, α)-expander graph. Let t be a positive integer. Build the graph G'_t on $N = nd^{t-1}$ vertices: each vertex corresponds to a $(t-1)$-random walk $x = (x_1, \cdots, x_t)$ on H (meaning that x_1 is chosen at random, and x_{i+1} is chosen randomly in the set of neighbors of x_i), and two vertices $x = (x_1, \cdots, x_t)$ and $y = (y_1, \cdots, y_t)$ in G'_t are adjacent iff $\{x_1, \cdots, x_t, y_1, \cdots, y_t\}$ is a clique in G. Then, the following holds.

Theorem 10. *[79] Let G be a graph on n vertices and H be a (n, d, α)-expander graph. If $b > 6\alpha$ then, denoting by $\omega(G)$ the clique-number (size of a maximum clique) of G, it holds that:*

- *if $\omega(G) \leqslant bn$ then $\omega(G'_t) \leqslant (b + 2\alpha)^t N$;*
- *if $\omega(G) \geqslant bn$ then $\omega(G'_t) \geqslant (b - 2\alpha)^t N$.*

We are well prepared now to prove the following theorem.

Theorem 11. *Let G be a graph on n vertices (for sufficiently large n) and $a > b$ be two positive real numbers. Then for any real $r > 0$ one can build in polynomial time a graph G_r and specify constants a_r and b_r such that:*

(i) G_r has $N \leqslant Cn$ vertices, where C is some constant independent of G (but may depend on r);

(ii) *if* $\omega(G) \leqslant bn$ *then* $\omega(G_r) \leqslant b_r N$*;*

(iii) *if* $\omega(G) \geqslant an$ *then* $\omega(G_r) \geqslant a_r N$*;*

(iv) $b_r/a_r \leqslant r$.

Proof. Set $k = \lceil \sqrt{n} \rceil$. We modify G by adding $k^2 - n$ dummy (isolated) vertices. Let G' be the new graph. It has $n' = k^2$ vertices. Note that $n' \leqslant (\sqrt{n}+1)^2 = n + 2\sqrt{n} + 1 = n + o(n)$. Let n be such that $1 - \epsilon \leqslant n/n' \leqslant 1$ for a small ϵ. Due to Lemma 14, we consider a (k^2, d, α)-expander graph H for a sufficiently small α (the value of which will be fixed later). According to Theorem 10 (applied on G') we build in polynomial time a graph G'_t on $N = n'd^t$ vertices such that (choosing $\alpha < b/6$):

- if $\omega(G) \leqslant bn$ then $\omega(G') = \omega(G) \leqslant bn'$, hence $\omega(G'_t) \leqslant (b + 2\alpha)^t N$;
- if $\omega(G) \geqslant an$ then $\omega(G') = \omega(G) \leqslant an'(1-\epsilon)$, hence $\omega(G'_t) \geqslant (a(1-\epsilon) - 2\alpha)^t N$.

We choose ϵ and α such that $a(1-\epsilon) - 2\alpha > b + 2\alpha$.
Then, we choose t such that $(a(1-\epsilon)-2\alpha)^t/(b+2\alpha)^t \leqslant r$. The number of vertices of G'_t is clearly linear in n (first point of the theorem). Then, $b_r = (b+2\alpha)^t$ and $a_r = (a(1-\epsilon) - 2\alpha)^t$ fulfill items (ii), (iii) and (iv) of theorem's statement. □

3.2 Some Consequences of (Almost-)Linear Size PCP System

3.2.1 Parameterized Inapproximability Bounds

It is shown in [38] that, under ETH, for any function f no algorithm running in time $f(k)n^{o(k)}$ can determine whether there exists an independent set of size k, or not (in a graph with n vertices). A challenging question is to obtain a similar result for approximation algorithms for MAX INDEPENDENT SET. In the sequel, we propose a reduction from MAX 3SAT to MAX INDEPENDENT SET that, based upon the negative result of Corollary 7, only gives a negative result for *some* function f (because Corollary 7 only avoids *some* subexponential running times and not, for instance, time $2^{m/\log m}$). However, this reduction gives the inapproximability result sought, if the consequence of LPC given in Lemma 12 (which strengthens Corollary 7 and seems to be a much weaker assumption than LPC) is used instead. We emphasize the fact that the results in

this section are valid as soon as a hardness result for MAX 3SAT as that in Lemma 12 holds.

The proof of the following theorem essentially combines the parameterized reduction in [38] and a classic gap-preserving reduction.

Theorem 12. *Under* LPC *and* ETH*, there exists $r < 1$ such that no approximation algorithm for* MAX INDEPENDENT SET *running in time $f(k)n^{o(k)}$ can achieve approximation ratio r in graphs of order n.*

Proof. We denote by N the number of vertices in a graph (to avoid confusion with the number of variables in a formula). We will show that the existence of such an algorithm for any $r' < 1$ would contradict the hardness result for MAX 3SAT in Lemma 12, hence ETH or LPC. Consider a constant $r < 1$. Let $0 < \epsilon < 1 - r$. We show that the existence of an $(r + \epsilon)$-approximation algorithm for MAX INDEPENDENT SET running in time $f(k)N^{o(k)}$ would allow to distinguish in time $2^{o(m)}$ between instances of MAX 3SAT where $(1 - \epsilon')m$ clauses are satisfiable and instances where at most $(r + \epsilon')m$ clauses are satisfiable, for some $\epsilon' > 0$. W.l.o.g., we can assume that f is increasing, and that $f(k) \geqslant 2^k$.

Take an instance I of MAX 3SAT, let K be an integer that will be fixed later. We build a graph G_I as follows:

- partition the m clauses into K groups $H_1, \cdots, H_K$ each of them containing roughly m/K clauses;
- each group H_i involves a number $s_i \leqslant 3m/K$ of variables; for all possible values of these variables, add a vertex in the graph G_I if these values satisfy at least $\lambda m/K$ clauses in H_i (the value of λ will also be fixed later);
- finally, add an edge between two vertices if they have one contradicting variable.

In particular, the vertices corresponding to the same group of clauses form a clique. It is easy to see that the so-constructed graph contains $N \leqslant K2^{3m/K}$ vertices.

The following easy claim holds.

Claim 1. If a variable assignment A satisfies at least $\lambda m/K$ clauses in at most s groups, then it satisfies at most $\lambda m + \left(s(1-\lambda)m/K\right)$ clauses.

Proof of claim. A satisfies at most m/K clauses in at most s groups, and at most $\lambda m/K$ in the other $K-s$ groups, so in total at most $sm/K + (K-s)\lambda m/K = \lambda m + s(1-\lambda)m/K$, that completes the proof of the claim. $\diamond$

Now, let us go back to the proof of the theorem. Assume an independent set of size at least t in G_I. Then one can achieve a partial solution that satisfies at least $\lambda m/K$ clauses in at least t groups. So, at least $t\lambda m/K$ clauses are satisfiable. In other words, if at most $(r+\epsilon')m$ clauses are satisfiable, then a maximum independent set in G_I has size at most $K \cdot (r+\epsilon')/\lambda$. Suppose that at least $(1-\epsilon')m$ clauses are satisfiable. Then, using the claim, there exists a solution satisfying at least $\lambda m/K$ clauses in at least $\left((1-\epsilon'-\lambda)/(1-\lambda)\right) \cdot K$ groups; otherwise, it should be $\lambda m + s(1-\lambda)m/K < (1-\epsilon')m$. Then, there exists an independent set of size $\left((1-\epsilon'-\lambda)/(1-\lambda)\right) \cdot K$ in G_I.

Now, set $K = \lceil f^{-1}(m)/(1-\epsilon^2) \rceil$. Set also $\lambda = 1-\epsilon$, and $\epsilon' = \epsilon^3$. Run the assumed $(r+\epsilon)$-approximation parameterized algorithm for MAX INDEPENDENT SET in G_I with parameter $k = (1-\epsilon^2)K$. Then, if at least $(1-\epsilon')m$ clauses are satisfiable, there exists an independent set of size at least $\left((1-\epsilon'-\lambda)/(1-\lambda)\right) \cdot K = (1-\epsilon^3/\epsilon)K = (1-\epsilon^2)K = k$; so, the algorithm must output an independent set of size at least $(r+\epsilon)k$. Otherwise, if at most $(r+\epsilon')m$ clauses are satisfiable, the size of an independent set is at most $K \cdot (r+\epsilon')/\lambda = K \cdot (r+\epsilon^3)/(1-\epsilon) = k \cdot (r+\epsilon^3)/((1-\epsilon)(1-\epsilon^2)) = k(r + r\epsilon + o(\epsilon))$.

So, for ϵ sufficiently small, the algorithm allows to distinguish between the two cases of MAX 3SAT (for ϵ'), i.e., whether at least $(1-\epsilon')m$ clauses are satisfiable, or at most $(r+\epsilon)m$ clauses.

The running time of the algorithm is $f(k)N^{o(k)}$, with $f(k) = f((1-\epsilon^2)K) = m$ and $N^{o(k)} = N^{k/\psi(k)}$, for some increasing and unbounded function ψ. So, $N^{o(k)} = (K2^{3m/K})^{k/\psi(k)} = K2^{3m(1-\epsilon^2)/\psi(k)} = O(2^{o(m)})$. $\square$

The following result follows from Lemma 13 and Theorem 12.

Corollary 8. *Under* LPC *and* ETH*, for any $r \in (0,1)$ there is no r-approximation parameterized algorithm for* MAX INDEPENDENT SET *(i.e., an algorithm that runs in time $f(k)poly(n)$ for some function f).*

Let us now consider MIN DOMINATING SET which is known to be W[2]-hard [51]. The existence of a parameterized constant-factor approximation algorithm for this problem is open [53].

Here, we present an approximation preserving reduction (fitting the parameterized framework) which, given a graph $G(V,E)$ on n vertices where V is a set of K cliques

$C_1, \cdots, C_K$, builds a graph $G'(V', E')$ such that G has an independent set of size α if and only if G' has a dominating set of size $2K - \alpha$. Using the fact that the graphs produced in the proof of Theorem 12 are of this form (vertex set partitioned into cliques), this reduction will allow us to obtain a lower bound (based on the same hypothesis) for the approximation of MIN DOMINATING SET.

The graph G' is built as follows:

- for each clique C_i in G, add a clique C'_i of the same size in G'; add also: an independent set S_i of size $3K$, each vertex in S_i being adjacent to all vertices in C'_i and a special vertex t_i adjacent to all the vertices in C'_i;
- for each edge $e = \{u, v\}$ with u and v *not* in the same clique in G, add an independent set W_e of size $3K$; suppose that $u \in C_i$ and $v \in C_j$; then, each vertex in W_e is linked to t_i and to all vertices in C'_i but u, and to t_j and to all vertices in C'_j but v.

Informally, the reduction works as follows. The set S_i ensures that we have to take at least one vertex in each C'_i, the fact that $|W_e| = 3K$ ensures that it is never interesting to take a vertex in W_e. If we take t_i in a dominating set, this will mean that we do not take any vertex in the set C_i in the corresponding independent set in G. If we take one vertex in C'_i (but not t_i), this vertex will be in the independent set in G. Let us state this property in the following lemma.

Lemma 17. *G has an independent set of size α if and only if G' has a dominating set of size $2K - \alpha$.*

Proof. Suppose that G has an independent set S of size α. Then, S has one vertex in α sets C_i, and no vertex in the other $K - \alpha$ sets. We build a dominating set T in G' as follows: for each vertex in S we take its copy in G'. For each clique C_i without vertices in S, we take t_i and an arbitrary vertex in C'_i. The set T has size $\alpha + 2(K - \alpha) = 2K - \alpha$. For each C'_i, one of its vertices is in T; so, vertices in C'_i, t_i and vertices in S_i are dominated. Now consider a vertex in W_e with $e = \{u, v\}$, $u \in C_i$ and $v \in C_j$. If $C_i \cap S = \emptyset$ (or $C_j \cap S = \emptyset$), then $t_i \in T$ (or $t_j \in T$) and, by construction, t_i is adjacent to all vertices in W_e. Otherwise, there exist $w \in S \cap C_i$ and $x \in S \cap C_j$. Since S is an independent set, either $w \neq u$ or $x \neq v$. If $w \neq u$, by construction w (its copy in C'_i) is adjacent to all vertices in W_e and, similarly, for x if $x \neq v$. So, T is a dominating set.

Conversely, suppose that T is a dominating set of size $2K - \alpha$. Since S_i is an independent set of size $3K$, T cannot contain S_i entirely, so at least one vertex in $N(S_i)$ has to be in T. But any vertex in $N(S_i)$ dominates all the vertices in S_i. Thus, we can assume that $T \cap S_i = \emptyset$ and the same occurs with W_e. In particular, there exists at least one vertex in T in each C'_i. Now, suppose that T has two different vertices u and v in the same C'_i. Then, we can replace v by t_i getting a dominating set (vertices in S_i are still dominated by u, and any vertex in some W_e which is adjacent to v is adjacent to t_i). So, we can assume that T has the following form: exactly one vertex in each C'_i, and $K - \alpha$ vertices t_i. Hence, there are α cliques C'_i, where t_i is not in T. We consider in G the set S constituted by the α vertices in T in these α sets. Take two vertices u and v in S with, say, $u \in C_i$ and $v \in C_j$ (with $t_i \notin T$ and $t_j \notin T$). If there were an edge $e = \{u, v\}$ in G, neither u nor v would have dominated a vertex in W_e (by construction). Since neither t_i nor t_j is in T, this set would not have been a dominating set, a contradiction. So, S is an independent set. □

Theorem 13. *Under* LPC *and* ETH*, there exists an $r > 1$ such that there is no r-approximation algorithm for* MIN DOMINATING SET *running in time $f(k)n^{o(k)}$ where n is the order of the graph.*

Proof. In the proof of Theorem 12, we produce a graph G_I which is made of K cliques and such that: if at least $(1 - \epsilon)m$ clauses are satisfiable in I, then there exists an independent set of size $(1 - O(\epsilon))K$; otherwise (at most $(r + \epsilon)m$ clauses are satisfiable in I), the maximum independent set has size at most $(r + O(\epsilon))K$. The previous reduction transforms G_I in a graph G'_I such that, applying Lemma 17, in the first case there exists a dominating set of size at most $2K-(1-O(\epsilon))K = K(1+O(\epsilon))$ while, in the second case, the size of a dominating set is at least $2K - (r + O(\epsilon))K = K(2 - r - O(\epsilon))$. Thus, we get a gap with parameter $k' = K(1 + O(\epsilon))$. Note that the number of vertices in G'_I is $n' = n + K + 3K + 3K|E_I| = O(n^3)$ (where E_I is the set of edges in G_I). If we were able to distinguish between these two sets of instances in time $f(k')n'^{o(k')}$, this would allow to distinguish the corresponding independent set instances in time $f(k')n'^{o(k')} = g(k)n^{o(k)}$ since $k' = K(1 + O(\epsilon)) = k(1 + O(\epsilon))$ ($k = K(1 - \epsilon^3)$ being the parameter chosen for the graph G_I). □

Such a lower bound immediately transfers to SET COVER since a graph on n vertices for MIN DOMINATING SET can be easily transformed into an equivalent instance of SET COVER with ground set and set system both of size n.

Corollary 9. *Under* LPC *and* ETH*, there exists $r > 1$ such that there is no r-approximation algorithm for* SET COVER *running in time $f(k)m^{o(k)}$ in instances with m sets.*

3.2.2 On the Approximability of MAX INDEPENDENT SET and Related Problems in Subexponential Time

As mentioned in Section 3.1, an almost-linear size PCP construction [107] for 3SAT allows to get the negative result stated in Corollary 7. In this section, we present further consequences of Theorem 9, based upon a combination of known reductions with (almost) linear size amplifications of the instance.

First, Theorem 9 combined with the reduction in [7] showing inapproximability results for MAX INDEPENDENT SET in polynomial time and the gap amplification of Theorem 11, leads to the following result.

Theorem 14. *Under* ETH*, for any $r > 0$ and any $\delta > 0$, there is no r-approximation algorithm for* MAX INDEPENDENT SET *running in time $O(2^{n^{1-\delta}})$, where n is the order of the input graph.*

Proof. Again, to avoid confusion we denote in this proof by N the number of vertices in a graph. Given an $\epsilon > 0$, let ϵ' be such that $0 < \epsilon' < \epsilon$. Given an instance ϕ of 3SAT on n variables, we first apply the sparsification lemma [81] (with ϵ') to get $2^{\epsilon' n}$ instances ϕ_i on at most n variables. Since each variable appears at most $c_{\epsilon'}$ times in ϕ_i, the global size of ϕ_i is $|\phi_i| \leqslant c_{\epsilon'} n$.

Consider a particular ϕ_i, $r > 0$ and $\delta > 0$. We use the fact that 3SAT $\in$ PCP$_{1,r}[(1+o(1))\log|\phi| + D_r, E_r]$ (where D_r and E_r are constants that depend only on r), in order to build the following graph G_{ϕ_i} (see also [7]):

- for any random string R of size $(1 + o(1))\log|\phi| + D_r$, and any possible value of the E_r bits read by V, add a vertex in the graph if V accepts;
- if two vertices are such that they have at least one contradicting bit (they read the same bit which is 1 for one of them and 0 for the other one), add an edge between them.

In particular, the set of vertices corresponding to the same random string is a clique.

Assume that ϕ_i is satisfiable. Then there exists a proof for which the verifier accepts for any random string R. Take for each random string R the vertex in G_{ϕ_i}

corresponding to this proof. There is no conflict (no edge) between any of these $2^{|R|}$ vertices, hence $\alpha(G_{\phi_i}) = 2^{|R|}$ (where, in a graph G, $\alpha(G)$ denotes the size of a maximum independent set).

If ϕ_i is not satisfiable, then $\alpha(G_{\phi_i}) \leqslant r2^{|R|}$. Indeed, suppose that there is an independent set of size $\alpha > r2^{|R|}$. This independent set corresponds to a set of bits with no conflict, defining part of a proof that we can arbitrarily extend to a proof Π. The independent set has α vertices corresponding to α random strings (for which V accepts), meaning that the probability of acceptance for this proof Π is at least $\alpha/2^{|R|} > r$, a contradiction with the property of the verifier.

Furthermore, G_{ϕ_i} has $N \leqslant 2^{|R|}2^{E_r} \leqslant C'|\phi_i|^{1+o(1)} = Cn^{1+o(1)}$ vertices (for some constants C, C' that depend on ϵ') since $|\phi_i| \leqslant c_{\epsilon'}n$. Then, one can see that, for any $r' > r$, an r'-approximation algorithm for MAX INDEPENDENT SET running in time $O(2^{N^{1-\delta}})$ would allow to decide whether ϕ_i is satisfiable or not in time $O(2^{n^{1-\delta'}})$ for some $\delta' < \delta$. Doing this for each of the formula ϕ_i would allow to decide whether ϕ is satisfiable or not in time $poly(n)2^{\epsilon' n} + 2^{\epsilon' n}O(2^{n^{1-\delta'}}) = O(2^{\epsilon n})$. This is valid for any $\epsilon > 0$ so it would contradict ETH.

Combining this reduction with the gap amplification of Theorem 11 allows to create a gap with any constant in $(0, 1)$. Since the reduction in this amplification is linear with respect to the number of vertices, we get the claimed result. □

Let us note that the result of Theorem 14 has been powerfully improved very recently in [37], where it is proved that *under* ETH*, for any $\delta > 0$ any r larger than some constant, any r-approximation algorithm for* MAX INDEPENDENT SET *must run in at least $2^{n^{1-\delta}/r^{1+\delta}}poly(n)$ time*.

Note also that, since (for $k \leqslant n$), $n^{k^{1-\delta}} = O(2^{n^{1-\delta'}})$, for some $\delta' < \delta$, the following holds.

Corollary 10. *Under* ETH*, for any $r > 0$ and any $\delta > 0$, there is no r-approximation algorithm for* MAX INDEPENDENT SET *running in time $O(n^{k^{1-\delta}})$, where n is the order of the input graph, and k is the size of a maximum independent set.*

The results of Theorem 14 and Corollary 10 can be immediately extended to problems that are linked to MAX INDEPENDENT SET by approximation preserving reductions (that preserve at least constant ratios) that have linear amplifications of the sizes of the instances, as in the following proposition.

Proposition 8. *Under* ETH*, for any $r > 0$ and any $\delta > 0$, there is no r-approximation algorithm for either* MAX SET PACKING *or* MAX COMPLETE BIPARTITE SUBGRAPH *running in time $O(2^{n^{1-\delta}})$ in a graph of order n.*

Proof. Consider the following reduction from MAX INDEPENDENT SET to MAX COMPLETE BIPARTITE SUBGRAPH given in [118]. Let $G(V, E)$ be an instance of MAX INDEPENDENT SET of order n. Construct a graph $G'(V', E')$ for MAX COMPLETE BIPARTITE SUBGRAPH by taking two distinct copies of G (denote them by G_1 and G_2, respectively) and adding the following edges: a vertex v_{i_1} of copy G_1 is linked with a vertex v_{j_2} of G_2, if and only if either $i = j$ or $(v_i, v_j) \in E$. The graph G' has $2n$ vertices.

Let now S be an independent set of G. Then, obviously, taking the two copies of S in G_1 and G_2 induces a bipartite graph of size $2|S|$. Conversely, consider an induced bipartite graph in G' of size t, and take the largest among the two color classes. By construction, it corresponds to an independent set in G, whose size is at least $t/2$ (note that it cannot contain 2 copies of the same vertex). So, any r-approximate solution for MAX COMPLETE BIPARTITE SUBGRAPH in G' can be transformed into an r-approximate solution for MAX INDEPENDENT SET in G. Observe finally that the size of G' is two times the size of G. □

Dealing with minimization problems, Theorem 14 and Corollary 10 can be extended to COLORING, using the reduction given in [98].

Note that this reduction uses the particular structure of graphs produced in the inapproximability result in [7] (as in Theorem 14). Hence, the following result can be derived.

Proposition 9. *Under* ETH*, for any $r > 1$ and any $\delta > 0$, there is no r-approximation algorithm for* COLORING *running in time $O(2^{n^{1-\delta}})$ in a graph of order n.*

Proof. In [98] the following reduction is presented. Given a graph G whose vertex set is partitioned into K cliques each of size S, and given a prime number $q > S$, a graph H_q having the following properties can be built in polynomial time:

- the vertex set of H_q is partitioned into q^2K cliques, each of size q^3;
- $\alpha(H_q) \leqslant \max\{q^2\alpha(G); q^2(\alpha(G) - 1) + K; qK\}$;
- if $\alpha(G) = K$, then $\chi(H_q) = q^3$.

Fix a ratio $r > 1$, and let $r_{IS} > 0$ be such that $r_{IS} + r_{IS}^2 \leqslant 1/r$. Start from the graph G_{ϕ_i} produced in the proof of Theorem 14 for ratio r_{IS}. The vertex set of G_{ϕ_i} is partitioned into $K = 2^{|R|}$ cliques, each of size at most 2^{E_r}. By adding dummy vertices (a linear number, since E_r is a fixed constant), we can assume that each clique has the same size $S = 2^{E_r}$, so the number of vertices in G_{ϕ_i} is $N = KS = 2^{|R|}2^{E_r}$.

Let $q > \max\{S, 1/r_{IS}\}$ be a prime number, and consider the graph H_q produced from G_{ϕ_i} by the reduction in [98] mentioned above. If ϕ_i is satisfiable, $\alpha(G_{\phi_i}) = K$ and then by the third property of the graph H_q, $\chi(H_q) = q^3$. Otherwise, by the second property $\alpha(H_q) \leqslant \max\{q^2\alpha(G_\phi); q^2(\alpha(G_\phi) - 1) + K; qK\}$. Formula ϕ_i being not satisfiable, $\alpha(G_{\phi_i}) \leqslant r_{IS}K$. By the choice of q, $qK \leqslant q^2 r_{IS}K$, so $\alpha(H_q) \leqslant q^2 r_{IS}K + K = (q^2 r_{IS} + 1)K$. Since the number of vertices in H_q is Kq^5, we get that $\chi(H_q) \geqslant q^5/(q^2 r_{IS}+1)$. The gap created for the chromatic number in the two cases is then at least:

$$\frac{q^5}{(q^2 r_{IS} + 1)\, q^3} = \frac{1}{r_{IS} + 1/q^2} \geqslant \frac{1}{r_{IS} + r_{IS}^2} \geqslant r$$

The result follows since H_q has Kq^5 vertices and q is a constant (that depends only on the ratio r and on the constant number of bits p read by V), so the size of H_q is linear in the size of G_{ϕ_i}. □

Concerning the approximability of MIN VERTEX COVER and MIN SAT in subexponential time, the following holds.

Proposition 10. *Under* ETH*, for any $\epsilon > 0$ and any $\delta > 0$, there is no $((7/6) - \epsilon)$-approximation algorithm for* MIN VERTEX COVER *running in time $O(2^{n^{1-\delta}})$ in graphs of order n, nor for* MIN SAT *running in time $O(2^{m^{1-\delta}})$ in CNF formulæ with m clauses.*

Proof. We combine the following theorem with a well known reduction.

Theorem 15. *[107] Under* ETH*, for every $\epsilon > 0$, and $\delta > 0$, it is impossible to distinguish between instances of* MAX 3-LIN *with m equations where at least $(1-\epsilon)m$ are satisfiable from instances where at most $((1/2) + \epsilon)m$ are satisfiable, in time $O(2^{m^{1-\delta}})$.*

Consider an instance I of MAX 3-LIN on m equations. Build the following graph G_I:

- for any equation and any of the eight possible values of the 3 variables in it, add a vertex in the graph if the equation is satisfied;
- if two vertices are such that they have one contradicting variable (the same variable has value 1 for one vertex and 0 for the other one), then add an edge between them.

In particular, the set of vertices corresponding to the same equation is a clique. Note that each equation is satisfied by exactly 4 values of the variables in it. Then, the number of vertices in the graph is $N = 4m$. Consider an independent set S in the graph G_I. Since there is no conflict, it corresponds to a partial assignment that can be arbitrarily completed into an assignment τ for the whole system. Each vertex in S corresponds to an equation satisfied by τ (and S has at most one vertex per equation), so τ satisfies (at least) $|S|$ equations. Reciprocally, if an assignment τ satisfies α clauses, there is obviously an independent set of size α in G_I. Hence, if $(1-\epsilon)m$ equations are satisfiable, there exists an independent set of size at least $(1-\epsilon)m$, i.e., a vertex cover of size at most $N-(1-\epsilon)m = N(3/4+\epsilon/4)$. If at most $((1/2)+\epsilon)m$ equations are satisfiable, then each vertex cover has size at least $N-((1/2)+\epsilon)m = N(7/8-\epsilon/4)$.

We now handle the MIN SAT problem via the following reduction [101]. Given a graph G, build the following instance on MIN SAT. For each edge $\{v_i, v_j\}$ add a variable x_{ij}. For each vertex v_i add a clause c_i. Variable x_{ij} appears positively in c_i and negatively in c_j. Then, take a vertex cover V^* of size k; for any x_{ij}, fix the variable to true if $v_i \in V^*$, to false otherwise. Consider a clause c_j with $v_j \notin V^*$. If $\overline{x_{ij}}$ is in c_j then v_i is in V^*, hence x_{ij} is true; if x_{ji} is in c_j then, by construction, x_{ji} is false. So c_j is not satisfied, and the assignment satisfies at most k clauses. Conversely, consider a truth assignment that satisfies k clauses $c_{i_1}, \cdots, c_{i_k}$. Consider the vertex set $V^* = \{v_{i_1}, \cdots, v_{i_k}\}$. For an edge $\{v_i, v_j\}$, if x_{ij} is set to true, then c_i is satisfied and v_i is in V^*; otherwise, c_j is satisfied and v_j is in V^*; so V^* is a vertex cover of size k. Since the number of clauses in the reduction equals the number of vertices in the initial graph, the result is concluded. □

All the results given in this section are valid under ETH and rule out some ratios in subexponential time of the form $2^{n^{1-\delta}}$. It is worth noticing that if LPC holds, then all these results would hold for *any* subexponential time (in contrast to the result of [37] for MAX INDEPENDENT SET that holds only for the form $2^{n^{1-\delta}}$). Note that this is in some sense optimal since it is easy to see that, for any increasing and unbounded function

$r(n)$, MAX INDEPENDENT SET is approximable within ratio $1/r(n)$ in subexponential time (simply consider all the subsets of V of size at most $n/r(n)$ and return the largest independent set among these sets).

Corollary 11. *Under* LPC *and* ETH *Theorem 14 and Propositions 8, 9 and 10 hold for any time complexity* $2^{o(n)}$.

Indeed, using LPC, the same proof as in Theorem 14 creates for each ϕ_i a graph on $N = O(n)$ variables with either an independent set of size αN (if ϕ_i is satisfiable) or a maximum independent set of size at most $(\alpha/2)N$ (if ϕ_i is not satisfiable). Then using expander graphs, usual arguments allow to amplify this gap from $1/2$ to any constant $r > 0$ while preserving the linear size of the instance (see Theorem 11). Results for the other problems immediately follow from the same arguments as above.

3.3 Subexponential Approximation Preserving Reducibility

In this section, we study subexponential approximation preserving reducibility. Recall that APETH(Π) (Hypothesis 1) states that it is hard to approximate in subexponential time problem Π, within some constant ratio r. We exhibit that a set of problems are APETH-equivalent using the notion of *approximation preserving sparsification*. We then link APETH with approximation in subexponential FPT-time.

3.3.1 Approximation Preserving Sparsification and APETH Equivalences

We first informally describe the basic idea behind sparsification [81] and its use for deriving lower bounds in exact computation. Assuming a reference problem Π' cannot be solved in $O^*(\lambda^n)$, for some $\lambda > 1$, we are interested in showing that another problem Π cannot be solved in $O^*(f(\lambda)^n)$. For instance, if the reference problem is SAT and $\lambda = 2$, our assumption is the Strong ETH (SETH).

For doing this, we use reductions from Π' to Π. Note that one can easily derive negative results if there exists a linear reduction from Π' to Π (i.e., a reduction with linear instance-size amplification). Although, linear reductions are quite rare, so that approach is limited. Yet, reductions where Π' is a graph problem, amplifying the instance to a size $O(n + m)$ where n is the number of vertices and m the number of edges (or, dealing with some satisfiability problem, n is the number of variables, and m

the number of clauses) are much less rare. But, in general m is not linear in n but quadratic.

A way to overcome that issue is to "*sparsify*" instances of Π', producing, from an instance I, $\gamma(n)$ instances where the number of edges is linear to n and to prove that, for at least one of them, an optimal solution is also (or can be transformed in time at most $O^*(\gamma(n))$ into) an optimal solution for I. We then apply the reduction to all of these sparsified instances.

The sparsifier for SAT, presented in [81], shows that *for every integer $k \geqslant 3$, and every $\varepsilon > 0$ there exists a constant $C_{\varepsilon,k}$ and $2^{\varepsilon n}\ C_{\varepsilon,k}$-sparse instances of k-SAT whose disjunction is equivalent to the initial instance*. But, as noticed above this idea does not work for approximation.

Recall that the sparsification lemma for 3SAT reduces a formula ϕ to a set of formulas ϕ_i with bounded occurrences of variables such that solving the instances ϕ_i would allow to solve ϕ. We attempt to build an analogous construction for subexponential approximation using the notion of *approximation preserving sparsification*.

Given an optimization problem Π and some parameter of the instance, Π-B denotes the problem restricted to instances where the parameter is at most B. For example, we can prescribe the maximum degree of a graph or the maximum number of literal occurrences in a formula as the parameter.

Note that we could consider a more general definition, leading to the same theorem, by allowing:

- a slight amplification of the size of I_i ($n_i \leqslant \alpha n$ for some fixed α in item 1)
- an expansion of the ratio in item 3 (if S_i is r-approximate S is $h(r)$ approximate where $h(r)$ goes to 1 when r goes to 1)
- a computation time $2^{\epsilon n}poly(n)$ for g in item 4.

With a slight abuse of notation, let APETH(Π-B) denote the hypothesis: $\exists B$ such that APETH(Π-B), meaning that Π is hard to approximate in subexponential time even for some bounded parameter family of instances. Then the following holds:

Theorem 16. *If there exists an approximation preserving sparsification from* Π *to* Π-B*, then* APETH(Π) *if and only if* APETH(Π-B).

Proof. Obviously, APETH(Π) is implied by APETH(Π-B). Now, assume APETH(Π) holds, for some ratio r. We show that APETH(Π-B) holds for the same ratio. Let

Definition 9. An approximation preserving sparsification from a problem Π to a bounded parameter version Π-B of Π is a pair (f, g) of functions such that, given any $\epsilon > 0$ and any instance I of Π:

1. f maps I into a set $f(I, \epsilon) = (I_1, I_2, \ldots, I_t)$ of instances of Π, where $t \leqslant 2^{\epsilon n}$ and $n_i = |I_i| \leqslant n$; moreover, there exists a constant B_ϵ (independent on I) such that any I_i has parameter at most B_ϵ;
2. for any $i \leqslant t$, g maps a solution S_i of an instance I_i (in $f(I, \epsilon)$) into a solution S of I;
3. there exists an index $i \leqslant t$ such that if a solution S_i is an r-approximation in I_i, then $S = g(I, \epsilon, I_i, S_i)$ is an r-approximation in I;
4. f is computable in time $2^{\epsilon n} poly(n)$, and g is computable in time polynomial in $|I|$.

$\epsilon > 0$, $\epsilon' = \epsilon/2$, and suppose that Π-B is r-approximable in time $2^{\epsilon' n} poly(n)$. Then given an instance I of Π, compute $f(I, \epsilon')$ (in time $2^{\epsilon' n} poly(n)$). For each of the t instances I_i, compute an r-approximate solution S_i in time $2^{\epsilon' n_i} poly(n_i) = 2^{\epsilon' n} poly(n)$, and use g to transform S_i into a solution S for I. Let S^* be the best of these solutions. We obtain S^* in time $2^{\epsilon' n} 2^{\epsilon' n} poly(n) = 2^{\epsilon n} poly(n)$. By item 3 of Definition 9, S^* is an r-approximation of I. We can do this for any ϵ, leading to a contradiction. □

We now illustrate this technique on some problems. It is worth noticing that the sparsification lemma for 3SAT in [81] is *not* approximation preserving[2]; one cannot use it to argue that approximating MAX 3SAT (in subexponential time) is equivalent to approximating MAX 3SAT with bounded occurrences.

Proposition 11. *There exists an approximation preserving sparsification from* MAX INDEPENDENT SET *to* MAX INDEPENDENT SET-B *and one from* MIN VERTEX COVER *to* MIN VERTEX COVER-B.

[2]One of the reasons is that when a clause C is contained in a clause C', a reduction rule removes C', that is safe for the satisfiability of the formula, but not when considering approximation.

Proof. Let $\epsilon > 0$. It is well known that the positive root of $1 = x^{-1} + x^{-1-B}$ goes to one when B goes to infinity. Then, consider a B_ϵ such that this root is at most 2^ϵ. Our sparsification is obtained via a branching tree: the leaves of this tree will be the set of instances I_i; f consists of building this tree; a solution of an instance in the leaf corresponds, via the branching path leading to this leaf, to a solution of the root instance, and that is what g makes.

More precisely, for MAX INDEPENDENT SET, consider the following usual branching tree, starting from the initial graph G: as long as the maximum degree is at least B_ϵ, consider a vertex v of degree at least B_ϵ, and branch on it: either take v in the independent set (and remove $N[v]$), or do not take it. The branching stops when the maximum degree of the graph induced by the unfixed vertices is at most $B_\epsilon - 1$. When branching, at least $B_\epsilon + 1$ vertices are removed when taking v, and one when not taking v; thus the number of leaves is $t \leqslant 2^{\epsilon n}$ (by the choice of B_ϵ). Then, f and g satisfy items 1 and 2 of the definition. For item 3, it is sufficient to note that g maps S_i in S by adding adequate vertices. Then, if we consider the path in the tree corresponding to an optimal solution S^*, leading to a particular leaf G_i, we have that $|S^*| = |S^* \cap G_i| + k$ for some $k \geqslant 0$, and the solution S computed by g is of size $|S| = |S_i| + k$. So, $|S|/|S^*| \geqslant |S_i|/|S^* \cap G_i| \geqslant r$ if S_i is an r-approximation for G_i. The same argument holds also for MIN VERTEX COVER. □

Analogous arguments apply more generally to any problem where we have a "sufficiently good" branching rule when the parameter is large. Indeed, suppose we can ensure the decrease in instance size by $g(B)$ for non decreasing and unbounded function g in all (possibly except for one) branches. Then such a branching rule can be utilized to yield an approximation preserving sparsification as in Proposition 11.

We give another approximation preserving sparsification, where there is no direct branching rule allowing to remove a sufficiently large number of vertices.

Let GENERALIZED DOMINATING SET be defined as follows: given a graph $G(V, E)$ where V is partitioned into V_1, V_2, V_3, we ask for a minimum size set of vertices $V' \subseteq V_1 \cup V_2$ which dominates all vertices in $V_2 \cup V_3$. Of course, the case $V_2 = V$ corresponds to the usual MIN DOMINATING SET problem. Note that GENERALIZED DOMINATING SET is also a generalization of SET COVER, with $V_2 = \emptyset$, V_3 being the ground set and V_1 being the set system.

Proposition 12. *There exists an approximation preserving sparsification from* GENERALIZED DOMINATING SET *to* GENERALIZED DOMINATING SET-B.

Proof. Let $\epsilon > 0$, and consider the following branching algorithm, where $B' \geqslant 4$ will be specified later (as a function of ϵ):

1. remove all edges between two vertices in V_1, as well as all edges between two vertices in V_3;
2. if there exists a vertex $v \in V_1$ of degree at least B', branch on it;
3. otherwise, if there exists a vertex $v \in V_2$ of degree at least B'^2, branch on it;
4. otherwise, if there exists a vertex $v \in V_3$ of degree at least B'^3, branch on a neighbor of v.

Note that branching on a vertex v in V_1 or V_2 means that if v is taken, then v is removed from the graph, its neighbors in V_2 are transferred to V_1 (they are already dominated), while its neighbors in V_3 are removed from the graph. If v is not taken, if it is in V_1 then it is removed from the graph, and if it is in V_2 then it is transferred to V_3 (we still need to dominate it).

By principle, in a leaf of the tree, each vertex in V_1 has degree at most B', while each vertex in V_2 has degree at most B'^2, and each vertex of V_3 has degree at most B'^3. Then the graph has bounded maximum degree $B = B'^3$.

However, when branching it might be the case that only at most one vertex is removed from the graph in each branch. To show that the number of leaves in the tree is indeed sufficiently small, we change the branching measure by introducing appropriate weights on the vertices of the graph. Let $w_1 = \min\{1/2, 1/4 + d(v)/4B'\}$ be the weights of vertices in V_1, $w_2 = \min\{1, 3/4 + d(v)/4B'\}$ and $w_3 = 1/2$ be the weights of vertices in V_2 and V_3 respectively. Then the global weight of G is $W(G) \leqslant n$.

Consider a branching step on a vertex $v \in V_1$ corresponding to item 2 of the algorithm: if v is taken, the weight of the instance is reduced by at least $(1/2) + (B'/4)$ ($1/2$ for v, and at least $1/4$ for each of its neighbors). If v is not taken, then the weight is reduced by $1/2$.

In a branching step on a vertex $v \in V_2$ corresponding to item 3 of the algorithm, if v is taken, the weight of the instance is reduced by at least $1 + B'^2/B' = 1 + B'$. Indeed, there is a weight-reduction of $1/2$ for v, and of at least $1/B'$ for each of its neighbors, since we know that every vertex in V_1 has degree at most $B' - 1$. If v is not taken, the weight reduces by at least $1/4$.

In a branching step on a vertex $w \in V_1 \cup V_2$ neighbor of v corresponding to item 4, when w is taken v is removed, so the degree of at least B'^3 vertices decreases by 1.

Since vertices in V_1 and V_2 have degree at most $B' - 1$ and $B'^2 - 1$ respectively, the total weight is reduced by at least ${B'^3}/{B'^2} = B'$. When w is not taken, the weight is reduced by at least $1/4$.

Then, it suffices to choose B' sufficiently large such that the branching factor of these three branchings is at most 2^ϵ.

The fact that an approximate solution on a leaf can be transferred to an approximate solution to the root is completely similar to the case of independent set. □

Combining Proposition 12 with some reductions, the following can be shown.

Lemma 18. APETH*(*MIN DOMINATING SET*) implies* APETH*(*MAX INDEPENDENT SET-B*).*

Proof. Using Proposition 12, it holds that:

$$\begin{aligned} \text{APETH(MIN DOMINATING SET)} &\Rightarrow \text{APETH(GENERALIZED DOMINATING SET)} \\ &\Rightarrow \text{APETH(GENERALIZED DOMINATING SET-}B\text{)} \end{aligned}$$

Consider an instance $G = (V_1, V_2, V_3, E)$ of GENERALIZED DOMINATING SET-B, and use the following reduction (adapted from [109] to this generalized version). Build a graph $G' = (V', E')$ where:

- for each vertex v in $V_2 \cup V_3$, consider a clique C_v of size $|N[v] \cap (V_1 \cup V_2)|$, where each vertex of C_v corresponds to one vertex in $N[v] \cap (V_1 \cup V_2)$ (note that cliques are disjoint; if a vertex is in the neighborhood of two such vertices, there will be two different vertices in G'); such vertices will be informally referred to as *vertices in the cliques*;
- for each vertex v in $V_1 \cup V_2$, add a vertex v' in G', and link v' to all its homologous vertices in the cliques (there is at most one per clique); hence, if $v \in V_1 \cup V_2$ has t neighbors in $V_2 \cup V_3$, v' will be linked to t vertices; such vertices v' will be informally referred to as *vertices not in the cliques* or *vertices outside the cliques*.

Note that the size of each clique C_v is at most B, so there is at most Bn vertices in all the cliques. There are $|V_1| \leqslant n$ vertices v', so $|V'| \leqslant (B+1)n$, the reduction has linear size (with respect to n). Each vertex in a clique has degree at most $(B-1)+1 = B$, and each vertex v' has degree at most B, so G' has degree at most B.

Let D be a generalized dominating set of G. For each vertex v in $V_2 \cup V_3$, there exists a vertex $w \in D$ dominating it. We select the corresponding vertex in G' in the clique C_v. This adds up to $|V_2 \cup V_3|$ vertices. Moreover, for each vertex v in $V_1 \cup V_2$ which is not in D, we select the corresponding vertex v'; hence, we select $|V_1 \cup V_2| - |D|$ more vertices. By construction, this is an independent set S in G' of size $|S| = |V_1| + 2|V_2| + |V_3| - |D|$.

Conversely, take an independent set S of G'. Suppose that S contains no vertex from a clique C_u. Then we can add a vertex from C_u to S, and (possibly) remove the vertex v' which was adjacent to it. We get an independent set of at least the same size. By repeating the argument, we can assume that S takes one vertex from each clique C_u. Consider in G the set D of vertices that corresponds to vertices v' (which are not in cliques) in G' that are not in S. Note that S is made of $|V_2| + |V_3|$ vertices in the cliques and $|V_1| + |V_2| - |D|$ vertices outside the cliques. So, we have $|D| = |V_1| + 2|V_2| + |V_3| - |S|$. Consider now a vertex v in $V_2 \cup V_3$. There is a vertex $w \in S$ in the clique C_v, so the vertex v' adjacent to this vertex w is not in S, hence its corresponding vertex is in D. Then, D is a generalized dominating set.

Suppose that we have an r-approximate solution S in G': $|S| \geqslant r\alpha(G')$, we can build a solution D of size $|D| \leqslant |V_1| + 2|V_2| + |V_3| - r\alpha(G') = r\gamma(G) + (1-r)(|V_1| + 2|V_2| + |V_3|)$ where $\gamma(G)$ is the size of a generalized dominating set in G. Since vertices in V_1 and V_2 have degree at most B, we know that $\gamma(G) \geqslant {}^{(|V_2|+|V_3|)}/_{B}$. Note that each vertex in V_1 has at least one neighbor (otherwise, it can be removed from the graph), so that there are at most $|V_1| \leqslant B(|V_2| + |V_3|)$. Then $|V_1| + 2|V_2| + |V_3| \leqslant (B+2)(|V_2| + |V_3|) \leqslant B(B+2)\gamma(G)$. Putting all the above together, we get $|D| \leqslant \gamma(G)(r + (1-r)B(B+2))$. □

Note that similarly, APETH(Set Cover) implies APETH(Max Independent Set-B), when the complexity of Set Cover is measured by $n + m$.

Then, we have the following set of equivalent problems.

Theorem 17. Set Cover, Max Independent Set, Max Independent Set-B, Min Vertex Cover, Min Vertex Cover-B, Min Dominating Set, Min Dominating Set-B, Max Cut-B, Max kSat-B *(for any $k \geqslant 2$) are* APETH*-equivalent.*

Proof. Equivalence between Min Vertex Cover-B, Max Independent Set-B, Max Cut-B, Max-3Sat-B, 2Sat-B, Min Dominating Set-B follow immediately from [109]. Indeed, for these problems [109] provides L-reductions with linear

size amplification. The equivalence between MAX kSAT-B problems is also well known (just replace a clause of size k by $k-1$ clauses of size 3).

The equivalence between MAX INDEPENDENT SET and MAX INDEPENDENT SET-B, MIN VERTEX COVER and MIN VERTEX COVER-B follows from Proposition 11. Finally, Lemma 18 allows us to conclude for MIN DOMINATING SET. □

3.3.2 APETH and Parameterized Approximation

The equivalence drawn in Theorem 17 gives a first intuition that the corresponding problems should be hard to approximate in subexponential time for some ratio. In this section we show another argument towards this hypothesis: if it fails, then *any* MaxSNP problem admits for *any* $r < 1$ a parameterized r-approximation algorithm in subexponential time $2^{o(k)}$, which would be quite surprising. The following theorem can be construed as an extension of [33].

Theorem 18. *The following statements are equivalent:*

(i) APETH(Π) holds for one (equivalently all) problem(s) in Theorem 17;

(ii) there exist a MaxSNP-complete problem Π, some ratio $r < 1$ and a constant $\epsilon > 0$ such that there is no parameterized r-approximation algorithm for Π with running time $O(2^{\epsilon k} poly(|I|))$;

(iii) for any MaxSNP-complete problem Π, there exist a ratio $r < 1$ and an $\epsilon > 0$ such that no parameterized r-approximation algorithm for Π can run in time $O(2^{\epsilon k} poly(|I|))$.

Proof. (i) $\Rightarrow$ (ii): We show it for $\Pi =$MAX INDEPENDENT SET-B, which is MaxSNP-complete. Suppose that for any r and any ϵ there is a parameterized r-approximation algorithm $\mathcal{A}$ which runs in time $O(2^{\epsilon k})$. Given an instance G of MAX INDEPENDENT SET-B, we run $\mathcal{A}$ on the instance (G, k) for $k = 1$ to n. Consider the largest k for which an independent set is given: it has size at least $\rho \cdot k$, while the optimum is at most k since no solution is output for $k+1$. Since $k \leqslant n$, the overall iteration takes $n \cdot 2^{o(n)}$-time.

(ii) $\Rightarrow$ (iii): suppose that (iii) is false, and consider a MaxSNP-complete problem Π_2 which admits for every $\epsilon' > 0$ and every $r' < 1$ a parameterized r-approximation algorithm running in time $2^{\epsilon k} poly(|I|)$. Then, as we will show, this is true for any MaxSNP problem, contradicting (ii).

Indeed, let Π_1 be a MaxSNP problem. There exists an L-reduction from Π_1 to Π_2, let α and β be the constants of the L-reduction. Let (I_1, k) be an instance of Π_1 and let $(I_2, \alpha \cdot k)$ be the instance of Π_2, where $I_2 := f(I_1)$ defined by the L-reduction. Let $r \in (0,1)$ and $\epsilon > 0$, and let $\mathcal{A}$ be a parameterized r'-approximation of Π_2 which runs in time $2^{\epsilon' k} poly(|I|)$ where $r' = 1 - {}^{(1-r)}/_{(\alpha\beta)} < 1$ and $\epsilon' = {}^{\epsilon}/_{\alpha}$. We present an algorithm which uses $\mathcal{A}$ as a subroutine and produces in time $2^{\epsilon k} poly(|I|)$ a solution of Π_1 of size at least rk whenever $opt(I_1) \geqslant k$.

Suppose that $opt(I_1) \geqslant k$. We iteratively run $\mathcal{A}$ over the instances $(I_2, \alpha k), (I_2, \alpha k - 1), \cdots$ by decreasing the parameter. Let $lb \geqslant \alpha k$ be the first integer for which that $\mathcal{A}$ returns a solution, let us call it sol_2, of size at least $r' lb$ upon (I_2, lb). Let $sol_1 := g(sol_2)$, where g is defined by the L-reduction. Note that if $opt(I_2) > \alpha k$ then $sol_2 \geqslant \alpha r' k$; if $opt(I_2) \leqslant \alpha k$, then $lb \geqslant opt(I_2)$ hence $sol_2 \geqslant r' opt(I_2)$.

Now, from the property of L-reduction, we have $opt(I_1) - sol_1 \leqslant \beta(opt(I_2) - sol_2)$, or equivalently $sol_1 \geqslant opt(I_1) - \beta(opt(I_2) - sol_2)$. By considering the two previous cases, and the fact that $opt(I_2) \leqslant \alpha opt(I_1)$ we easily get that whenever $opt(I_1) \geqslant k$, the iterative applications of $\mathcal{A}$ combined with the algorithm g returns a solution sol_1 of size at least $(1 - \alpha\beta(1 - r'))k = rk$. It is easily verified that the overall algorithms performs $O(2^{\epsilon k} \cdot poly(|I_1|))$ steps.

(iii) $\Rightarrow$ (i): Suppose that for any r and any ϵ there is an r-approximation algorithm for MAX INDEPENDENT SET-B with running time $O(2^{\epsilon n})$. Given a graph G and an integer k, if $k \leqslant {}^{n}/_{(B+1)}$ we output an independent set of size ${}^{n}/_{(B+1)}$ (any maximal independent set). Otherwise, we compute an r-approximate solution S in time $O(2^{\epsilon' n}) = O(2^{\epsilon k})$ for $\epsilon' = {}^{\epsilon}/_{(B+1)}$. If $|S| \geqslant rk$ we output it, otherwise $ropt(G) \leqslant |S| < rk$, hence $opt(G) < k$. This contradicts (iii) for MAX INDEPENDENT SET-B. □

As an interesting complement of the above theorem, we show that trade-offs between (exponential) running time and approximation ratio do exist for any MaxSNP problem. In [31], it is shown that every MaxSNP problem Π is fixed-parameter tractable in time $2^{O(k)}$ for the standard parameterization, while in [109] it is shown that Π is approximable in polynomial time within a constant ratio ρ_Π. We prove here that there exists a family of parameterized approximation algorithms achieving ratio $\rho_\Pi + \epsilon$, for any $\epsilon > 0$, and running in time $2^{O(\epsilon k)}$. This is obtained as a consequence of a result in [91].

Proposition 13. *Let Π be a standard parameterization of a MaxSNP-complete problem. For any $\epsilon > 0$, there exists a parameterized $(\rho_\Pi + \epsilon)$-approximation algorithm for Π running in time $\gamma^{\epsilon k} \cdot poly(|I|)$ for some constant γ.*

Proof. Given a parameter k and a set of constraints with at most c variables per constraint, the problem MAX c-CSP ABOVE AVERAGE asks if there is a variable assignment that satisfies at least $\rho \cdot m + k$ constraints. Here ρ is the expected fraction of constraints satisfied by a uniform random assignment. In [91], the following theorem is proved.

Theorem 19. *([91]) For every $c \geqslant 2$,* MAX c-CSP ABOVE AVERAGE *can be solved in time $O(\gamma^k \cdot m)$, where γ is a constant depending only on c.*

Let Π be a problem in the class MaxSNP, defined in the standard way by $\max_S |\{x : \phi(x, G, S)\}|$. As shown in [109], for each of the (polynomially many) possible values x_i of x, consider the corresponding formula $\phi_i(G, S) = \phi(x_i, G, S)$. Since ϕ is fixed, this is a fixed size formula involving (at most) a fixed number t of variables (corresponding to the predicate S). The goal is then to find S satisfying the largest number of formulas ϕ_i. Let ρ_Π be the expected fraction of constraints satisfied by a uniform random assignment. It is easy to find deterministically an assignment satisfying as many formulas as a random one, so Π is ρ_Π-approximable in polynomial time. Note that Π can be interpreted as a MAX c-CSP parameterized by the number of satisfied constraints.

To get the claimed $(\rho_\Pi + \epsilon)$-approximation algorithm for $0 \leqslant \epsilon \leqslant 1 - \rho_\Pi$, we run the algorithm $\mathcal{A}$ given in Theorem 19 on the instance $(\{\phi_i : 1 \leqslant i \leqslant m\}, k')$ (where m is the number of formulas ϕ_i). We take k' so that it satisfies $\rho_\Pi \cdot m + k' = k(\rho_\Pi + \epsilon)$. If k formulas are satisfiable, then, clearly, $k(\rho_\Pi + \epsilon)$ formulas are also satisfiable, so the algorithm will output an assignment satisfying at least this number of constraints (formulas). The running time is $\gamma^{k'} poly(n)$. The claim holds since $k' = \epsilon k - \rho_\Pi(m-k)$ and $k \leqslant m$. □

3.4 Recent Advances

Soon after the results of the previous sections of this chapter were published, Theorem 14 has been powerfully improved by [37], where an implementation of PCP [107] leads to the following theorem.

Theorem 20. *[37] Under ETH, in graphs of order n with maximum degree Δ:*

1. *(General graphs) for any $\delta > 0$ and any r larger than some constant, any r-approximation algorithm for* MAX INDEPENDENT SET *runs in time at least $O^*(2^{n^{1-\delta}/r^{1+\delta}})$;*

2. *(Δ-sparse graphs[3]) for any sufficiently small $\varepsilon > 0$, there exists a constant Δ_ε, such that for any $\Delta \geqslant \Delta_\varepsilon$,* MAX INDEPENDENT SET *on Δ-sparse graphs is not $\Delta^{1-\varepsilon}$-approximable in time $O^*(2^{n^{1-\varepsilon}/\Delta^{1+\varepsilon}})$.*

Our goal now is to capitalize on those results to derive inapproximability in subexponential time results for several other fundamental problems as MIN DOMINATING SET, MIN FEEDBACK VERTEX SET, etc. To do so, we propose two *new* sparsifiers in Section 3.5 and Section 3.6.

The first sparsifier, called *superlinear sparsifier*, generalizes the (linear) sparsifier introduced in Section3.3. The superlinear sparsifier relaxes the requirement that B_ϵ has to be constant and allows the sparsification tree to stop even for non-constant degrees. For simplicity, we present this sparsifier for the case of MAX INDEPENDENT SET and MIN VERTEX COVER, but similar sparsifiers can be developed for several other problems, in particular for the APETH-equivalent problems. As previously, this more general sparsifier allows the transfer of negative results to problems linked to MAX INDEPENDENT SET, or to MIN VERTEX COVER, by approximability preserving reductions building instances of size $O(n+m)$, where m denotes the number of edges of the input graph.

The second sparsifier devised in Section 3.6, is called *k-step sparsifier* and runs in polynomial time. It deals with problems whose solutions satisfy some domination property (as MAX INDEPENDENT SET, MIN DOMINATING SET, MIN INDEPENDENT DOMINATING SET, and MIN VERTEX COVER) and gives quite interesting results when handling maximization problems.

Using either superlinear or k-step sparsifier, together with gap-preserving reductions, we prove in Section 3.7 rather strong negative subexponential inapproximability results for several fundamental problems.

More precisely:

- via *superlinear sparsifier* we show that *under ETH, and for any $\varepsilon > 0$, none of* MIN DOMINATING SET, MIN SET COVER, MIN HITTING SET, MIN FEEDBACK

[3] Graphs where the maximum degree is bounded by Δ.

VERTEX SET, MIN INDEPENDENT DOMINATING SET, *and* MIN FEEDBACK ARC SET *can be* $(7/6-\varepsilon)$*-approximable in time* $O^*(2^{n^{1-2\varepsilon}})$;

- via k*-step sparsifier* we show that *under ETH, for any* $\varepsilon>0$ *and any* $\Delta<\Delta_\varepsilon$, *in* Δ*-sparse graphs and in time* $O^*(2^{O(n^{1-\varepsilon}/\Delta_\varepsilon^{1+\varepsilon})})$, MAX INDEPENDENT SET, MAX ℓ-COLORABLE INDUCED SUBGRAPH *and* MAX INDUCED PLANAR SUBGRAPH *are inapproximable within ratios* $\Delta/2-(\Delta_\varepsilon/2-\Delta_\varepsilon^{1-\varepsilon})$, $\Delta/2-(\Delta_\varepsilon/\ell-\Delta_\varepsilon^{1-\varepsilon})$ *and* $\Delta-(\Delta_\varepsilon-\Delta_\varepsilon^{1-\varepsilon})$, *respectively*;

- finally, using Item 1 of Theorem 20 we show that *under ETH, for any* $\delta>0$ *and any* $r\geqslant n^{1/2-\delta}$, MAX MINIMAL VERTEX COVER *and* MIN INDEPENDENT DOMINATING SET *are inapproximable within ratios* $(c+r)/(1+c)$ *and* $1/(1-c)$ *respectively, in less than* $O^*(2^{n^{1-\delta}/r^{1+\delta}})$ *time, in a graph of order* nr, *with* c *the stability ratio of the* MAX INDEPENDENT SET*-instance of [37].*

Our technique for proving negative results via approximation preserving sparsification (on graph problems) can be outlined as follows. Let Π be some problem inapproximable in time $O^*(2^{n^{1-\epsilon}})$, for any $\epsilon>0$, Π' be some problem such that Π reduces to Π' by some approximation preserving reduction R that works in polynomial time and builds instances of Π' of size $n+m$, and let $\mathcal{F}$ be a superlinear approximation preserving sparsifier for Π. Then, for an instance G of Π we do the following:

– apply $\mathcal{F}$ to G in order to build at most $O^*(2^{n^{1-\epsilon}})$ n^ϵ-sparse instances G_i;

– transform any sparse instance G_i into an instance G_i' of Π';

– if Π is not approximable in time $O^*(2^{n^{1-\epsilon}})$ within ratio r and if R transforms any ratio r' for Π' into ratio $r=c(r')$ for some invertible function c, then Π' is no more approximable in time $O^*(2^{n^{1-\epsilon}})$ within ratio $c^{-1}(r)$.

3.5 Superlinear Sparsifier

Given an optimization graph problem Π and some parameter of the instance (this can be, for instance, the maximum, or the average degree) let Π-B be the problem restricted to instances where the parameter is at most B (we use the same notations as in Section 3.3). Then, a superlinear sparsifier can be defined as follows.

Definition 10. An approximation preserving superlinear sparsification from a graph problem Π to its bounded parameter version Π-B is a pair (f, g) of functions such that, given any function ϕ computable in polynomial time, sublinear in n, and any instance G of Π:

- f maps G into a set $f(G, \phi) = (G_1, G_2, \ldots, G_t)$ of instances of Π, where $t \leqslant 2^{\phi(n)}$ and the orders n_i of the G_is are all bounded by n; moreover, there exists a function ψ (depending on ϕ) such that any G_i has parameter at most $\psi(n)$ (for instance, if the parameter is the degree of the graph, the number of edges of G_i's is linear in n, if ψ is constant, superlinear otherwise);

- for any $i \leqslant t$, g maps a solution S_i of an instance $G_i \in f(G, \phi)$ into a solution S of G;

- there exists an index $i \leqslant t$ such that if a solution S_i is an r-approximation for G_i, then $S = g(G, G_i, S_i)$ is an r-approximation for G;

- f is computable in time $O^*(2^{\phi(n)})$, and g is polynomial in $|n|$.

We observe that, if the parameter considered is, say, the degree of the graph, the graph G_i is $\psi(n)$-sparse but not necessary $\psi(|G_i|)$-sparse.

The sparsifier can be extended to problems defined on set-systems, as MIN SET COVER, MIN HITTING SET, or SET PACKING. Here, parameters can be the cardinality of the largest set, or the frequency. It can also be extended to fit optimum satisfiability problems, where as parameter B can be considered the maximum occurrence of a variable in the input formula. The soundness of this sparsifier relies on the following lemma.

Lemma 19. *A branching algorithm* Π *with branching vector* $(1, \psi(n))$ *has running time* $O^*(2^{(\ln(\psi(n)+1))n/\psi(n)})$.

Proof. Let $T(x)$ denote a bound on the running time of Π when the instance is of size x. $T(n) \leqslant T(n-1) + T(n-\psi(n)) \leqslant T(n-2) + T(n-\psi(n)-1) + T(n-\psi(n)) \leqslant \ldots \leqslant T(n-\psi(n)) + T(n-2\psi(n)+1) + T(n-2\psi(n)+2) + \ldots + T(n-\psi(n)) \leqslant (\psi(n)+1)T(n-\psi(n))$. Thus, the running time of Π is bounded by $O^*((\psi(n)+1)^{n/\psi(n)}) = O^*(2^{(\ln(\psi(n)+1))n/\psi(n)})$. □

Lemma 20. *For any $\varepsilon > 0$, there exists an approximation preserving n^{ε}-sparsification for* MAX INDEPENDENT SET *and* MIN VERTEX COVER *computable in time $O^*(2^{n^{1-\varepsilon}\varepsilon \log n})$.*

Proof. While the maximum degree Δ of the surviving graph exceeds n^{ε}, the standard branching has vector better than $(1, n^{\varepsilon})$ and is approximation preserving.

For MAX INDEPENDENT SET, this branching consists in either including a vertex v of maximum degree to the solution and removing $N[v]$ ($\Delta+1$ vertices are so removed), or not including v in the solution and removing it from the graph (1 vertex removed).

For MIN VERTEX COVER, either include a vertex v of maximum degree in the solution and remove it from the graph (1 vertex removed), or discard v and mandatorily include $N(v)$ to the solution and remove $N[v]$ ($\Delta + 1$ vertices fixed).

By Lemma 19, this branching takes time $O^*(2^{n^{1-\varepsilon}\varepsilon \log n})$. □

One of the main characteristics of the classical notions of reducibility used for proving **NP**-completeness (i.e., Karp- or Turing-reducibility) is the superlinear amplification of the instance sizes. This fact constitutes a major drawback for using these reductions in order to transfer (in)approximability results between problems. Most of the approximation preserving reductions (see [9] for an extensive presentation and discussion of such reductions) manage to limit this amplification in such a way that, in most cases, it remains (almost) linear. In this sense, a reduction which transforms a graph G of order n into an instance of size $O(m)$, has very few chances to be approximation preserving (the bounded-degree requirement of the L-reductions in [109] basically guarantees that m remains linear in n).

As we show in the following Theorem 21, allowing the approximation preserving sparsifier to stop before the degree becomes a constant, enables us to exploit approximation preserving reductions amplifying the instance "more than linearly", and more precisely in $O(n + m)$. Note that, for short, the theorem handles approximability preserving reductions from Π to Π' that transform some ratio r' for Π' into ratio $r = c(r') = r'$ for Π, i.e., c is the identity function.

Theorem 21. *Under ETH:*

1. *if there exists an approximation preserving reduction from* MAX INDEPENDENT SET *to a problem Π building instances of size $O(n + m)$, then, for any $\varepsilon > 0$, and any r larger than some constant satisfying $r \leqslant n^{1/2-\varepsilon}$, Π cannot be $c(r)$-approximable in time $O^*(2^{n^{1-2\varepsilon}/r^{1+\varepsilon}})$;*

2. *if there exists an approximation preserving reduction from* MIN VERTEX COVER *to a problem* Π *building instances of size* $O(n+m)$*, then, for any* $\varepsilon > 0$*,* Π *is not* $c(7/6-\varepsilon)$*-approximable in time* $O^*(2^{n^{1-2\varepsilon}})$.

Proof. We first handle the case of reductions from MAX INDEPENDENT SET. For any $\varepsilon > 0$, take a value $\eta = \varepsilon^+$ arbitrary close to ε such that $\eta > \varepsilon$. Apply Lemma 20 to obtain n^η-sparse instances in time $O^*(2^{n^{1-\eta}\eta\log n})$. Reduce all those instances to Π; instances of size $O(n+nn^\eta) = \alpha n^{1+\varepsilon}$, for some constant α, are so built.

Assume, one can compute an r-approximation for Π in time $O^*(2^{n^{1-2\varepsilon}/r^{1+\varepsilon}})$. Then, one would r-approximate MAX INDEPENDENT SET in time $O^*(2^{n^{1-\eta}\eta\log n}$ $2^{((\alpha n^{1+\eta})^{1-2\varepsilon})/r^{1+\varepsilon}}) = O^*(2^{n^{1-\eta}\eta\log n+(\alpha^{1-2\varepsilon}n^{1+\eta-2\varepsilon-2\varepsilon\eta})/r^{1+\varepsilon}}) = O^*(2^{n^{1-\varepsilon}/r^{1+\varepsilon}})$; a contradiction to the inapproximability of MAX INDEPENDENT SET [37].

We now handle reductions from MIN VERTEX COVER. Beforehand let us do the following important remark. The instance of MAX INDEPENDENT SET built in [37] to ensure the inapproximability gap for MAX INDEPENDENT SET, cannot be used to produce some gap for MIN VERTEX COVER that is greater than $7/6$, the gap of Proposition 10. Indeed, using this instance, the negative result that can be derived for MIN VERTEX COVER is just the impossibility of a subexponential time approximation schema. So, in what follows, we will design gap-preserving reductions from MIN VERTEX COVER.

Suppose that Π is $(7/6-\varepsilon)$-approximable in time $O^*(2^{n^{1-2\varepsilon}})$ for some $\varepsilon > 0$. Again, take a value $\eta = \varepsilon^+$ arbitrary close to ε such that $\eta > \varepsilon$. Apply Lemma 20 to obtain n^η-sparse instances in time $O^*(2^{n^{1-\eta}\eta\log n})$. Reduce all those instances to Π; $2^{n^{1-\eta}\eta\log n}$ instances of size $O(n+nn^\eta) = \alpha n^{1+\eta}$ are so built. By assumption, in time $O^*(2^{n^{1-\eta}\eta\log n}2^{(\alpha n^{1+\eta})^{1-2\varepsilon}}) = O^*(2^{n^{1-\eta}\eta\log n+\alpha^{1-2\varepsilon}n^{1+\eta-2\varepsilon-2\varepsilon\eta}}) = O^*(2^{n^{1-\varepsilon}})$, one can $(7/6-\varepsilon)$-approximate all those subinstances and therefore one can $(7/6-\varepsilon)$-approximate MIN VERTEX COVER, a contradiction with Proposition 10. □

3.6 A k-Step Sparsifier for Maximization Subset Graph Problems

The superlinear sparsifier developed in Section 3.5 works in superpolynomial time. In what follows, we develop, a simple approximability preserving sparsifier, working in polynomial time. Here also, sparsification is done with respect to the maximum degree Δ of the input graph G.

We deal with maximization graph problems where feasible solutions are subsets of the vertex-set verifying some specific property (we consider hereditary property); we call informally these problems "subset problems". Furthermore, we suppose that non-trivial feasible solutions dominate the rest of vertices of the graph. The degree decreasing (sparsification) is done thanks to this domination characteristic of the solution. For reasons of simplicity, we describe the sparsifier for the case of MAX INDEPENDENT SET, but it can be identically applied for any subset problem whose non-trivial solutions dominate the rest of the vertices of the input graph.

Consider a graph G with degree Δ and a constant $k < \Delta$. Then the sparsifier, builds an instance of MAX INDEPENDENT SET-$\Delta - k$ running the following procedure:

> *for* $1 \leqslant i \leqslant k$*, repeatedly excavate maximal (for inclusion) independent sets* X_i*, until the degree of the surviving graph becomes equal to* $\Delta - k$.

Denote by $G'(V', E')$ the instance of MAX INDEPENDENT SET-$\Delta - k$, so-built. Note that, since maximal independent sets dominate the vertices of the graph where they are excavated, their removal reduces the maximum degree. Hence, at the end of the sparsification, G' has degree $\Delta - k$. Furthermore, the sparsifier iterates k times, that is polynomial in n.

Remark that non-trivial solutions of several maximization subset graph-problems verify vertex-domination property. This is the case, for instance of MAX ℓ-COLORABLE INDUCED SUBGRAPH, or of MAX INDUCED PLANAR SUBGRAPH. Indeed if there exists a vertex non dominated by a vertex-set V' inducing an ℓ-colorable subgraph, it suffices to add it in one of the color-classes. The graph $G[V' \cup \{x\}]$ always remains ℓ-colorable. The same holds for MAX INDUCED PLANAR SUBGRAPH.

Theorem 22. *Let* $\mathcal{P}(\Pi, r', \Delta - k)$ *be the following property: "*if problem Π is approximable within ratio r' in time $f(n)$ on $(\Delta - k)$-sparse graphs then, on Δ-sparse graphs, it is $(r' + 1)$-approximable in time $O(f(n) + n^2)$*". Then:*

1. $\mathcal{P}$(MAX INDEPENDENT SET, $r', \Delta - 2$)*;*
2. $\mathcal{P}$(MAX ℓ-COLORABLE INDUCED SUBGRAPH, $r', \Delta - \ell$)*;*
3. $\mathcal{P}$(MAX INDUCED PLANAR SUBGRAPH, $r', \Delta - 1$).

Proof. Let $G(V, E)$ be a graph on n vertices with maximum degree Δ. Let S^* be a maximum independent set of G. Run the k-step sparsifier for two steps and stop it (this

obviously takes polynomial time). It computes two maximal independent sets S_1 in G, and S_2 in $G[V \setminus S_1]$; $G' = G[V \setminus (S_1 \cup S_2)]$ has degree degree at most $\Delta - 2$. Set $B = G[S_1 \cup S_2]$, the bipartite subgraph of G induced by the union of S_1 and S_2.

Since B is bipartite, a maximum independent set S_B^* in B can be computed in polynomial time. If $|S_B^*| \geqslant \alpha(G)/r$, then S_B^* is an r-approximation MAX INDEPENDENT SET in G.

Assume now that $|S_B^*| < \alpha(G)/r$ and consider the graph $G' = G[V \setminus (S_1 \cup S_2)]$. Let $S^{*'}$ be the part of S^* contained in G'. Since $|S_B^*| < \alpha(G)/r$, and since S_B^* has size at least equal to the size of the part of S^* that belongs to B, $|S^{*'}| > (1 - 1/r)\alpha(G)$.

The graph G' has degree at most $\Delta - 2$, since if a vertex v has degree Δ, or $\Delta - 1$ in $G[V \setminus (S_1 \cup S_2)]$, then it has no neighbors in either S_1, or S_2 and this contradicts the maximality of at least one of them.

Run in G' the r'-approximation algorithm (with complexity $f(n)$) assumed for $(\Delta - 2)$-sparse graphs and denote by S' the solution returned. Since S' is an r'-approximation, $|S'| \geqslant |S^*|/r'$, so, $|S'| > ((1 - 1/r)1/r')\alpha(G)$. The independent set S' is obviously a solution also for G and guarantees ratio $r \cdot r'/r-1$.

Finally, take the best among independent sets S_B^* and S' as solution for G.

Equality of ratios r and $r \cdot r'/r-1$ derives $r = r' + 1$. Since ratio r' is achieved in time $f(n)$ and the application of the sparsification step takes time $O(n^2)$, ratio r is achieved for MAX INDEPENDENT SET in G in time $O(f(n) + n^2)$ as claimed.

For MAX ℓ-COLORABLE INDUCED SUBGRAPH, let $G(V, E)$ be a graph on n vertices with maximum degree Δ. Let L^* be an optimal solution for MAX ℓ-COLORABLE INDUCED SUBGRAPH on G. Run the the k-step sparsifier for MAX INDEPENDENT SET for ℓ steps. It iteratively excavates ℓ maximal independent sets $S_1, S_2, \ldots S_\ell$. Set $V' = S_1 \cup S_2 \cup \ldots \cup S_\ell$, and $G' = G[V']$, the ℓ-colorable subgraph of G induced by V'. Denote by $L^{*'}$ the part of L^* belonging to L^*.

If $|L^{*'}| \geqslant L^*/r$ then, since $|V'| \geqslant |L^{*'}| \geqslant L^*/r$, V' is an r-approximation MAX INDEPENDENT SET in G.

Assume now $|L^{*'}| < L^*/r$ and consider the graph $G'' = G[V \setminus V']$. Let $L^{*''}$ be the part of L^* contained in G''. Since $|L^{*'}| < L^*/r$, $|L^{*''}| > (1 - 1/r)|L^*|$.

The graph G'' has degree at most $\Delta - \ell$ and the rest of the proof remains similar to the corresponding part of that of the first item.

For MAX INDUCED PLANAR SUBGRAPH, one just excavates only one independent set. An independent set is a planar graph. The rest of the proof of the third item is the

same as above. □

3.7 Some More Subexponential Inapproximability Results

3.7.1 Via Superlinear Sparsification

Combining the superlinear sparsifier of Definition 10 in Section 3.5 together with approximation preserving reductions from MIN VERTEX COVER to several problems, the following theorem can be proved.

Theorem 23. *Under ETH, and for any $\varepsilon > 0$, none of* MIN DOMINATING SET, MIN SET COVER *and* MIN HITTING SET, MIN FEEDBACK VERTEX SET, MIN INDEPENDENT DOMINATING SET, *and* MIN FEEDBACK ARC SET *is $(7/6 - \varepsilon)$-approximable in time $O^*(2^{n^{1-2\varepsilon}})$.*

Proof. For MIN DOMINATING SET, let $G(V, E)$ be an instance of MIN VERTEX COVER and assume G is connected. Build a graph $G'(V', E')$ as follows. Start from a copy of G and for each edge $e = (u, v) \in E$, add two dummy vertices y_e and z_e in V' and link those vertices to u and v. The graph G' so built has order $n + 2m$.

A minimum dominating set in G' does not contain any dummy vertex. Indeed, if a solution S contains $y_{(u,v)}$ or $z_{(u,v)}$, then $S \setminus \{y_{(u,v)}, z_{(u,v)}\} \cup \{u\}$ is still a dominating set of at most equal cardinality. Thus, a minimum dominating set in G' naturally maps to a subset of V which covers all the edges, hence a vertex cover of the same size. Furthermore, given an r-approximation of MIN DOMINATING SET in G', one can start by removing the potential dummy vertices as explained above, and then obtain an r-approximation for MIN VERTEX COVER. Item 2 of Theorem 21 suffices for completing the proof.

The result for MIN SET COVER immediately follows from a well-known approximation preserving reduction from MIN DOMINATING SET (function c being the identity function). Given an instance $G(V, E)$ of MIN DOMINATING SET, one can construct an instance $(\mathcal{S}, C)$ of MIN SET COVER, where $\mathcal{S}$ is a set-system over the ground set C, by taking $\mathcal{S} = V$, $C = V$ and, for each vertex $v_i \in V$, the corresponding set $S_i \in \mathcal{S}$ contains as elements $c_j \in C$ such that vertex v_j is either v_i or $v_j \in N(v_i)$.

For MIN HITTING SET, just observe is the problem is similar to MIN SET COVER where roles of $\mathcal{S}$ and C are interchanged.

Notice that the previous reduction still works for MIN FEEDBACK VERTEX SET. In G', every subset of vertices containing non-dummy vertex is a dominating set, iff it is a feedback vertex set[4].

For MIN INDEPENDENT DOMINATING SET, tune the previous reduction by deleting all the edges in the copy of the graph G. In other words, build G' from an independent set V of size $n = |V|$ where each vertex corresponds to a vertex in V, and link all the vertices $u \in V$ to an independent set I_e with 2 dummy vertices for each edge $e = (u, v)$. Again, an optimal solution contains only copy vertices (no dummy vertices). Furthermore, in G', every subset containing non-dummy vertex is an independent dominating set iff it is a vertex cover in G.

For MIN FEEDBACK ARC SET, the reduction in [87] is approximation preserving with c the identity function. The graph $G'(V', E')$ for MIN FEEDBACK ARC SET is built with:

$$\begin{aligned} V' &= V \times \{0, 1\} \\ E' &= \{((u, 0), (u, 1)) : u \in V\} \cup \{((u, 1), (v, 0)) : (u, v) \in E\} \end{aligned}$$

In any solution, an arc $((u, 1), (v, 0))$ can be advantageously replaced by arc $((v, 0), (v, 1))$. Indeed, a cycle containing edge $((u, 1), (v, 0))$, necessarily contains also edge $((v, 0), (v, 1))$ since the vertex $(v, 0)$ has out-degree 1. Thus, removing $((v, 0), (v, 1))$ destroys the same cycles (plus potentially others). We can therefore assume that a solution is $\{((v, 0), (v, 1)) : v \in S\}$, for some $S \subseteq V$. Now, S is a vertex cover, and an r-approximation for MIN FEEDBACK ARC SET transforms into an r-approximation for MIN VERTEX COVER. □

Let us note that using the classical reduction from MIN VERTEX COVER to MIN SAT [101] a similar result can be derived for MIN SAT.

3.7.2 Via k-Step Sparsification

Revisit Item 2 of Theorem 20. There, Δ_ε is related to ε in the following way: there exists a universal constant C such that $\Delta_\varepsilon = 2^{C/\varepsilon}$. Our purpose in this section is to strengthen this item deriving inapproximability for MAX INDEPENDENT SET, MAX ℓ-COLORABLE INDUCED SUBGRAPH and MAX INDUCED PLANAR SUBGRAPH, in subexponential time $O^*(2^{O(n^{1-\varepsilon}/\Delta_\varepsilon^{1+\varepsilon})})$ with a *smaller* bounded degree.

[4]These reductions rely on the fact that, in graphs without isolated vertices, a vertex cover is both a dominating set and a feedback vertex set.

Theorem 24. *Under ETH, for any $\varepsilon > 0$ and any $\Delta < \Delta_\varepsilon$, in Δ-sparse graphs,* MAX INDEPENDENT SET, MAX ℓ-COLORABLE INDUCED SUBGRAPH *and* MAX INDUCED PLANAR SUBGRAPH *are inapproximable within ratios $\Delta/2 - (\Delta_\varepsilon/2 - \Delta_\varepsilon^{1-\varepsilon})$, $\Delta/2 - (\Delta_\varepsilon/\ell - \Delta_\varepsilon^{1-\varepsilon})$ and $\Delta - (\Delta_\varepsilon - \Delta_\varepsilon^{1-\varepsilon})$, respectively, in time $O^*(2^{O(n^{1-\varepsilon}/\Delta_\varepsilon^{1+\varepsilon})})$.*

Proof. By Item 2 of Theorem 20, for any $\varepsilon > 0$, MAX INDEPENDENT SET on Δ_ε-sparse graphs, is inapproximable within ratio $\Delta_\varepsilon^{1-\varepsilon}$ in time $O^*(2^{n^{1-\varepsilon}/\Delta_\varepsilon^{1+\varepsilon}})$, with $\Delta_\varepsilon = 2^{C/\varepsilon}$ for some constant C.

For any Δ, run the k-step sparsifier on a Δ_ε-sparse graph G for $(\Delta_\varepsilon - \Delta)/2$ steps, from Δ_ε down to Δ, in order to get a Δ-sparse instance G' of MAX INDEPENDENT SET. Combination of the Item refmain1 of Theorem 22 and of Item 2 of Theorem 20 directly derives inapproximability of MAX INDEPENDENT SET in G within ratio $\Delta_\varepsilon^{1-\varepsilon} - (\Delta_\varepsilon - \Delta)/2 = \Delta/2 - (\Delta_\varepsilon/2 - \Delta_\varepsilon^{1-\varepsilon})$ in time $O^*(2^{O(n^{1-\varepsilon}/\Delta_\varepsilon^{1+\varepsilon})})$.

Consider now the following simple reduction from MAX INDEPENDENT SET to MAX ℓ-COLORABLE INDUCED SUBGRAPH. Let $G(V, E)$ be an instance of MAX INDEPENDENT SET of order n. We keep G as the instance of MAX ℓ-COLORABLE INDUCED SUBGRAPH. Any independent set S of G can be considered as an ℓ-colorable graph with empty the $\ell - 1$ of its color classes. Conversely, given an ℓ-colorable graph on sets $S_1, S_2, \ldots, S_\ell$, all them are independent sets and the largest among them has size more than $1/\ell$ times the size of the ℓ-colorable graph. So, any ratio r for MAX ℓ-COLORABLE INDUCED SUBGRAPH becomes ratio ℓr for MAX INDEPENDENT SET.

In the same spirit, one can devise a reduction from MAX INDEPENDENT SET to MAX INDUCED PLANAR SUBGRAPH. An independent set is a planar graph per se. On the other hand since any planar graph is 4-colorable, a solution $G' = G[S]$ of MAX INDUCED PLANAR SUBGRAPH can be transformed into an independent set by coloring the vertices of S with four colors and taking the largest of them. So an approximation ratio r for MAX INDUCED PLANAR SUBGRAPH is transformed into ratio $4r$ for MAX INDEPENDENT SET.

The proofs for MAX ℓ-COLORABLE INDUCED SUBGRAPH and MAX INDUCED PLANAR SUBGRAPH above of the theorem immediately derive from the remarks above. □

Note that the inapproximability bound for MAX INDEPENDENT SET of Theorem 24 (Item 1) cannot be derived by Theorem 20 for $\Delta \geqslant 2^{C/\varepsilon}(1/2 - 2^{-C})$. So, Theorem 24 extends the result of [37] to degree $\Delta_\varepsilon/2$.

Also, from the discussion of for MAX ℓ-COLORABLE INDUCED SUBGRAPH and MAX INDUCED PLANAR SUBGRAPH in the proof of Theorem 24, the following corollary holds.

Corollary 12. *Under ETH, and for any $\varepsilon > 0$, neither* MAX ℓ-COLORABLE INDUCED SUBGRAPH *nor* MAX INDUCED PLANAR SUBGRAPH *is r-approximable in time $O^*(2^{n^{1-\delta/r^{1+\delta}}})$, where r is the approximability-gap of* MAX INDEPENDENT SET.

3.7.3 Via Theorem 20

Similar results as those of Corollary 12 can be obtained for several other problems linked to MAX INDEPENDENT SET by approximability-preserving reductions.

For instance, for SET PACKING, take $\mathcal{S} = V$, $C = E$ and, for any set $S_i \in \mathcal{S}$, $S_i = \{c_j : e_j \text{ incident to } v_i\}$. This very classical reduction transforms any independent set of G to an equal-cardinality set-packing of $(\mathcal{S}, C)$, and vice-versa.

For MAX UNUSED SETS, observe that its optimal value is an affine transformation of the optimum for MIN SET COVER. Since this latter problem is a generalization of MIN VERTEX COVER (indeed MIN VERTEX COVER can be seen as a MIN SET COVER problem where all ground elements have frequency 2), MAX UNUSED SETS is a generalization of MAX INDEPENDENT SET.

In what follows in this section, we handle inapproximability bounds for two problems that are closely linked between them, MIN INDEPENDENT DOMINATING SET and MAX MINIMAL VERTEX COVER. In fact, they are related in the same way as MAX INDEPENDENT SET and MIN VERTEX COVER.

Let us first consider MAX MINIMAL VERTEX COVER and revisit the following reduction from MAX INDEPENDENT SET given in [21]. Given an instance $G(V, E)$ of MAX INDEPENDENT SET, link any $v_i \in V$ to $n + 1$ new vertices. The so-built graph H for MAX MINIMAL VERTEX COVER has size $n^2 + 2n$. Then, by considering a MAX MINIMAL VERTEX COVER-solution for H consisting of taking the out-of-G neighbors of some some independent set S of G together with $V \setminus S$ as solution for MAX MINIMAL VERTEX COVER, one can guarantee the following:

$$\begin{array}{rcl} \mathrm{sol}(H) & \leqslant & n \cdot |S| + n \\ \mathrm{opt}(H) & \geqslant & n \cdot \alpha(G) + n \end{array} \tag{3.1}$$

where $\mathrm{sol}(H)$ and $\mathrm{opt}(H)$ denote the sizes of an approximate and of an optimal solutions for MAX MINIMAL VERTEX COVER, respectively. Then, using expressions

in (3.1) and considering G the MAX INDEPENDENT SET-instance of [37], one easily derives the following.

Proposition 14. *Under ETH, for any $\delta > 0$ and any $r \geqslant n^{1/4-\delta}$,* MAX MINIMAL VERTEX COVER *is inapproximable within ratio r in less than $O^*(2^{n^{1/2-\delta}/r^{1+\delta}})$ time.*

Observe that, in the reduction above, $\Delta(H) \geqslant n \approx \sqrt{n(H)}$. So, the following corollary derives from Proposition 14.

Corollary 13. *Furthermore, under ETH, for any $\delta > 0$ and any $r \geqslant \Delta^{1/2-\delta}$,* MAX MINIMAL VERTEX COVER *is inapproximable within ratio r in less than $O^*(2^{\Delta^{1-\delta}/r^{1+\delta}})$ time.*

The result of Proposition 14 can be further strengthened by slightly changing the reduction of [21]. Denote by c the stability ratio $\alpha(G)/n$ of G. Then the following holds.

Proposition 15. *Under ETH, for any $\delta > 0$ and any $r \geqslant n^{1/2-\delta}$, in any graph of order nr,* MAX MINIMAL VERTEX COVER *is inapproximable within ratio $(c+r)/(1+c)$ in time less than $O^*(2^{n^{1-\delta}/r^{1+\delta}})$, where c the stability ratio of the* MAX INDEPENDENT SET*-instance of [37].*

Proof. Consider the MAX INDEPENDENT SET-instance of Theorem 20 and link any of its vertices to $r+1$ new vertices where r is as in Item 1 of Theorem 20. The MAX MINIMAL VERTEX COVER-instance H has now $n(r+1)$ vertices. Set $\rho'(H) = \mathrm{sol}(H)/\mathrm{opt}(H)$, the inverse of the approximation ratio for MAX MINIMAL VERTEX COVER in H. Then, using (3.1), it holds that:

$$\frac{|S|}{\alpha(G)} \geqslant \rho'(H) - \frac{(1-\rho'(H))n}{r\alpha(G)} \tag{3.2}$$

As one can see in the proof of Item 1 of Theorem 20, $\alpha(G)$ is linear in n, i.e., $\alpha(G) \geqslant cn$ for some fixed (independent on n) $c < 1$. So, (3.2) becomes:

$$\frac{1}{r} \geqslant \frac{|S|}{\alpha(G)} \geqslant \rho'(H) - \frac{(1-\rho'(H))c}{r} \geqslant \rho'(H) - \frac{c}{r} \tag{3.3}$$

where the first inequality above is due to the inapproximability bound $1/r$ for MAX INDEPENDENT SET in the graph of Item 1 of Theorem 20. Then some simple algebra derives $\rho(H) = 1/\rho'(H) \geqslant (c+r)/(1+c)$, as claimed. □

Interestingly enough, although MIN INDEPENDENT DOMINATING SET is one of the hardest problems for polynomial approximation, only subexponential inapproximability within ratio $7/6 - \varepsilon$ can be proved for it, using sparsification. The following proposition gives a stronger subexponential inapproximability bound for MIN INDEPENDENT DOMINATING SET using the fact that an independent dominating set in some graph G is the complement of a minimal vertex cover of G.

Proposition 16. *Under ETH, for any $\delta > 0$ and any $r \geqslant n^{1/2-\delta}$, in any graph of order nr,* MIN INDEPENDENT DOMINATING SET *is inapproximable within ratio $1/(1-c)$ in time less than $O^*(2^{n^{1-\delta/r^{1+\delta}}})$, where c is the stability ratio $\alpha(G)/n$ of the* MAX INDEPENDENT SET*-instance of [37].*

Proof. Consider again the graph G built in Item 1 of Theorem 20 and the reduction of Proposition 15 to MAX MINIMAL VERTEX COVER. Denote by c the stability ratio of G, i.e., $c = \alpha(G)/n$, and recall that c is a fixed constant [37]. Then:

$$\iota(H) = \alpha(G) + (n - \alpha(G))(r+1) = (1-c)n(r+1) \tag{3.4}$$

Denote by $\iota'(H)$, the independent dominating set associated with the approximate minimal vertex cover of H, i.e., $\iota'(H) = n(r+1) - \mathrm{sol}(H)$ and by b the inverse of the inapproximability bound for MAX MINIMAL VERTEX COVER ($b < 1$). Then, using (3.4), we get:

$$\begin{aligned} b \geqslant \frac{\mathrm{sol}(H)}{\mathrm{opt}(H)} \geqslant \frac{n(r+1) - \iota'(H)}{n(r+1) - \iota(H)} \quad &\Longrightarrow \quad \frac{\iota'(H)}{\iota(H)} \geqslant \frac{n(r+1)(1-b)}{\iota(H)} + b \\ &\geqslant \quad \frac{1-b}{1-c} + b \sim \frac{1}{1-c} \end{aligned}$$

where the last approximation for b is due to the fact that $b = o(1)$. □

3.8 More About Sparsifiers

If one revisits the informal description of sparsification in Section 3.5, the sparsifier designed in [81] may yield very weak lower bounds, in the sense that $f(\lambda)$ may be very close to 1.

Suppose that there exists a polynomial time reduction R from k-SAT to a problem Π, and two integers α and β such that, for an instance ϕ of k-SAT with n variables

and m clauses, $\mathsf{R}(\phi)$ is of size $\alpha n + \beta m$. To solve an instance of k-SAT on ϕ, one can sparsify it, reduce all the $2^{\varepsilon n}$ sparsified formulas, and solve each instance of Π built by application of R to any sparse instance produced from ϕ. This takes time $O^*((2^{\varepsilon}\lambda^{\alpha+\beta C_k^{\varepsilon}})^n)$. Assuming ETH, let λ_k be the smallest real number such that k-SAT is solvable in $O^*(\lambda_k^n)$. Then, $2^{\varepsilon}\lambda^{\alpha+\beta C_k^{\varepsilon}} \geqslant \lambda_k$. Adjusting ε to get the best possible lower bound for λ, one gets $\lambda - 1 < 10^{-10}$, for plausible values of α and β. So, one only shows that Π is not solvable in, say, $O^*((1 + 10^{-10})^n)$.

We show that the superlinear sparsifier of Section 3.5 may be used to produce stronger lower bounds than those get by the sparsifier of [81].

In order to do that, we will use the central problem MAX INDEPENDENT SET.

Assume $\mathcal{H}_{\mathrm{IS}}(\lambda)$ is the hypothesis that MAX INDEPENDENT SET is not solvable in time $O^*(\lambda^n)$, and $g : (1, 2) \to \mathbb{N}$ maps any real value x in $(1, 2)$ to the smallest integer p such that the positive root $X^{p+1} - X^p - 1 = 0$ is smaller than x. The superlinear sparsifier can be used to show the following.

Proposition 17. *Let Π be problem such that there exists a polynomial time reduction R from* MAX INDEPENDENT SET *to Π and two positive numbers α and β satisfying, for all instances G of* MAX INDEPENDENT SET, *$|\mathsf{R}(G(V, E))| \leqslant \alpha|V| + \beta|E| = \alpha n + \beta m$. Under $\mathcal{H}_{\mathrm{IS}}(\lambda)$, if Π is solvable in $O^*(\mu^n)$, then $\mu > \lambda^{1/\alpha+\lfloor g(\lambda)/2 \rfloor \beta}$*

Proof. Use the superlinear sparsifier with the threshold $\Delta = g(\lambda)$, that is, stop the branching when the degree of the graph becomes strictly less than $g(\lambda)$. The branching factor is the positive root of $X^{g(\lambda)+1} - X^{g(\lambda)} - 1 = 0$ which, by construction, is smaller than λ. At a leaf of the branching tree, if the number of vertices is $n - k$, then the number of edges in the remaining graph is at most $\lfloor g(\lambda)/2 \rfloor (n - k)$.

Thus, by performing the reduction R on the instances at each leaf of the branching tree, and then solving the obtained instances of Π, one gets an algorithm solving MAX INDEPENDENT SET in time $O^*(\lambda^k \mu^{(\alpha+\lfloor g(\lambda)/2 \rfloor \beta)(n-k)})$. So, $\mu > \lambda^{1/\alpha+\lfloor g(\lambda)/2 \rfloor \beta}$, otherwise $\lambda^k \mu^{(\alpha+\lfloor g(\lambda)/2 \rfloor \beta)(n-k)} \leqslant \lambda^n$. □

Since the superlinear sparsifier is approximation preserving, if reduction R from MAX INDEPENDENT SET to Π preserves approximation, one can obtain relative exponential time lower bounds even for approximation issues. The following proposition provides a lower bound to the best currently known complexity (function of the number of clauses) of MAX 3SAT, under $\mathcal{H}_{\mathrm{IS}}$. Note that the best known running time for MAX 3SAT is $O^*(1.324^m)$.

Proposition 18. *Under $\mathcal{H}_{\mathrm{IS}}(\lambda)$,* MAX 3SAT *is not solvable in $O^*(\lambda^{(1/1+\lfloor g(\lambda)/2 \rfloor)n})$.*

Proof. We recall the reduction in [109]. An instance $(G(V,E),k)$ of the decision version of MAX INDEPENDENT SET is transformed into an instance of the decision version of MAX 3SAT in the following way: each vertex $v_i \in V$ encodes a variable X_i and for each edge $(v_i,v_j) \in E$ we add a clause $\neg X_i \vee \neg X_j$. Finally, we add the 1-clause X_i for all $v_i \in V$. In the so built instance of MAX 3SAT we wish to satisfy at least $k+m$ clauses. This reduction builds $n+m$ clauses, so $\alpha = \beta = 1$. Hence, under $\mathcal{H}_{\mathrm{IS}}$, and according to Proposition 17, one cannot solve MAX 3SAT in time $O^*(\mu^n)$ when $\mu = \lambda^{1/1+\lfloor g(\lambda)/2 \rfloor}$. □

Suppose that Π is a problem (like MAX 3SAT when considering its complexity in terms of m) with a reduction from MAX INDEPENDENT SET in $n+m$ ($\alpha = \beta = 1$), and Π is solvable in $O^*(\mu^n)$. Then, the following table gives some values of μ as function of λ.

λ	Infeasible value for μ
1.1	1.0073
1.18	1.027
1.21	1.038

We conclude by pointing out that the k-step sparsifier of Section 3.6 has also some interesting consequences when handling parameterized issues. MAX INDEPENDENT SET can be solved in time $O^*((\Delta+1)^\alpha)$ with a standard branching algorithm [108] (here $\alpha = \alpha(G)$ is the size of a maximum independent set, or equivalently the natural parameter for MAX INDEPENDENT SET). The excavation performed by the k-step sparsifier can be used to obtain an algorithm running in time $O^*(2^{(\Delta-2)\alpha})$. Indeed, one can excavate consecutively $\Delta-2$ maximal independent sets S_1 to $S_{\Delta-2}$, where each S_i is a maximal independent set in $G[V \setminus \bigcup_{k=1\ldots i-1} S_k]$. By hypothesis, for all i, $|S_i| \leqslant \alpha$, so an exhaustive search on $\bigcup_{k=1\ldots\Delta-2} S_i$ takes time $O^*(2^{(\Delta-2)\alpha})$. Graph $G[V \setminus \bigcup_{k=1\ldots\Delta-2} S_k]$ is a graph with degree 2, hence it takes polynomial time to complete a solution by finding a maximum independent set on this part of the graph. This algorithm improves the branching algorithm for $\Delta \leqslant 4$. as the following table shows.

Δ	Exhaustive branching	Sparsification
3	4^α	2^α
4	5^α	4^α

3.9 Conclusion

More interesting questions remain untouched in the junction of approximation and (sub)exponential-time/FPT-time computations. Among a range of problems to be tackled, we propose the following.

- Our inapproximability results are conditional upon the Linear PCP Conjecture. Is it possible to relax the condition to a more plausible one?
- Or, we dare ask whether (certain) inapproximability results in FPT-time imply strong improvement in the PCP theorem. For example, would the converse of Lemma 12 hold?
- Can we design approximation preserving sparsifications for problems like MAX CUT or MAX 3SAT? It seems to be difficult to design a sparsifier based on branching rules, so a novel idea is needed.

Note that we have considered constant approximation ratios. As noted earlier, ratio $1/r(n)$ is achievable in subexponential time for any increasing and unbounded function r for MAX INDEPENDENT SET. However, dealing with parameterized approximation algorithms, achieving a non-constant ratio is also an open question. More precisely, finding in FPT-time an independent set of size $g(k)$ when there exists an independent set of size k is not known for *any* unbounded and increasing function g.

Finally, let us note that, in the same vein of the first part of our work, [103] studied a proof checking view of parameterized complexity. Possible links between these two approaches are worth being investigated in future works.

4 Complexity of Games

4.1 Complexity of Trick-Taking Card Games

Determining the complexity of perfect information trick taking card games is a long standing open problem. This question is worth addressing not only because of the popularity of these games among human players, e.g., DOUBLE DUMMY BRIDGE, but also because of its practical importance as a building block in state-of-the-art playing engines for CONTRACT BRIDGE, SKAT, HEARTS, and SPADES.
We define a general class of perfect information two-player trick taking card games dealing with arbitrary numbers of hands, suits, and suit lengths. We investigate the complexity of determining the winner in various fragments of this game class.
Our main result is a proof of **PSPACE**-completeness for a fragment with bounded number of hands, through a reduction from Generalized Geography.

Determining the complexity class of games is a popular research topic [76], even more so when the problem has been open for some time and the game is actually of interest to players and researchers. For instance, the game of AMAZONS was proved **PSPACE**-complete by three different research groups almost simultaneously [69, 76]. In the following sections, we investigate the complexity of trick taking card games. The class of *trick taking card games* encompasses a large number of popular card games

such as CONTRACT BRIDGE, HEARTS, SKAT, SPADES, TAROT, and WHIST.[1]

The rules of the quintessential trick taking card game are fairly simple. A set of players is partitioned into teams and arranged around a table. Each player is dealt a given number of cards n called *hand*, each card being identified by a *suit* and a *rank*. The game consists in n tricks in which every player plays a card. The first player to play in a given trick is called lead, and the other players proceed in the order defined by the seating. The single constraint is that players should follow the lead suit if possible. At the end of a trick, whoever put the highest ranked card in the lead suit wins the trick and leads the next trick. When there are no cards remaining, after n tricks, we count the number of tricks each team won to determine the winner.[2]

Assuming that all hands are visible to everybody, is there a strategy for the team of the first player to ensure winning at least k tricks?

Despite the demonstrated interest of the general population in trick taking card games and the significant body of artificial intelligence research on various trick taking card games [28, 72, 66, 93, 99], most of the corresponding complexity problems remain open. This stands in stark contrast with other popular games such as CHESS or GO, the complexity of which was established early [65, 95, 114].

There are indeed very few published hardness results for card games. We only know of a recent paper addressing UNO [48], a card game not belonging to the category of trick taking card games, and Frank and Basin [67]'s result on the best defense model. They show that given an imperfect information game tree and an integer w, and assuming the opponent has perfect information, determining whether one has a pure strategy winning in at least w worlds is **NP**-complete.

As for tractability, after a few heuristics were proposed [86], Wästlund's performed an in-depth combinatorial study on fragments of perfect information two-hands WHIST proving that some important fragments of trick taking card games are polynomial [126, 127].

Note that contrary to the hypotheses needed for Frank and Basin [67]'s **NP**-completeness result, we assume perfect information and a compact input, namely the hands and an integer k. There are several reasons for focusing on perfect information. First, it provides a lower bound to the imperfect information case when compact

[1] A detailed description of these games and many other can be found on `http://www.pagat.com/class/trick.html`.

[2] There are more elaborate point-based variants where tricks might have different values, possibly negative, based on cards comprising them. We focus on the special case where each card has the same positive value.

input is assumed. More importantly, perfect information trick taking card games actually do appear in practice, both among the general population in the form of DOUBLE DUMMY BRIDGE problems, but also in research as perfect information Monte Carlo sampling is used as a base component of virtually every state-of-the-art trick taking game engine [94, 72, 121, 96].

It is rather natural to define fragments of this class of decision problems, for instance, by limiting the number of different suits, the number of hands, or even limiting the number of cards within each suit. We will soon define the lattice of such fragments.

In Section 4.2, we show that the general problem is **PSPACE**-complete and it remains so even if the number of cards per suit is 2. The proof is a rather straightforward reduction from Generalized Geography (GG). Our main result is a more involved reduction from GG to address the fragment with bounded number of hands, it is presented in Section 4.3.

4.1.1 Trick-Taking Game

Definition 11. A *card* c is a pair of two integers representing a *suit* (or color) $s(c)$ and a *rank* $r(c)$. A *position* p is defined by a tuple of hands $h = (h_1, \ldots, h_n)$, where a *hand* is a set of cards, and a *lead turn* $\tau \in [1, n]$. We further assume that all hands in a given position have the same size $\forall i, j \in [1, n], |h_i| = |h_j|$ and do not overlap: $i \neq j \Rightarrow h_i \cap h_j = \emptyset$.

An example position with 4 hands and 12 total cards is given in Figure 4.1. The position is written as a diagram, so for instance, hand h_3 contains 3 cards $\{(s_1,\text{A}), (s_1,\text{J}), (s_2,\text{K})\}$.

Definition 12. Playing a *trick* consists in selecting one card from each hand starting from the lead: $c_\tau \in h_\tau, c_{\tau+1} \in h_{\tau+1}, \ldots, c_n \in h_n, c_1 \in h_1, \ldots, c_{\tau-1} \in h_{\tau-1}$. We also require that *suits are followed*, i.e., each played card has the same suit as the first card played by hand τ or the corresponding hand h_i does not have any card in this suit: $s(c_i) = s(c_\tau) \vee \forall c \in h_i, s(c) \neq s(c_\tau)$.

Definition 13. The *winner of a trick* is the index corresponding to the card with highest rank among those having the required suit. The position resulting from a trick with cards $C = \{c_\tau, \ldots, c_{\tau-1}\}$ played in a position p can be obtained by removing the selected cards from the hands and setting the new lead to the winner of the trick.

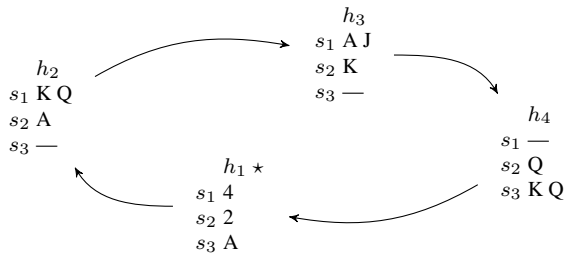


Figure 4.1: Example of a trick-taking game position with 4 hands, 3 suits, and 1 as lead turn. If team A controls h_1 and h_3 and team B controls h_2 and h_4, then team A can make all three remaining tricks by starting with (s_3, A)

In the example in Figure 4.1, the lead is to 1. A possible trick would be $(s_3$,A$), (s_1$,Q$), (s_1$,J$), (s_3$,Q$)$; note that only hand h_4 can follow suit, and that 1 is the winner so remains lead.

Definition 14. A *team mapping* σ is a map from $[1 \dots n]$ to $\{A, B\}$ where n is the number of hands, A is the existential player, and B is the universal player. A (perfect information, plain) *trick-taking game* is pair consisting of a position and a team mapping σ.

For simplicity of notation, team mappings will be written as words over the alphabet $\{A, B\}$. For instance, $1 \mapsto A, 2 \mapsto B, 3 \mapsto A, 4 \mapsto B$ is written $ABAB$.

Definition 15. A trick is won by team A if its winner is mapped to A with σ. The *value* of a game is the number of tricks that team A wins, if both team A and team B play optimally with respect to making the greatest number of tricks.

The value of the game presented in Figure 4.1 is 3 as team A can ensure making all remaining tricks with the following strategy known as *squeeze*. Start with $(s_3$,A$)$ from h_1 and play $(s_1$,J$)$ from h_3, then start the second trick in the suit where h_2 elected to play.

4.1.2 Decision Problem and Fragments

The most natural decision problem associated to trick-taking games is to compute whether the value of a game is larger or equal to a given value ν. Put another way,

is it possible for some team to ensure capturing more than ν tricks? We will see in Section 4.2 that the general problem is **PSPACE**-hard, but there are several dimensions along which one can constrain the problem. This should allow to better understand where the complexity comes from.

Team mappings only allow team mappings belonging to a language $\mathcal{L} \subseteq \{A, B\}^*$, typically $\mathcal{L} = \mathcal{L}_i = \{(AB)^i\}$ or $\mathcal{L} = \_ = \{A, B\}^*$.

Number of suits the total number of distinct suits s is bounded by a number $s = S$, or unbounded $s = \_$.

Length of suits the maximal number of ranks over all suits l is bounded by a number $l = L$, or unbounded $l = \_$.

Symmetry for each suit, each hand needs to have the same number of cards pertaining to that suit.

The fragments of problems respecting such constraints are denoted by $\mathcal{B}(\mathcal{L}, s, l)$.
The largest class, that is, the set of all problems without any restriction is $\mathcal{B}(\_, \_, \_)$.

Example 1. The class of double-dummy Bridge problems is exactly $\mathcal{B}(\mathcal{L}_2, 4, 13)$.

Proposition 19. *$\mathcal{B}(\_, \_, \_)$ is in **PSPACE**.*

Proof. The game ends after a polynomial number of moves. It is possible to perform a minimax search of all possible move sequences using polynomial space to determine the maximal number of tricks team A can achieve. □

4.1.3 Generalized Geography

Generalized Geography (GG) is a zero-sum two-player game over a directed graph with one vertex token. The players take turn moving the token towards an adjacent vertex and thereby removing the origin vertex. The player who cannot play anymore loses. An example of a GG instance on a bipartite graph is given in Figure 4.2.

Deciding the winner of a GG instance is **PSPACE**-complete [116], and GG was used to prove **PSPACE**-hardness for numerous games including GO [95], OTHELLO [83], AMAZONS [69], UNO [48]. Lichtenstein and Sipser have shown that GG remains **PSPACE**-hard even if the graph is assumed to be bipartite [95].

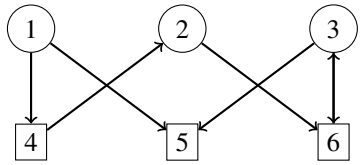


Figure 4.2: Example of an instance of GG.

4.2 Unbounded Number of Hands

We present a polynomial reduction ϕ from bipartite GG on graphs of degree 3 to $\mathcal{B}(\_, \_, 2)$.

An instance of GG on a bipartite graph is given by $(G = (V_A \cup V_B, E_{AB} \cup E_{BA}), v_1)$ where $v_1 \in V_A$ denotes the initial location of the token. Let $m = m_{AB} + m_{BA} = |E_{AB}| + |E_{BA}|$ the number of edges and $n = n_A + n_B = |V_A| + |V_B|$ the number of vertices. We construct an instance of $\mathcal{B}(\_, \_, 2)$ using $m + n(m + 1)$ suits, and $2n$ hands with $m + 1$ cards each as follows.

Each vertex $v \in V_A$ (resp. $\in V_B$) is encoded by a hand h_v owned by team A (resp. B). We add n additional dummy hands, hand h_{A_1} up to hand $h_{A_{n_B}}$ for team A and h_{B_1} up to hand $h_{B_{n_A}}$ for team B.

Each edge $(s, t) \in E_{AB}$ (resp. E_{BA}) is encoded by a suit $s_{s,t}$ of length 2, for instance $\{\text{AK}\}$. The cards in suit $s_{s,t}$ are dealt such that hand h_s receives K, hand h_t receives A. We add $n(m + 1)$ additional dummy suits $s_{A_{i,j}}$ for all $i \in \{1, \ldots, n_B\}$ and $j \in \{1, \ldots, m + 1\}$ and $s_{B_{i,j}}$ for all $i \in \{1, \ldots, n_A\}$ and $j \in \{1, \ldots, m + 1\}$.

For all $i \in \{1, \ldots, n_B$ (resp. $n_A)\}$, hand h_{A_i} (resp. h_{B_i}) receives A in all suits $s_{A_{i,j}}$ for $j \in \{1, \ldots, m + 1\}$, while hand h_i (resp. h_{n_A+i}) receives K in all suits $s_{A_{i,j}}$ for $j \in \{1, \ldots, m - \deg(v_i)\}$. Recall that $1 \leqslant \deg(v_i) \leqslant 3$ and note that the suits $s_{A_{i,j}}$ with $m - 2 \leqslant j$ might only feature A (with no K).

The goal of team A is to make at least $m_{BA} + 1$ tricks. Intuitively, for team A (resp. B) playing in a suit $s_{B_{i,j}}$ (resp. $s_{A_{i,j}}$) makes them lose all the remaining tricks (provided, of course, that hand h_{B_i} (resp. h_{A_i}) has not discarded its corresponding A) so it cannot be good. The interesting and difficult part of this bridge game would only occur in playing accurately the suits $s_{s,t}$ between hands h_i for $i, s, t \in \{1, \ldots, n\}$, that is the non dummy suits and the non dummy hands. Remark that this part simulates GG on the instance (G, v_1).

h_1		h_2		h_3	
$s_{B_{1,1}}$	K	$s_{B_{2,1}}$	K	$s_{B_{3,1}}$	K
$\vdots$		$\vdots$		$\vdots$	
$s_{B_{1,6}}$	K	$s_{B_{2,6}}$	K	$s_{B_{2,5}}$	K
$s_{1,4}$	K	$s_{2,6}$	K	$s_{3,5}$	K
$s_{1,5}$	K	$s_{4,2}$	A	$s_{3,6}$	K
				$s_{6,3}$	A

h_{A_1}		h_{A_2}		h_{A_3}	
$s_{A_{1,1}}$	A	$s_{A_{2,1}}$	A	$s_{A_{3,1}}$	A
$s_{A_{1,2}}$	A	$s_{A_{2,2}}$	A	$s_{A_{3,2}}$	A
$\vdots$		$\vdots$		$\vdots$	
$s_{A_{1,8}}$	A	$s_{A_{2,8}}$	A	$s_{A_{3,8}}$	A

h_4		h_5		h_6	
$s_{A_{1,1}}$	K	$s_{A_{2,1}}$	K	$s_{A_{3,1}}$	K
$\vdots$		$\vdots$		$\vdots$	
$s_{A_{1,6}}$	K	$s_{B_{2,6}}$	K	$s_{A_{3,5}}$	K
$s_{1,4}$	A	$s_{1,5}$	A	$s_{2,6}$	A
$s_{4,2}$	K	$s_{3,5}$	A	$s_{3,6}$	A
				$s_{6,3}$	K

h_{B_1}		h_{B_2}		h_{B_3}	
$s_{B_{1,1}}$	A	$s_{B_{2,1}}$	A	$s_{B_{3,1}}$	A
$s_{B_{1,2}}$	A	$s_{B_{2,2}}$	A	$s_{B_{3,2}}$	A
$\vdots$		$\vdots$		$\vdots$	
$s_{B_{1,8}}$	A	$s_{B_{2,8}}$	A	$s_{B_{3,8}}$	A

Figure 4.3: Reduction from Figure 4.2.

Lemma 21. *If Player 1 has a winning strategy in (G, v_1), then team A can make $m_{BA} + 1$ tricks in $\phi(G, v_1)$.*

Proof. Let ψ be the winning strategy of player 1, mapping a path $v_1 \dots v$ ending in V_A to a vertex v' in V_B. We define the following winning strategy for team A in $\phi(G, v_1)$. When in a hand h_{A_i} for some i, cash all the remaining A (all the remaining tricks). When in a hand h_i for some i, cash all A (they are in suits $s_{s,t}$). Then ψ tells you which of the K in a non dummy suit (suits of the form $s_{s,t}$) to play. Keeping track of which hands have taken the lead so far (without counting several times a hand which cashes some A) $h_1 h_{k_2} h_{k_3} \dots h_i$, play the K in $s_{i,t}$ where $\psi(v_1 v_{k_2} v_{k_3} \dots v_i)$ is the t-ieth vertex.

As for discarding, hands h_i for $i \in \{1, \dots, n_A\}$ can throw away any of the dummy K in suits s_{B_j} in any order. Hands h_{A_i} for $i \in \{1, \dots, n_B\}$ have to be more careful. They can start by discarding the A in suits $s_{A_{i,m-\deg(v_i)+1}}$ up to $s_{A_{i,m+1}}$. Then, they can discard in the same suit hand h_{n_A+i} has discarded its K at some previous trick. Indeed, hand h_{n_A+i} does not discard at most $\deg(v_i)$ times in the part of the game in non dummy hands.

From a hand h_j, team B cannot play towards a non dummy hand h_i of team A which has already taken the lead, since by construction, h_i has cashed all the A in

non dummy suits, and in particular the one in $s_{i,j}$, so the K owns by team B has gone. Team B, had therefore two losing options: follow an actual GG game simulation where he will eventually lose, or play in a dummy suit and lose all the remaining tricks. All in all, team B cannot cash more than its number of A in non dummy suits which is equal to m_{AB}. So, team A will make at least the complement $m_{BA} + 1$. □

The same result also applies to team B. Therefore team A has a winning strategy in $\phi(G, v_1)$ if and only if the first player has a winning strategy in the instance (G, v_1) of GG. The reduction is thus complete, leading to **PSPACE**-hardness.

Theorem 25. $\mathcal{B}(\_, \_, 2)$ *is* ***PSPACE****-complete.*

4.3 Bounded Number of Hands

The *trick balance* in an intermediate position is the number of tricks made by team A so far minus the number of tricks made by team B so far.

The basic idea of this reduction is that we have a termination gadget that allows both team to end the game by splitting the remaining tricks evenly. However, using the termination gadget comes with a small cost. So each team tries to achieve a sufficiently high trick balance before terminating the game. The termination gadget involves two suits s_A and s_B and four hands h_3 through h_6 that do not otherwise influence the game.

Besides the termination gadget, we have two hands, one per team, and one suit s_v for each vertex v of the GG instance. A team, say A, can threaten to increase the trick balance in their favor by playing in the attacking gadget of a suit s_v. This can only be defended by having the opponent team, say B, counter-attacking in a suit $s_{v'}$. B can choose $s_{v'}$, but for the defense to be successful, v' needs to be a neighbor of v in GG. After this exchange is performed, team B has priority to attack but can only attack in suit $s_{v'}$'s attacking gadget. The same process goes on until the defending team cannot find an appropriate counter-attacking suit. At that moment, the attacking team manages to increase the trick balance enough to safely terminate the game in their favor. We see that picking the counter-attacking suits emulates a game of GG on G.

Let $G = ((V = V_A \cup V_B, E), v_0)$, a directed bipartite graph and one of its vertex v_0, be an instance of GG. Let $n = n_A + n_B$ the number of vertices, and $N(v)$ denote the set of neighbors of a vertex v. We construct in polynomial time an equivalent instance of $\mathcal{B}(\mathcal{L}_3, \_, \_)$ using 6 hands and $s = n + 2$ suits. In the instance we create, the seating order does not have any influence so we will represent gadgets and positions

Hand	Suit	Ranks
h_1	s_A	1
h_2	s_B	1
h_3	s_A	2t+1 $\frac{2}{—}$ 3
h_4	s_A	2t $\frac{2}{—}$ 2
h_5	s_B	2t+1 — 2t-3ω+1 $\frac{2}{—}$ 3
h_6	s_B	2t-3ω $\frac{2}{—}$ 3ω+1 — 2

Figure 4.4: The termination gadget.

simply by listing the cards in each hand in a table. In the following, we will use ω to represent a large number, for instance we can set $\omega = 8n$. We will also use t to represent the total number of tricks to be made.[3] Team A wins if they make strictly more than $t/2 + \omega/4$ tricks.

4.3.1 Presentation of the Gadgets

Unless the gadgets are not symmetrical, we only describe one team's version of the gadgets. Assume an arbitrary ordering on the vertices in V_A and in V_B, that is $V_A = \{v_1, v_2, \dots v_{n_A}\}$ and $V_B = \{v'_1, v'_2, \dots v'_{n_B}\}$.

The termination gadget. The suit s_A has length $2t + 1$ and is possessed only by hands h_1, h_3 and h_4. Hand h_1 only has the smallest card in the suit, hand h_3 has all the other cards of the suit with an odd rank and hand h_4 has all the cards of that suit with an even rank. Thus, hands h_3 and h_4 have only cards in the suit s_A. The suit s_B is owned only by hands h_2, h_5 and h_6. It is defined similarly except h_5 has 3ω top cards to cash in that suit, then the cards are interleaved.

The following two lemmas allow us to focus on hands h_1 and h_2 in the rest of the reduction.

Lemma 22. *If h_1 leads and the trick balance is ω, then team A can ensure winning the game.*

Proof. Team A can play in the suit s_A then the rest of the game will hold between hands h_3 and h_4. It is easy to see that team A wins half (rounded up) of the remaining

[3] t's exact value can be computed and is polynomial in the input.

Hand	Suit	Ranks	Concise notation
h_1	s	7 — 1	$3\|4$
h_2	s	10 — 8 xxxx	

Figure 4.5: A $3|4$-block in a suit s for team A. Only the hands and the suits involved in this gadget are displayed. h_2 does have cards in suit s, but they are not displayed in the compact notation, as they can be deduced from the cards in h_1.

Hand	Suit	Ranks	Concise notation
h_1	s	27 — 20 15 — 9 5 — 1	$3\|5$ $4\|3$ $3\|2$
h_2	s	30 — 28 19 — 16 8 — 6 xx...	

Figure 4.6: Concatenation of $3|5$-, $4|3$- and $3|2$-blocks in s for team A. There are $5 + 3 + 2 = 10$ x in suit s in hand h_2.

tricks. □

Lemma 23. *If h_2 leads and the trick balance is -2ω, then team B can ensure winning the game.*

Proof. Team B can play in the suit s_B then the rest of the game will hold between hands h_5 and h_6. Team B first loses 3ω tricks, then the remaining tricks are split. □

The $e|w$-block. An $e|w$-block in a suit s for team A is the possibility for team A to cash w tricks by playing e times in the suit. In other words, hand h_2 has the e top cards and h_1 has the $e + w$ following top cards. e stands for *establish* and w for *winners*. Figure 4.5 provides an example of a $3|4$-block.

We can concatenate several $e|w$-blocks for the same team in the same suit. For instance, Figure 4.6 shows how blocks are concatenated and provides a more concise notation.

Given a vertex $v \in V$, a concatenation of $e|w$ blocks with various values for e allows to encode the index of v in an *attacking* gadget. It also allows to encode which vertex v', v is adjacent to in a *counter-attacking* gadget via their indices. If $v \in V_B$ (resp. V_A), the (counter-)attacking gadgets will be for team A (resp. B) and we say that the corresponding suit is a defensive suit for team B (resp. A). The $e|w$-blocks we need in the following have e an integer in $[1, \ldots, 6]$, and w a fraction of ω.

Suit	Counter-attacking		Attacking	
	Word c	Gadget $\mathcal{C}$	Word a	Gadget $\mathcal{A}$
s_{v_1}	1 000 1	$3\|\omega\ 2\|\omega\ 2\|\omega\ 1\|\omega$	1 100 0	$2\|\omega\ 3\|\omega\ 2\|\omega\ 2\|\omega$
s_{v_2}	1 100 1	$2\|\omega\ 3\|\omega\ 2\|\omega\ 1\|\omega$	1 010 0	$3\|\omega\ 1\|\omega\ 3\|\omega\ 2\|\omega$
s_{v_3}	1 001 1	$3\|\omega\ 2\|\omega\ 1\|\omega\ 2\|\omega$	1 001 0	$3\|\omega\ 2\|\omega\ 1\|\omega\ 3\|\omega$
s_{v_4}	1 100 1	$2\|\omega\ 3\|\omega\ 2\|\omega\ 1\|\omega$	1 100 0	$2\|\omega\ 3\|\omega\ 2\|\omega\ 2\|\omega$
s_{v_5}	1 101 1	$2\|\omega\ 3\|\omega\ 1\|\omega\ 2\|\omega$	1 010 0	$3\|\omega\ 1\|\omega\ 3\|\omega\ 2\|\omega$
s_{v_6}	1 011 1	$3\|\omega\ 1\|\omega\ 2\|\omega\ 2\|\omega$	1 001 0	$3\|\omega\ 2\|\omega\ 1\|\omega\ 3\|\omega$

Figure 4.7: Counter-attacking and attacking gadgets in the instance corresponding to Figure 4.2.

The counter-attacking and attacking gadgets. Consider two words over $\{0, 1\}$ for each suit s_v with $v \in V_A$. The attacking word for suit s_{v_i} is a such that $a(0) = 1$, $a(n_A + 1) = 0$, and for each $j \neq i, j \in [1, n_A]$, $a(j) = 0$ and $a(i) = 1$. The counter-attacking word for suit s_{v_i} is c such that $c(0) = 1$, $c(n_B + 1) = 1$, and for each $j \in [1, n_B]$, if $v_i \in N(v_j)$ then $c(j) = 1$ else $c(j) = 0$.

The gadgets can be built by looking at adjacent letters in these words. If these letters are 11 or 00, put a $2|\omega$-block. If they are 10, put $3|\omega$-block, and if they are 01, put $1|\omega$. We thus define for each suit s_v, a *counter-attacking* gadget $\mathcal{C}(v)$ and an *attacking* gadget $\mathcal{A}(v)$. The words and gadgets for the GG instance in Figure 4.2 are given in Figure 4.7.

Let $v \in V_A$ and $v' \in V_B$. Observe that the sum of the e parts of the $\mathcal{A}(v)$ gadgets is equal to $2(n_A + 1) + 1$ and that of the $\mathcal{C}(v')$ gadgets is $2(n_A + 1)$. Similarly, the sum of the w parts is $\omega(n_A + 1)$. The same holds for $\mathcal{A}(v')$ and $\mathcal{C}(v)$, replacing n_B with n_A.

Assume one hand leads (s, r), the other team is said to *duck* if it has cards higher than r in suit s but plays a card lower than r, thereby refusing to take the trick.

Lemma 24. *When a team sets up tricks in a suit, there is no point in ducking.*

Proof. Anyway, the tricks will be established. Ducking only enables the opponent to get her winners earlier. □

In the next two lemmas, we assume optimal play from both teams subject to leading from a single suit.

Lemma 25. *Assume the initial trick balance is* 0*, team B starts, team A only leads cards from* $\mathcal{A}(v)$*, and team B only leads cards from* $\mathcal{C}(v')$*. The trick balance remains* $\geqslant -\omega$*. If* $v' \in N(v)$*, it remains* $\leqslant 1$*, else it reaches* $\omega + 1$*.*

Proof. It is optimal to play blocks from the highest to the lowest ranked when in lead, and to take any trick offered when not in lead. Let i the index of v. Observe that team A needs to $2j + 1$ tempi to establish the jth block if $j \neq i$, and $2i$ tempi for the ith block. Team B needs $2j$ tempi to establish the jth block if $v' \in N(v_j)$, and $2j + 1$ tempi otherwise. □

Lemma 26. *Assume the initial trick balance is* $-3\omega/2$*, team A starts, team A only leads cards from* $\mathcal{C}(v)$*, and team B only leads cards from* $\mathcal{A}(v')$*. The trick balance remains* $\leqslant -\omega/2$*. If* $v \in N(v')$*, it remains* $\geqslant -3\omega/2 - 1$*, else it reaches* $-5\omega/2$*.*

When team A attacks in the suit s_v and team B does not play in an admissible counter-attacking suit, team A establishes ω tricks before her (Lemma 25) and wins by termination (Lemma 22). Conversely, if team A does not play in a neighboring suit when team B attacks, team A loses (Lemma 23, 26). Thus, Lemmas 25 and 26 give the graph structure to the suits.

Combining the attacking and counter-attacking gadgets. We now need to complete the picture so that the assumptions of Lemmas 25 and 26 are met.

In each suit s_v of hand h_1 (resp. h_2) but the one corresponding to the starting vertex v_0, we start by the counter-attacking gadget $\mathcal{C}(v)$ surrounded by the fixed sequences $5|\omega\ 1|^{3\omega}/_2$ and $2|\omega$ (resp. $4|^{\omega}/_2\ 3|2\omega$ and $2|2\omega$) and call it *first part* of the suit. We then add to each suit, including s_{v_0}, the attacking gadget $\mathcal{A}(v)$ surrounded by the fixed sequences $4|\omega\ 3|^{3\omega}/_2\ 1|^{3\omega}/_2$ and $1|\omega$ (resp. $6|\omega\ 1|^{3\omega}/_2$ and $1|^{3\omega}/_2$) and call it *second part* of the suit. Figure 4.8 displays the combination resulting from the GG instance in Figure 4.2.

These fixed starting sequences ensure that once a team leads in suit s_v, they will continue leading only in s_v until the suit is emptied. They also ensure, that while one team chooses the attacking suit first, the opponent actually starts leading in the counter-attacking gadget.

The ending sequences, on the other hand, ensure that after the attacking suit s_v and the first part of the counter-attacking suit $s_{v'}$ have been emptied, the situation corresponds to the reduction from the GG instance with the edges adjacent to v removed and v' as a starting vertex.

Hand	Suit	Ranks	
	s_{v_1}		$6\|\omega\ 1\|^{3\omega/2}\ \mathcal{A}(v_1)\ 1\|^{3\omega/2}$
h_2	s_{v_2}	$4\|^{\omega/2}\ 3\|2\omega\ \mathcal{C}(v_2)\ 2\|2\omega$	$6\|\omega\ 1\|^{3\omega/2}\ \mathcal{A}(v_2)\ 1\|^{3\omega/2}$
	s_{v_3}	$4\|^{\omega/2}\ 3\|2\omega\ \mathcal{C}(v_3)\ 2\|2\omega$	$6\|\omega\ 1\|^{3\omega/2}\ \mathcal{A}(v_3)\ 1\|^{3\omega/2}$
	s_{v_4}	$5\|\omega\ 1\|^{3\omega/2}\ \mathcal{C}(v_4)\ 2\|\omega$	$4\|\omega\ 3\|^{3\omega/2}\ 1\|^{3\omega/2}\ \mathcal{A}(v_4)\ 1\|\omega$
h_1	s_{v_5}	$5\|\omega\ 1\|^{3\omega/2}\ \mathcal{C}(v_5)\ 2\|\omega$	$4\|\omega\ 3\|^{3\omega/2}\ 1\|^{3\omega/2}\ \mathcal{A}(v_5)\ 1\|\omega$
	s_{v_6}	$5\|\omega\ 1\|^{3\omega/2}\ \mathcal{C}(v_6)\ 2\|\omega$	$4\|\omega\ 3\|^{3\omega/2}\ 1\|^{3\omega/2}\ \mathcal{A}(v_6)\ 1\|\omega$

Figure 4.8: Combination of attacking and counter-attacking gadgets for the instance corresponding to Figure 4.2, with v_1 as starting vertex.

Lemma 27. *A trick balance of ω cannot be achieved by leading in defensive suits.*

Ensuring the players simulate GG.

Let P a position resulting from one constructed from a GG instance.Assume there exists a suit s_v (and v the corresponding vertex in the original GG instance), such that for any suit s different from s_A, s_B, and s_v, s is dealt among hands h_1 and h_2 so as to form a first part and a second part in h_1 or in h_2. If s_v forms only a second part in h_2 (resp. h_1), h_2 (resp. h_1) is on the lead, and the trick balance is $\frac{1}{2}\omega$ (resp. 0), then we say that P is *A-clean* (resp. *B-clean*) and s_v is the current *starting suit*.

Lemma 28. *Let P a B-clean position with starting suit s_v, and a suit $s_{v'}$ such that $v' \in N(v)$. Assume team A can ensure winning with optimal play from P. If team B only leads from suit $s_{v'}$ and team A only leads from s_v until s_v is empty, then we reach a A-clean position P' with $s_{v'}$ as the starting suit. Moreover, team A can ensure winning with optimal play from P'.*

Proof. As P is a B-clean position, the trick balance is 0 and the lead is on h_1. Suppose team A plays in a suit s_u with $u \neq v$ and $u \in V_A$, before having established and cashed the ω tricks of the $6|\omega$-block of the color s_v (first block of the second part of that color). After 6 tempi, team B has had time to cash the $\frac{5\omega}{2}$ tricks of the two first blocks $5|\omega$ and $1|^{\frac{3\omega}{2}}$ of the color $s_{v'}$, while team A has at most cash the $\frac{\omega}{2}$ winners of the first block $4|^{\frac{\omega}{2}}$ of s_u. Consequently, the balance trick takes a value smaller than -2ω and team B wins accordingly to Lemma 23. In particular, team A cannot ensure winning. Thus, team A has to use the 6 first tempi to play in s_v. At this point, team B threatens to enter in $\mathcal{C}(v')$ while team A cannot yet enter $\mathcal{A}(v)$. That forces team A to play again

in v. Then, team B enters $\mathcal{C}(v')$ immediately followed by team A cashing $\frac{3\omega}{2}$ tricks (the trick balance is now 0) and entering $\mathcal{A}(v)$. According to Lemma 25, team A has to play solely in $\mathcal{A}(v)$ until emptying this gadget.

In the end, the balance is $-\omega$, the lead is in h_2 and the only remaining cards in h_1 in the color s_v are $\frac{3\omega}{2}$ winners. So the trick balance is virtually $\frac{\omega}{2}$ and we reach a A-clean position winning for team A. □

Proof. As P is an A-clean position, the trick balance is $\frac{\omega}{2}$ and the lead is on h_2. Suppose team B plays in a suit $s_{u'}$ with $u' \neq v'$ and $u' \in V_B$, before having established and cashed the ω tricks of the $4|\omega$-block of the color $s_{v'}$ (first block of the second part of that color). After 4 tempi, team A has had time to cash the $\frac{\omega}{2}$ tricks of the two first blocks $4|\frac{\omega}{2}$ of the color s_v, while team B could not cash any winners of a block. Consequently, the balance trick takes a value greater than ω and team A wins accordingly to Lemma 22. Thus, team B has to use the 4 first tempi to play in s_v. For similar considerations on the balance trick, team B has to use the next 3 tempi playing in s_v. Note that the trick balance does not reach a lethal value: taking values $\omega 2$, $-\frac{\omega}{2}$, $\frac{3\omega}{2}$, $\frac{\omega}{2}$. At this point, team A threatens to enter in $\mathcal{C}(v)$ while team B cannot yet enter $\mathcal{A}(v')$. That forces team B to play again in $\mathcal{A}(v)$. Then, team A enters $\mathcal{C}(v)$ immediately followed by team B cashing $\frac{3\omega}{2}$ tricks (the trick balance is now $-\omega$) and entering $\mathcal{A}(v')$. According to Lemma 26, team B has to play solely in $\mathcal{A}(v')$ until emptying this gadget. At last, the balance is $-\omega$, the lead is in h_1 and the only remaining cards in h_2 in the color $s_{v'}$ are ω winners. So, we reach a B-clean position which is necessarily winning for team B, and the trick balance is virtually 0. □

Lemma 29. *Let P a A-clean position with starting suit $s_{v'}$. Assume team A can ensure winning with optimal play from P. Then there exists a suit s_v such that $v \in N(v')$ and such that if team B only leads from suit $s_{v'}$ and team A only leads from s_v until $s_{v'}$ is empty, then we reach a B-clean position P' with s_v as the starting suit. Moreover, team A can ensure winning with optimal play from P'.*

Lemma 30. *Let G be an GG instance, and consider the corresponding $\mathcal{B}(\mathcal{L}_3, \_, \_)$ instance B. If team A can win in B, then the second player in G does not have a winning strategy.*

Proof. Let σ a strategy for the second player in G and let us show that σ is not winning. Assume team B plays according to σ in the B instance. Team A can answer by keeping simulating GG and still ensure winning (Lemma 28 and 29), thereby generating a

strategy in GG. Since there are only finitely many suits to be emptied, we will reach a B-clean position with starting suit s_v and without any suit $s_{v'}$ such that $v' \in N(v)$. This shows that the corresponding GG situation is lost and that σ is not a winning strategy. □

Lemma 31. *Let G be an GG instance, and consider the corresponding $\mathcal{B}(\mathcal{L}_3, \_, \_)$ instance B. If team B can win in B, then the first player in G does not have a winning strategy.*

Proof. Similar proof with the dual to Lemmas 28 and 29. □

These two propositions lead us to our main result.

Theorem 26. *The $\mathcal{B}(\mathcal{L}_3, \_, \_)$ fragment is **PSPACE**-hard.*

4.4 Connection Games

Numerous popular abstract strategy games ranging from HEX and HAVANNAH to LINES OF ACTION belong to the class of connection games. Still, very few complexity results on such games have been obtained since HEX was proved **PSPACE**-complete in the early eighties.
We study the complexity of two connection games among the most widely played. Namely, we prove that HAVANNAH and TWIXT are **PSPACE**-complete.
The proof for HAVANNAH involves a reduction from GENERALIZED GEOGRAPHY and is based solely on ring-threats to represent the input graph. On the other hand, the reduction for TWIXT builds up on previous work as it is a straightforward encoding of HEX.

A connection game is a kind of abstract strategy game in which players try to make a specific type of connection with their pieces [25]. In many connection games, the goal is to connect two opposite sides of a board. In these games, players take turns placing or/and moving pieces until they connect the two sides of the board. HEX, TWIXT, and SLITHER are typical examples of this type of game. However, a connection

game can also involve completing a loop (HAVANNAH) or connecting all the pieces of a color (LINES OF ACTION).

A typical process in studying an abstract strategy game, and in particular a connection game, is to develop an artificial player for it by adapting standard techniques from the game search literature, in particular the classical Alpha-Beta algorithm [4] or the more recent Monte Carlo Tree Search paradigm [26, 6]. These algorithms explore an exponentially large game tree are meaningful when optimal polynomial time algorithms are impossible or unlikely. For instance, tree search algorithms would not be used for NIM and SHANNON'S EDGE SWITCHING GAME which can be played optimally and solved in polynomial time [27].

Until now, connection games have received few attention. Besides Even and Tarjan's proof that SHANNON'S VERTEX SWITCHING GAME is **PSPACE**-complete [56] and Reisch's proof that HEX is **PSPACE**-complete [113], the only complexity results on connection games that we know of are the **PSPACE**-completeness of virtual connection detection [90] in HEX, the **NP**-completeness of dominated cell detection in SHANNON'S VERTEX SWITCHING GAME [12], as well as an unpublished note showing that a problem related to TWIXT is **NP**-complete [104].[4]

The two games that we study in Section 4.5 and Section 4.6 rank among the most notable connection games. They were the main topic of multiple master's theses and research articles [80, 104, 106, 124, 122, 97, 57], and they both gave rise to competitive play. High-level online competitive play takes place on `www.littlegolem.net`. Finally, live competitive play can also be observed between human players at the Mind Sports Olympiads where an international TWIXT championship has been organized every year since 1997, as well as between HAVANNAH computer players at the ICGA Computer Olympiad since 2009.[5]

4.5 Havannah

HAVANNAH is a 2-player connection game played on a hexagonal board paved by hexagons. White and Black place a stone of their color in turn in an unoccupied cell. Stones cannot be taken, moved nor removed. Two cells are neighbors if they share an edge. A group is a connected component of stones of the same color via the neighbor

[4] For a summary in English of Reisch's reduction, see Maarup's thesis [100].

[5] See `www.boardability.com/game.php?id=twixt` and `www.grappa.univ-lille3.fr/icga/game.php?id=37` for details.

relation. A player wins if they realize one of the three following different structures: a circular group, called *ring*, with at least one cell, possibly empty, inside; a group linking two corners of the board, called *bridge*; or a group linking three edges of the board, called *fork*.

As the length of a game of HAVANNAH is polynomially bounded, exploring the whole game tree can be done with polynomial space, so HAVANNAH is in **PSPACE**.

In our reduction, the HAVANNAH board is large enough that the gadgets are far from the edges and the corners. Additionally, the gadgets feature ring threats that are short enough that the bridges and forks winning conditions do not have any influence. Before starting the reduction, we define threats and make two observations that will prove useful in the course of the reduction.

A *simple threat* is defined as a move which threatens to realize a ring on the next move on a unique cell. There are only two kinds of answers to a simple threat: either win on the spot or defend by placing a stone in the cell creating this very threat. A *double threat* is defined as a move which threatens to realize a ring on the next move on at least two different cells. We will use *threat* as a generic term to encompass both simple and double threats. A *winning sequence of threats* is defined as a sequence of simple threats ended by a double threat for one player such that the opponent's forced move never makes a threat. Thus, when a player is not threatened and can initiate a winning sequence of threats, they do win. To be more concise, we will denote by $W : a_1$,a_2; a_3,a_4; ...; a_{2n-1}(,a_{2n}) the sequence of moves starting with White's move a_1, Black's answer a_2, and so on. a_{2n} is optional, for the last move of the sequence might be White's or Black's. Similarly, $B : a_1$,a_2; a_3,a_4; ...; a_{2n-1}(,a_{2n}) denotes the corresponding sequence of moves initiating by Black. We will use the following lemmas multiple times:

Lemma 32. *If a player is not threatened, playing a simple threat forces the opponent to answer on the cell of the threat.*

Proof. Otherwise, no matter what have played their opponent, placing a stone on the cell of the threat wins the game.

□

Lemma 33. *If a player is not threatened, playing a double threat is winning.*

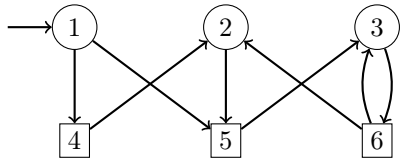


Figure 4.9: Example of an instance of GG with vertex 1 as initial vertex.

Proof. The player is not threatened, so their opponent can not win at their turn. Let u and v be two cells of the double threat. If their opponent plays in u, the player wins by playing in v. If their opponent plays somewhere else, the player wins by playing in u. □

4.5.1 Generalized Geography

Lichtenstein and Sipser have proved that the game remained **PSPACE**-hard even if G was assumed to be bipartite and of degree at most 3 [95]. This time, we will reduce from such a restriction of GG to show that HAVANNAH is **PSPACE**-hard.

We denote by $P(v)$ the set of predecessors of the vertex v in G, and $S(v)$ the set of successors of v. A vertex with in-degree i and out-degree o is called (i, o)-vertex. The degree of a vertex is the sum of the in-degree and the out-degree, and the degree of G is the maximal degree among all vertices of G. If V is the set of vertices of G and V' is a subset of vertices, then $G[V \setminus V']$ is the induced subgraph of G where vertices belonging to V' have been removed.

To limit the number of gadgets we need to create, we will also assume a few simplifications detailed below. An example of a simplified instance of GG can be found in Fig. 4.9.

Let (G, v_0) be an instance of GG with G bipartite and of degree at most 3. We can assume that there is no vertex v with out-degree 0 in G. Indeed, if $v_0 \in P(v)$ then (G, v_0) is trivially winning for Player 1. Else, $(G[V \setminus (\{v\} \cup P(v))], v_0)$ is an equivalent instance, since playing in a predecessor of v is losing.

All edges coming to the initial vertex v_0 can be removed to form an equivalent instance. So, v_0 is a $(0, 1)$-, a $(0, 2)$-, or a $(0, 3)$-vertex. If $S(v_0) = \{v'\}$, then $(G[V \setminus \{v_0\}], v')$ is a strictly smaller instance such that Player 1 is winning in (G, v_0) if and only if Player 1 is losing in $(G[V \setminus \{v_0\}], v')$. If $S(v_0) = \{v', v'', v'''\}$, then Player 1 is winning in (G, v_0) if and only if Player 1 is losing in at least one of the

three instances $(G[V \setminus \{v_0\}], v')$, $(G[V \setminus \{v_0\}], v'')$, and $(G[V \setminus \{v_0\}], v''')$. In those three instances v', v'', and v'' are not $(0,3)$-vertices since they had in-degree at least 1 in G. Therefore, we can also assume that v_0 is $(0,2)$-vertex.

We call an instance with an initial $(0,2)$-vertex and then only $(1,1)$-, $(1,2)$-, and $(2,1)$-vertices a *simplified* instance.

In the following subsections we propose gadgets that encode the different parts of a *simplified* instance of GG. These gadgets have starting points and ending points. The gadgets are assembled so that the ending point of a gadget coincides with the starting point of the next one. The resulting instance of HAVANNAH is such that both players must enter in the gadgets by a starting point and leave it by an ending point otherwise they lose.

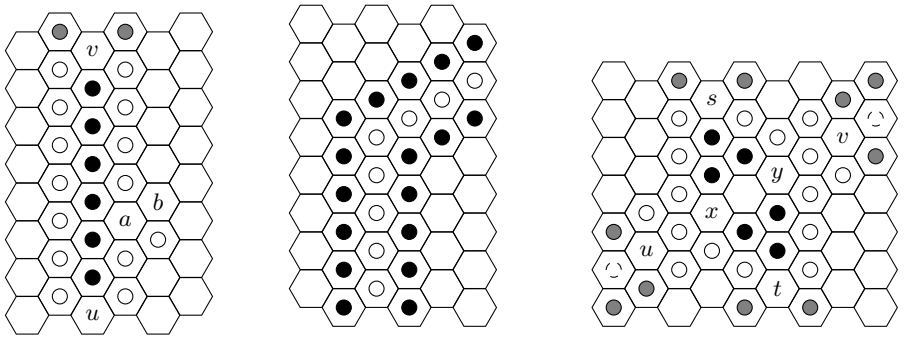
4.5.2 Edge Gadgets

Wires, curves, and crossroads will enable us to encode the edges of the input graph. In the representation of the gadgets, White and Black stones form the proper gadget. Dashed stones and gray stones are respectively White and Black stones setting the context.

In the HAVANNAH board we name the 6 directions: North, North-West, South-West, South, South-East, and North-East according to standard designation. While figures and lemmas are mostly presented from White's point of view, all the gadgets and lemmas work exactly the same way with colors reversed.

The wire gadget. Basically, a wire teleports moves: one player plays in a cell u and their opponent has to answer in a possibly remote cell v. u is called the starting point of the wire and v is called its ending point. A wire where White prepares a threat and Black answers is called a WB-wire (Fig. 4.10a); conversely, we also have BW-wires. We say that WB-wires and BW-wires are *opposite* wires. Note that wires can be of arbitrary length and can be curved with 120° angles (Fig. 4.10b). On an empty board, a wire can link any pair of cells as starting and ending point provided they are far enough from each other.

Lemma 34. *If White plays in the starting point u of a WB-wire (Fig. 4.10a), and Black does not answer by a threat, Black is forced to play in the ending point v (possibly with moves at a and b interleaved).*



(a) Entire WB-wire which starts in u and ends in v.

(b) Curved fragment for a BW-wire.

(c) Crossover gadget.

Figure 4.10: Edge gadgets.

Proof. If Black does not play neither in a nor in b, then White plays in v which makes a double threat in a and b and wins by Lemma 33. If Black plays in a (resp. in b), at the very least White can play in b (resp. in a) which forces Black to play in v by Lemma 32.

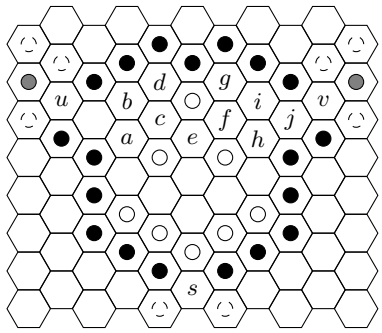
□

The crossover gadget. The input graph of GG might not be planar, so we have to design a crossover gadget to enable two chains of wires to cross. Fig. 4.10c displays a crossover gadget, we have a South-West BW-wire with starting point u which is linked to a North-East BW-wire with ending point v, and a North BW-wire with starting point s is linked to a South BW-wire with ending point t.

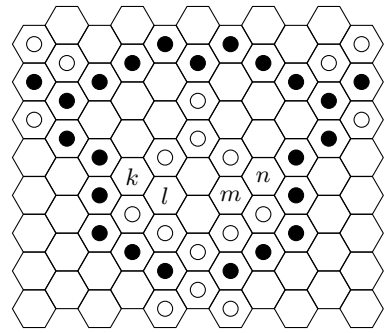
Lemma 35. *In a crossover gadget (Fig. 4.10c), if White plays in the starting point u, Black ends up playing in the ending point v and if White plays in the starting point s, Black ends up playing in the ending point t.*

Proof. By Lemma 32, if White plays in u, Black has to play in x, forcing White to play in y, forcing finally Black to play in v. If White plays in s, again by Lemma 32, Black has to play in t.

□



(a) Gadget before being used. The wires for the in-edges end in u and v, the wire for the out-edge starts in s.



(b) Gadget after being used and then reentered. White wins with a double threat.

Figure 4.11: The $(2,1)$-vertex gadget links three WB-wires. The North-West and North-East ones end in u and v, the South WB-wire starts in s.

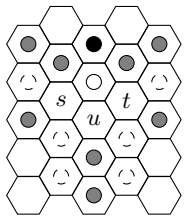
Note that the South wire is linked to the North wire irrespective of whether the other pair of wires has been used and conversely. That is, in a crossover gadget two paths are completely independent.

4.5.3 Vertex Gadgets

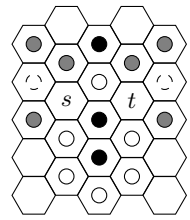
We now describe the gadgets encoding the vertices. Recall from Section 4.5.1 that simplified GG instances only feature $(1,2)$-, $(1,1)$-, and $(2,1)$-vertices, and a $(0,2)$-vertex. One can encode a $(1,1)$-vertex with two consecutive opposite wires. Thus, we will only present three vertex gadgets, one for $(2,1)$-vertices, one for $(1,2)$-vertices, and one for the $(0,2)$-vertex.

The (2,1)-vertex gadget. A $(2,1)$-vertex gadget receives two wire ending points. If a stone is played on either of those ending points, it should force an answer in the starting point of a third wire. That simulates a vertex with two edges going in and one edge going out.

Lemma 36. *If Black plays in one of the two possible starting points u and v of a $(2,1)$-vertex gadget (Fig. 4.11b), and White does not answer by a threat, White is forced to play in the ending point s.*



(a) The $(1, 2)$-vertex gadget. The wire for the in-edge ends in u.



(b) The $(0, 2)$-vertex gadget representing the starting vertex v_0.

Figure 4.12: In these choice gadgets, White can defend by playing in s or in t. A North-West BW-wire starts in s and a North-East BW-wire starts in t.

Proof. Assume Black plays in u and White answers by a move which is not in s nor a threat. This move from White has to be either in v or in j, otherwise, Black has a double threat by playing in s and wins by Lemma 33. Suppose White plays in v. Now, Black plays in s with a simple threat in j, so White has to play in j by Lemma 32. Then Black has the following winning sequence: B: a,b; c,d; h,i; f. Black has now a double threat in g and e and so wins by Lemma 33. If White plays in j instead of v, the argument is similar.

If Black plays the first move in v, the proof that White has to play in s is similar. □

The (1-2)-vertex and (0,2)-vertex gadgets. A $(1, 2)$-vertex gadget receives one ending point of a wire (Fig. 4.12a). If a stone is played on this ending point, it should offer the choice to defend either by playing in the starting point of a second wire, or by playing in the starting point of a third wire. That simulates a vertex with one edge going in and two edges going out. The $(0, 2)$-vertex gadget (or *starting-vertex* gadget) can be seen as a $(1, 2)$-vertex gadget where a stone has already been played on the ending point of the in-edge. The $(0, 2)$-vertex gadget represents the two possible choices of the first player at the beginning of the game.

Lemma 37. *If Black plays in the starting point u of a $(1, 2)$-vertex gadget (Fig. 4.12a), and White does not play a threat, White is forced to play in one of the two ending points s and t, then, if Black does not answer by a threat, they have to play in the other ending point.*

Proof. Black plays in u. Suppose White plays neither in s nor in t nor a threatening move. Then Black plays in s. By Lemma 32, White has to play in t but Black wins by playing in the ending point of the wire starting at s by Lemma 34.

Assume White's answer to u is to play in s. t can now be seen as the ending point of the in-wire, so Black needs to play in t or make a threat by Lemma 34.

□

Corollary 14. *If White is forced to play a threat or to open the game in one of the two opening points s and t of the $(0, 2)$-vertex gadget (Fig. 4.12b). Then, if Black does not play a threat, they are forced to play in the other opening point.*

4.5.4 Assembling the Gadgets Together

Let (G, v_0) be a simplified instance of GG, and n be its number of vertices. G being bipartite, we denote by V_1 the side of the partition containing v_0, and V_2 the other side. Player 1 moves the token from vertices of V_1 to vertices of V_2 and player 2 moves the token from V_2 to V_1. We denote by ϕ the reduction from GG to HAVANNAH. Let us describe the construction of $\phi((G, v_0))$. As an example, we provide the reduction from the GG instance from Fig. 4.9 in Fig. 4.13.

The initial vertex v_0 is encoded by the gadget displayed in Fig. 4.12b. Each player 1's $(2, 1)$-vertex is encoded by the $(2, 1)$-vertex gadget of Fig. 4.11a, and each player 2's $(2, 1)$-vertex is encoded by the same gadget in reverse color. Each player 1's $(1, 2)$-vertex is encoded by the $(1, 2)$-vertex gadget of Fig. 4.12a, and each player 2's $(1, 2)$-vertex is encoded by the same gadget in reverse color.

All White's vertex gadgets are aligned and all Black's vertex gadgets are aligned on a parallel line. Whenever (u, v) is an edge in G, we connect an exit of the vertex gadget representing u to an entrance of a gadget encoding v using wires and crossover gadgets. Let n be the number of vertices in G, since G is of degree 3, we know that the number of edges is at most $3n/2$. The minimal size in terms of HAVANNAH cells for a smallest wire and the size of a crossover are constants. Therefore the distance between Black's line and White's line is linear in n. Note that, two wires of opposite colors might be needed to connect two vertex gadgets or a vertex gadget and a crossover. Similarly, we can show that the distance between two vertices on Black's line or on White's line is constant.

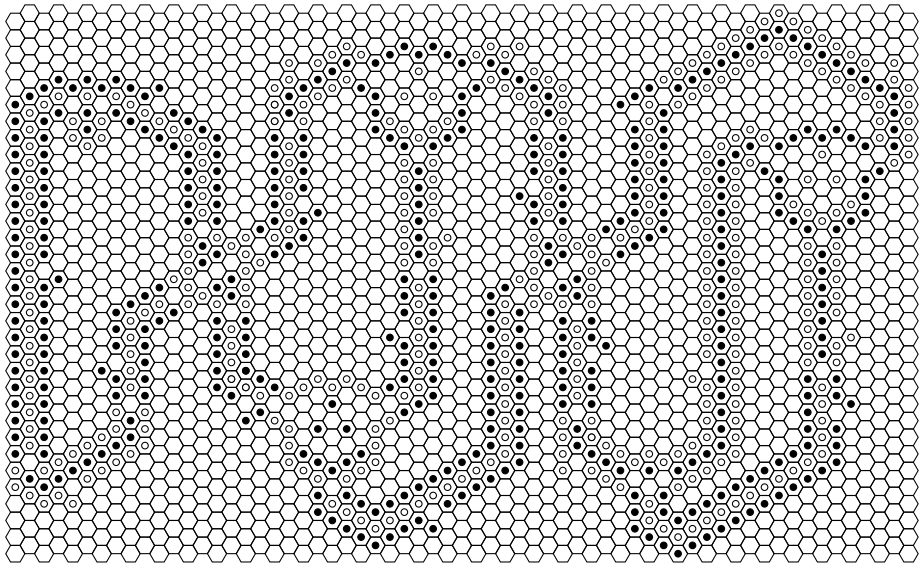


Figure 4.13: HAVANNAH gadgets representing the GG instance from Fig. 4.9.

Lemma 38. *If Black reenters a White's* $(2, 1)$*-vertex gadget (Fig. 4.11b), and Black has no winning sequence of threats elsewhere, White wins.*

Proof. If Black reenters a White's $(2, 1)$-vertex by playing in v, White plays in e. As Black cannot initiate a winning sequence, whatever he plays White can defend until Black is not threatening anymore. Then White plays in k or in l with a decisive double threat in m and n.

□

Theorem 27. HAVANNAH *is* ***PSPACE****-complete.*

Proof. We already mention that HAVANNAH $\in$ **PSPACE** and we just presented a polynomial time reduction from a **PSPACE**-complete problem. We shall now prove that the reduction is sound, that is: player 1 is winning in (G, v_0) if and only if White is winning in $\phi((G, v_0))$. First we show that the players in the game of HAVANNAH lose if they make a move which does not correspond to anything in the instance of GG. Such a move will be called a *cheating* move. The exhaustive list of non cheating

moves is: defending a threat, playing at the end of a wire when the opponent had just played at its starting point, choosing which wire starting point s or t to block when the opponent had just play in u (Fig. 4.12a), which forces them to take the other wire, and playing in the exit s of a $(2,1)$-vertex gadget when the opponent had just play in u or in v (Fig. 4.11a). In order to conclude by invoking Lemma 34 up to Corollary 14, we should show that making a threat is not helpful in all the above situations. Note that those Lemmas imply the following invariant: while White and Black play a legal game of GG, at their turn, a player is threatened or their opponent can initiate a winning sequence of threats. There is only two kinds of places where one can play a threat: the crossroad gadget (Fig. 4.10c) and the $(2,1)$-vertex gadget while already being entered (Fig. 4.11b).

Let us start by showing that playing a threat in a crossroad gadget which does not defend a threat, that is, the action was occurring in a different place, is losing. If White plays in s then Black plays in t which is the starting point of a BW wire. And now, they are at least two places where Black can initiate a winning sequence of threats, so White loses (after possibly playing some additional but harmless threats). The same holds by reversing the colors or by reversing s and t, and is not affected by whether or not stones have been played in u, x, y and v. If, instead, White plays in u, Black answers in x, White answers in y and Black plays in v, and again Black can initiate a winning sequence of threats in two places. If, instead, White plays in x, Black answers in u and again White is losing. If, instead, Black plays in x, White plays in y and Black plays in v, and now White plays their winning sequence of threats.

Now, let us show that the threats in the already entered $(2,1)$-vertex gadget are harmless. Consider now Fig. 4.11b. If Black plays in b, White answers in a and there is no more threats for Black. If Black plays in a, White answers in b. Black can threat again in c or d but White defends in d or c, respectively, and there are no more threats. Note that this does not affect the fact that reentering in the gadget is losing for Black. Summing up, White and Black has to simulate a proper game of GG in the instance (G, v_0).

We now show that if a player in the game of HAVANNAH has no more move in the corresponding GG instance, they lose. The only non cheating move would be to reenter in a $(2,1)$-vertex but it is losing by Lemma 38.

□

4.6 TwixT

Alex Randolph's TWIXT is one of the most popular connection games. It was invented around 1960 and was marketed as soon as in 1962 [80]. In his book devoted to connection games, Cameron Browne describes TWIXT as one of the most popular and widely marketed of all connection games [25]. We now briefly describe the rules of TWIXT and refer to Moesker's master's thesis for an introduction and a mathematical approach to the strategy, and the description of a possible implementation [106].

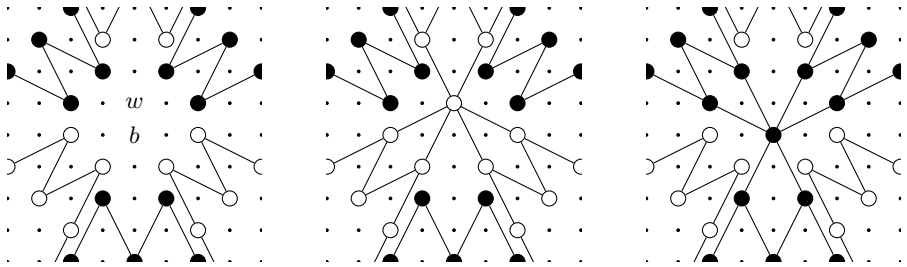
TWIXT is a 2-player connection game played on a GO-like board. At their turn, player White and Black place a pawn of their color in an unoccupied place. Just as in HAVANNAH and HEX, pawns cannot be taken, moved, nor removed. When 2 pawns of the same color are spaced by a knight's move, they are linked by an edge of their color, unless this edge would cross another edge. At each turn, a player can remove some of their edges to allow for new links. The goal for player White (resp. Black) is to link top and bottom (resp. left and right) sides of the board. Note that sometimes, a player could have to choose between two possible edges that intersect each other. The *pencil and paper* version TWIXTPP where the edges of a same color are allowed to cross is also famous and played online.

As the length of a game of TWIXT is polynomially bounded, exploring the whole tree can be done with polynomial space using a minimax algorithm. Therefore TWIXT is in **PSPACE**.

Mazzoni and Watkins have shown that 3-SAT could be reduced to single-player TWIXT, thus showing **NP**-completeness of the variant [104]. While it might be possible to try and adapt their work and obtain a reduction from 3-QBF to standard two-player TWIXT, we propose a simpler approach based on HEX. The **PSPACE**-completeness of HEX has already been used to show the **PSPACE**-completeness of AMAZONS, a well-known territory game [69].

We now present how we construct from an instance G of HEX an instance $\phi(G)$ of TWIXT. We can represent a cell of HEX by the 9×9 TWIXT gadgets displayed in Fig. 4.14. Let n be the size of a side of G, Fig. 4.15 shows how a TWIXT board can be paved by n^2 TWIXT cell gadgets to create a HEX board.

It is not hard to see from Fig. 4.14a that in each gadget of Fig. 4.15, move w (resp. b) is dominating for White (resp. Black). That is, playing w is as good for White as any other move of the gadget. We can also see that the moves that are not part of any gadget in Fig. 4.15 are dominated for both players. As a result, if player Black



(a) Empty cell. (b) White cell. (c) Black cell.

Figure 4.14: Basic gadgets needed to represent cells.

(resp. White) has a winning strategy in G, then player Black has a winning strategy in $\phi(G)$. Thus, G is won by Black if and only if $\phi(G)$ is won by Black. Therefore determining the winner in TWIXT is at least as hard as in HEX, leading to the desired result.

Theorem 28. TWIXT *is* ***PSPACE****-complete.*

Proof. We already mentioned that TWIXT $\in$ **PSPACE**. We presented a polynomial time reduction from a **PSPACE**-complete problem.

□

Observe that the proposed reduction holds both for the classic version of TWIXT as well as for the *pencil and paper* version TWIXTPP. Indeed, the reduction does not require the losing player to remove any edge, so it also proves that TWIXTPP is **PSPACE**-hard.

4.7 Conclusions and Perspectives

4.7.1 Trick-Taking Card Games

In his thesis, Hearn proposed the following explanation to the standing lack of hardness result for BRIDGE [2006, p122].

There is no natural geometric structure to exploit in BRIDGE *as there is in a typical board game.*

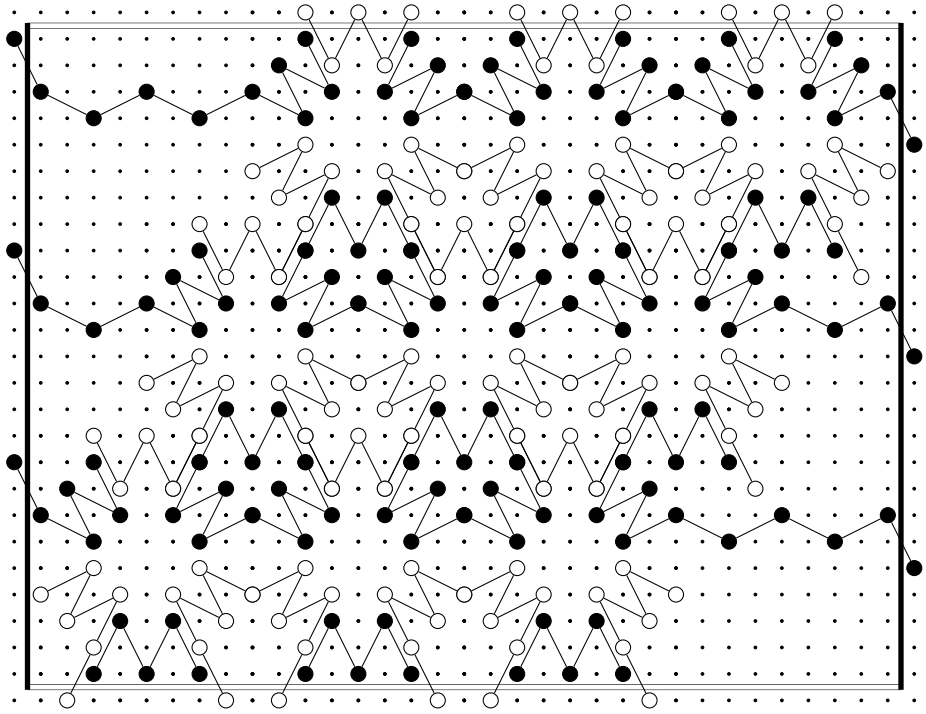


Figure 4.15: Empty 3×3 HEX board reduced to a TWIXT board.

Theorem 26 achieves a significant milestone in that respect. The gadgets in the reduction indeed show that it is possible to find a graphical structure within the suits. From all, the attacking and counter-attacking gadgets stand as the central idea, giving an adjacency list structure to suits, by means of a precise race to establishment. Termination gadgets make those races decisive.

Finding a **PSPACE**-hardness proof necessitating only 2 hands per team is very appealing. Another interesting problem is to find a hardness proof with a bounded number of suits.

Many actual trick taking card games also feature a *trump suit* and potentially different values for tricks based on which cards were involved. Such a setting can be seen as a direct generalization of ours, but remains bounded. Therefore our **PSPACE**-

completeness results carry over to point-based trick taking card games involving trumps.

4.7.2 Connection Games

We have established the **PSPACE**-hardness of HAVANNAH and TWIXT but the complexity of other notable connection games remains open. In particular, the following games seem to be good candidates for future work on the complexity of connection games.

In LINES OF ACTION, each player starts with two groups of pieces and tries to connect all their pieces by moving these pieces and possibly capturing opponent pieces [128]. While the goal of LINES OF ACTION clearly makes it a connection game, the mechanics distinguishes it from more classical connection games as no pieces are added to the board and existing pieces can be moved or removed. As a result, it is not even clear that LINES OF ACTION is in **PSPACE**.

SLITHER is closer to HEX but each move actually consists of putting a new stone on the board and possibly moving another one. Obtaining a **PSPACE**-hardness result for SLITHER is not so easy since the rules allow a player to influence two different areas of the board in a single turn.

5 Conclusion

The second chapter was mainly devoted to the *greediness-for-parameterization* technique. We hope that this technique may be useful for other problems than the ones tackled in that chapter. We are also glad that the formalism of local graph partitioning problems (LGPPs) have been adopted in another work [117]. This framework is a good way of generalizing algorithms or hardness results to the whole class of LGPPs. It might constitute an economy of energy, in the sense that it prevents us from showing a result for a specific local graph partitioning problem and observing after some time that the result, in fact, extends to many *similar* problems.

Another key idea concerning cardinality-constrained graph problems is the bound on the range of possible degrees in the subset of vertices from which an optimal solution has to be built (see Lemma 5 and Lemma 6). We are currently investigated the parameterized complexity of cardinality-constrained problems in other classes than bipartite graphs, and this observation proves quite useful.

The first half of the third chapter was a step towards tight inapproximability results within superpolynomial time (subexponential time and FPT time). The main result of [37] constitutes a major breakthrough in that direction. Indeed, it yields an almost tight result between the lower bound and the upper bound of approximating MAX INDEPENDENT SET in subexponential time. The second half of that chapter can be seen as a first attempt to extend this tight result to other optimization problems. Again, we hope that the framework of *approximation preserving sparsifiers* will serve in the near future.

Concerning the fourth chapter on the complexity of games, we would like to address some last observations. We want to give some additional motivations in studying the

complexity of natural fragments of trick taking card games. Usually (and it is almost always the case for board games) one can classify the complexity of an interesting game with perfect and complete information by asking: 1) Is the game singled-player? 2) Is the length of the games polynomially bounded? Then, if one gets two yeses, the game is likely to be **NP**-complete, if one gets exactly one yes, then the game is likely to be **PSPACE**-complete, and **EXP**-complete in case of two noes.

In fact, that only proves membership but the lower bound often matches it. Now, there are some fragments of trick taking card games where it is even hard to have an intuition. For instance, if both teams are limited to a single hand (that is $\mathcal{B}(\mathcal{L}_1, \_, \_)$), it is not clear if there is a polynomial time algorithm or if the problem remains **PSPACE**-complete. Indeed, it is close to the polynomial fragment $\mathcal{B}(\mathcal{L}_1, \_, \_))$ restricted to symmetric hands (i.e., in each suit the two players have the same number of cards) [127]. But, it seems also close to the **PSPACE**-hardness result for $\mathcal{B}(\mathcal{L}_3, \_, \_))$ of Theorem 26, since of the three hands of each player, only one is really significant.

Finally, we list some open questions inspired by this thesis:

- What is the complexity of MIN k-VERTEX COVER with respect to the value of the solution p?
- In general, is there a unifying view of a superset of local graph partitioning problems or another set of problems which may be useful to produce generic results (and not to have to write s algorithms for s problems)?
- Can we establish the parameterized complexity with respect to the size of the solution of each LGPP in chordal graphs?
- Can we use the trick of Lemma 5 and Lemma 6 to get FPT results for other classes of graphs?
- What is the parameterized complexity of MAX SAT-k with respect to $k + f$ where k is the number of variables that we can set to true, and f is the maximum number of occurrences among all the variables.
- Can we establish a tight result on the approximability in subexponential time status of MIN INDEPENDENT DOMINATING SET?
- Same question for other minimization graphs problems.

- Can we design an approximation preserving sparsifier for SAT?
- Can we adapt the proof of Theorem 26 to show that $\mathcal{B}(\mathcal{L}_2, \_, \_)$ is **PSPACE**-hard?
- What is the complexity of $\mathcal{B}(\mathcal{L}_1, \_, \_)$? Can we venture a guess?
- What is the complexity of Lines of Actions?

6 Appendix

Decision Problems

CLIQUE

Input: A graph $G = (V, E)$ and an integer k.

Question: Does G contain a clique of size (at least) k? (i.e., a subset of vertices in V which are pairwise adjacent)

MAX c-CSP ABOVE AVERAGE

Input: A collection of m boolean functions on the variable set V, where each function depends on at most c variables, and a nonnegative integer k.

Question: Does there exist a boolean assignment of V that satisfies at least $\rho \cdot m$ functions, where ρ is the average fraction of functions satisfied by a uniform random assignment?

INDEPENDENT SET

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

Question: Does G contain an independent set of size (at least) k? (i.e., a set of vertices which are pairwise nonadjacent)

PRIME

Input: An integer n.

Question: Is n prime?

QBF

Input: A quantified CNF formula ϕ on the variable set V.

Question: Does there exist a truth assignment of the free variables in V satisfying ϕ?

2SAT

Input: A 2-CNF formula ϕ on the variable set V.

Question: Does there exist a truth assignment of V satisfying all clauses of ϕ?

3SAT

Input: A 3-CNF formula ϕ on the variable set V.

Question: Does there exist a truth assignment of V satisfying all clauses of ϕ?

kSat

Input: A k-CNF formula ϕ on the variable set V.

Question: Does there exist a truth assignment of V satisfying all clauses of ϕ?

SAT

Input: A CNF formula ϕ on the variable set V.

Question: Does there exist a truth assignment of V satisfying all clauses of ϕ?

SAT-k

Input: A CNF formula ϕ on the variable set V, and an integer k.

Goal: Is there a truth assignement of V, setting at most k variables to true, satisfying all the clauses?

Optimization Problems

GENERALIZED k-DENSEST

Input: A graph $G = (V, E)$, a subset $V' \subseteq V$, and an integer k.

Goal: Find a set $S \supseteq V'$ of k vertices maximizing the number $|E(S)|$ of inner edges.

GENERALIZED k-SPARSEST

Input: A graph $G = (V, E)$, a subset $V' \subseteq V$, and an integer k.

Goal: Find a set $S \supseteq V'$ of k vertices minimizing the number $|E(S)|$ of inner edges.

GENERALIZED MAX $(k, n-k)$-CUT

Input: A graph $G = (V, E)$, a subset $V' \subseteq V$, and an integer k.

Goal: Find a set $S \supseteq V'$ of k vertices maximizing the number $|E(S, V \setminus S)|$ of edges in the cut.

GENERALIZED MIN DOMINATING SET

Input: A graph $G = (V, E)$ with a partition $V = (V_1, V_2, V_3)$ (some of the sets being possibly empty).

Goal: Find a smallest set of vertices $V' \subseteq V_1 \cup V_2$ which dominate all vertices in $V_2 \cup V_3$.

GENERALIZED MIN $(k, n-k)$-CUT

Input: A graph $G = (V, E)$, a subset $V' \subseteq V$, and an integer k.

Goal: Find a set $S \supseteq V'$ of k vertices minimizing the number $|E(S, V \setminus S)|$ of edges in the cut.

k-DENSEST

Input: A graph $G = (V, E)$, and an integer k.

Goal: Find a set S of k vertices maximizing the number $|E(S)|$ of inner edges.

k-SPARSEST

Input: A graph $G = (V, E)$, and an integer k.

Goal: Find a set S of k vertices minimizing the number $|E(S)|$ of inner edges.

LOCAL GRAPH PARTITIONING PROBLEM

Input: A graph $G = (V, E)$, two reals α_1 and α_2, an objective opt $\in \{\min, \max\}$, and an integer k.

Goal: Find a set S of k vertices opt-imizing the number $\alpha_1|E(S)| + \alpha_2|E(S, V \setminus S)|$.

MAX CLIQUE

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

Goal: Find the clique of maximum size, i.e., where a clique is a subset of vertices in V which are pairwise adjacent.

MAX COMPLETE BIPARTITE SUBGRAPH

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

Goal: Find an induced bipartite subgraph of G containing a maximum number of vertices.

MAX c-CSP

Input: A collection of m boolean functions on the variable set V, where each function depends on at most c variables.

Goal: Find a boolean assignment of V that satisfies a maximum number of equations.

MAX CUT

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

Goal: Find a set $S \subseteq V$ such that the number of edges having exactly one endpoint in S is maximized.

MAX INDEPENDENT SET

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

Goal: Find a largest independent set in G, i.e., a set of vertices which are pairwise nonadjacent.

MAX INDUCED PLANAR SUBGRAPH

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

Goal: Find a largest subset $S \subseteq V$, such that $G[S]$ is planar.

MAX k-DOMINATING SET

Input: A graph $G = (V, E)$ and an integer k.

Goal: Find a set S of k vertices maximizing the number $|N(S)|$ of vertices dominated by S.

MAX $(k, n-k)$-CUT

Input: A graph $G = (V, E)$ and an integer k.

Goal: Find a set S of k vertices maximizing the number $|E(S, V \setminus S)|$ of edges in the cut.

MAX kSAT

Input: A CNF formula ϕ on the variable set V containing at most k literals per clause.

Goal: Find a truth assignement of V that satisfies a maximum number of clauses.

MAX k-SET COVER

Input: A universe X, a collection $\mathcal{S}$ of subsets of X, and an integer k.

Goal: Find k subsets in $\mathcal{S}$ whose union has the maximum cardinality.

MAX k-VERTEX COVER

Input: A graph $G = (V, E)$ and an integer k.

Goal: Find a set S of k vertices maximizing the number $|E(S)| + |E(S, V \setminus S)|$ of edges covered by S.

MAX ℓ-COLORABLE INDUCED SUBGRAPH

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

Goal: Find a largest subset $S \subseteq V$ such that $G[S]$ is ℓ-colorable.

MAX MINIMAL VERTEX COVER

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

Goal: Find a largest subset $S \subseteq V$ such that S is a minimal vertex cover, that is, S is a vertex cover and for any $T \subsetneq S$, T is not a vertex cover.

MAX SAT

Input: A CNF formula ϕ on the variable set V.

Goal: Find a truth assignement of V that satisfies a maximum number of clauses.

MAX SAT-k

Input: A CNF formula ϕ on the variable set V, and an integer k.

Goal: Find a truth assignement of V, setting at most k variables to true, that satisfies a maximum number of clauses.

MAX UNUSED SETS

Input: A universe X, a collection $\mathcal{S}$ of subsets of X.

Goal: Find a set $\mathcal{T} \subseteq \mathcal{S}$ of maximum size, such that $\mathcal{S} \setminus \mathcal{T}$ covers X.

Max 2Sat

Input: A CNF formula ϕ on the variable set V containing at most 2 literals per clause.

Goal: Find a truth assignement of V that satisfies a maximum number of clauses.

MAX 3LIN

Input: A system $Az = b$ of linear equations in the variable set V over $\mathbb{F}_2$, each equation involving exactly 3 variables.

Goal: Find an assignment of values to V satisfying a maximum number of equations.

Max 3Sat

Input: A CNF formula ϕ on the variable set V containing at most 3 literals per clause.

Goal: Find a truth assignement of V that satisfies a maximum number of clauses.

MIN BISECTION

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

Goal: Find a partition (V_1, V_2) of V such that $||V_1| - |V_2|| \leqslant 1$, minimizing the number $|E(V_1, V_2)|$ of edges in the cut.

MIN COLORING

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

Goal: Find a proper (vertex-)coloring of G, i.e., a coloring where no adjacent vertices get the same color, using a smallest number of colors.

MIN CUT

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

Goal: Find a set $S \subseteq V$ such that the number of edges having exactly one endpoint in S is minimized.

MIN DOMINATING SET

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

Goal: Find a smallest dominating set in G, i.e., a set of vertices S such that every vertex in $V \setminus S$ has a neighbor in S.

MIN FEEDBACK ARC SET

Input: A directed graph $G = (V, A)$.

Goal: Find a smallest feedback arc set in G, i.e., a set of arcs F such that $(V, A \setminus F)$ is acyclic.

MIN FEEDBACK VERTEX SET

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

Goal: Find a smallest feedback vertex set in G, i.e., a set of vertices S such that $G[V \setminus S]$ is a forest.

MIN HITTING SET

Input: A universe X and a collection $\mathcal{S}$ of subsets of X.

Goal: Find a smallest subset of elements $C \subseteq X$ such that C intersects every subset in $\mathcal{S}$.

MIN INDEPENDENT DOMINATING SET

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

Goal: Find a smallest set of vertices which is simultaneously an independent set and a dominating set in G.

MIN k-DOMINATING SET

Input: A graph $G = (V, E)$ and an integer k.

Goal: Find a set S of k vertices minimizing the number $|N(S)|$ of vertices dominated by S.

MIN $(k, n-k)$-CUT

Input: A graph $G = (V, E)$ and an integer k.

Goal: Find a set S of k vertices minimizing the number $|E(S, V \setminus S)|$ of edges in the cut.

MIN k-VERTEX COVER

Input: A graph $G = (V, E)$ and an integer k.

Goal: Find a set S of k vertices minimizing the number $|E(S)| + |E(S, V \setminus S)|$ of edges covered by S.

MIN SAT

Input: A CNF ϕ on the variable set V.

Goal: Find a truth assignement of V that satisfies a minimum number of clauses.

MIN SET COVER

Input: A universe X and a collection $\mathcal{S}$ of subsets of X.

Goal: Find a minimum number of sets from $\mathcal{S}$ whose union is X.

MIN VERTEX COVER

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

Goal: Find a smallest vertex cover of G, i.e., a set C of vertices such that for every $e = \{u, v\} \in E$, $C \cap \{u, v\} \neq \emptyset$.

SET PACKING

Input: A universe X and a collection $\mathcal{S}$ of subsets of X.

Goal: Find a maximum number of sets from $\mathcal{S}$ which are pairwise disjoint.

Bibliography

[1] Alexander A. Ageev and Maxim Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In *IPCO*, pages 17–30, 1999.

[2] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. PRIMES is in P. *Ann. of Math*, 2:781–793, 2002.

[3] Noga Alon, Raphael Yuster, and Uri Zwick. Color coding. In *Encyclopedia of Algorithms*. 2008.

[4] Vadim V. Anshelevich. A hierarchical approach to computer Hex. *Artificial Intelligence*, 134(1-2):101–120, 2002.

[5] Nicola Apollonio and Bruno Simeone. The maximum vertex coverage problem on bipartite graphs. *Discrete Applied Mathematics*, 165(0):37 – 48, 2014. ISSN 0166-218X. doi: http://dx.doi.org/10.1016/j.dam.2013.05.015. URL `http://www.sciencedirect.com/science/article/pii/S0166218X13002448`. 10th Cologne/Twente Workshop on Graphs and Combinatorial Optimization (CTW 2011).

[6] Broderick Arneson, Ryan B. Hayward, and Philip Henderson. Monte Carlo tree search in Hex. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):251–258, 2010.

[7] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and intractability of approximation problems. *J. Assoc. Comput. Mach.*, 45(3):501–555, 1998.

[8] Bengt Aspvall, Michael F. Plass, and Robert E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8 (3):121–123, 1979.

[9] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. *Complexity and approximation. Combinatorial optimization problems and their approximability properties*. Springer-Verlag, Berlin, 1999.

[10] Claude Berge. *Graphs and Hypergraphs*. Elsevier Science Ltd., Oxford, UK, UK, 1985. ISBN 0720404797.

[11] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2):546–563, 2009.

[12] Yngvi Björnsson, Ryan Hayward, Michael Johanson, and Jack van Rijswijck. Dead cell analysis in Hex and the Shannon game. In Adrian Bondy, Jean Fonlupt, Jean-Luc Fouquet, Jean-Claude Fournier, and Jorge L. Ramírez Alfonsín, editors, *Graph Theory in Paris*, pages 45–59. Springer, 2007.

[13] Markus Bläser. Computing small partial coverings. *Inform. Process. Lett.*, 85 (6):327–331, 2003.

[14] Édouard Bonnet and Vangelis Th. Paschos. Sparsification and subexponential approximation. *CoRR*, abs/1402.2843, 2014.

[15] Édouard Bonnet and Vangelis Th. Paschos. Parameterized (in)approximability of subset problems. *Oper. Res. Lett.*, 42(3):222–225, 2014.

[16] Édouard Bonnet, Bruno Escoffier, Eun Jung Kim, and Vangelis Th. Paschos. On subexponential and fpt-time inapproximability. In *IPEC*, pages 54–65, 2013.

[17] Édouard Bonnet, Bruno Escoffier, Vangelis Th. Paschos, and Émeric Tourniaire. Multi-parameter complexity analysis for constrained size graph problems: Using greediness for parameterization. In *IPEC*, pages 66–77, 2013.

[18] Édouard Bonnet, Florian Jamain, and Abdallah Saffidine. Havannah and TwixT are PSPACE-complete. In *Computers and Games*, pages 175–186, 2013.

[19] Édouard Bonnet, Vangelis Th. Paschos, and Florian Sikora. Multiparameterizations for max k-set cover and related satisfiability problems. *CoRR*, abs/1309.4718, 2013.

[20] Édouard Bonnet, Florian Jamain, and Abdallah Saffidine. On the complexity of trick-taking card games. In Francesca Rossi, editor, *23rd International Joint Conference on Artificial Intelligence (IJCAI)*, Beijing, China, August 2013. AAAI Press.

[21] Nicolas Boria, Federico Della Croce, and Vangelis Th. Paschos. On the max min vertex cover problem. In *Proc. WAOA'13*, Lecture Notes in Computer Science. Springer-Verlag, 2013.

[22] Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Efficient approximation by "low-complexity" exponential algorithms. Cahier du LAMSADE 271, LAMSADE, Université Paris-Dauphine, December 2007. Available at `http://www.lamsade.dauphine.fr/cahiers/PDF/cahierLamsade271.pdf`.

[23] Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Efficient approximation of MIN COLORING by moderately exponential algorithms. *Inform. Process. Lett.*, 109(16):950–954, 2009.

[24] Nicolas Bourgeois, Bruno Escoffier, and Vangelis Th. Paschos. Approximation of MAX INDEPENDENT SET, MIN VERTEX COVER and related problems by moderately exponential algorithms. *Discrete Appl. Math.*, 159(17):1954–1970, 2011.

[25] Cameron Browne. *Connection Games: Variations on a Theme*. AK Peters, 2005.

[26] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, March 2012. ISSN 1943-068X. doi: 10.1109/TCIAIG.2012.2186810.

[27] John Bruno and Louis Weinberg. A constructive graph-theoretic solution of the Shannon switching game. *Circuit Theory, IEEE Transactions on*, 17(1):74–81, 1970.

[28] Michael Buro, Jeffrey R. Long, Timothy Furtak, and Nathan Sturtevant. Improving state evaluation, inference, and search in trick-based card games. In *21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009.

[29] Leizhen Cai. Parameter complexity of cardinality constrained optimization problems. *The Computer Journal*, 51:102–121, 2008.

[30] Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation: A new method for solving fixed-cardinality optimization problems. In *IWPEC*, pages 239–250, 2006.

[31] Liming Cai and Jianer Chen. On fixed-parameter tractability and approximability of NP optimization problems. *J. Comput. Syst. Sci.*, 54(3):465–474, 1997.

[32] Liming Cai and Xiuzhen Huang. Fixed-parameter approximation: Conceptual framework and approximability results. *Algorithmica*, 57(2):398–412, 2010.

[33] Liming Cai and David W. Juedes. On the existence of subexponential parameterized algorithms. *J. Comput. Syst. Sci.*, 67(4):789–807, 2003.

[34] Bugra Caskurlu and K. Subramani. On partial vertex cover on bipartite graphs and trees. *CoRR*, abs/1304.5934, 2013.

[35] Kevin Cattell and Michael J. Dinneen. A characterization of graphs with vertex cover up to five. In *ORDAL*, pages 86–99, 1994.

[36] Marco Cesati. The turing way to the parameterized intractability, 2001.

[37] Parinya Chalermsook, Bundit Laekhanukit, and Danupon Nanongkai. Independent set, induced matching, and pricing: Connections and tight (subexponential time) approximation hardnesses. In *FOCS*, pages 370–379, 2013.

[38] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *J. Comput. Syst. Sci.*, 72(8):1346–1367, 2006.

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

[40] Yijia Chen, Martin Grohe, and Magdalena Grüber. On parameterized approximability. *Electronic Colloquium on Computational Complexity (ECCC)*, 14 (106), 2007.

[41] Rajesh Hemant Chitnis, MohammadTaghi Hajiaghayi, and Guy Kortsarz. Fixed-parameter and approximation algorithms: A new look. In *IPEC*, pages 110–122, 2013.

[42] Alonzo Church. A note on the entscheidungsproblem. *J. Symb. Log.*, 1(1):40–41, 1936.

[43] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[44] Derek G. Corneil and Yehoshua Perl. Clustering and domination in perfect graphs. *Discrete Appl. Math.*, 9:27–39, 1984.

[45] Marek Cygan and Marcin Pilipczuk. Exact and approximate bandwidth. *Theoret. Comput. Sci.*, 411(40–42):3701–3713, 2010.

[46] Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed parameter tractable. *CoRR*, abs/1311.2563, 2013.

[47] Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: algorithmic combinatorial game theory. In Richard J. Nowakowski, editor, *Games of No Chance III*, pages 3–56. Cambridge University Press, 2009.

[48] Erik D. Demaine, Martin Demaine, Ryuhei Uehara, Takeaki Uno, and Yushi Uno. Uno is hard, even for a single player. In *Fun with Algorithms*, pages 133–144. Springer, 2010.

[49] Irit Dinur. The PCP theorem by gap amplification. *J. Assoc. Comput. Mach.*, 54 (3), 2007.

[50] Irit Dinur and Shmuel Safra. The importance of being biased. In *STOC*, pages 33–42, 2002.

[51] Rodney G. Downey and Michael R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer, New York, 1999.

[52] Rodney G. Downey, Vladimir Estivill-Castro, Michael R. Fellows, Elena Prieto, and Frances A. Rosamond. Cutting up is hard to do: The parameterized complexity of k-cut and related problems. In *Electronic Notes in Theoretical Computer Science 78*, pages 205–218. Elsevier Science Publishers, 2003.

[53] Rodney G. Downey, Michael R. Fellows, and Catherine McCartin. Parameterized approximation problems. In H. L. Bodlaender and M. A. Langston, editors, *Proc. International Workshop on Parameterized and Exact Computation, IWPEC'06*, volume 4169 of *Lecture Notes in Computer Science*, pages 121–129. Springer-Verlag, 2006.

[54] Rodney G. Downey, Michael R. Fellows, Catherine McCartin, and Frances A. Rosamond. Parameterized approximation of dominating set problems. *Inf. Process. Lett.*, 109(1):68–70, 2008.

[55] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17: 449–467, February 1965. doi: 10.4153/CJM-1965-045-4.

[56] Shimon Even and Robert E. Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the ACM (JACM)*, 23(4):710–719, 1976.

[57] Timo Ewalds. Playing and solving Havannah. Master's thesis, University of Alberta, 2012.

[58] U. Feige. A threshold of $\ln n$ for approximating set cover. *J. Assoc. Comput. Mach.*, 45:634–652, 1998.

[59] Uriel Feige and Michael Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *J. Algorithms*, 41(2):174–211, 2001.

[60] Uriel Feige, Robert Krauthgamer, and Kobbi Nissim. On cutting a few vertices from a graph. *Discrete Applied Mathematics*, 127:2003, 2001.

[61] Michael R Fellows. On the complexity of vertex set problems. Technical report, Technical report, Computer Science Department, University of New Mexico, 1988.

[62] Michael R. Fellows and Michael A. Langston. Nonconstructive advances in polynomial-time complexity. *Inf. Process. Lett.*, 26(3):155–162, 1987.

[63] Fedor V. Fomin, Dieter Kratsch, and Gerhard J. Woeginger. Exact (exponential) algorithms for the dominating set problem. In *WG*, pages 245–256, 2004.

[64] Fedor V. Fomin, Petr A. Golovach, and Janne H. Korhonen. On the parameterized complexity of cutting a few vertices from a graph. In *MFCS*, pages 421–432, 2013.

[65] Aviezri S. Fraenkel and David Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. *Journal of Combinatorial Theory, Series A*, 31(2):199–214, 1981.

[66] Ian Frank and David Basin. Search in games with incomplete information: A case study using bridge card play. *Artificial Intelligence*, 100(1):87–123, 1998.

[67] Ian Frank and David Basin. A theoretical and empirical investigation of search in imperfect information games. *Theoretical Computer Science*, 252(1):217–256, 2001.

[68] Martin Fürer, Serge Gaspers, and Shiva Prasad Kasiviswanathan. An exponential time 2-approximation algorithm for bandwidth. *Theor. Comput. Sci.*, 511:23–31, 2013.

[69] Timothy Furtak, Masashi Kiyomi, Takeaki Uno, and Michael Buro. Generalized Amazons is PSPACE-complete. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *19th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 132–137, 2005.

[70] Ofer Gabber and Zvi Galil. Explicit constructions of linear-sized superconcentrators. *J. Comput. Syst. Sci.*, 22(3):407–420, 1981.

[71] Michael R. Garey and David S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[72] Matthew L. Ginsberg. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research (JAIR)*, 14:303–358, 2001.

[73] Jiong Guo, Iyad A. Kanj, and Stefan Kratsch. Safe approximation and its relation to kernelization. In *IPEC*, pages 169–180, 2011.

[74] MohammadTaghi Hajiaghayi, Rohit Khandekar, and Guy Kortsarz. The foundations of fixed parameter inapproximability. *CoRR*, abs/1310.2711, 2013.

[75] Magnús M. Halldórsson. Approximating the minimum maximal independence number. *Inform. Process. Lett.*, 46:169–172, 1993.

[76] Robert A. Hearn. *Games, Puzzles, and Computation*. PhD thesis, Massachusetts Institute of Technology, 2006.

[77] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A K Peters, July 2009.

[78] Dorit S. Hochbaum, editor. *Approximation algorithms for NP-hard problems*. PWS, Boston, 1997.

[79] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S*, 43:439–561, 2006.

[80] James R. Huber. Computer game playing: the game of TwixT. Master's thesis, Rochester Institute of Technology, 1983.

[81] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

[82] Robert W. Irving. On approximating the minimum independent dominating set. *Inf. Process. Lett.*, 37(4):197–200, 1991.

[83] Shigeki Iwata and Takumi Kasai. The othello game on an $n \times n$ board is pspace-complete. *Theoretical Computer Science*, 123(2):329–340, 1994.

[84] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974.

[85] Gwenaël Joret and Adrian Vetta. Reducing the rank of a matroid. *CoRR*, abs/1211.4853, 2012.

[86] Jeff Kahn, Jeffrey C. Lagarias, and Hans S. Witsenhausen. Single-suit two-person card play. *International Journal of Game Theory*, 16(4):291–320, 1987.

[87] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of computer computations*, pages 85–103. Plenum Press, New York, 1972.

[88] Subash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\varepsilon$. *J. Comput. System Sci.*, 74(3):335–349, 2008.

[89] Subhash Khot. On the unique games conjecture (invited survey). In *IEEE Conference on Computational Complexity*, pages 99–121, 2010.

[90] Stefan Kiefer. Die Menge der virtuellen Verbindungen im Spiel Hex ist PSPACE-vollständig. Studienarbeit Nr. 1887, Universität Stuttgart, July 2003.

[91] Eun Jung Kim and Ryan Williams. Improved parameterized algorithms for above average constraint satisfaction. In *IPEC*, pages 118–131, 2011.

[92] Christian Komusiewicz and Manuel Sorge. Finding dense subgraphs of sparse graphs. In *IPEC*, pages 242–251, 2012.

[93] Sebastian Kupferschmid and Malte Helmert. A skat player based on Monte-Carlo simulation. In H.Jaap van den Herik, Paolo Ciancarini, and H.H.L.M.(Jeroen) Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 135–147. Springer-Verlag, Berlin Heidelberg, 2006.

[94] David N. L. Levy. The million pound bridge program. In *Heuristic Programming in Artificial Intelligence: The First Computer Olympiad*, pages 95–103. Ellis Horwood, 1989.

[95] David Lichtenstein and Michael Sipser. Go is polynomial-space hard. *Journal of the ACM (JACM)*, 27(2):393–401, 1980.

[96] Jeffrey Long, Nathan R. Sturtevant, Michael Buro, and Timothy Furtak. Understanding the success of perfect information Monte Carlo sampling in game tree search. In *24th AAAI Conference on Artificial Intelligence (AAAI)*, pages 134–140, 2010.

[97] Richard J. Lorentz. Improving Monte-Carlo tree search in Havannah. In *Computers and Games*, pages 105–115, 2010.

[98] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. Assoc. Comput. Mach.*, 41(5):960–981, 1994.

[99] Mitja Luštrek, Matjaž Gams, and Ivan Bratko. A program for playing tarok. *ICGA Journal*, 23(3):190–197, 2003.

[100] Thomas Maarup. Hex: Everything you always wanted to know about Hex but were afraid to ask. Master's thesis, Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark, 2005.

[101] Madhav V. Marathe and S. S. Ravi. On approximation algorithms for the minimum satisfiability problem. *Inf. Process. Lett.*, 58(1):23–29, 1996.

[102] Daniel Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1):60–78, 2008.

[103] Luke Mathieson. A proof checking view of parameterized complexity. *CoRR*, abs/1206.2436, 2012.

[104] Dominic Mazzoni and Kevin Watkins. Uncrossed knight paths is NP-complete. 1997.

[105] Kurt Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, volume 2 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984.

[106] Kevin Moesker. TwixT: theory, analysis and implementation. Master's thesis, Maastricht University, 2009.

[107] Dana Moshkovitz and Ran Raz. Two-query PCP with subconstant error. *J. ACM*, 57(5), 2010.

[108] Rolf Niedermeier. Ubiquitous parameterization - invitation to fixed-parameter algorithms. In *MFCS*, pages 84–103, 2004.

[109] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation and complexity classes. *J. Comput. System Sci.*, 43:425–440, 1991.

[110] Azaria Paz and Shlomo Moran. Non deterministic polynomial optimization problems and their approximations. *Theoret. Comput. Sci.*, 15:251–277, 1981.

[111] Venkatesh Raman and Saket Saurabh. Short cycles make W-hard problems hard: Fpt algorithms for W-hard problems in graphs with no short cycles. *Algorithmica*, 52(2):203–225, 2008.

[112] Omer Reingold, Salil Vadhan, and Avi Wigderson. Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In *Annals of Mathematics*, pages 157–187, 2000.

[113] Stefan Reisch. Hex ist PSPACE-vollständig. *Acta Informatica*, 15:167–191, 1981.

[114] John M. Robson. The complexity of Go. In *IFIP*, pages 413–417, 1983.

[115] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.

[116] Thomas J. Schaefer. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences*, 16(2):185–225, 1978. ISSN 0022-0000. doi: 10.1016/0022-0000(78)90045-4.

[117] Hadas Shachnai and Meirav Zehavi. Parameterized algorithms for graph partitioning problems. *CoRR*, abs/1403.0099, 2014.

[118] Hans-Ulrich Simon. On approximate solutions for combinatorial optimization problems. *SIAM J. Disc. Math.*, 3(2):294–310, 1990.

[119] Christian Sloper and Jan A. Telle. An overview of techniques for designing parameterized algorithms. *Comput. J.*, 51(1):122–136, 2008.

[120] Jan A. Stankiewicz, Mark H. M. Winands, and Jos W. H. M. Uiterwijk. Monte-carlo tree search enhancements for havannah. In *ACG*, pages 60–71, 2011.

[121] Nathan R. Sturtevant and Adam M. White. Feature construction for reinforcement learning in hearts. In H.Jaap Herik, Paolo Ciancarini, and H.H.L.M.(Jeroen) Donkers, editors, *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, pages 122–134. Springer, Berlin Heidelberg, 2006. ISBN 978-3-540-75537-1. doi: 10.1007/978-3-540-75538-8_11.

[122] Fabien Teytaud and Olivier Teytaud. Creating an upper-confidence-tree program for Havannah. In H. Jaap van den Herik and Pieter Spronck, editors, *Advances in Computer Games*, volume 6048 of *Lecture Notes in Computer Science*, pages 65–74. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12992-6. doi: 10.1007/978-3-642-12993-3_7.

[123] Alan M. Turing. Computability and lambda-definability. *J. Symb. Log.*, 2(4): 153–163, 1937.

[124] Jos Uiterwijk and Kevin Moesker. Mathematical modelling in Twixt. In Benedikt Löwe, editor, *Logic and the Simulation of Interaction and Reasoning*, page 29, 2009.

[125] Vijay V. Vazirani. *Approximation algorithms*. Springer, Berlin, 2001.

[126] Johan Wästlund. A solution of two-person single-suit whist. *The Electronic Journal of Combinatorics*, 12(1):R43, 2005.

[127] Johan Wästlund. Two-person symmetric whist. *The Electronic Journal of Combinatorics*, 12(1):R44, 2005.

[128] Mark H. M. Winands, Yngvi Björnsson, and Jahn-Takeshi Saito. Monte Carlo tree search in Lines of Action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):239–250, 2010.

[129] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proc. STOC'06*, pages 681–690, 2006.