Problèmes NP-difficiles : approximation modérément exponentielle et complexité paramétrique

Emeric Tourniaire

Université Paris-Dauphine — École doctorale de Dauphine

Problèmes NP-Difficiles : approximation modérément exponentielle et complexité paramétrique

Émeric TOURNIAIRE

Manuscrit pour l'obtention d'un doctorat ès sciences

Spécialité informatique

17 juin 2013

Vangelis Th. Paschos ..... Directeur de thèse

Bruno Escoffier ..... Co-Directeur de thèse

Éric Angel ..... Rapporteur

Federico Della Croce ..... Rapporteur

Mathieu Liedloff ..... Membre du jury

Yann Vaxès ..... Membre du jury

Stéphane Vialette ..... Membre du jury

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

"I've had nothing yet," Alice replied in an offended tone, "so I can't take more."
"You mean you can't take *less*," said the Hatter: "it's very easy to take *more* than nothing."

Lewis Caroll
Alice's adventures in Wonderland

[. . .] All I can hope is that future historians note that one of the core empirical points providing the intellectual foundation for the global move to austerity in the early 2010s was based on someone accidentally not updating a row formula in Excel.

Mike Konczal

Remerciements

Je souhaite remercier tout d'abord Vangelis et Bruno pour m'avoir encadré pendant ces trois années. Leur patience et leur disponibilité ont été la clé de la réussite de cette thèse, et nos discussions ont largement dépassé le strict cadre professionnel.

Je remercie tous les membres du jury pour avoir accepté d'être présents à ma soutenance. Un merci particulier à Federico et Éric pour avoir relu et commenté mon manuscrit.

Au cours de cette thèse, j'ai échangé avec un grand nombre de chercheurs au sein du LAMSADE, sur des questions scientifiques ou techniques. Je tiens à les remercier ici, et tout particulièrement Jérôme Monnot, dont l'optimisme et la bonne humeur sont communicatives.

L'organisation du séminaire transversal a été une expériences très enrichissante. Un grand merci à Edouard, Abdallah, Florian J., Florian S., et Renaud pour avoir contribué à sa création et à son existence. Merci à tous les gens qui y sont venus, et à tous ceux qui ont participé aux réflexions sur sa conception. Merci également à tous les autres collègues du laboratoire pour toutes ces conversations sur des sujets aussi variés (le graphe du réseau ferré francilien, l'autonomie d'un nouveau-né, les bienfaits de l'agriculture biologique, les limites de l'humour, le rôle de l'AERES[1], le journalisme en Afrique du Sud, ...).

La recherche ne donne pas toujours les résultats auxquels on s'attend. Au cours des trois années, toutes les pistes que nous avons explorées n'ont pas abouti, et cet apprentissage nécessaire s'est fait parfois au détriment de mon naturel enjoué. Heureusement, de nombreuses personnes m'ont entouré de leurs attentions bienveillantes. Claude, Véro, Anne-Laure, Tiphanie, Tristan, Manon : j'espère que vous savez à quel point je vous suis reconnaissant d'être comme vous êtes. Je vous aime très fort. Nico : je n'aurais jamais fait tout ça

[1] Ok, celle-ci n'a pas beaucoup duré.

sans toi[2]. Je tiens à remercier en vrac Alice, Aliocha, Benjamin, Boka, Claire, Claire, Claire, Crocell, Erwan, Fabrice, Héloïse, Jym, Marc, Margaux, Marie, Maxime, Myriam, Sophie, et Sophie pour leur amitié et leur bienveillance. Enfin, je remercie tous les autres amis, parents, proches, camarades, ou collègues avec qui j'ai passé du temps pendant ces trois ans, et dont la liste serait trop longue pour figurer extensivement ici.

[2]Et tu le sais, même si tu nies.

Contents

1 Introduction

*In this first part, we present an overview of the field of this manuscript. You may find here a vulgarization on complexity, a list of **NP**-hard problems that are studied in the latter parts, and a brief insight of techniques used in the field. We present in particular three main topics : exact algorithms, moderately exponential algorithms and parameterized complexity.*

Cette partie présente le contexte dans lequel les résultats de cette thèse ont été obtenus. Vous trouverez ici un préambule introductif qui constitue une tentative de vulgarisation de ce qu'est l'optimisation combinatoire, suivi d'un bestiaire de problèmes d'optimisation combinatoire que j'ai étudiés (ou qui présentent des résultats intéressants pour le cadre de la thèse), et un sommaire état de l'art sur les trois techniques principales étudiées : algorithmes exacts ou approchés faiblement exponentiels, et algorithmes paramétriques.

Table des matières

1.1 En guise de préambule

Une organisation politique crapuleuse[1] a trouvé une combine pour détourner de l'argent des conseils généraux[2] : elle vole l'argent dédié aux lignes de bus, et laisse le département voisin payer. Tant que la comptabilité n'est pas trop regardante, la combine passe inaperçue. Cependant, il n'est pas possible d'utiliser cette technique au-delà d'un montant, identique dans tous les départements. De plus, on ne peut pas l'utiliser dans deux départements voisins : les chauffeurs ne seraient plus payés. L'objectif est donc de trouver un nombre aussi grand possible de départements sans avoir deux départements voisins.

Pour simplifier, on oubliera les DOM-TOM (les organisations politiques crapuleuses les oublient souvent de toute façon).

1.1.1 Modélisation

L'outil adapté pour décrire ce problème est le graphe. Un graphe est un objet mathématique étudié depuis le XVIII^{e} siècle, constitué d'un ensemble de sommets (qui vont ici représenter un département chacun) reliés ou non par des arêtes (ici, on va relier ensemble deux départements s'ils sont voisins). Je donne un exemple de cette transformation sur la Figure 1.1. Sur la figure, le graphe est constitué des sommets $\{1,7,26,38,42,69,73,74\}$, et les arêtes sont $\{(1,38),(1,69),(1,73),(1,74),(7,26),\ldots\}$.

[1] Cet exemple est bien sûr totalement fictif.
[2] Assemblées constituantes des départements.

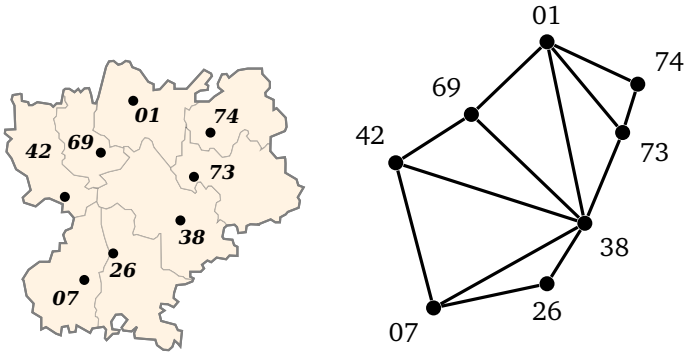


FIGURE 1.1 – Transformation du problème en graphe, sur une région.

Ainsi, la forme précise des départements n'a plus d'importance, il suffit de regarder le graphe pour savoir quels départements peuvent ou ne peuvent pas être pris simultanément.

Mathématiquement, si on appelle V l'ensemble des sommets et E l'ensemble des arêtes[3], on cherche le plus grand sous-ensemble $X \subset V$ qui vérifie

$$\forall i,j \in X, (i,j) \notin E$$

On dit d'un tel sous-ensemble qu'il est « stable », et comme on cherche à le rendre le plus grand possible, le problème qu'on cherche à résoudre s'appelle MAXIMUM INDEPENDENT SET (MAX IS).

L'objet de l'optimisation combinatoire est d'étudier de tels problèmes, et de chercher à les résoudre. MAX IS est dit **NP**-difficile, ce qui signifie qu'on ne peut pas espérer résoudre ce problème très efficacement. Sur un « petit » graphe comme celui de la France, avec seulement une centaine de sommets, on peut le faire, comme on le verra plus bas. Mais si on augmentait trop le nombre de sommets (par exemple en faisant l'analyse au niveau des communes et plus au niveau des départements), le problème deviendrait trop difficile à résoudre, même pour le meilleur supercalculateur ; même en donnant un tel supercalculateur à chaque humain sur terre ; même en leur laissant plusieurs milliers d'année.

Mais donc, que faire ? Nous allons aborder quelques techniques notables.

[3] V pour « vertex » et E pour « edge », respectivement.

1.1.2 L'abandon

Cette première technique n'est sans doute pas très satisfaisante intellectuellement, mais quand on ne peut vraiment pas trouver la solution, on est parfois contraint d'abandonner. Cela peut arriver par exemple :

- parce que les règles du problème ont changé (finalement, on a acheté les juges de tout le pays, donc on peut détourner depuis tous les conseils généraux sans la contrainte de discrétion).

- parce qu'on ne peut pas y arriver avec la contrainte de temps imposée (si le parti veut faire ça avant la réforme du territoire, il lui faut obtenir une solution rapidement)

- parce qu'on ne sait pas résoudre le problème, même sans contrainte de temps. Cela peut arriver si le problème est trop difficile, par exemple parce qu'un nombre infini de stratégies serait possible. Cela dit, on ne traite pas souvent de tels problèmes en optimisation combinatoire, et la recherche exhaustive (voir plus bas) est dans la pratique possible pour tous les problèmes abordés dans cette thèse.

Beaucoup de problèmes **NP**-difficiles sont ainsi abandonnés parce qu'on ne sait pas les résoudre. Ainsi, un GPS sait trouver le chemin le plus court pour aller d'un endroit à un autre, mais ne propose pas de trouver le chemin le plus court qui passe par différents points. C'est normal, le premier problème est facile, le deuxième difficile.

1.1.3 Recherche exhaustive

Pour notre problème, vu qu'on cherche un sous-ensemble des départements, on peut toujours essayer de regarder **tous** les sous-ensembles possibles. Il ne restera alors qu'à éliminer ceux qui ne sont pas stables, et prendre le plus grand dans ceux qui restent.

Sur le cas particulier de la Figure 1.1, on peut facilement énumérer les différents sous-ensembles (il y en a 256), et on peut facilement retrouver les plus grands (ils sont de taille 3, et il y en a 5, par exemple $\{74,69,07\}$).

Bien entendu, si on généralise ça aux 96 départements de la métropole, on aura alors 2^{96} sous-ensembles possibles (soit environ 10^{29}), et il est à nouveau impossible d'espérer tous les faire analyser par un ordinateur.

Par contre, cette méthode, lorsqu'elle est réalisable, donne le plus d'informations : elle permet de savoir si la solution est unique, et si ce n'est pas le cas, elle donne toutes les solutions. Ce ne sera pas le cas des méthodes suivantes.

1.1.4 Algorithme glouton

Les algorithmes gloutons sont les algorithmes dans lesquels on cherche, à chaque étape, à faire les choses le mieux possible, mais sans aucun plan d'ensemble. L'Algorithme 1.1 est un exemple d'algorithme glouton pour le problème MAX IS.

ALGORITHME 1.1 – Algorithme glouton pour MAX IS

Entrées: un graphe
Sorties: un ensemble stable
tant que *Il reste des sommets dans le graphe* **faire**
    Ajouter un sommet dans la solution.
    Retirer les voisins de ce sommet du graphe.
fin

Cet algorithme fonctionne très rapidement. Cependant, il n'est pas très efficace. Par exemple, si l'on considère la Figure 1.2, l'algorithme a choisi deux sommets, alors que la meilleure solution en comporte trois.

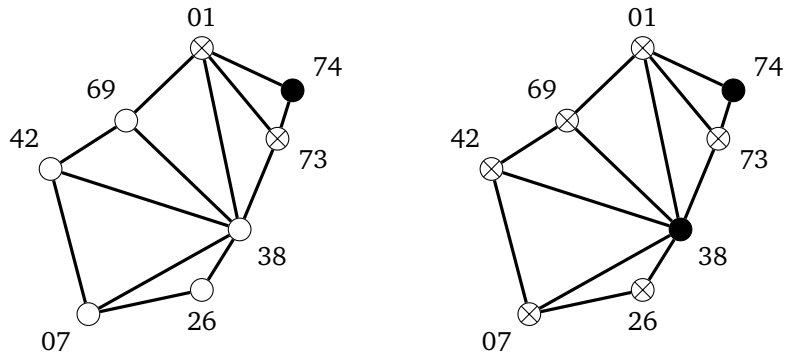
Dans certaines situations l'algorithme glouton donne la meilleure solution. C'est par exemple le cas pour rendre la monnaie : en choisissant toujours la plus grosse pièce possible, vous rendrez toujours le nombre minimum de pièces (ce ne serait pas le cas si on n'avait que des pièces de 1, 3 et 4€[4]). Mais quand le problème est **NP**-difficile, l'algorithme glouton n'est pas efficace.

1.1.5 Heuristique

Dans les cas où la solution de l'algorithme glouton n'est pas satisfaisante, on a parfois des idées pour améliorer la situation une fois qu'il est terminé. L'Algorithme 1.2 est un exemple d'une heuristique.

Sur le problème de la Figure 1.1, cet algorithme permettrait sans doute d'obtenir la solution optimale. Mais c'est un cas très particulier, et cela ne marche pas dans le cas général.

[4]Par exemple, pour rendre 6€.



(a) Le sommet 74 est sélectionné. Ses voisins sont retirés.

(b) Le sommet 38 est sélectionné. Ses voisins sont retirés, l'algorithme est terminé.

FIGURE 1.2 – Application d'un algorithme glouton. Les sommets ● sont rajoutés, et les sommets ⊗ sont éliminés.

ALGORITHME 1.2 – Heuristique pour MAX IS

Entrées: un graphe
Sorties: un ensemble stable
Appliquer l'algorithme glouton et obtenir une solution S.
faire *un grand nombre de fois*
    Essayer d'échanger un sommet au hasard de la solution avec un autre sommet au hasard du graphe.
    si *On peut rajouter un autre sommet* **alors**
        Enregistrer l'échange, rajouter l'autre sommet.
    sinon
        Annuler l'échange
    fin
finfaire

1.1.6 Algorithmes exacts

On a vu plus haut une méthode qui permettait de résoudre le problème en cherchant exhaustivement toutes les solutions possibles. C'est une méthode efficace, mais très lente : s'il y a n départements, il faudra passer en revue 2^n sous ensembles. Cela revient à dire que dès qu'on rajoute un département, on multiplie par deux le temps de calcul, ou encore que chaque département est choisi ou éliminé, et qu'on travaille ensuite avec $n-1$ départements dans les deux cas. Comme l'analyse d'un ensemble donné est rapide, on mesure la difficulté du problème en comptant le nombre de sous-ensembles à examiner, et on note $T(n)$ ce nombre, si on a n départements. On a alors l'équation :

$$T(n) = T(n-1) + T(n-1)$$

et on retrouve bien entendu après résolution $T(n) = 2^n$.

L'objectif de la résolution par algorithmes exacts est d'avoir des algorithmes qui n'examinent plus 2^n ensembles, mais seulement $1{,}5^n$, ou $1{,}1^n$, ou $1{,}001^n$. La Table 1.1 donne des exemples de tailles de problème qu'on peut résoudre si on a l'une ou l'autre de ces complexités, et justifie l'intérêt qu'on porte à ce domaine. Cependant, il faut garder à l'idée que l'objectif de ces études est également d'accéder à une connaissance théorique plus complète.

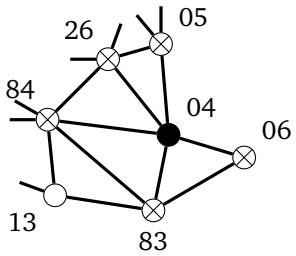
TABLE 1.1 – Tailles de problème (n) qu'on peut résoudre avec un temps de calcul de l'ordre de γ^n

γ	n
2	60
1,618	85
1,5	100
1,38	129
1,1	430
1,001	40 000

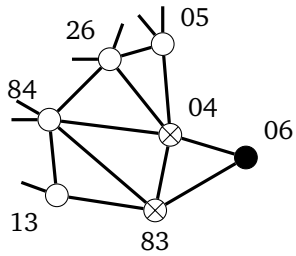
Je présente ici deux propriétés qui permettent d'obtenir une complexité inférieure au 2^n de l'algorithme exhaustif. On utilise un branchement : on choisit un département, que l'on va soit rajouter à la solution (dans ce cas, on raye ses voisins), soit éliminer. Si le département a d voisins, cela conduit à l'équation $T(n) \leqslant T(n-1) + T(n-1-d)$. Et bien entendu, plus d est grand, plus rapide sera l'algorithme.

On peut déjà éliminer le cas où un département n'aurait aucun voisin. Dans ce cas, on peut toujours l'ajouter, et reprendre l'algorithme : ce sera forcément mieux que de ne pas le prendre. On a donc toujours $d \geqslant 1$, et ça nous donne une complexité en 1.618^n.

Imaginons maintenant un département X qui n'aurait qu'un seul voisin Y. On ne peut pas prendre X et Y ; si on prend Y, on risque de ne plus pouvoir prendre ses voisins. Si on prend X, on n'a pas ce risque. On va donc toujours préférer prendre X directement. Un raisonnement similaire s'applique par exemple dans les départements du sud-est de la France (voir Figure 1.3).



(a) Si on choisit de prendre le département 04, on se prive de départements potentiellement intéressants



(b) En revanche, prendre le département 06 ne peut pas nuire à la solution.

FIGURE 1.3 – Simplifications dans le Sud-Est

Au final, on peut montrer assez facilement que les sommets n'ayant qu'un ou deux voisins peuvent être traités avant de commencer l'algorithme. Ainsi, on assure que d vaut au moins 3. On a donc l'équation $T(n) \leqslant T(n-1) + T(n-4)$, d'où un nombre d'ensembles total à examiner inférieur à 1.38^n.

Des analyses plus complexes, plus récentes et présentées dans la suite de l'introduction, permettent de descendre encore cette valeur à 1.22^n.

1.1.7 Algorithmes approchés

Si on ne peut pas obtenir la meilleure solution, on peut essayer d'obtenir une solution approchée. Comme avec une heuristique, on ne cherche plus la meilleure solution, mais cette fois, on veut des garanties qu'on n'aura pas une solution « trop mauvaise ».

Dans la pratique, cela se fait en fixant un rapport d'approximation (par exemple 80%), et en exigeant ensuite d'avoir au moins ce rapport entre la solution que l'on trouve et la meilleure possible (si la meilleure solution est de 20 départements, notre solution en aura au moins 16).

Parfois, obtenir une telle approximation est rapide, mais parfois, même une telle solution est difficile à obtenir. On espère alors simplement avoir l'approximation plus rapidement que ce qui serait nécessaire pour calculer la meilleure solution. Je donne (Algorithme 1.3) un exemple naïf d'une telle approximation pour MAX IS avec un ratio de 50%.

ALGORITHME 1.3 – Algorithme approché à 50% pour MAX IS

Entrées: un graphe
Sorties: un ensemble stable approché
1 Séparer le graphe en deux parties (par exemple le nord et le sud).
2 Calculer le MAX IS sur chacune des deux parties, en ignorant l'autre partie du graphe.
3 Donner la meilleure des deux solutions (celle qui a le plus de sommets).

Ce qu'on obtient est bien une approximation : considérons la solution optimale sur le graphe complet. Imaginons qu'elle a 14 sommets au nord, et 12 au sud, soit 26 au total. Alors notre algorithme trouvera au moins 14 sommets sur la moitié nord, et au moins 12 sur la moitié sud. Je peux alors affirmer qu'au moins une de ces deux solution contient plus (ou autant) que 13 sommets, et on a bien le bon ratio. Finalement, cela fonctionne parce qu'une des moitiés du graphe contiendra au moins la moitié de la solution.

Et du coup, au lieu de résoudre le problème complet (en examinant 1.22^n sous-ensembles d'après le paragraphe précédent), on peut se contenter de résoudre sur des moitiés d'instance, ce qui réduit le nombre d'ensembles à $1.22^{n/2} = 1.1^n$.

On a des techniques qui permettent de généraliser ce résultat pour des ratios différents, mais bien entendu la complexité augmente quand on cherche à être plus précis.

1.1.8 Algorithmes paramétriques

On a vu qu'obtenir la solution peut parfois prendre très longtemps. Mais il arrive parfois que le temps de calcul dépende de la taille de la solution, du nombre moyen de départements voisins, ou d'autres paramètres du graphe.

L'objet de la complexité paramétrique, sans rentrer dans les détails, est de trouver des algorithmes qui sont rapides quand un paramètre choisi à l'avance est petit. De tels algorithmes garantissent que le temps de calcul reste petit quand le paramètre l'est, même si la taille du problème augmente. Mathématiquement, si on appelle ce paramètre k, et la taille n ; cela revient à avoir un temps de calcul de la forme $f(k)g(n)$ où g est une fonction « petite », et f peut être très grosse (mais ce ne serait pas très grave dans les cas où k reste petit).

Un exemple classique est de prendre pour k la taille de la solution, et la question qui se pose alors est : « A-t-on des algorithmes efficaces lorsque la solution est petite ? ». On donne des exemples de ceci dans la section 1.4.3 (page 38).

1.1.9 Est-ce que P $=$ NP ?

Depuis le début de l'introduction, j'utilise les mots « difficile » ou « facile » pour décrire des problèmes informatiques. La classification des problèmes en ces deux catégories n'est pas arbitraire : elle correspond à des critères très précis.

On dit qu'un problème est dans la classe **P** (« facile ») si on peut le résoudre en temps polynomial[5]. Il suffit par contre qu'on sache vérifier la solution d'un problème pour qu'il soit dans **NP** (« difficile »)[6].

Bien entendu, si un problème est dans **P**, il est également dans **NP** : si je dois vérifier une solution d'un problème de **P**, je peux commencer par trouver la solution, et je compare les deux.

Inversement, on peut se poser la question de savoir si les problèmes de **NP** sont également dans **P**. Ce problème est ouvert depuis 1971, il est devenu en 2001 un des 7 problèmes du millénaire, et est donc mis à prix pour un million de dollars. Malgré cela, on n'a pour le moment aucune preuve. On sait en revanche qu'il n'est pas nécessaire d'obtenir des résultats sur tous les problèmes **NP** : il suffirait de démontrer qu'un problème dit « **NP**-Complet » est, ou n'est pas dans **P** pour répondre à ce problème, et devenir riche. On connait aujourd'hui beaucoup de problèmes **NP**-Complets (pour commencer, presque tous les problèmes présentés dans cette thèse le sont). Si, contre l'intuition de la plupart des chercheurs, quelqu'un prouvait que **P** $=$ **NP**, cela rendrait tous les travaux de cette thèse obsolètes, ce qui serait embêtant.

[5] Cela signifie grossièrement que le temps de calcul ne va pas augmenter trop vite quand la taille du problème augmente.

[6] C'est aussi une définition grossière. Une définition précise est donnée au 1.2.2.

1.2 Un aperçu du domaine

1.2.1 Qu'est-ce que l'optimisation combinatoire ?

L'optimisation en mathématiques consiste à maximiser ou minimiser une quantité dépendant de variables. La complexité d'un problème d'optimisation dépend beaucoup de l'expression de la quantité et des valeurs autorisées pour les variables. Dans certains cas particuliers, on dispose d'outils puissants : par exemple, l'algorithme du simplexe introduit dans [39] permet de résoudre efficacement des problèmes d'optimisation linéaire[7]. Il existe également certaines méthodes numériques d'optimisation qui permettent d'obtenir des résultat approchés, et certaines ont été utilisées pour l'élaboration de cette thèse.

En optimisation combinatoire, le nombre de variables considéré est fini, et le nombre de valeurs possibles pour chaque variable est également fini. Cette définition est assez générale pour englober énormément de problèmes. Nous montrons ici sur un cas particulier comment on peut faire correspondre un problème d'optimisation combinatoire à cette définition.

MINIMUM TRAVELING SALESMAN PROBLEM (MIN TSP) consiste à trouver la chaîne la plus courte passant par tous les sommets d'un graphe pondéré (la longueur d'une chaîne étant la somme des poids des arêtes qui le constituent). Si on appelle $G(V,E)$ le graphe, où $V = (v_1, \dots v_n)$ est l'ensemble des sommets et $E = (e_1, \dots, e_m)$ l'ensemble des arêtes, et le poids de l'arête e_i est simplement $p(e_i)$. On crée m variables $(x_1, \dots x_m)$ qui peuvent prendre deux valeurs : 0 ou 1. Pour estimer la valeur d'une solution, on considère les arêtes e_i telles que $x_i = 1$.

- Si ces arêtes forment un circuit qui passe par tous les sommets une seule fois, alors la valeur de cette solution est la somme des poids des arêtes choisies ($\sum_{x_i=1} p(e_i)$).
- Sinon, la solution a pour valeur $+\infty$.

On se convaincra aisément que minimiser la valeur d'une solution revient précisément à résoudre le problème MIN TSP. D'autres problèmes d'optimisation combinatoire sont présentés en section 1.3 (toutefois sans le formalisme ci-dessus).

[7] Un cas particulier dans lequel la quantité à optimiser et les contraintes sur les variables s'expriment linéairement par rapport à celles-ci.

1.2.2 Qu'est-ce que la complexité ?

Depuis les années 1930, des mathématiciens ont formalisé la notion de *calculabilité*. Celle-ci permet de distinguer les problèmes dont on peut *calculer* la solution à l'aide d'une suite finie d'opérations, et un des travaux pionniers dans le domaine est un article d'Alan Turing [89] qui définit la notion de machine de Turing.

Une machine de Turing est une machine virtuelle, constituée d'un ruban infini, d'une tête de lecture/écriture mobile, et d'un *programme*, un automate fini qui commande les déplacements de la tête sur le ruban, ainsi que ses écritures. Le comportement de la machine est complètement défini par le programme et les données présentes initialement sur le ruban. Formellement, Alan Turing montre qu'une telle machine ne peut pas résoudre tous les problèmes possibles[8].

En optimisation combinatoire, en revanche, il est toujours possible d'énumérer les valeurs possibles pour toutes les variables, et de choisir la solution optimale, ce qu'une machine de Turing peut toujours faire. Cela semble une bonne nouvelle, puisque nos ordinateurs savent donc, pour peu qu'ils aient assez de mémoire, résoudre tous les problèmes de notre domaine.

Cependant, deux facteurs limitants importants apparaissent :

- les ordinateurs n'ont pas une mémoire infinie, et surtout
- les humains n'ont pas autant de temps qu'ils le voudraient pour obtenir les résultats attendus.

La notion de complexité apparait logiquement dans les années 1960, après la création des premiers ordinateurs. Elle a pour objectif de mesurer l'efficacité d'un algorithme, c'est-à-dire le nombre nécessaire d'étapes pour qu'une machine de Turing puisse résoudre un problème. Presque simultanément, A. Cobham ([30]) et J. Edmonds ([46]) définissent la classe **P** qui regroupe tous les problèmes qu'on peut résoudre en temps polynomial par rapport à la taille des données. Il s'agit pour Edmonds de donner un critère pratique de réalisabilité[9] :

> There is an obvious finite algorithm, but that algorithm increases in difficulty exponentially. (...) For practical purposes the difference between algebraic and exponential order is often more crucial than the difference between finite and non-finite. J. Edmonds

[8] Il démontre en particulier que le problème de l'arrêt n'est pas calculable.

[9] Nous reviendrons toutefois sur cette notion plus loin.

et pour Cobham, les raisons invoquées sont algébriques[10] :

> For several reasons the class **P** seems a natural one to consider. For one thing, if we formalize the definition relative to various general classes of computing machines we seem always to end up with the same well-defined class of functions. A. Cobham

Quelques années plus tard, S. Cook introduit dans [31] la classe **NP** des problèmes qui peuvent être résolus en temps polynomial par une machine de Turing non déterministe. Intuitivement, cette classe regroupe les problème dont on peut vérifier la validité en temps polynomial, et contient donc en particulier tous les problèmes de la classe **P**. Dès lors, la question de l'égalité $\mathbf{P} = \mathbf{NP}$ était posée, et est encore à ce jour non-résolue.

Cependant, de nombreux problèmes ont pu être classifiés grace à deux outils principaux : les réductions polynomiales (développées par Karp dans [69]), et la **NP**-complétude (introduite par Cook dans [31]).

Définition 1.1 (Réduction polynomiale)**.** Soient deux problèmes Π_1 et Π_2. On dit que Π_1 se réduit polynomialement à Π_2 s'il existe une fonction f calculable en temps polynomial telle que toute instance I de Π_1 admet une solution si et seulement si $f(I)$ admet une solution pour le problème Π_2.

Définition 1.2 (**NP**-complétude)**.** Un problème est dit **NP**-complet s'il est dans **NP**, et que si problème de **NP** peut se réduire polynomialement à lui.

Cook montra en particulier que le problème SATISFIABILITY (SAT) est **NP**-complet, et Karp démontra la **NP**-complétude de 21 problèmes dont certains seront présentés dans le bestiaire (page 20).

Par la suite, la classification s'est étoffée d'un grand nombre de catégories, délimitant les problèmes qu'on peut résoudre avec une catégorie d'algorithmes ou une autre, et [1] en dénombre à ce jour plus de 400.

Nous retiendrons essentiellement :

- **NPO** est la classe naturelle des problèmes d'optimisation, c'est à dire des problèmes qui consistent à trouver une solution x qui minimise une fonction de coût $C(x)$ calculable en temps polynomial. (voir [7]).

[10] La classe est invariante si on change de modèle computationnel. Par exemple certains modèles de machine de Turing ont plusieurs rubans ; les problèmes résolubles en temps polynomial sont néanmoins les même que sur des machine de Turing à un seul ruban.

- **SUBEXP** (resp. **EXP**) est la classe des problèmes qu'un algorithme fonctionnant en temps sous-exponentiel (resp. exponentiel) peut résoudre.
- **FPT**, **W[1]**, **W[k]**, et **XP** sont des classes liées à la complexité paramétrique, et seront vues plus en détail ultérieurement (Section 1.4.3, page 38).
- **APX** est la classe des problèmes admettant un algorithme d'approximation polynomial à ratio constant, et seront vus plus en détail ultérieurement (Section 1.4.2, page 32).

1.2.3 À propos de complexité exponentielle

Les problèmes que nous considérons dans ce manuscrit sont tous des problèmes **NP**-difficile. À moins de démontrer que **P** $=$ **NP**, on ne peut pas espérer trouver d'algorithme polynomial qui les résolvent.

Une première approche possible a été de chercher des algorithmes polynomiaux offrant des solutions aussi proches que possible de l'optimal. On trouve des algorithmes stochastiques ou des heuristiques très efficaces sur certains problèmes, et on a parfois même des garanties de performance dans le pire des cas. Généralement, cette performance se mesure comme l'« écart » entre la solution produite et la solution optimale, et on utilise en particulier le rapport standard. Sur une instance particulière I, avec une fonction objectif f positive à maximiser, on mesure la valeur de la solution produite par l'algorithme $f(S_I)$ et la valeur de la solution optimale $f(\mathrm{OPT}(I))$. Le ratio d'approximation sur I est alors $\rho(I) = f(S_I)/f(\mathrm{OPT}(I))$, et le ratio d'approximation de l'algorithme est[11] :

$$\rho = \inf_I \frac{f(S_I)}{f(\mathrm{OPT}(I))}.$$

La section 1.4.2 page 32 donne des exemples précis de tels algorithmes. Néanmoins, cette approche a ses limites, et nous verrons en particulier qu'il n'est pas toujours possible d'atteindre n'importe quel ratio d'approximation pour n'importe quel problème.

Dans les cas où l'approximation polynomiale échoue, il devient nécessaire d'envisager l'utilisation d'algorithmes non-polynomiaux. La question de la réalisabilité pratique devient alors cruciale : les algorithmes exponentiels sont facilement dépassés par des tailles modérées de problèmes.

[11] Avec une fonction à minimiser, on prend le $\sup$ au lieu de l'$\inf$.

On pourrait penser que ce problème est voué à être résolu par l'amélioration constante des moyens technologiques (depuis les années 70, la puissance de calcul des ordinateurs double tous les deux ans environ), qui devraient permettre tôt ou tard de résoudre n'importe quel problème, fût-il exponentiel.

En fait, c'est plutôt le contraire que l'on constate : l'amélioration des moyens technologiques fait bien peu de choses en regard de l'amélioration des algorithmes eux-même, et l'étude des problèmes d'optimisation combinatoire est donc indispensable pour nous permettre d'obtenir des solutions efficacement.

Illustrons cela avec un problème classique de théorie des graphes, MAX IS, déjà rencontré dans le préambule. Dans un graphe $G(V,E)$, la résolution par recherche exhaustive nécessite d'examiner tous les $2^{|V|}$ sous-graphes[12]. Un supercalculateur récent peut, en un an, effectuer $567\,648\,000\,000\,000\,000\,000$ opérations. C'est un nombre impressionnant, mais il permettrait au mieux d'examiner les sous-ensembles d'un graphe de 80 sommets. À l'inverse, le meilleur algorithme connu pour résoudre ce problème peut se contenter de l'ordre de $1.2127^{|V|}$ opérations ([20]). La même puissance permet alors de résoudre des instances de 280 sommets. Enfin, attendre que des ordinateurs soient assez puissants pour résoudre des instances de 280 sommets avec la méthode exhaustive prendrait de l'ordre de 700 ans...

Les algorithmes exponentiels efficaces ont donc un intérêt quand ils sont comparés aux algorithmes exponentiels inefficaces. Mais ils peuvent rivaliser aussi contre des algorithmes polynomiaux. Ainsi, la table 1.2 montre que pour certaines tailles d'instance, il vaut sans doute mieux utiliser un algorithme qui nécessite $1{,}1^n$ opérations qu'un algorithme qui en nécessite n^3.

TABLE 1.2 – Comparaison du nombre d'opération entre un algorithme polynomial et un algorithme exponentiel efficace

Taille de l'instance	n^3	$1{,}1^n$
$n = 30$	$27\,000$	18
$n = 50$	$125\,000$	118
$n = 100$	10^6	$13\,781$
$n = 200$	8×10^6	2×10^8

Cette différence n'est pas que théorique : il existe des cas où des algorithmes exponentiels sont préférés en pratique à des algorithmes polynomiaux. Un

[12] $|E|$ désigne le cardinal de l'ensemble E.

exemple spectaculaire est l'utilisation en programmation linéaire de l'algorithme du simplexe (exponentiel), préféré à la méthode des ellipses (polynomiale). Les schémas d'approximation polynomiaux sont également connus pour être inefficaces en principe (avec une complexité en $O(n^{1/(1-\epsilon)})$, pour un ratio d'approximation ϵ).

L'habitude veut que les notations de Landau sont utilisées en complexité : le nombre d'opérations réalisées par un algorithme s'exprime généralement en $O(f(n))$, ce qui signifie que le rapport entre le nombre effectif d'opérations élémentaires et $f(n)$ est borné[13]. En complexité exponentielle, s'intéresser à un coefficient polynomial est de peu d'intérêt : une complexité $O(p(n) \times 2^n)$, où p serait un polynôme d'aussi grand degré que l'on veut, sera toujours une complexité $O(2{,}000 \ldots 001^n)$. On utilise donc la notation $O^*(f(n))$, qui signifie que le rapport est borné *à polynôme près*.

Inversement à ce que l'on écrivait plus haut, donc, il peut aussi arriver que des algorithmes étudiés aient une complexité exponentielle intéressante, comme $O^*(1{,}1^n)$, mais avec un facteur polynomial si grand qu'il n'est pas possible de l'utiliser en pratique, même pour des instances de taille modérée. Nous garderons alors à l'esprit l'intérêt théorique de ces classifications.

1.3 Bestiaire de problèmes

Cette partie présente une liste de problèmes qu'on rencontre en optimisation combinatoire, et pour lesquels des résultats ont pu être trouvés par la suite. Tous ces problèmes sont des problèmes **NPO**, ce qui justifie de sortir du cadre polynomial. Dans la mesure du possible, des exemples d'application découlant directement de ces problèmes seront donnés, ainsi que des résultats du domaine les concernant.

Pour des raisons de commodité, les noms des problèmes sont conservés dans leur forme anglaise. Les lecteurs familiers avec le domaine connaitront sans doute mieux ceux-ci que leurs formes françaises.

1.3.1 Sur des graphes

Dans toute cette partie, nous employons les notations suivantes. Si $G(V,E)$ est un graphe, V est l'ensemble de ses sommets, $E \subset V^2$ ses arêtes. On notera $n = |V|$ et $m = |E|$. Pour tout sous-ensemble $W \subset V$, on notera $G[W]$ le

[13] On ne cherche pas à connaitre une borne de ce rapport, parce qu'elle varie facilement en fonction de ce qu'on considère vraiment comme une opération élémentaire.

graphe induit par W sur G, et $E(W) = E \cap W^2$ les arêtes de $G[W]$. Si v est un sommet, on notera $N(v)$ son voisinage ouvert, et $N[v] = N(v) \cup \{v\}$ son voisinage fermé. On notera $d(v)$ le degré de v. Le graphe complémentaire de G est $G(V,V^2 \setminus (E \cup \{(i,i)\}))$ (on inverse les arêtes, sauf les boucles).

MAXIMUM INDEPENDENT SET et MAXIMUM CLIQUE

MAX IS consiste à trouver un ensemble indépendant de taille maximale (c'est-à-dire un ensemble W de taille maximale tel que $E(W) = \emptyset$).

Ce problème est l'un des plus anciens et des mieux étudiés de théorie des graphes, et ce en particulier parce que c'est l'un de ceux pour lesquels les résultats négatifs sont les plus forts. Il est en particulier inapproximable en temps polynomial avec un ratio $n^{1-\varepsilon}$ (pour tout $\varepsilon > 0$) si **P** $\neq$ **NP** ([93]).

MAXIMUM CLIQUE (MAX CLIQUE) consiste à trouver un sous-graphe complet de taille maximale (c'est-à-dire un ensemble W de taille maximale tel que $E(W) = W^2$). Il est équivalent au problème MAX IS en passant au graphe complémentaire, et les deux ont donc même complexité, et même ratio d'approximation par des algorithmes approchés.

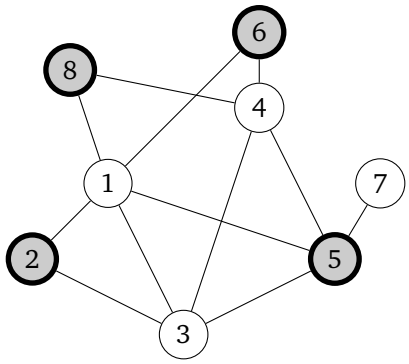
Ces deux problèmes ont des applications très variées en sélection de projets, en taxonomie, en théorie des codes, en économie, en vision par ordinateur, en transmission de signal, en biologie moléculaire, etc. On trouvera dans [85], section 7, des références plus précises sur ces applications.

Ces deux problèmes sont montrés **NP**-difficile dans [69]. Jusqu'à présent, les meilleurs résultats pour résoudre le problème en espace polynomial sont ceux obtenus par Bourgeois *and al.* dans [20], qui proposent un algorithme en $O^*(1.22^n)$. On trouve également dans [87] algorithme en espace exponentiel sait le résoudre en $O^*(1.19^n)$

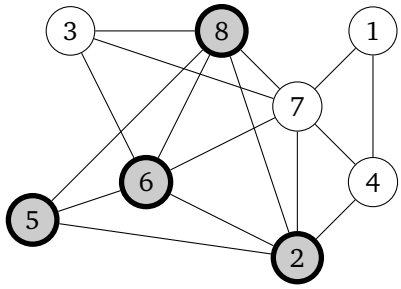
MINIMUM VERTEX COVER

MINIMUM VERTEX COVER (MIN VC) consiste à trouver une couverture minimale des arêtes par les sommets, c'est-à-dire un ensemble $W \subset V$ aussi petit que possible tel que $\forall (v_i,v_j) \in E, v_i \in W$ ou $v_i \in W$.

Il se trouve que le complémentaire d'un ensemble stable est une couverture, on peut donc passer d'une solution de MAX IS à une solution pour MIN VC (voir figure 1.4) en passant au complémentaire dans l'ensemble des sommets. Cela garantit une complexité identique au problème précédent pour un algorithme exact (mais pas pour des algorithmes approchés). Cela démontre aussi que le problème est **NP**-difficile, et on trouve également ce résultat dans [69].



(a) Exemple de MAX IS : $\{2,5,6,8\}$ est un stable maximum, $\{1,3,4,7\}$ est une couverture minimum.



(b) En inversant les arêtes, la solution précédente devient une clique maximale (les sommets sont déplacés pour lisibilité)

FIGURE 1.4 – MAX IS, MAX CLIQUE

Il est approximable polynomialement avec un ratio 2 très facilement : l'algorithme glouton qui consiste à rajouter les deux extremités d'une arête qui n'est pas encore couverte permet d'atteindre ce ratio. De plus, si **P** $\neq$ **NP**, on peut montrer que ce problème ne peut pas être approché polynomialement avec un ratio meilleur que 1.3606 ([42]). Sous une autre hypothèse classique (l'Unique Games Conjecture), il est impossible d'obtenir un ratio meilleur que 2 ([71]).

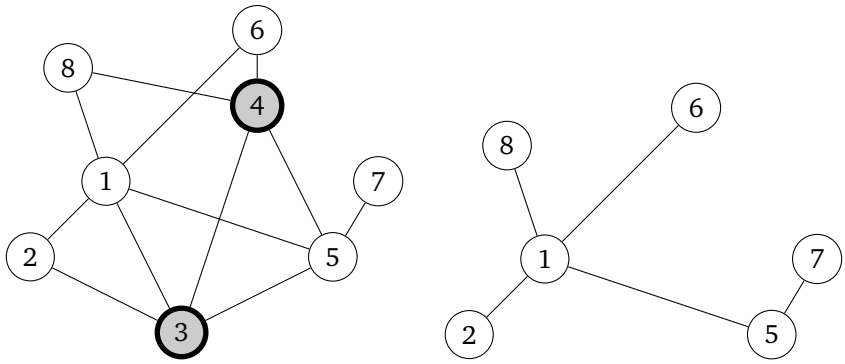
En plus des applications du problème précédent, MIN VC a également des applications dans la résolution de *race conditions*[14] (voir par exemple [83]).

MINIMUM FEEDBACK VERTEX SET

MINIMUM FEEDBACK VERTEX SET (MIN FVS) consiste à trouver un sous-ensemble minimal de sommets qui intersecte chaque cycle du graphe. Alternativement, cela revient à trouver un sous-ensemble de sommets dont la suppression rend le graphe acyclique (*i.e.* une forêt, voir figure 1.5). Ce problème peut se rencontrer avec des graphes orienté ou non, et les deux versions ne sont pas tout à fait équivalentes. Elles sont toutes les deux **NP**-difficile ([69]).

Le problème admet une 2-approximation polynomiale ([11]) dans les deux cas. Pour la résolution exacte, on connait un algorithme pour les graphes orientés

[14] Ou situations de compétition en bon français.



(a) Exemple de MIN FVS : tous les cycles de ce graphe contiennent le sommet 1 ou le sommet 4. $\{1,4\}$ est donc la solution.

(b) Le retrait de ces deux sommets transforme le graphe en forêt.

FIGURE 1.5 – MIN FVS

avec une complexité $O^*(1.9977^n)$ ([86]). Pour les graphes non-orientés, on a un algorithme en $O^*(1.7347^n)$ ([56]).

MIN FVS a des applications pour la conception de systèmes d'exploitations (plus précisément pour optimiser des *deadlock recovery*[15]), dans des bases de données, dans la conception de puces électroniques, et en séquençage de génome.

MAXIMUM CUT et ses variantes

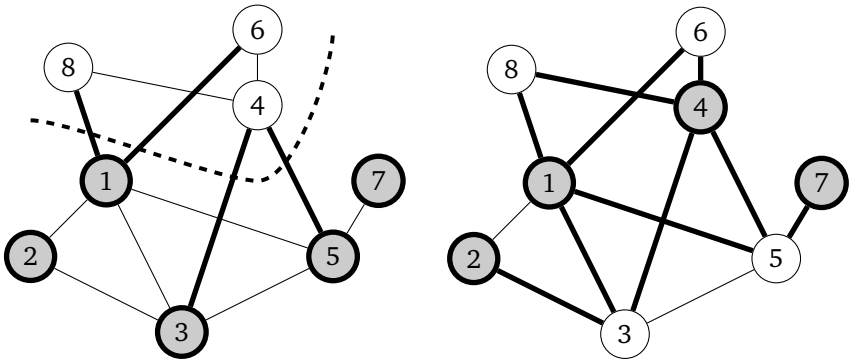
Les problèmes de coupe consistent à partitionner les sommets d'un graphe en deux ensembles V_1 et V_2, et à considérer les arêtes à cheval sur les deux ensembles. On dénombre beaucoup de problèmes dans cette catégorie.

MINIMUM CUT (MIN CUT) (resp. MAXIMUM CUT (MAX CUT)) consiste à choisir les ensembles de sorte à minimiser (resp. maximiser) ce nombre d'arêtes[16]. Ces deux problèmes sont illustrés sur la figure 1.6. MIN CUT est un problème polynomial, identique au problème de flot maximal, et traditionnellement associé aux algorithmes de Ford-Fulkerson ([59]) et d'Edmonds-Karp ([47]). En plus des applications immédiates pour la recherche de flot maximal (on remonte l'origine de ce problème aux réseaux ferroviaires russes en 1930...), on trouve

[15] Ici, je ne vois pas de traduction satisfaisante.

[16] Dans le cas de MIN CUT, on impose un sommet s dans V_1 et un sommet t dans V_2.

des applications pour des problèmes d'affectation, de chemins indépendants, ou de gestion de contraintes. Inversement, MAX CUT est **NP**-difficile ([69]), et difficile à approcher : si **P** $\neq$ **NP**, alors on ne peut pas l'approcher en temps polynomial avec un ratio meilleur que 0.941 ([64]). En admettant l'unique games conjecture, il devient impossible de faire mieux que 0.878 ([70]), et on sait atteindre ce ratio ([63]). MAX CUT trouve des applications en physique statistique ([9]) ou en conception de circuits ([26]).



(a) Exemple de MIN CUT dans lequel on a imposé $1 \in V_1$ et $4 \in V_2$. On trouve alors $V_1 = \{1,2,3,5,7\}$.

(b) MAX CUT sur le même graphe, donne cet autre résultat.

FIGURE 1.6 – MIN CUT et MAX CUT

Les problèmes MINIMUM $(k,n-k)$-CUT (MIN K,N-K CUT) et MAXIMUM $(k,n-k)$-CUT (MAX K,N-K CUT) sont identiques aux deux précédents, sauf qu'on rajoute un paramètre entier k à l'instance, et qu'on impose la taille de V_1 à k. Il s'agit alors à nouveau de maximiser ou minimiser le nombre d'arêtes de la coupe. Le problème MIN K,N-K CUT a été introduit par [51], où il est également nommé « CUTTING A FEW VERTICES FROM A GRAPH »[17]. Le cas particulier où $k = n/2$ est noté MINIMUM BISECTION (MIN BISECTION) (respectivement MAXIMUM BISECTION (MAX BISECTION)) et est montré **NP**-difficile dans [62][18]. La **NP**-difficulté du cas général en découle, même si le problème devient polynomial quand on fixe la

[17] Nous continuerons cependant à le noter MIN K,N-K CUT par cohérence avec la version maximale.

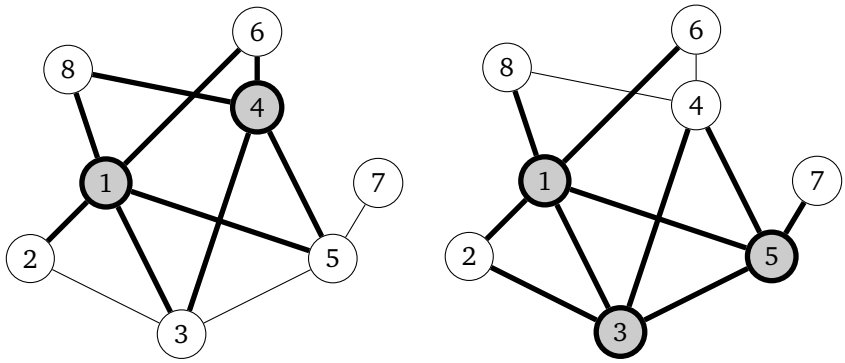
[18] La **NP**-difficulté de MAX BISECTION découle de celle de MIN BISECTION, en inversant les arêtes du graphe.

valeur de k. On ne connait pas d'algorithme d'approximation polynomial à ratio constant, mais des résultats plus spécifiques sont détaillés dans le chapitre 4.

MAXIMUM k-COVERAGE

Le problème MAXIMUM k-COVERAGE (MAX K-COVERAGE) consiste à choisir un ensemble $S \subset V$ de sommets du graphe, avec $|S| = k$, qui couvre un nombre maximum d'arêtes (c'est-à-dire qui maximise le nombre d'arêtes $(i,j) \in E$ tel que $i \in S$ ou $j \in S$).

Ce problème admet un ratio d'approximation $3/4$ ([52]), mais n'admet pas de schéma polynomial.



(a) Exemple de MAX K-COVERAGE avec $k = 2$ (b) Idem, avec $k = 3$.

FIGURE 1.7 – MAX K-COVERAGE

1.3.2 Problèmes généraux

SATISFIABILITY et ses variantes

Le problème SAT est un problème essentiel en complexité et calculabilité. Prenons des variables booléennes que l'on nomme $x_1, x_2, \ldots, x_n$. Chacune de ces variables peut prendre deux valeurs : VRAI (V) ou FAUX (F). On dispose également de trois opérateurs : la conjonction, la disjonction et la négation, dont les valeurs de vérité sont données dans le tableau 1.3.

À l'aide de ces éléments, on peut construire des formules booléennes, et la question se pose alors de savoir si on peut affecter une valeur à chaque variable

TABLE 1.3 – Tableaux de vérité des opérateurs booléens classiques

(a) Disjonction $x_1 \vee x_2$

x_1	x_2	$x_1 \vee x_2$
F	F	F
F	V	V
V	F	V
V	V	V

(b) Conjonction $x_1 \wedge x_2$

x_1	x_2	$x_1 \wedge x_2$
F	F	F
F	V	F
V	F	F
V	V	V

(c) Négation $\neg x_1$

x_1	$\neg x_1$
F	V
V	F

x_i telle que la valeur de la formule soit V. Un *littéral* est soit une variable, soit sa négation (x_i ou $\neg x_i$).

Dans la pratique, on exprime souvent les formules sous *Forme Normale Conjonctive* (CNF), c'est-à-dire sous la forme d'une conjonction de disjonctions de littéraux. Historiquement, ce problème est le premier à avoir été démontré **NP**-difficile par Cook ([31]), en construisant explicitement la formule « associée » à l'état final d'une machine de Turing non-déterministe (après un nombre connu d'étapes).

Par la suite, d'autres versions de ce problème ont été décrites. On citera en particulier 3-SATISFIABILITY (3-SAT) (resp. k-SATISFIABILITY (k-SAT)), dans lequel on cherche la satifiabilité d'une formule constituée d'une conjonction de disjonctions d'au plus 3 (resp. k) littéraux. Cette version est montrée **NP**-difficile dans [69], alors que le problème est polynomial quand on limite les formules à deux littéraux.

Il existe également des versions **NPO** de ces problèmes : dans MAXIMUM SATISFIABILITY (MAX SAT), on cherche à maximiser le nombre de formules satisfaites, alors qu'elles sont toutes des disjonctions de littéraux. Dans MINIMUM SATISFIABILITY (MIN SAT), on cherche à minimiser ce nombre. Enfin, on a les variantes MINIMUM k-SATISFIABILITY (MIN k-SAT) et MAXIMUM k-SATISFIABILITY (MAX k-SAT) dans lesquelles on limite le nombre de littéraux à k dans chaque clause. Ces deux problèmes sont également **NP**-difficiles, même quand $k = 2$.

La complexité pour résoudre exactement tous ces problème a fait l'objet de nombreuses publications. À l'heure où ces lignes sont écrites, on ne connait pas d'algorithme meilleur que $O^*(2^n)$ pour résoudre SAT. On dispose cependant d'heuristiques : un concours pour comparer des solveurs est organisé réguliè-

rement[19], et les efficacités de ces heuristiques sont comparées. Du point de vue théorique, signalons également deux points importants : la complexité est ici envisagée du point de vue du nombre de variables (n), mais elle peut l'être également du point de vue du nombre de clauses (m). On a par exemple un algorithme en $O^*(2^{n(1-1/\log(2m))})$ pour résoudre SAT ([38]). De même, il arrive souvent qu'on considère des instances particulières de SAT. Dans le cas de 3-SAT, on dispose d'un algorithme en $O^*(1.330^n)$ ([78]), par exemple. Il est actuellement conjecturé qu'il n'est pas possible de résoudre 3-SAT en temps sous-exponentiel, et même qu'il n'est pas possible de résoudre SAT en temps inférieur à $O^*(2^n)$ (il s'agit respectivement de l'*Exponential Time Hypothesis* et de sa version forte).

Les versions d'optimisation de ces problèmes ont des complexités encore différentes. Signalons qu'on ne sait pas faire mieux que $O^*(2^n)$ non plus pour MAX SAT ou MIN SAT, et qu'on dispose d'algorithmes dépendant du nombre de clauses, en particulier [27] donne un algorithme en $O^*(1.325^m)$ dont nous nous servirons abondament dans le chapitre 2. Enfin, en terme d'approximation, on sait facilement obtenir une $1/2$-approximation de MAX SAT en temps polynomial[20], mais on dispose également d'algorithmes plus performants, dont l'un fournit une 0.7846-approximation ([4]). Pour MIN SAT, on ne dispose pas de si bon résultat, mais on sait toutefois approcher MIN k-SAT avec un ratio $2(1-2^{-k})$ (et donc avec un raio 2 pour MIN SAT).

Beaucoup de problèmes d'optimisation peuvent se réduire assez naturellement à l'une ou l'autre des variantes de SAT, et ses applications sont difficiles à délimiter précisément, mais on peut en citer en conception électronique, en vérification formelle, ou en résolution de contraintes.

MINIMUM SET COVER

Dans MINIMUM SET COVER (SET COVER), on se donne un ensemble de n objets E, et une collection de m sous-ensembles de E $E_1, \ldots, E_m$. L'objectif est de trouver un nombre minimum de ces sous-ensembles tels que la réunion vaut E, autrement dit des indices $(a_1, \ldots, a_k)$ tels que k soit minimal et $\bigcup_{i=1}^{k} E_{a_i} = E$. Ce problème est illustré sur la figure 1.8.

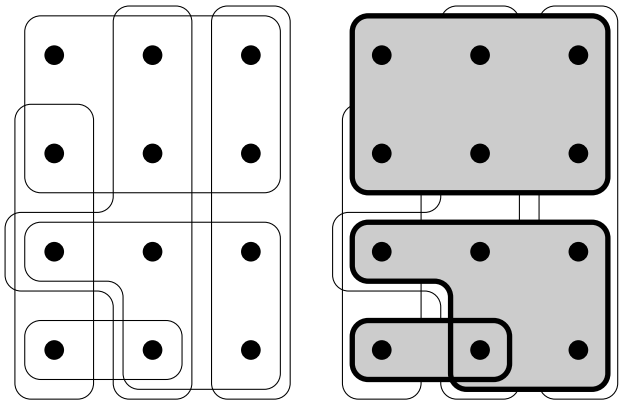
Ce problème est lui aussi montré **NP**-difficile dans [69]. Certaines variantes limitent le nombre d'élément dans chaque ensemble (le problème reste **NP**-difficile si on limite à 3 ce cardinal, il est dans **P** si on limite le nombre d'éléments à deux), ou bien la fréquence d'apparition de chaque élément.

[19] `http://www.satcompetition.org/`

[20] Il suffit de choisir la valeur de variable de manière gloutonne.

En terme d'approximation, on démontre (sous l'hypothèse que **P** $\neq$ **NP**) qu'il n'est pas possible d'approcher SET COVER avec un ratio constant quelconque ([77] par exemple).

Ce problème a de nombreuses applications dont une pratique est détaillée ici : imaginons une compagnie aérienne qui doit réaliser un certain nombre de trajets. Un équipage peut assurer plusieurs de ces trajets mais pas n'importe quelle combinaison de ceux-ci. On peut minimiser le nombre d'équipage en se ramenant à SET COVER, où les ensembles E_i sont les ensembles de trajets compatibles. De même, on peut facilement réduire le problème MINIMUM DOMINATING SET (MIN DS) (qui consiste à choisir un nombre minimum de sommets qui couvre tous les autres) à SET COVER : on construit un ensemble par sommet, qui contient le sommet et son voisinage (et on a donc $m = n$ dans ce cas).



(a) Représentation graphique d'une instance de SET COVER

(b) Illustration d'une solution (optimale) de valeur 3

FIGURE 1.8 – SET COVER

1.4 État de l'art

Dans cette dernière partie de l'introduction, je propose une revue d'algorithmes ou de techniques existantes pour certains problèmes présentés dans le bestiaire.

1.4.1 Algorithmes exacts

Nous considérons ici des algorithmes exacts pour résoudre des problèmes **NP**-difficile. Pour un aperçu plus détaillé, je renvoie au livre de Fomin et Kratsch ([58]).

Recherche exhaustive

La première méthode, la plus générale, est de considérer toutes les solutions pour un problème donné. D'après la définition d'un problème d'optimisation combinatoire vue section 1.2.1 (page 15). Il y a encore des problèmes pour lesquels on ne connait pas d'algorithme exact plus rapide, par exemple MAX SAT (voir page 25).

Les algorithmes de ce genre sont faciles à mettre en pratique, et assez rapides sur des très petites instances : en ce sens ils sont parfois la meilleure solution pour résoudre des problèmes complexes sur des instances très petites.

Branchement

Les algorithmes de branchement consistent à « réduire » une instance de taille n à deux ou plusieurs instances de taille inférieure à n. Ici, « réduire » s'entend dans le sens où la solution sur les sous-instances permet de trouver en temps polynomial une solution sur l'instance de départ. Un formalisme plus précis est proposé de ce qu'est une règle de branchement dans le chapitre 3.

Les algorithmes de branchement sont la solution naturelle pour implémenter des recherches exhaustives : considérons par exemple le problème MAX IS. Si on considère une instance $G(X,E)$ du problème MAX IS, on peut choisir un sommet v au hasard, et considérer deux sous-instances[21] $G_1 = G[X \backslash v]$ et $G_2 = G[X \backslash N[v]]$. Ces deux sous-instances correspondent à la situation naturelle si on choisit de prendre le sommet v dans la solution, ou de l'en exclure. Elles sont alors résolues récursivement, donnant les solutions S_1 et S_2 ; la solution finale est le maximum entre $|S_1|$ et $|S_2| + 1$.

Précisons également que si un branchement aboutit à k instances dont la taille est réduite de $a_1, \ldots, a_k$, alors la complexité du branchement $T(n)$ doit vérifier

$$T(n) \geqslant \sum_{i=1}^{k} T(n - a_i) \tag{1.1}$$

[21] Les notations sont définies page 20.

Cela conduit à une complexité $O^*(\Gamma^n)$, où Γ est la solution minimale de l'inéquation

$$1 \geqslant \sum_{i=1}^{k} \Gamma^{-a_i} \tag{1.2}$$

Des techniques pour améliorer les algorithmes de branchement ou leur analyse sont connues. Nous les détaillons ici.

1. Étude de cas

Cette technique consiste à étudier différents cas possibles au moment du branchement. Elle est spécifique à chaque problème, on en trouvera un exemple dans [27] pour MAX SAT. Ainsi, pour ce dernier problème, considéré du point de vue du nombre m de clauses dans l'instance, on passe progressivement de $T(m) \geqslant T(m-1) + T(m-1)$ (branchement naïf) de complexité $O^*(2^m)$ au branchement $T(m) \geqslant T(m-1)+T(m-2)$ (branchement naïf après élimination des littéraux de fréquence 2) de complexité $O^*(1.7^m)$. L'article cité propose une décomposition en 17 cas dont le pire conduit à une complexité en moins de $O^*(1.32^m)$.

2. Programmation dynamique et mémorisation

La programmation dynamique est apparue dans les années 1940, et consiste à calculer les solutions sur des petites sous-instances, puis à mémoriser les solutions pour ne plus réeffectuer ces calculs. Les solutions sont donc calculées « du bas vers le haut ». Cette technique est utilisée dans des algorithmes polynomiaux aussi bien que des algorithmes exponentiels.

Par exemple, l'algorithme de Levenshtein ([74]) permet de calculer la distance d'édition entre deux chaines de caractère en temps polynomial (quadratique en la taille de la chaine la plus longue).

Dans la partie exponentielle, on peut citer l'algorithme qui permet de résoudre le problème MIN TSP avec une complexité $O^*(2^n)$, alors que le nombre de chemins possibles est $n!$. Pour cela, l'algorithme considère tous les sous-ensembles de sommets à visiter, ordonnés par taille croissante, et calcule pour chacun les chemin les plus court passant par ces sommets, et se terminant par un sommet choisi. Le nombre de telles sous-solutions est exponentiel (borné par $n \times 2^n$), mais le calcul de chaque solution est polynomial.

L'inconvénient d'une telle méthode est qu'elle nécessite beaucoup d'espace de calcul : tous les résultats intermédiaires doivent être stockés. Il y a des exemples d'algorithmes pour lesquels un compromis peut être ajusté entre temps de calcul et mémoire nécessaire.

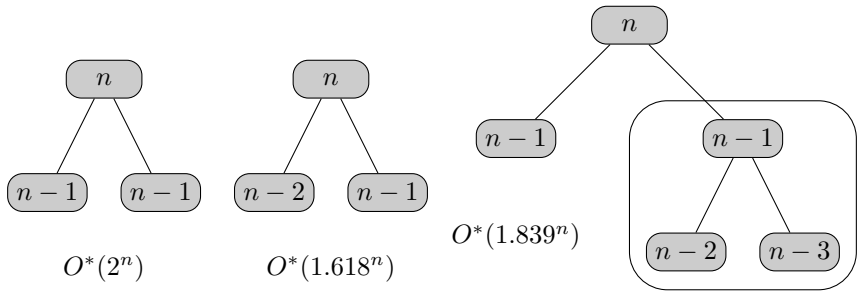
On peut aussi combiner l'approche dynamique avec l'approche par arbre de branchement de la façon suivante : on commence par résoudre (et mémoriser)

un problème donné sur toutes les sous-instances de taille inférieure à un k donné, puis on effectue l'algorithme de branchement sur le graphe initial. L'algorithme de branchement s'arrête plus tôt grâce aux résultats mémorisés, et complète en fouillant dans sa base de solutions. Une application de ce principe se trouve dans [57], où les auteurs fournissent un algorithme de complexité $O^*(1.5263^n)$ en espace polynomial, mais peuvent descendre à $O^*(1.5137^n)$ en espace exponentiel, en mémorisant les résultats au fur et à mesure.

3. Branchements multiples

Pour certains problèmes, on dispose de plusieurs cas de branchement, et le plus mauvais donne une complexité médiocre. Il est parfois possible d'améliorer l'analyse si le pire des cas ne peut pas se produire plusieurs fois de suite. On choisit alors de regarder la complexité sur plusieurs branchements de suite, et on garde à nouveau la pire.

Comme exemple simple, on propose la situation suivante : le problème Π amène deux situations de branchement. La première a deux alternatives, diminuant chacune la taille de l'instance de 1. La deuxième a deux alternatives, diminuant la taille respectivement de 1 et de 2. Les deux branchements ont une complexité respective (d'après 1.2) de $O^*(2^n)$ et $O^*(1.618^n)$. Le pire cas est donc le premier. Cependant, si on sait qu'après un « pire » branchement, une au moins des branches conduira immédiatement au meilleur branchement, on pourra améliorer l'analyse (voir figure 1.9) : le pire branchement possède alors trois branches : l'une dans laquelle la taille diminue de 1, la deuxième dans laquelle la taille diminue de deux, et la troisième dans laquelle la taille a diminué de 3. La complexité est alors $O^*(1.839^n)$.



(a) Deux branchements sont possibles.

(b) Si on enchaine les deux branchements, on a une meilleure complexité.

FIGURE 1.9 – Exemple de branchement multiple.

Cette technique a par exemple été mise en œuvre sous le nom de complexité amortie dans [29].

4. Measure and Conquer

La méthode du *Measure and Conquer* est introduite en 2005 dans [57]. Les auteurs constatent que l'analyse des algorithmes (en particulier des algorithmes de branchement) s'effectue toujours vis-à-vis de mesures simples (typiquement, la taille de l'instance). Ils proposent alors une mesure différente, qui dépend de caractéristiques locales des éléments de l'instance (typiquement, le degré d'un sommet, ou la taille d'un ensemble, ...).

L'analyse des branchements est alors raffinée, et conduit parfois à de larges améliorations dans la complexité. Cela peut se comprendre intuitivement parce que cette nouvelle mesure prend en compte un sommet dont le degré diminuerait (respectivement d'un ensemble dont la taille décroit) après un branchement, choses impossible à prendre en compte si la seule mesure est le nombre de sommets (respectivement le nombre d'ensembles).

Dans l'article cité, cela diminue la complexité pour résoudre le problème MINIMUM INDEPENDENT DOMINATING SET (MIN IDS) de $O^*(2^{0.850n})$ à $O^*(2^{0.598n})$. Cette méthode a par la suite été fructueusement appliquée (voir [55] ou [90] par exemple) à d'autres problèmes.

5. Bottom-up

La méthode du *bottom up* a été utilisée pour améliorer l'analyse d'un algorithme pour MAX IS. Dans ce problème, l'analyse était confrontée à un paradoxe : s'il existe un sommet de degré élevé, on peut brancher avantageusement sur lui (quand on ajoute un sommet, on supprime tous ses voisins, ce qui diminue beaucoup la taille de l'instance).

Inversement, si tous les degrés sont faibles, on dispose d'algorithmes efficaces. Dans [20], les auteurs proposent de tirer avantage de cette situation pour obtenir de meilleurs complexités pour des instances de degré moyen croissant.

1.4.2 Algorithmes approchés

Exigence de garantie

Comme on l'a vu page 9, les algorithmes approchés n'offrent pas tous la garantie de qualité que l'on peut souhaiter. La spécificité de notre approche est de pouvoir borner l'erreur commise par notre algorithme.

La manière classique de le faire est de mesurer le *ratio d'approximation*, à savoir le pire rapport entre une solution proposée par un algorithme donné et la solution optimale pour cette instance.

On notera $I \in P$ une instance du problème P, OPT_I la valeur de sa solution optimale et $\mathscr{S}_I$ la valeur de la solution apportée par l'algorithme. On définit alors le ratio d'approximation ρ pour un problème P de minimisation par :

$$\rho = \max_{I \in P} \frac{\mathscr{S}_I}{\text{OPT}_I} \quad (\rho \geqslant 1)$$

et inversement pour un problème de maximisation :

$$\rho = \min_{I \in P} \frac{\mathscr{S}_I}{\text{OPT}_I} \quad (\rho \leqslant 1)$$

Pour donner un exemple, nous proposons ici un exemple avec l'algorithme 1.4, qui résoud MIN VC avec un ratio d'approximation 2.

ALGORITHME 1.4 – Algorithme approché à 50% pour MIN VC

Entrées: un graphe $G(V,E)$
Sorties: une couverture minimale (ratio 2)

```
S ⟵ ∅
for (i,j) ∈ E do
    if i ∉ S and j ∉ S then
        S ⟵ S ∪ {i,j}
    end
end
Renvoyer S
```

Par construction, cet algorithme renvoie bien une solution au problème posé. Il se trouve qu'il rajoute des sommets d'arêtes non adjacentes dans le graphe (et il en rajoute deux par arête). Il est facile de montrer que pour chacune de ces arêtes, il était nécessaire d'avoir au moins une de ces extrémités dans la solution. On a donc rajouté, dans le pire des cas, deux fois trop d'arêtes. Par conséquent, MIN VC est approximable en temps polynomial avec un ratio 2.

Dans certaines situations, on veut un ratio d'approximation qui dépende de paramètres de l'instance (typiquement, la taille). Ainsi, l'algorithme 1.5 réalise une approximation pour MAX IS avec un ratio $\sqrt{n}$: s'il existe un stable dont la taille est comprise entre $\sqrt{n}$ et n, alors l'algorithme renvoie un stable de taille $\sqrt{n}$ (ce qui assure le bon ratio). Sinon, il renvoie un stable de taille 1, ce qui assure ici encore le ratio demandé. Le nombre de sous-ensembles à estimer est $\binom{\sqrt{n}}{n}$, ce qui conduit à une complexité en $O^*(2^{\sqrt{n}\log(n)})$.

ALGORITHME 1.5 – Algorithme approché avec ratio $\sqrt{n}$ pour MAX IS

Entrées: un graphe $G(V,E)$
Sorties: un stable maximum (ratio $\sqrt{n}$)
for *S sous-ensemble de taille $\sqrt{n}$ de V* **do**
  if *S est stable* **then**
    Renvoyer S
  end
end
Renvoyer un ensemble avec un unique sommet.

Approximation polynomiale et limites

Dans le domaine de l'approximation à ratio fixé, les algorithmes capables de tourner en temps polynomial par rapport à la taille de l'instance (comme l'algorithme 1.4) forment un ensemble très étudié. On dit qu'un problème est **APX** s'il existe un tel algorithme, et on trouve pour certains problèmes des améliorations régulières dans les ratios d'approximation. Je donne la table 1.4, largement inspirée de [4], comme indication des améliorations régulières qu'on obtient dans ce domaine.

TABLE 1.4 – Amélioration des ratios d'approximation pour la résolution de MAX SAT

Année	Ratio	Référence
1974	0.5	[68]
1981	0.618	[76]
1989	0.666	[73]
1992	0.75	[92]
1995	0.7584	[63]
1996	0.765	[6]
1997	0.77	[5]
2002	0.7846	[4]

Cependant, ce domaine connait aussi des résultats négatifs. Pour certains problèmes, en effet, même une résolution approchée à ratio constant est un résultat **NP**-difficile : c'est le cas de MIN TSP, auquel se réduit le problème de

trouver un chemin hamiltonien dans un graphe (il suffit de rendre le graphe complet, en mettant un poids très élevé aux arêtes qu'on rajoute : la résolution du MIN TSP ne peut pas emprunter de telle arête s'il existe un chemin qui s'en prive, c'est-à-dire un chemin hamiltonien).

En 2001, un prix Gödel a été décerné à 9 chercheur-se-s, pour leur travail sur le théorème **PCP**. Ce théorème, qui donne une égalité entre deux classes de complexité, a permis de montrer la difficulté à approcher un grand nombre de problèmes d'optimisation combinatoire. En particulier, sous réserve que **P** $\neq$ **NP**, on peut par exemple montrer que MAX IS est inapproximable à ratio constant (et mieux : on ne peut pas l'approcher pour des ratios variables de la forme $1/n^{1-\varepsilon}$, voir [93]). Une autre conséquence classique est l'inapproximabilité d'une variante de MAXIMUM 3-SATISFIABILITY (MAX 3-SAT) (où toutes les clauses ont exactement trois littéraux) avec un ratio strictement meilleur que $7/8$.

Approximation exponentielle

Devant les résultats négatifs précédents, deux idées naturelles font surface. La première consiste à abandonner l'idée de ratio garanti, et de fournir des algorithmes polynomiaux approchés. Souvent, la qualité de ces algorithmes est mesurée via des expérimentations sur des problèmes concrets, le pire des cas étant soit très mauvais, soit trop difficile à évaluer précisément.

L'idée que nous avons suivie et à laquelle cette partie est consacrée consiste à conserver la garantie exigée, mais à renoncer à avoir des algorithmes polynomiaux. À la place, des algorithmes exponentiels sont utilisés, mais avec une complexité asymptotique meilleure que celle de la résolution exacte.

Un premier résultat dans ce domaine est donné dans [37], sur le problème MAX SAT. Par la suite, cette thématique a été explorée et appliquée à des problèmes variés pendant la thèse de Nicolas Bourgeois (MAX IS, MIN VC, SET COVER, voir [18], [17] or [16]).

Un algorithme est alors présenté comme un couple (ρ,γ), signifiant qu'on sait obtenir un ratio d'approximation ρ avec une complexité $O^*(\gamma^n)$. Il est souvent pratique de présenter ces couples sous forme d'un graphique, comme on en trouvera dans les pages suivantes.

Propriétés héréditaires

Un des résultats d'approximabilité modérément exponentielle est la notion de *splitting*, qu'on peut utiliser sur des problèmes concernant des ensembles avec des propriétés héréditaires.

Définition 1.3 (Propriété héréditaire)**.** Une propriété π est héréditaire si tout sous-ensemble d'un ensemble vérifiant π vérifie également π.

Ainsi, la propriété d'être indépendant pour un ensemble de sommets d'un graphe est bien héréditaire : tout sous-ensemble d'un ensemble indépendant est lui-même indépendant. On a alors le théorème suivant, dans [19] :

Théorème 1.4. Soit le problème P, qui consiste à trouver un sous-ensemble maximum vérifiant une propriété héréditaire π, et $\mathscr{A}$ un algorithme qui fonctionne en $O^*(\gamma^n)$ pour résoudre ce problème.

Pour tout $\rho \in \mathbb{Q}$, $\rho \leqslant 1$, on peut trouver un algorithme donnant une solution ρ-approchée, avec une complexité $O^*(\gamma^{\rho n})$.

Démonstration. La preuve est constructive : une analyse de l'algorithme 1.6 montre qu'il répond à la question pour un ratio $1/2$. Le détail de la preuve ainsi que sa généralisation sont donnés dans le chapitre 2. □

ALGORITHME 1.6 – Algorithme approché (ratio $1/2$) pour le problème P

Entrées: un ensemble E de taille n
Sorties: un sous-ensemble maximum de E vérifiant π (ratio $1/2$)
1 Partager E en deux ensembles E_1 et E_2 de taille identique.
2 Appliquer l'algorithme A sur E_1 et stocker la solution S_1.
3 Appliquer l'algorithme A sur E_2 et stocker la solution S_2.
4 Renvoyer la plus grande solution entre S_1 et S_2.

On peut représenter ce résultat sous forme de graphique, comme sur la figure 1.10

Notion de schémas

Dans le cas précédent, on a la chance d'avoir affaire à un problème pour lequel on peut facilement ajuster la complexité permise dans l'algorithme pour obtenir le ratio d'approximation voulu. On définit ainsi la notion de schéma exponentiel d'approximation :

Définition 1.5. Un problème de maximisation admet un schéma exponentiel d'approximation si, pour chaque ratio d'approximation ρ, on peut trouver un algorithme ρ-approché avec une complexité asymptotique meilleure que la complexité d'un algorithme exact (y compris à polynome près).

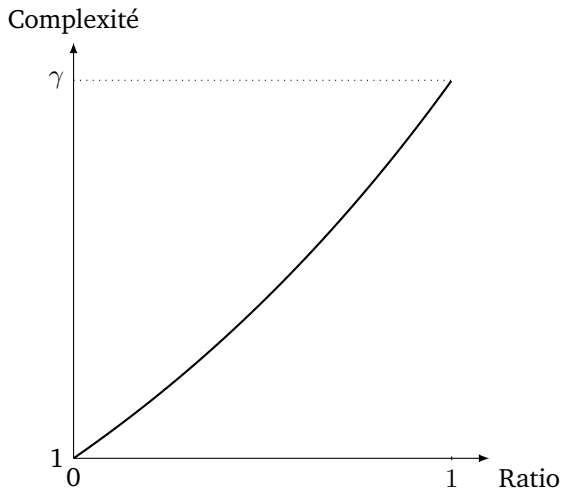


FIGURE 1.10 – Analyse de la méthode de *splitting* pour un problème pour lequel on dispose d'un algorithme exact de complexité $O^*(\gamma^n)$.

Cette définition s'inspire de la complexité polynomiale, et en particulier de la classe **PTAS**. Formellement, l'existence d'un schéma assure de gagner en complexité dès qu'on relâche le ratio d'approximation. Dans la pratique, on peut parfois espérer « relier » de manière continue un algorithme exact exponentiel à un algorithme polynomial approché à ratio garanti.

Un mot sur les tests pratiques

Contrairement à une grande partie des publications en approximation sans garantie, les articles constituant cette thèse ne contiennent pas de tests pratiques des algorithmes proposés, mais uniquement des démonstrations formelles de leur efficacité. Il y a principalement deux raisons à ça :

a) Tout d'abord, on pourrait envisager d'utiliser de tels résultats expérimentaux pour améliorer l'analyse des algorithmes. Ainsi, on pourrait imaginer disposer d'une courbe de complexité d'un de nos algorithmes, constater qu'elle « ressemble » à une courbe de la forme $O^*(\gamma^n)$, et se servir de cette intuition pour améliorer notre analyse de l'algorithme. Malheureusement, ce serait utopique : il est difficile d'obtenir un jeu de données suffisant pour des tailles très variées, parce que les tailles critiques des problèmes qu'on peut considérer sont

assez faibles. Pour ces tailles, les facteurs polynomiaux que nous masquons en complexité asymptotique ne sont pas du tout négligeables, et brouilleraient ces mesures.

b) Ensuite, on serait tenté d'obtenir de tels résultats expérimentaux pour confronter nos algorithmes approchés à d'autres algorithmes existants, pour montrer que les nôtres sont plus rapides. Malheureusement, ici aussi, nous n'espérons pas grand chose d'une telle mise en pratique : les algorithmes que nous produisons sont très simplifiés pour permettre une analyse efficace. Il faudrait leur ajouter des heuristiques, des coupes, des améliorations pour les rendre compétitifs, et cela demanderait un travail complet (voir par exemple [3] comme exemple de tel travail sur le MIN VC). Bien que très utile, cette approche sortait du cadre de cette thèse, et nous avons choisi, pour leur lisibilité, de réduire les algorithmes à leur plus simple expression.

1.4.3 Algorithmes paramétriques

Paramétrisation et classe FPT

Un dernier aspect pour conclure cette introduction concerne les algorithmes paramétriques. Comme leur nom l'indique, dans ce domaine, on considère des instances d'un problème avec un paramètre particulier. Celui-ci peut-être le degré maximum, ou minimum des sommets d'un graphe, ou bien la taille de la solution espérée (on parle alors de paramètre standard), ou même la taille de la solution à un autre problème d'optimisation (on parlera en particulier du *treewidth* d'un graphe, ou de la taille d'une couverture comme paramètre).

Le but est alors d'exprimer la complexité d'un algorithme non plus simplement en fonction de la taille de l'instance n (complexité classique), mais également en fonction du paramètre k. On espère ainsi avoir une complexité meilleure, au moins pour certaines valeurs de k.

Illustrons cela avec un exemple. Considérons le problème MAX IS avec le paramètre standard. Il s'agit alors d'un problème de décision : « Peut-on, dans un graphe donné de taille n, trouver un ensemble indépendant de taille k ? ». Un algorithme naïf qui consiste à énumérer tous les sous-ensembles de taille k, soit $\binom{k}{n}$ sous-ensembles. On peut majorer grossièrement ce nombre et obtenir une complexité $O^*(n^k)$, ce qui donne une complexité qui peut être plus intéressante que le $O^*(2^n)$ de l'algorithme en complexité classique. On dit en l'occurence que le problème MAX IS muni du paramètre standard est dans la classe **XP**.

Définition 1.6 (XP). Un problème paramétrique P est dans **XP** s'il existe un algorithme polynomial résolvant P en temps polynomial pour toute valeur fixée

du paramètre.

Un grand nombre de problèmes paramétriques sont dans **XP**, c'est le cas de bien des problèmes à cardinalité fixée munis de leur cardinal comme paramètre (par exemple, MAX K-COVERAGE muni de k).

À l'inverse, le problème de coloration d'un graphe n'est pas dans **XP** : détecter une coloration, même à trois couleurs, est **NP**-difficile dans le cas général (et on ne dispose donc pas d'algorithme polynomial). De même, en se servant du fait que MAX IS est **NP**-difficile même dans des graphes de degré 3, on en conclut que MAX IS, muni du degré du graphe n'est pas dans **XP**.

Dans [44], les auteurs introduisent une autre classe essentielle à la complexité paramétrique :

Définition 1.7 (FPT). Un problème paramétrique est dans **FPT** (pour Fixed Parameter Tractable) s'il existe un algorithme résolvant une instance de taille n avec un paramètre k en temps $O(f(k)p(n))$, où p est une fonction polynomiale.

Cette définition est donc plus restreinte que **XP** : le degré du polynôme ne peut pas varier en fonction du paramètre. On peut donner comme exemple l'algorithme 1.7 pour le problème MIN VC. L'analyse de cet algorithme est simple : il cherche une couverture minimale en procédant arête par arête. La décroissance du paramètre impose de relancer l'étape 3 au plus 2^k fois, ce qui assure un temps de calcul en $O^*(2^k)$ au total (le reste des opérations est polynomial).

Plus généralement, l'intuition qui se trouve souvent derrière l'appartenance à la classe **FPT**, c'est l'existence d'un algorithme de branchement tel que chaque nœud a un nombre borné de branches, et chaque branche assure une augmentation de la solution. Dans l'exemple du MIN VC, notre algorithme considère une arête, et on sait qu'une de ses deux extrémités sera dans la solution.

Il existe cependant une autre caractérisation de la classe **FPT**, que je présente maintenant.

Kernelisation et hiérarchie W[]

Cette autre caractérisation s'appuie sur la notion de Kernel.

Définition 1.8. Un problème paramétrique admet un *kernel* si toute instance de taille n (avec un paramètre k) peut se réduire en temps polynomial à une autre instance dont la taille est bornée par une fonction de k.

Dit autrement, la kernelisation est le procédé qui réduit un problème à son « noyau de difficulté » de faible taille.

ALGORITHME 1.7 – Algorithme paramétrique pour MIN VC

Entrées: un ensemble $G(V,E)$, un paramètre k
Sorties: « G admet-il une couverture de taille k ? »

$S \leftarrow \emptyset$
$c \leftarrow k$
tant que *Il reste des arêtes dans* E *et* $c \geqslant 0$ **faire**
  Choisir une arête (i,j) dans E
  si $i \in S$ *ou* $j \in S$ **alors**
    Retirer (i,j) de E
  sinon
    Relancer l'étape 3 avec $S \leftarrow S \cup \{i\}$ et $c \leftarrow c-1$
    Relancer l'étape 3 avec $S \leftarrow S \cup \{j\}$ et $c \leftarrow c-1$
  fin
fin
si *Une des branches de l'algorithme termine avec* $E = \emptyset$ **alors**
  Renvoyer **Oui**
sinon
  Renvoyer **Non**
fin

Théorème 1.9. Un problème est **FPT** si et seulement si il admet un kernel.

On trouvera la preuve de ce théorème dans [44], par exemple. Cependant, le théorème ne dit rien sur la taille du kernel par rapport au paramètre. Un pan important de la complexité paramétrique s'articule autour des tailles de kernels (savoir si un problème admet un kernel polynomial par exemple, voir [14]).

Il existe, enfin, un ensemble de classes de complexité, les classes **W[1]**, **W[2]**, ... Ces classes[22] croissantes pour l'inclusion, sont préservées par un type particulier de réductions (les ***FPT**-réductions*, qui sont polynomiales et préservent le paramètre). Leur intérêt réside dans l'existence de problèmes difficiles suivant ces réductions, vers lesquels on peut réduire d'autres problèmes pour conclure à la non-existence d'algorithme **FPT**. Ainsi, MAX IS est **W[1]**-difficile, MIN DS est **W[2]**-difficile, ... Je renvoie à nouveau vers [44] pour plus de détails.

[22] La classe **W[i]** est définie comme l'ensemble des problèmes qu'on peut réduire à la satisfiabilité d'un circuit électronique utilisant moins de i portes logiques à nombre non-borné d'entrées sur chaque chemin, et moins de k entrées mises à 1.

1.5 Plan de cette thèse

La partie que vous venez de lire était une introduction au domaine. Le reste de ce manuscrit est constitué des différents articles rédigés ou publiés au cours de ma thèse, développés et adaptés à la mise en page de ce manuscrit. On retrouvera en particulier dans les chapitres successifs :

- Des algorithmes modérément exponentiels pour approcher le problème MAX SAT.
- Une formalisation des classes d'approximation exponentielle, une méthode générale pour produire des algorithmes d'approximation, et des exemples d'application de cette méthode.
- Une étude de la complexité paramétrique du problème $(k,n-k)$-CUT (K,N-K CUT).

2 Approximating MAXIMUM SATISFIABILITY

I can't get no satisfaction.

Rolling Stones, 1965

Dans ce premier chapitre, nous considérons le problème MAX SAT pour lequel nous donnons des algorithmes approchés modérément exponentiels. L'objectif général est de dépasser les résultats d'inapproximabilité polynomiale avec des temps de calcul meilleurs que ceux des algorithmes exacts. Nos algorithmes permettent d'obtenir des ratios d'approximation aussi proche que l'on veut de 1, le temps de calcul dépendant du ratio voulu.

We study approximation of the MAX SAT problem by moderately exponential algorithms. The general goal of the issue of moderately exponential approximation is to catch-up on polynomial inapproximability, by providing algorithms achieving, with worst-case running times importantly smaller than those needed for exact computation, approximation ratios unachievable in polynomial time. We develop several approximation techniques that can be applied to MAX SAT in order to get approximation ratios arbitrarily close to 1.

Contents

2.1 Introduction and notations

Optimum satisfiability problems are of great interest from both theoretical and practical points of view. Let us only note that several subproblems of MAXIMUM SATISFIABILITY (MAX SAT) and MINIMUM SATISFIABILITY (MIN SAT) are among the first complete problems for many approximability classes [7, 8, 49, 91]. On the other hand, in many fields (including artificial intelligence, database system, mathematical logic, ...) several problems can be expressed in terms of versions of SATISFIABILITY (SAT) [10].

Satisfiability problems have in particular drawn a major attention in the field of polynomial time approximation as well as in the field of parameterized and exact solution by exponential time algorithms. Our goal in this paper is to develop approximation algorithms for MAX SAT with running times which, though being exponential, are much lower than those of exact algorithms, and with a better approximation ratio than the one achieved in polynomial time. This approach has already been considered for MAX SAT in [37], where interesting tradeoffs between running time and approximation ratio are given. It has also been considered for several other well known problems such as MINIMUM SET COVER (SET COVER) [18, 35], MINIMUM COLORING (MIN COLOR) [13, 17], MAXIMUM INDEPENDENT SET (MAX IS) and MINIMUM VERTEX COVER (MIN VC) [16], MINIMUM BANDWIDTH (MIN BW) [36, 60], etc. Similar issues arise in the field of FPT algorithms, where approximation notions have been introduced, for instance, in [24, 45]. In this article, we propose several improvements of the result of [37] using various algorithmic techniques.

Given a set of variables and a set of disjunctive clauses, MAX SAT consists of finding a truth assignment for the variables that maximizes the number of satisfied clauses. In what follows, we denote by $X = \{x_1, x_2, \ldots, x_n\}$ the set of variables and by $C = \{C_1, C_2, \ldots, C_m\}$ the set of clauses. Each clause consists of a disjunction of literals that are either a variable x_i or the negation of a variable $\neg x_i$. A ρ-approximation algorithm for MAX SAT (with $\rho < 1$) is an algorithm that finds an assignment satisfying at least a fraction ρ of the maximal number of simultaneously satisfied clauses. The best known ratio guaranteed by a polynomial time approximation algorithm is $\alpha = 0.784$ obtained in [4].

Dealing with exact solution, [27] gives an exact algorithm working in time $O^*(1.3247^m)$, which is the best known bound so far. Dealing with the number of variables, the trivial $O^*(2^n)$ bound has not yet been broken down, and this constitutes one of the main open problems in the field of exact exponential algorithms[1]. The parameterized version of MAX SAT consists, given a set of clauses C and an integer k, of finding a truth assignment that satisfies at least k clauses, or to output an error if no such assignment exists. In [27] the authors give a parameterized algorithms for MAX SAT running in time $O^*(1.3695^k)$. Intuitively, the first algorithm is faster because it benefits from the empty clauses that are removed from the instance, whereas the parameterized algorithm can only take into account satisfied clauses.

Using the same notation as in [27], we say that that a variable x is an (i, j)-variable if it occurs positive in exactly i clauses and negative in exactly j clauses. For any instance C of MAX SAT, we will denote by OPT(C) (or OPT if no ambiguity occurs) an optimal set of satisfied clauses. We use notation $f(n) = O^*(g(n))$, if and only if $\exists c \in \mathbb{R}, f(n) = O(g(n)n^c)$. Finally, we denote by α the ratio guaranteed by a polynomial time approximation algorithm. In general, ρ will denote the approximation ratio of an algorithm, and, when dealing with exponential complexity, γ will be the basis of the exponential expressing it.

In order to fix ideas, let us give the first simple algorithm 2.1, useful to understand some of our results. In particular, it is one of the basic stones of the results in [37]. It is based upon the following two well known reduction rules. It is illustrated on figure 2.1.

Rule 2.1. Any clause containing an $(h, 0)$- (resp. a $(0, h)$-literal), $h \geqslant 1$, can be removed from the instance. This is correct because we can set this literal to true (resp. false) and satisfy the clauses that contain it.

Rule 2.2. Any $(1, 1)$-literal can be removed too. Let $C_1 = x_1 \vee x_2 \vee \cdots \vee x_p$ and $C_2 = \neg x_1 \vee x'_2 \vee \cdots \vee x'_q$ be the only two clauses containing the variable x_1. If there exist two opposite literals in C_1 and C_2, then we can satisfy these clauses by choosing a correct value for x_1 and therefore we can remove these clauses. Otherwise, we can replace these clauses by $C = x_2 \vee \cdots \vee x_p \vee x'_2 \vee \cdots \vee x'_q$. The optimum in the initial instance is the optimum in the reduced instance plus 1.

To evaluate the complexity of Algorithm 2.1, we count the number of leaves in the tree. Note that if the number of leaves is $T(n)$, then the algorithm

[1] This is known as the Strong Exponential Time Hypothesis

ALGORITHM 2.1 – A simple branching algorithm for MAX SAT

Input : An instance for the MAX SAT problem
Output : An optimal truth assignment for this instance

1 Build a tree as follows : each node is labeled with an instance of MAX SAT. The root is the initial instance. The empty instances are the leaves. For each node whose label is a non-empty sub-instance, if one of the reductions above applies, then the node has one child labeled with the resulting (reduced) sub-instance. Else, a variable x is arbitrarily chosen and the node has two children: in the first one, the instance has been transformed by setting x to false (the literals $\neg x$ have been removed and the clauses containing the literal $\neg x$ are satisfied); in the second one, x is set to true and the contrary happens.
2 As each node represents a partial truth assignment, an optimal solution is a truth assignment corresponding to a leaf that has the largest number of satisfied clauses.
3 Output this optimal solution.

obviously works in time $O^*(T(n))$. In the sequel, in order to simplify notations we will use $T(n)$ to denote both the number of leaves (when we express recurrences) and the complexity. There are two ways to count the number of leaves. The former is by means of the variables. Each node has two children for which the number of unaffected variables decreases by 1. This leads to a number of leaves $T(n) \leqslant 2 \times T(n-1)$ and therefore $T(n) = O^*(2^n)$. The second is by means of the clauses. On each node, if the chosen variable is an (i, j)-variable, then the first child will have its number of clauses decreased by at least i and the second child by at least j. The worst case, using the two reduction rules given above, is $i = 1$ and $j = 2$ (or the contrary), that leads to $T(m) = T(m-1) + T(m-2)$ and therefore $T(m) = O^*(1.618^m)$.

In [37], the authors showed a way to transform any polynomial time approximation algorithm (with ratio α) into an approximation algorithm with ratio ρ (for any $\alpha \leq \rho \leq 1$) and running time $O^*(1.618^{(\rho-\alpha)(1-\alpha)^{-1}m})$. The basic idea of this algorithm is to build the same tree as in Algorithm 2.1 up to the fact that we stop the branching when enough clauses are satisfied. Then the α-approximation polynomial algorithm is applied on the resulting sub-instances. As already mentioned, the best value of α is 0.784 [4].

The paper is organized as follows. In Section 2.2 two first results are presented: the first one uses the same technique as in [37] while the second

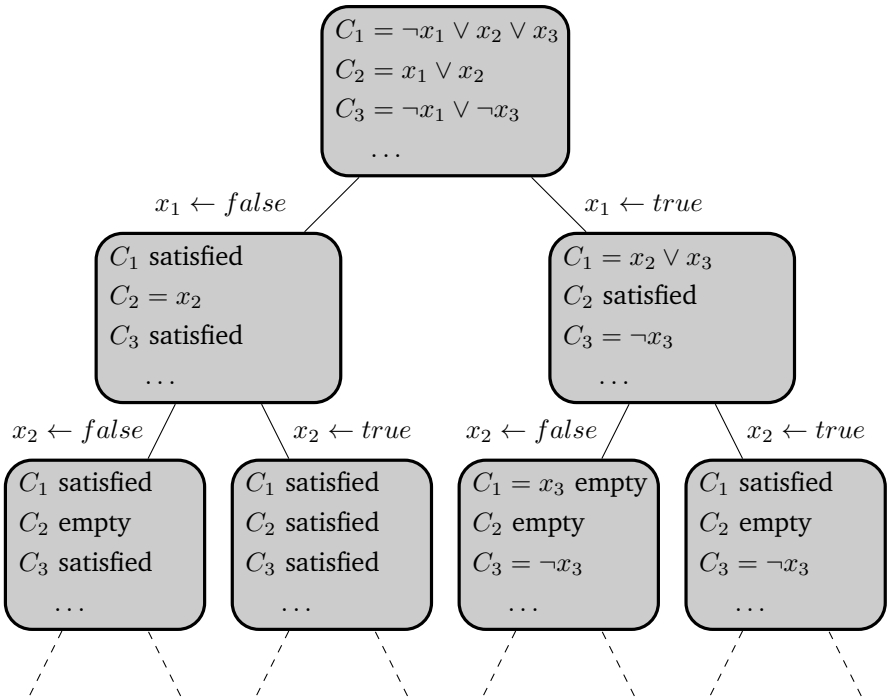


Figure 2.1: A simple branching algorithm.

one uses a different approach consisting of splitting the instance in "small" sub-instances. In Section 2.3, we further improve these results for some ratios using another technique consisting of approximately pruning a search tree. All these results deal with complexity depending on the number of clauses. In Section 2.4, we obtain the same kind of results when we are interested in running times that depend on the number of variables. We conclude the article in Section 2.5 where we also briefly discuss the MIN SAT problem.

2.2 First results

We provide in this section two first improvements of the result given in [37]. The first one, given in Section 2.2.1, uses the same idea as [37] while the second one uses a completely different technique and achieve improved running times

(for some approximation ratios) by splitting the initial instance in sub-instances of smaller size.

2.2.1 Using a better parameterized algorithm

In this section we briefly mention that the same technique as in [37] leads to an improved result when we build the search tree according to the algorithm from [27] instead of the branching tree presented in Section 2.1. We so derive the following algorithm, that is strictly better than the one of [37]. This is illustrated on figure 2.2.

ALGORITHM 2.2 – Improving on [37]

Input : An instance for the MAX SAT problem
Output : An truth assignment wich achieve an approximation ratio ρ

1 Build a search-tree as the parameterized algorithm of [27] does. Stop the development of this tree at each node where at least $(m(\rho-\alpha)/(1-\alpha))$ clauses are satisfied (α is the best known polynomial approximation ratio for MAX SAT), or when the instance is empty.
2 For each leaf of this tree, apply a polynomial α-approximation algorithm to complete the assignment of the remaining variables
3 Thus, each leaf of the tree corresponds to a complete truth assignment. Return the assignment satisfying the largest number of clauses.

Proposition 2.3. For any ρ such that $\alpha \leq \rho \leq 1$, Algorithm 2.2 achieves approximation ratio ρ in time $O^*(1.3695^{m(\rho-\alpha)/(1-\alpha)})$.

Proof. Consider first the running time. The parameterized algorithm of [27] builds a search tree where the worst case recurrence relation is $T(k) \leq 2T(k-3)+2T(k-7)$, where the parameter k is the number of satisfied clauses, leading to a global complexity of $O^*(1.3695^k)$. Here, we build this tree and stop the construction in each leave where $m(\rho-\alpha)/(1-\alpha)$ clauses are satisfied. This leads to a running time of $O^*(1.3695^{m(\rho-\alpha)/(1-\alpha)})$.

We now handle the approximation ratio. First, if the number |OPT| of the clauses satisfied by an optimal solution OPT is less than $m(\rho-\alpha)/(1-\alpha)$, then Algorithm 2.2 obviously finds an optimum solution. Otherwise, let us consider the branch of the branching tree where the leaf corresponds to a partial optimal truth assignment satisfying clauses in OPT. Denote by k_0 the number of clauses satisfied in this leaf ($k_0 \geqslant m(\rho-\alpha)/(1-\alpha)$), i.e. by the partial assignment

corresponding to this leaf. Using this assignment, we get a resulting instance in which it is possible to satisfy $|\text{OPT}| - k_0$ clauses (because the optimal assignment satisfies $|\text{OPT}|$ clauses). Consequently, the α-approximation algorithm called by Algorithm 2.2 will satisfy at least $\alpha\,(|\text{OPT}| - k_0)$ more clauses. So, finally, at least $k_0 + \alpha\,(|\text{OPT}| - k_0) = k_0(1-\alpha) + \alpha|\text{OPT}| \geqslant m\,(\rho - \alpha) + \alpha|\text{OPT}| \geqslant |\text{OPT}|(\rho - \alpha) + \alpha|\text{OPT}| = \rho|\text{OPT}|$ clauses will be satisfied. □

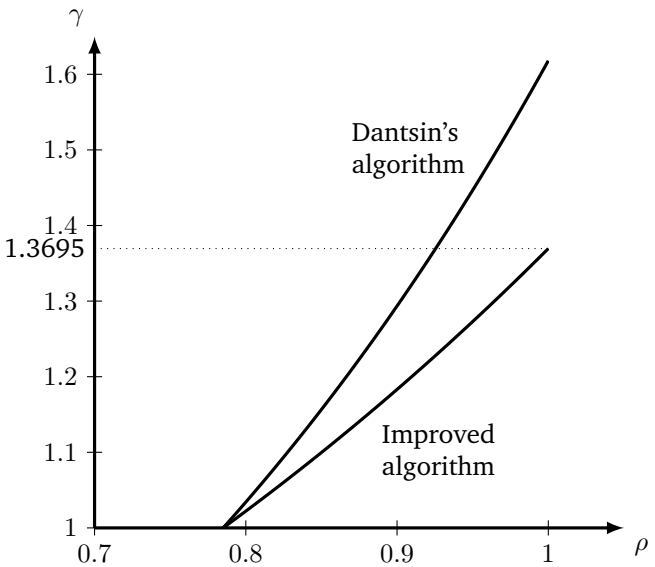


Figure 2.2: Comparison between the algorithm of [37] and Algorithm 2.2.

2.2.2 Splitting the clauses

In [16, 15], it is shown that a generic method can give interesting moderately exponential approximation algorithms if applied in (maximization) problems satisfying some hereditary property (a property is said to be hereditary if for any set A satisfying this property, and any $B \subset A$, B satisfies this property too).

MAX SAT can be seen as searching for a maximum subset of clauses satisfying the property "can be satisfied by a truth assignment", and this property is clearly hereditary. Therefore, we can adapt the splitting method introduced in [16, 15]

to transform any exact algorithm into a ρ-approximation algorithm, for any rational ρ, and with running time exponentially increasing with ρ.

ALGORITHM 2.3 – Splitting the clauses

Input : An instance for the MAX SAT problem
Output : An truth assignment wich achieve an approximation ratio ρ
1 Let p, q be two integers such that $\rho = p/q$.
2 Split the set of clauses into q pairwise disjoint subsets $A_1, \ldots, A_q$ of size m/q (at most $\lceil m/q \rceil$).
3 Consider the q subsets $S_i = A_i \cup A_{i+1} \cup \cdots \cup A_{i+p-1}$ (if the index is larger than q, take it modulo q).
4 **for** *every subset* S_i **do**
5 | Apply some exact algorithm for MAX SAT on S_i.
6 **end**
7 Return the best assignment among them as solution for the whole.

Proposition 2.4. Given an exact algorithm for MAX SAT running in time $O^*(\gamma^m)$, Algorithm 2.3 achieves approximation ratio ρ in time $O^*(\gamma^{\rho m})$.

Proof. Algorithm 2.3 calls q times an exact algorithm (whose running time is $O^*(\gamma^m)$). Then the bound of the running time easily follows from the fact that each subset S_i contains at most $p\lceil m/q \rceil \leq \rho m + p$ clauses.

For the approximation ratio, note first that if we restrict an instance with a set C of clauses to a new instance with a new set $C' \subset C$ of clauses, then an optimal solution for C' satisfies at least the same amount of clauses in C' than an optimal solution for C (in other words, the restriction of any solution for C to C' is feasible for C'), i.e., $|\text{OPT}(C) \cap C'| \leqslant |\text{OPT}(C')|$. In our case, for $i = 1, \ldots, q$, $|\text{OPT}(S_i)| \geqslant |\text{OPT}(C) \cap S_i|$[2].

Now, by construction of the S_i's, we easily see that each clause appears in exactly p among the q subsets $S_1, S_2, \ldots, S_q$, and this holds in particular for any clause in OPT. Thus, $\sum_{i=1}^{q} |\text{OPT}(C) \cap S_i| = p \times |\text{OPT}(C)|$. Using the argument above, $\sum_{i=1}^{q} |\text{OPT}(S_i)| \geqslant p \times |\text{OPT}(C)|$. Since $\sum_{i=1}^{q} |\text{OPT}(S_i)| \leqslant q \times \max_{i=1}^{q} |\text{OPT}(S_i)|$, then $\max_{i=1}^{q} |\text{OPT}(S_i)| \geqslant \frac{p}{q} |\text{OPT}(C)|$.

□

It is worth noticing that Algorithm 2.3 is faster than Algorithm 2.2 for ratios close to 1 (see Figure 2.3 in Section 2.3).

[2] We denote here by OPT(C) the optimal set of clauses for a particular instance of MAX SAT.

2.3 Approximating by pruning the search tree

Informally, the idea of the technique discussed in this section is based upon the fact that, if we seek, say, a $1/2$-approximation for a maximization problem, then when a search-tree based algorithm selects a particular datum d for inclusion in the solution, one may remove one other datum d' from the instance (without, of course, including it in the solution). At worst, d' is part of an optimal solution and is lost by our solution. Thus, globally, the number of data in an optimum solution is at most two times the number of data in the built solution. On the other hand, with the removal of d', the size of the surviving instance is reduced not by 1 (due to the removal of d) but by 2. This method have been studied more widely in the chapter 3.

This method can be adapted to MAX SAT in the following way: revisit Algorithm 2.1 and recall that its worst case with respect to m is to branch on a $(1,2)$-literal and to fix 1 (satisfied) clause on the one side and 2 (satisfied) clauses on the other side. If we decide to also remove 1 more clause (arbitrarily chosen) in the former case and 2 more clauses (arbitrarily chosen) in the latter one, this leads to a running time $T(m)$ satisfying $T(m) \leqslant T(m-2)+T(m-4)$, i.e., $T(m) \leqslant O^*(1.27^m)$. Since in the branches we have satisfied at least $s \geq 1$ clause (resp., $s \geq 2$ clauses) while the optimum satisfies at most $s+1$ clauses (resp., $s+2$ clauses), we get an approximation ratio 0.5.

This basic principle is not sufficient to get interesting result for MAX SAT, but it can be improved as follows. Let us consider the left branch where the $(1,2)$-literal is set to true, satisfying a clause C_1. Instead of throwing away one other clause, we pick two clauses C_2 and C_3 such that C_2 contains a literal ℓ and C_3 contains the literal $\neg\ell$, and we remove these two clauses. Any truth assignment satisfies either C_2 or C_3, meaning that in this branch we will satisfy at least 2 clauses (C_1 and one among C_2 and C_3), while at worst the optimum will satisfy these three clauses. In the other branch where 2 clauses are satisfied, we pick two pairs of clauses containing opposite literals and we remove them. This trick improves both the approximation ratio and the running time: now we have an approximation ratio $2/3$ (2 clauses satisfied among 3 clauses removed in one branch, 4 clauses satisfied among 6 clauses removed in the other branch), and the running time satisfies $T(m) \leqslant T(m-3)+T(m-6)$, i.e., $T(m) = O^*(1.17^m)$.

In what follows, we generalize the ideas sketched above in order to work for any ratio $\rho \in \mathbb{Q}$.

Note that it might be the case that at some point it is impossible to find p pairs of clauses with opposite literals. But this means that (after removing $q < p$ pairs) each variable appears only positively or only negatively, and the

ALGORITHM 2.4 – Pruning the search tree with basic branching rules

Input : An instance for the MAX SAT problem
Output : An truth assignment wich achieve an approximation ratio ρ

1 Let p, q be two integers such that $\frac{p}{q} = \frac{\rho-1}{1-2\rho}$.
2 Build the search tree of algorithm 2.1 and on every node, add a label counting the number of clauses satisfied since the root (we will *not* count here the clauses removed).
3 Each time a node's label reaches a multiple of q, pick p pairs of clauses with opposite literals and remove them from the remaining sub-instance.
4 If a node has an empty subinstance, but some variables have no truth assignment, then pick up one at random.
5 Finally, return the best truth assignment among the leaves of the tree.

remaining instance is clearly easily solvable in linear time.

Theorem 2.5. Algorithm 2.4 satisfies at least $\rho|\text{OPT}|$ clauses and runs in time $O^*(1.618^{m(2\rho-1)})$.

Proof. Consider the leaf where the variables affected in step 2 of algorithm 2.4 are set like in an optimum solution. In this leaf, assume that the number of satisfied clauses is $s \times q + s'$ (where $s' < q$) - again, we do not count the clauses that have been arbitrarily removed. Then, the algorithm has removed $s \times 2p$ clauses arbitrarily, among which at least $s \times p$ are necessarily satisfied. In the worst case, the $s \times p$ other clauses are in OPT; hence, $|\text{OPT}| \leqslant 2sp + sq + s'$. So, the approximation ratio of Algorithm 2.4 is at least: $(sq+sp+s')/(sq+2sp+s') \geq \rho$.

We now estimate the running time of Algorithm 2.4. For each node i of the tree, denote by m_i the number of clauses left in the surviving sub-instance of this node, by z_i the number of satisfied clauses since the root of the tree (we do not count the clauses that have been arbitrarily removed) and set $t_i = m_i - (2p/q)(z_i \mod q)$.

For the root of the tree, $z_i = 0$ and therefore $t_i = m$. Let i be a node with two children j (at least one clause satisfied) and g (at least two clauses satisfied). Let us examine quantity t_j when exactly one clause is satisfied. In this case, $z_j = z_i + 1$. On the other hand: i) If $z_j \mod q \neq 0$, then we do not have reached the threshold necessary to remove the $2p$ clauses. Then, $m_j = m_i - 1$ and $t_j = m_j - 2p/q(z_j \mod q) = m_i - 1 - 2p/q((z_i$

$\mod q) + 1) = t_i - 1 - 2p/q$. If $z_j \mod q = 0$, then $z_i \mod q = q - 1$ and the threshold has been reached; so $2p$ clauses have been removed. Then, $m_j = m_i - 1 - 2p$, $t_j = m_j = m_i - 1 - 2p$ and $t_i = m_i - 2p/q(q-1) = m_i - 2p + 2p/q$. Finally, $t_j = t_i - 1 - 2p/q$. Therefore, in both cases i) and ii), $t_j \leqslant t_i - 1 - 2p/q$. Of course, by a similar argument, if we satisfy g clauses, then the quantity t_i is reduced by $g(1 + 2p/q)$. This leads to a running time $T(t) \leqslant T(t - 1 - 2p/q) + T(t - 2 - 4p/q)$ and hence $T(t) = 1.618^{t/(1+2p/q)}$. Since initially $t = m$, we get $T(m) = 1.618^{m/(1+2p/q)}$. Taking into account that $p/q = (\rho - 1)/(1 - 2\rho)$, we get immediately $1/(1 + 2p/q) = 2\rho - 1$. □

Algorithm 2.4 can be improved if instead of using the simple branching rule in the tree, the more involved case analysis of [27] is used. This derives the following algorithm 2.5.

ALGORITHM 2.5 – Pruning the search tree with efficients branching rules

Input : An instance for the MAX SAT problem
Output : An truth assignment wich achieve an approximation ratio ρ

1 Let p, q be two integers such that $\frac{p}{q} = \frac{\rho-1}{1-2\rho}$.
2 Build the search tree of [27] and on every node, add a label counting the number of clauses satisfied since the root (we will *not* count here the clauses removed).
3 Each time a node's label reaches a multiple of q, pick p pairs of clauses with opposite literals and remove them from the remaining sub-instance.
4 If a node has an empty subinstance, but some variables have no truth assignment, then pick one for them at random.
5 Finally, return the best truth assignment among the leaves of the tree.

To estimate the running time of Algorithm 2.5, we use nearly the same analysis as in [27]. The only difference is that, at each step of the branching tree, [27] counts without distinction the satisfied and the unsatisfied clauses (because left empty), whereas we have to make a difference in the complexity analysis: a satisfied clause reduces the quantity t by $1 + 2p/q$ in Algorithm 2.5, while an unsatisfied clause reduces it by only 1.

We illustrate this analysis for the case 4.2 of [27] ("there is $(2, 2)$-literal x that occurs at least once as a unit clause"). In this case, a branching is done on the variable x. On the one side, 2 clauses are satisfied while, on the other side, 2 are satisfied and 1 becomes empty and thus it is removed from the instance. For [27], this leads to a complexity $T(m) \leqslant T(m-2) + T(m-3)$. For

Algorithm 2.5, this gives $T(m) \leqslant T(m-2-4p/q)+T(m-3-4p/q)$. To simplify these results, set $\chi = 2p/q$[3]. Then $T(m) \leqslant T(m-2-2\chi)+T(m-3-2\chi)$, which leads to $T(m) = O^*(\gamma^m)$ with γ the largest real solution of the equation $\gamma^{2\chi+3} - \gamma - 1 = 0$.

A comparative study between the algorithm of [27] and Algorithm 2.5 is summarized in Table 2.1. Its third column gives equations whose largest real solutions are the worst case running times for Algorithm 2.5.

Table 2.1: Running times for the algorithm of [27] and Algorithm 2.5.

Case	[27]	Algorithm 2.5
4.0 a)	$T(m) = T(m-1)+T(m-5)$	$T(m) = T(m-1-\chi)+T(m-5-4\chi)$
4.0 b)	$T(m) = T(m-1)+T(m-7)+T(m-10)$	$T(m) = T(m-1-\chi)+T(m-7-6\chi)+T(m-10-9\chi)$
4.1	$T(m) = 2T(m-3)$	$T(m) = 2T(m-3-3\chi)$
4.2	$T(m) = T(m-2)+T(m-3)$	$T(m) = T(m-2-2\chi)+T(m-3-2\chi)$
4.3	$T(m) = 2T(m-6)+T(m-2)$	$T(m) = 2T(m-6-6\chi)+T(m-2-2\chi)$
4.4	$T(m) = T(m-3)+T(m-2)$	$T(m) = T(m-3-3\chi)+T(m-2-2\chi)$
4.5	Same as 4.3	
4.6	Same as 4.4	
4.7	$T(m) = 2T(m-5)$	$T(m) = 2T(m-5-5\chi)$
4.8	$T(m) = 2T(m-5)+2T(m-7)$	$T(m) = 2T(m-5-5\chi)+2T(m-7-6\chi)$
4.9 a)	Same as 4.1	
4.9 b)	Same as 4.0 a)	
4.10	$T(m) = T(m-1)+1$	$T(m) = T(m-1)+1$
4.11 a)	$T(m) = 2T(m-4)$	$T(m) = 2T(m-4-4\chi)$
4.11 b)	Same as 4.0 a)	
4.12 a)	Same as 4.0 a)	
4.12 b)	$T(m) = 2T(m-8)+T(m-1)$	$T(m) = 2T(m-8-7\chi)+T(m-1-\chi)$

[3]Intuitively, χ represents the amount of reduction of the instance when a clause is satisfied.

Depending on the ratio ρ we seek, the running time of the algorithm is given by the worst case of all the cases given in Table 2.1. However, one can show that for any ρ the worst case is always reached by the case 4.2. Let us show an example (the other cases are similar or simpler). Consider the equations $f_{4.2}(X) = X^{2\chi+3} - X - 1 = 0$ and $f_{4.0}(X) = X^{4\chi+5} - X^{3\chi+4} - 1 = 0$. The largest real solution of the former is always larger than the largest real solution of the latter one. Indeed, let χ be any positive value. Remark first that $f'_{4.0}(X) = (4\chi+5)X^{4\chi+4} - (3\chi+4)X^{3\chi+3} > 0$; hence, function $f_{4.0}$ is increasing with $X > 1$. What we now need to show is that if $f_{4.2}(X) = 0$, then $f_{4.0}(X) \geqslant 0$ (this means that the zero of $f_{4.0}$ is before that of $f_{4.2}$):

$$\begin{aligned}
f_{4.2}(X) = 0 &\Leftrightarrow X^{2\chi+3} = X + 1 \quad \Leftrightarrow \quad X^{3\chi+4} = X^{\chi+2} + X^{\chi+1} \\
&\Leftrightarrow X^{4\chi+5} = X^{2\chi+3} + X^{2\chi+2} \quad \Leftrightarrow \quad X^{4\chi+5} = X^{2\chi+2} + X + 1 \\
&\Rightarrow X^{4\chi+5} - X^{3\chi+4} - 1 = X^{2\chi+2} + X - X^{\chi+2} - X^{\chi+1} \\
&\Rightarrow f_{4.0}(X) = X^{2\chi+2} + X - X^{\chi+2} - X^{\chi+1} \\
&\Rightarrow f_{4.0}(X) = \left(X^{\chi+1} - 1\right)\left(X^{\chi+1} - X\right) \geqslant 0
\end{aligned}$$

and the result follows.

On the other hand, the claim of Theorem 2.5 dealing with the approximation ratio of Algorithm 2.4 identically applies also for Algorithm 2.5. Putting all the above together, the following theorem holds and concludes the section.

Theorem 2.6. For any $\rho < 1$, Algorithm 2.5 achieves approximation ratio ρ on MAX SAT with running time $T(m) = O^*(\gamma^m)$, where γ is real solution of the equation $X^{2\alpha+3} - X - 1 = 0$ and $\alpha = \dfrac{2\rho - 2}{1 - 2\rho}$.

Figure 2.3 illustrates the relationship approximation ratio - running time of the different methods seen so far. The numeric analysis shows that, with the current value of α, and with the algorithm of [27] as the best currently known algorithm for MAX SAT, Algorithm 2.2 is the most efficient for ratios less than 0.967 while Algorithm 2.5 dominates the other ones for ratios above this value.

Note that the alogrithm 2.3 seems bad on the graph, but could be improved with any new exact algorithm for the MAX SAT problem, whereas the 2.5 can only benefit from some particular algorithms (branching algorithms, that is).

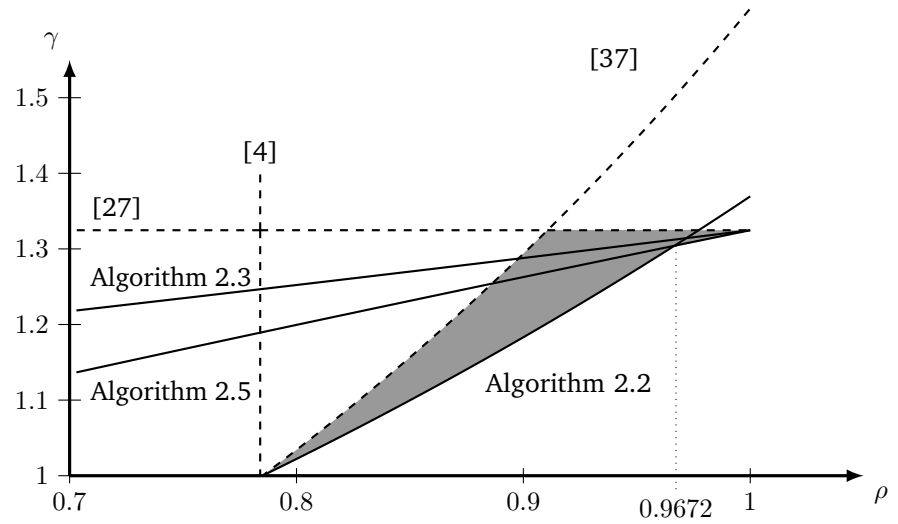


Figure 2.3: Evaluation of the running times for the algorithms in this part depending on the approximation ratio. Dashed lines are algorithms from the literature. The grey part is the improvement due to our results.

2.4 Splitting the variables

In this section, we present two algorithms that approximate MAX SAT within any approximation ratio smaller than 1, and with a computation time depending on n (the number of variables). The first algorithm of this section builds several trees. Then, in each of them, as for Algorithm 2.2 in Section 2.2.1, it cuts the tree at some point and completes variables' assignment using a polynomial approximation algorithm.

Each of the trees built by Algorithm 2.6 is a binary tree and has depth roughly pn/q (at most $pn/q + p$ to be precise). So its running time is $O^*(2^{np/q})$. Note also that, on each of these trees, at least one leaf is a partial assignment of an optimal (global) truth assignment. We will call such a leaf an *optimal leaf*.

Lemma 2.7. At least one among the optimal leaves has at least $\frac{p}{q} \times |\text{OPT}|$ satisfied clauses (before applying the polytime approximation algorithm).

Proof. Remark that every clause C_i in OPT contains at least one true literal; pick one of them from each clause C_i and denote the variable corresponding to

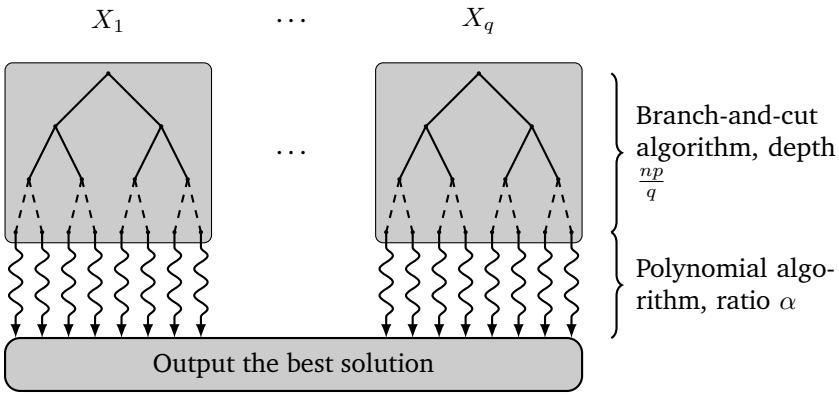


Figure 2.4: Illustration of Algorithm 2.6. For each subset of variables X_i, the upper part uses an exact branch-and-cut algorithm, but stops after some point, and uses a polynomial approximation algorithm on the remaining instances.

ALGORITHM 2.6 – Splitting the variables

Input : An instance for the MAX SAT problem
Output : An truth assignment wich achieve an approximation ratio ρ

1 Let p, q be two integers such that $\frac{p}{q} = \frac{\rho-\alpha}{1-\alpha}$.
2 Build q subsets $X_1, \ldots, X_q$ of variables, each one containing roughly $p/q \times n$ variables, and each variable appears in exactly p subsets (as in Algorithm 2.3 page 50).
3 For each subset X_i, construct a complete branching tree, considering only the variables in the subset (i.e. the depth of each of these tree is exactly $|X_i|$)
4 **for** *each of the leaves of these trees* **do**
5 | Run a polynomial time algorithm guaranteeing a ratio α on the surviving sub-instance.
6 **end**
7 Return the best truth assignment among those built.

this literal by Var(C_i). Let, for each variable x, $C(x)$ be the set of clauses from OPT for which x or $\neg x$ is the picked literal, i.e :

$$\forall x \in X, C(x) = \{C_i \in \text{OPT}/\text{Var}(C_i) = x\}$$

. Based upon this, OPT $= \bigcup_{x \in X} C(x)$.

In the tree obtained on the set X_i, denote by Λ_i the set of satisfied clauses on some optimal leaf and set $\lambda_i = |\Lambda_i|$. Then

$$\bigcup_{x \in X_i} C(x) \subseteq \Lambda_i$$

By construction,

$$\forall i, j, C\left(x_i\right) \cap C\left(x_j\right) = \emptyset$$

We so have:

$$\lambda_i \geqslant \sum_{x \in X_i} |C(x)|$$

and

$$\sum_{i=1}^{q} \lambda_i \geqslant \sum_{i=1}^{q} \sum_{x \in X_i} |C(x)|$$

As every x belongs to exactly p subsets among the q sets X_i, it holds that

$$\sum_{i=1}^{q} |\lambda_i| \geqslant p \times \sum_{x \in X} |C(x)| = p \times |\text{OPT}|$$

From what it is immediately derived that:

$$\max_{i=1}^{q} |\lambda_i| \geqslant \frac{1}{q} \sum_{i=1}^{q} |\lambda_i| \geqslant \frac{p}{q} \times \sum_{x \in X} |C(x)| = \frac{p}{q} \times |\text{OPT}|$$

□

Theorem 2.8. For any $\rho \leqslant 1$, Algorithm 2.6 achieves an approximation ratio ρ with running time $O^*\left(2^{n\frac{\rho-\alpha}{1-\alpha}}\right)$.

Proof. As seen before, the running time is $O^*(2^{np/q} = O^*(2^{n(\rho-\alpha)/(1-\alpha)}))$.

For the approximation ration, by Lemma 2.7, among all the optimal leaves, at least one satisfies $\lambda \geqslant \frac{p}{q} \times |\text{OPT}|$ clauses. As an optimal leaf corresponds to

an optimal truth assignment, it is possible to complete this assignment into an optimal (global) solution. In other words, there exist $|\text{OPT}|-\lambda$ remaining clauses that become true on the surviving sub-instance. If the polynomial algorithm called by Algorithm 2.6 achieves approximation ratio α, it will compute a solution that satisfies at least $\alpha \times (|\text{OPT} - |\lambda|)$ clauses. Hence, the number of satisfied clauses will be at least:

$$|\lambda| + \alpha \times (|\text{OPT} - |\lambda|) = \alpha|\text{OPT}| + (1-\alpha)\lambda \geqslant \alpha|\text{OPT}| + (1-\alpha)\frac{p}{q}|\text{OPT}|$$

that leads to an approximation ratio of $\alpha + (1-\alpha)\frac{p}{q} = \rho$. □

Algorithm 2.6 builds a full branching tree on each subset of variables. In particular, when the seek ratio ρ tends to 1, the basis of the exponent in the complexity tends to 2. Then, one might ask the following question: suppose that there is an exact algorithm solving MAX SAT in $O^*(\gamma^n)$ (for some $\gamma < 2$), is it possible to find a ρ approximation algorithm in time $O^*(\gamma_\rho^n)$ where $\gamma_\rho < \gamma$ for some $\rho \in]\alpha, 1]$? for any $\rho \in]\alpha, 1]$? This kind of reduction from an approximate solution to an exact one would allow to take advantage of any possible improvement of the exact solution of MAX SAT, which is not the case in Algorithm 2.6. Note again that finding an exact algorithm in time $O^*(\gamma^n)$ for some $\gamma < 2$ is a famous open question for MAX SAT as well as for some other combinatorial problems. It has very recently received a positive answer for the Hamiltonian cycle problem in [12].

Indeed, we propose in what follows a ρ-approximation algorithms working in time $O^*(\gamma_\rho^n)$ with $\gamma_\rho < \gamma$ for any $\rho \in]\alpha, 1]$.

Lemma 2.9. If there is a MAX SAT-algorithm working in time $O^*(\gamma^n)$, then the instances of MWS in Algorithm 2.7 can be solved with the same bound on the running time.

Proof. Note that the only weights assigned by Algorithm 2.7 are 1 and 2. In such a weighted instance, we can add a new variable x_0 and replace each clause c of weight 2 by three new clauses: c, $c \vee x_0$ and $c \vee \neg x_0$. Thus, if c is satisfied, then it will count in the new instance as three satisfied clauses. Otherwise, exactly one of the three new clauses will be satisfied. Thus, the so-built instance of MAX SAT is equivalent to the initial MWS-instance built by Algorithm 2.7. Furthermore, although the number of clauses can be multiplied by 2, the number of variables can only increase by one, guaranteeing an equivalent running time. □

ALGORITHM 2.7 – Splitting the variables improved

Input : An instance for the MAX SAT problem
Output : An truth assignment wich achieve an approximation ratio ρ

Let p, q be two integers such that $\frac{p}{q} = 2\rho - 1$.
Build q subsets $X_1, \ldots, X_q$ of variables, each one containing roughly $p/q \times n$ variables, and each variable appears in exactly p subsets (as in Algorithm 2.6 page 57).
for *each* X_i **do**
    Assign weight 2 to every clause
    Remove from the instance the variables not in X_i
    for *each clause missing at least one variable from* X_i **do**
        Set its weight to 1
    end
    Remove empty clauses
    Solve exactly this MWS instance, obtaining a truth assignment for the variables in X_i.
    Complete it with a greedy algorithm (for each (i, j)-literal in a remaining instance, if $i > j$ then set the literal to true, else set it to false.
end
Return the best among the truth assignments so produced.

Theorem 2.10. Algorithm 2.7 achieves approximation ratio ρ with running time $O^*(\gamma^{(2\rho-1)n})$, where $O^*(\gamma^n)$ is the running time of an exact algorithm for MAX SAT.

Proof. For the running time: we apply q times an exact algorithm $O^*(\gamma^n)$ on instances of size $(2\rho - 1)n$.

For the approximation ratio, using the same notation as before, consider one particular literal in each clause satisfied by some optimum solution OPT, and let $C(x)$ be the subset of these clauses such that the picked literal is x or $\neg x$. Then, as shown before, there exists a subset X_i such that $\sum_{x \in X_i} |C(x)| \geqslant \frac{p}{q}|\text{OPT}|$. Consider now such a X_i, and denote by (see Figure 2.5): A the subset of clauses containing only variables in $X \setminus X_i$, A_+ (resp., A_-) the subset of clauses from A that are in the optimum (resp., are not in the optimum); B the subset of clauses containing at least one variable in X_i and one variable in $X \setminus X_i$; B^1_+ (resp., B^2_+) the subset of clauses from B that are in the optimum

and whose chosen variable Var(c) is in X_i (resp., not in X_i); B_- the subset of clauses from B that are not in the optimum; C the remaining clauses, i.e., the clauses that contain only variables in X_i, C_+ (resp., C_-) the subset of clauses from C that are in the optimum (resp., are not in the optimum). Note that when removing variables in $X \setminus X_i$, clauses in A become empty, so the remaining clauses are exactly those in $B \cup C$. With these notations, OPT $= A_+ \cup B^1_+ \cup B^2_+ \cup C_+$ and $\sum_{x \in X_i} |C(x)| = |B^1_+| + |C_+|$. Then, for the chosen X_i:

$$\left|B^1_+\right| + |C_+| \geqslant \frac{p}{q}|\text{OPT}| = \frac{p}{q}\left(|A_+| + \left|B^1_+\right| + \left|B^2_+\right| + |C_+|\right) \tag{2.1}$$

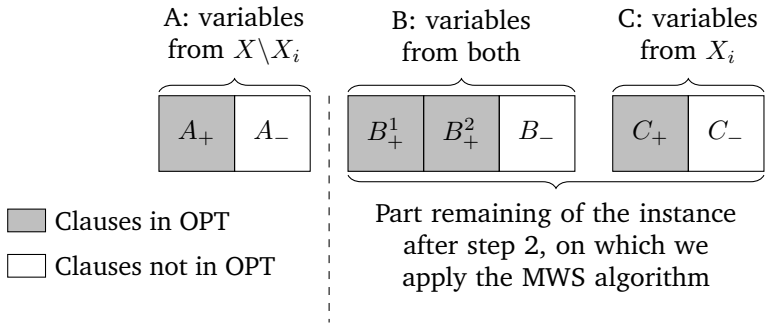


Figure 2.5: Division of clauses according to a subset X_i of variables.

With respect to step 10 of Algorithm 2.7, denote by B_1 the subset of satisfied clauses from B and by C_1 the subset of satisfied clauses from C. As OPT is a particular solution of weight $|B^1_+| + 2|C_+|$ for this weighted sat problem, we have:

$$|B_1| + 2\,|C_1| \geqslant \left|B^1_+\right| + 2\,|C_+| \tag{2.2}$$

The greedy algorithm in step 11 will satisfy at least half of the remaining clauses containing at least one literal from $X \setminus X_i$, i.e., the set $(B \setminus B_1) \cup A$. Finally, the number of satisfied clauses is at least:

$$\begin{aligned} |B_1| + |C_1| + \frac{|B| - |B_1| + |A|}{2} = & \quad |C_1| + \frac{|B_1|}{2} + \frac{|B|}{2} + \frac{|A|}{2} \\ \overset{(2.2)}{\geqslant} & \quad \frac{\left|B^1_+\right|}{2} + |C_+| + \frac{|B|}{2} + \frac{|A|}{2} \\ \geqslant & \quad \left|B^1_+\right| + |C_+| + \frac{\left|B^+_2\right|}{2} + \frac{|A^+|}{2} \end{aligned}$$

So, the approximation ratio achieved is at least:

$$\frac{\left|B_+^1\right| + |C_+| + \frac{\left|B_+^2\right| + |A_+|}{2}}{\left|B_+^1\right| + |C_+| + \left|B_+^2\right| + |A_+|} = \frac{1}{2}\left(1 + \frac{\left|B_+^1\right| + |C_+|}{\left|B_+^1\right| + |C_+| + \left|B_+^2\right| + |A_+|}\right)$$
$$\overset{(2.1)}{\geqslant} \frac{1}{2}\left(1 + \frac{p}{q}\right) = \frac{q+p}{2q} = \rho$$

that completes the proof. □

The Algorithm 2.7 may seem less efficient than the previous one (Algorithm 2.6). Its main advantage is that it follows the best known algorithm to solve exactly MAX SAT. For instance, if MAX SAT becomes solvable in $O^*(1.657^n)$[4], then Algorithm 2.7 achieves a 0.9-approximation in time $O^*(1.576^n)$ while Algorithm 2.6 achieves the same ratio in time $O^*(1.703^n)$.

2.5 Discussion

We have proposed in this paper several algorithms that constitute a kind of "moderately exponential approximation schemata" for MAX SAT. They guarantee approximation ratios that are unachievable in polynomial time unless $\mathbf{P} = \mathbf{NP}$. To obtain these schemata, several techniques have been used coming either from the polynomial approximation or from the exact computation. Furthermore, Algorithm 2.7 in Section 2.4 is a kind of polynomial reduction between exact computation and moderately exponential approximation transforming exact algorithms running on "small" sub-instances into approximation algorithms guaranteeing good ratios for the whole instance. We think that research in moderately exponential approximation is an interesting research issue for overcoming limits posed to the polynomial approximation due to the strong inapproximability results proved in the latter paradigm.

All the algorithms given in this paper have exponential running time when we seek for approximation ratios either better than the known ones, or unachievable in polynomial time. Can we expect for better results? For several problems that are hard to approximate in polynomial time within constant ratios (as MAX INDEPENDENT SET, MIN COLORING, ...), subexponential time can be easily reached for ratios depending on the input size (see Algorithm 1.5 page 34). In any case, an interesting question is what is doable in subexponential time for

[4]This is the running time to solve Hamiltonian cycle in [12].

the approximation of problems admitting bounds in their approximability. For instance, is it possible to devise a constant approximation algorithm working in subexponential time for, say MIN COLORING? An easy argument shows that this is very unlikely, at least for small (constant) ratios, and hence achievement of good approximations in subexponential time for *any* NP-hard problem seems to be impossible. In fact, the existence of subexponential approximation algorithms (within ratio better than 4/3) is quite improbable for MIN COLORING since it would imply that 3-COLORING can be solved in subexponential time, contradicting so the "exponential time hypothesis" [65]. We conjecture that this is true for any constant ratio for MIN COLORING. Anyway, the possibility of devising subexponential approximation algorithms for **NP**-hard problems, achieving ratios forbidden in polynomial time or of showing impossibility of such algorithms is an interesting open question that deserves further investigation.

We conclude this paper with a word about another very well known optimum satisfiability problem, the MIN SAT problem that, given a set of variables and a set of disjunctive clauses, consists of finding a truth assignment that minimizes the number of satisfied clauses. A ρ-approximation algorithm for MIN SAT (with $\rho > 1$) is an algorithm that finds an assignment satisfying at most ρ times the minimal number of simultaneously satisfied clauses.

In [33] an approximability-preserving reduction between MIN VC and MIN SAT is presented transforming any ρ-approximation for the former problem into a ρ-approximation for the latter problem. This reduction can be used to translate any result on the MIN VC problem into a result on the MIN SAT, the number of vertices in the MIN VC instance being the number of clauses in the MIN SAT instance. For instance, the results from [16] for MIN VC lead to the following parameterized approximation result for MIN SAT: *for every instance of MIN SAT and for any $r \in \mathbb{Q}$, if there exists a solution for MIN SAT satisfying at most k clauses, it is possible to determine with complexity $O^*(1.28^{rk})$ a $2 - r$-approximation of it*.

We also note that the method used in Algorithm 2.6 can be applied as well to MIN SAT with as in the algorithm 2.8 below.

The complexity of this algorithm is the same as that of Algorithm 2.6, i.e., $O^*(2^{n(\alpha-\rho)/(\alpha-1)})$ (the best known ratio is $\alpha = 2$). Furthermore Lemma 2.7 still holds and a similar analysis derives an approximation ratio $\alpha - (\alpha - 1)\frac{p}{q} = \rho$.

ALGORITHM 2.8 – Splitting the variables for MIN SAT

Input : An instance for the MIN SAT problem
Output : An truth assignment wich achieve an approximation ratio ρ

1 Let p, q be two integers such that $\frac{p}{q} = 2\rho - 1$.
2 Build q subsets $X_1, \ldots, X_q$ of variables, each one containing roughly $p/q \times n$ variables, and each variable appears in exactly p subsets (as in Algorithm 2.3 page 50).
3 For each subset X_i, construct a complete branching tree, considering only the variables in the subset (i.e. the depth of each of these tree is exactly $|X_i|$)
4 **for** *each of the leaves of these trees* **do**
5 | Run a polynomial time algorithm guaranteeing a ratio α on the surviving sub-instance.
6 **end**
7 Return the best truth assignment among those built.

3 Exponential time approximation and branching algorithms

> However, it was not clear to me from reading the paper if branching algorithms are common and are used for a variety of problems in the world of exact algorithms.
>
> Anonymous reviewer

Dans cet article, nous discutons d'une méthode générale pour concevoir des algorithms exponentiels d'approximation. En particulier, nous montrons que sous certaines conditions, une variante d'algorithme de branchements permet d'obtenir facilement des résultats intéressants.

We study links between approximation, exponential time computation and fixed parameter tractability. In particular, rather than focusing on one particular optimization problem, we tackle the question of finding sufficient conditions for a problem to admit "good" approximation algorithms in exponential time. In particular, we focus on the existence of "approximation schemata" (ratios $1 \pm \epsilon$ for arbitrarily small ϵ) and we exhibit conditions under which a technique of devising approximate branching algorithms reaches interesting results.

Contents

3.1 Introduction

Polytime approximation, exponential time computation and fixed parameter tractability (FPT) are three main fields aiming at coping with **NP**-hard problems. Considering an optimisation problem, a ρ-approximation algorithm is an algorithm that, on any instance, outputs a solution whose value is at least (for a maximisation problem, at most for a minimization problem) ρ times the optimal value.

Polytime approximation is the field of studying whether one can derive approximation algorithms working in polynomial time for an **NP**-hard problem (see for instance [7]). The goal of exponential time computation (see [58]) is to devise exact algorithms whose worst case complexity, though not polynomially bounded, is as low as possible. For many **NP**-hard problems, the best known algorithm works in exponential time $O^*(\gamma^n)$ (meaning $O(\gamma^n p(n))$ for some polynomial p). A natural objective is then to minimize the value of γ. FPT is in some sense a complementary field where an instance is given together with a parameter $k \in \mathbb{N}$; the goal is to know whether the problem is solvable by an algorithm working in time $O(f(k)p(n))$ for some function f and some polynomial p [44]. Such an algorithm is called an FPT algorithm, and if such an algorithm exists the problem is in the **FPT** class (for this particular parameterization). A usual parameter (called standard parameterization) is the value of the sought solution. Formally, for the standard parameterization[1], a problem is in **FPT** if there exists an algorithm such that given an instance I of size n and an integer k, (i) it outputs YES if there is a solution of value at least k (for a maximization problem, at most k for a minimization problem) and NO otherwise; (ii) it works in time $O(f(k)p(n))$ for some function f and polynomial p.

Interestingly, negative results have been obtained in these three fields. For polytime approximation, tight lower bounds are known for many optimization problem, under the hypothesis **P** $\neq$ **NP**. For instance, MAXIMUM

[1] We will use the standard parameterization in all this article, so we will omit to precise it.

3-SATISFIABILITY (MAX 3-SAT)[2] is well known to be approximable in polytime within ratio $7/8$, and not approximable in polytime within ratio $7/8 + \epsilon$ unless **P** $\neq$ **NP** for any $\epsilon > 0$ [64]. Dealing with exponential time computation, [65] have introduced the EXPONENTIAL TIME HYPOTHESIS (ETH)which supposes that there is no algorithm for MAX 3-SAT that works in time $2^{o(n)}$ where n is the number of variables. Under ETH, many classical optimization problems do not have subexponential time algorithms[3]. Finally, the parameterized complexity theory considers the hypothesis **W[1]** $\neq$ **FPT** (see [44] for the definition of the class **W[[]**1]) under which it is shown that many optimization problems do not admit FPT algorithms (under the standard parameterization for instance).

In recent years, a growing interest appeared in trying to combine these approaches. The first way that has been considered is to devise (sub)exponential approximation algorithms. In this topic, we take a problem that is inapproximable with some ratio ρ (in polytime), and for which the best known complexity is $O^*(\gamma^n)$. A natural question is whether there exists a ρ-approximation algorithm that works in time much smaller than $O^*(\gamma^n)$. This issue has been considered for several optimization problems such as MAX SAT [37, 50], MAX IS and MIN VC [19], MIN BW [36, 60], ... In these articles, approximation algorithms working in exponential time are provided. Interestingly, the existence of subexponential time approximation algorithms (achieving constant ratios forbidden in polytime) seems to be unlikely for several classical optimization problems (see [81, 48]).

Another research direction introduced in [24, 45] aims at combining approximation algorithms and fixed parameter tractability. In the parameterized framework, given an instance I of an optimization problem and an integer k, a ρ-approximation algorithm either outputs a solution of value at least ρk (for a maximization problem, at most ρk for a minimization problem), or answers NO, asserting in this latter case that there is no solution of value at least (resp., at most) k. Then, if a problem is both **W[1]**-hard and hard to approximate within ratio ρ in polytime, the existence of an FPT ρ-approximation algorithm is worth being considered. Both positive and negative results have been obtained (see the survey [80]). If a problem is solvable in time $O^*(f(k))$

but hard to approximate within ratio ρ in polytime, it is also interesting to

[2]Given a set of binary variables and a set of clauses, MAX SAT is to find a truth assignment of variables that maximizes the number of satisfied clauses. MAXIMUM k-SATISFIABILITY (MAX k-SAT) is a restricted version where each clause contains exactly k literals.

[3]Note that speaking of (sub)exponential time algorithm requires to have defined a parameter representing the size of the instance, such as the number of vertices in a graph, with respect to which the complexity is measured.

seek a parameterized ρ-approximation algorithm working in time $O^*(f'(k))$ for some function f' significantly lower than f. In this direction, positive answers have been provided for instance for MIN VC [22, 53].

In this work, we further study the link between approximation algorithms, exponential time computation and fixed parameter tractability. In particular, rather than focusing on one particular optimization problem, we tackle the question of finding sufficient conditions for a problem to admit "good" approximation algorithms in exponential time. Especially, we focus on the existence of "approximation schemata" (ratios $1 \pm \epsilon$ for arbitrarily small ϵ). The article is organized as follows. We define approximation classes and give some basic properties in Section 3.2. In Section 3.3, we give sufficient conditions to derive approximation schemata in exponential time using branching algorithms, which is the main result of this article. We conclude the article in Section 3.4 where we mention in particular FPT approximation algorithms.

3.2 Approximation classes and first results

3.2.1 Approximation classes

We focus on the well known class **NPO** of optimization problems whose decision versions are in **NP**. An **NPO** problem P is defined on a set $\mathscr{I}$ of instances. To each instance $I \in \mathscr{I}$ corresponds a set $\mathscr{S}(I)$ of feasible solutions. A function f associates for each instance I and each feasible solution $S \in \mathscr{S}(I)$ a value $f(I, S) \in \mathbb{N}$, to be either maximized or minimized. Furthermore, (i) instances of P are recognized in polynomial time; (ii) the size of feasible solutions are polynomially bounded in the size of the instance and, moreover, it can be checked in polynomial time whether a given S is in $\mathscr{S}(I)$ or not; (iii) f is polynomially computable.

An optimal solution of an instance I will be denoted $\mathrm{OPT}(I)$, or simply OPT if there is no ambiguity. We use the notation $f(I)$ for $f(I, \mathrm{OPT}(I))$.

Following the discussion in introduction, considering an optimization problem that is solvable in time $O^*(\gamma^n)$, we may wonder whether the problem is ρ-approximable in time $O^*(\gamma_\rho^n)$ with $\gamma_\rho < \gamma$ for some ratio ρ, or even for any ratio $\rho \neq 1$, or not. This motivates the definition of the following two classes. Note that we give the definition for maximization problems, but everything said in this section can be easily translated to minimization problems.

Class 3.1 (EAS). For $\delta > 1$, a maximization **NPO** problem is in **EAS**[δ] if, for any $\rho < 1$, there exists a ρ-approximation algorithm with computation time

$O^*(\delta_\rho^n)$, with $\delta_\rho < \delta$.

This defines a notion of approximation schema, the existence of which of course depends on the parameter δ which is the "target" basis of the exponential function expressing the running time.

Dealing with approximation for a specific ratio, we define the class **EAX**$[\gamma, \rho]$.

Class 3.2 (EAX). For $\gamma > 1$, $\rho \neq 1$, an **NPO** problem is in **EAX**$[\gamma, \rho]$ if there exists a ρ-approximation algorithm with computation time $O^*(\gamma^n)$.

Let us begin with some properties of these classes. First, note that of course any problem that can be solved exactly in time $O^*(\delta^n)$ is in **EAS**$[\delta']$ for any $\delta' > \delta$. On the other hand, one may expect at first sight that if a problem is in **EAS**$[\delta]$, then the problem is solvable in $O^*(\delta^n)$. For several exponential approximation schemata obtained so far in the literature problems shown to be in **EAS**$[\delta]$ are indeed solvable in $O^*(\delta^n)$, but the following result is worth being mentioned: MAXIMUM UNUSED COLORS (MAX UC)[4] can be approximated with ratio $(1-\epsilon)$ in time $O^*(2^n)$ and polynomial space for any $\epsilon > 0$, whereas the best known exact algorithm in polynomial space is in $O^*(2.25^n)$ [17].

However, we say that a problem has a *full γ-exponential approximation schema*, if it is possible to perform a $(1-\epsilon)$ approximation in time $O(p(1/\epsilon) \times \gamma_\epsilon^n)$ for some polynomial p and $\gamma_\epsilon < \gamma$. Then, we have the following easy result.

Property 3.3. If an **NPO** maximization problem has a full γ-exponential approximation schema and is polynomially bounded[5], then it can be solved in time $O^*(\gamma^n)$.

Proof. If the value is bounded by $P_m(n)$, with P_m a polynomial, then using the schema, we can get a $(1 + 1/P_m(n))$-approximation in time $O(P(P_m(n))\gamma^n)$, which is a computation time $O^*(\gamma^n)$. With this ratio, the difference between the optimal solution and the obtained solution would be less than one, which means that we have an exact solution. □

Now, let us make some observations on the classes **EAS** and **EAX**.

[4] Given a graph G on n vertices, the goal is to find a proper coloring of G maximizing $n - k$ where k is the number of colors of the coloring.

[5] This means that the value of a solution is polynomialy bounded in the size of the instance.

Property 3.4. The following holds: (i) $\forall \delta < \delta' \mathbf{EAS}[\delta] \subseteq \mathbf{EAS}[\delta']$; (ii) $\forall \gamma < \gamma', \forall \rho, \mathbf{EAX}[\gamma, \rho] \subseteq \mathbf{EAX}[\gamma', \rho]$; (iii) $\forall \gamma, \forall \rho' < \rho, \mathbf{EAX}[\gamma, \rho] \subseteq \mathbf{EAX}[\gamma, \rho']$. Furthermore, assuming the *SETH*[6], the above inclusions are strict.

Proof. Inclusions are trivial. We prove the strict inclusions under *SETH*. We first deal with the first inclusion Let δ and δ' be two real numbers such that $1 < \delta < \delta'$, and δ'' be such that $\delta < \delta'' < \delta'$. We create a problem where the value of solutions is either 0 or 1, and such that it is solvable in $O^*(\delta''^n)$ (and hence is in $\mathbf{EAS}[\delta']$) but not in $O^*(\delta^n)$ under *SETH* (and hence not in $\mathbf{EAS}[\delta]$ since obviously any approximate solution allows to solve optimally the problem).

Let $P_{\delta,\delta''}$ be the following problem. An instance of it, is a set of n variables $(x_1, \cdots, x_n)$, n nonnegative integers $(a_1, \cdots, a_n)$ such that the average value $\overline{a_i} = \sum_{i=1}^{n} a_i/n$ lies in the interval $(\ln(\delta)/\ln(2), \ln(\delta'')/\ln(2)]$, and a polynomially computable function $f : \mathbb{N}^n \to \{0, 1\}$. Every variable x_i can take values from 0 to $2^{a_i} - 1$ (this defines the set of feasible solutions), and the value of a solution is $f(x_1, \cdots, x_n)$, to be maximized. This problem is obviously in **NPO**.

Note that $\prod_{i=1}^{n} 2^{a_i} = 2^{\sum_{i=1}^{n} a_i} \leq \delta''^n$; so, an exhaustive search allows us to solve the problem in time $O^*(\delta''^n)$.

Now, take an instance of SAT with N variables. Let $n = \lceil \ln(2)N/\ln(\delta'') \rceil$. Note that $\ln(2)N/\ln(\delta'') \leq n < \ln(2)N/\ln(\delta'')+1$, meaning that $\ln(2)/\ln(\delta'') \leq n/N < \ln(2)/\ln(\delta'') + 1/N \leq \ln(2)/\ln(\delta)$ for N large enough. Then choose n nonnegative integers a_i such that $\sum_{i=1}^{n} a_i = N$. Intuitively, if for instance $a_i = 2$ this means that our variable x_i will have value in $\{0, 1, 2, 3\}$ which allows to simulate 2 variables of the SAT instance. This way, since $\sum_{i=1}^{n} a_i = N$, each variable of the SAT instance is simulated in our set of n variables with value in $\{0, \cdots, 2^{a_i} - 1\}$. The function f is now trivial: its value is 1 if $(x_1, \cdots, x_n)$ corresponds to a truth value that satisfies the SAT instance, and 0 otherwise. Now, note that $\delta^n < 2^{\sum_{i=1}^{n} a_i} = 2^N$, so an algorithm solving $P_{\delta,\delta'}$ in $O^*(\delta^n)$ would contradict *SETH*.

The same argument shows the second inclusion claimed. Just take a problem that is not solvable in $O^*(\gamma^n)$ under *SETH* like in the above construction, but define the values to be either 1 or ρ (instead of either 1 or 0). □

[6]The strong exponential time hypothesis (*SETH* [65]) claims that there is no constant $\gamma < 2$ such that SAT is solvable in time $O^*(\gamma^n)$.

3.2.2 FPT and exponential time approximation schemata

As explained in introduction, our main concern in this article is to provide sufficient conditions for a problem solvable in time $O^*(\gamma^n)$ to be in **EAS**[γ].

In this line of work, a first general result has been obtained in [19] for hereditary graph problems[7]: using a method consisting of partitioning the instance, the authors show that if an hereditary graph problem is solvable in $O^*(\gamma^n)$ then it is ρ-approximable in time $O^*(\gamma^{\rho n})$ for any ratio ρ, and hence it is in **EAS**[γ].

As a first simple result here, we show that exponential time approximation schemata can be easily derived from FPT problems having some basic properties.

Property 3.5. Let P be a maximization (resp., minimization) problem in **FPT** solvable in $O^*(\delta^k)$ such that: (i) the value of any feasible solution is at most n (n is the size of the instance); (ii) if there exists a solution of value k, then there exists a solution of size l, for any $l \leqslant k$ (resp., for any $k \leqslant l \leqslant n$).

Then P is also in **EAS**[δ], and it is possible to reach any approximation ratio ρ in computation time $O^*(\delta^{\rho n})$ (resp., $O^*(\delta^{n/\rho})$).

Proof. Let us first consider that P is a maximization problem. Given a $\rho \in (0, 1)$, we apply the parameterized algorithm with every value for k between 0 and ρn. If the best solution has value at most ρn, we find it. Otherwise, the optimum value is between ρn and n, hence there is a solution of value ρn. We will find such a solution, that trivially achieves an approximation ratio ρ. The complexity is clearly in $O^*(\delta^{\rho n})$.

If P is a minimization problem, for $\rho \geq 1$, we apply the parameterized algorithm with every value for k between 0 and n/ρ. If no solution is found, we output any feasible solution (by hypothesis we know how to compute a feasible solution in polynomial time). If the optimal value is at most n/ρ we find it. Otherwise, any feasible solution achieves an approximation ratio ρ. The complexity is $O^*(\delta^{n/\rho})$. □

The conditions in Proposition 3.5 are not very restrictive so that this proposition applies to a large range of problems, including for instance MIN VC, MAX SAT, ...

[7] A graph property π is hereditary if the following holds: if π holds for a graph G, then π holds for any induced subgraph of G. Given an hereditary property π, a usual goal is to find the largest induced subgraph of a given graph that satisfies the property π.

3.3 Approximation schemata by branching algorithms

Hereditary problems and FPT problems having additional properties admit approximation schemata in exponential time. However, many other problems are known to be (exactly) solvable in exponential time, and a classical way to achieve these exponential time algorithms is to use a search tree (see [58] for instance). Then, a natural idea is to extend this technique to derive exponential time approximation algorithms. This has been done for MAX SAT in [37, 50]. In this section, we formalize and generalize this technique, and show a sufficient condition on the problem dealt that makes the technique efficient.

Example 3.6. We give a basic overview of the schema on MAX IS. On this problem, using classical reduction rules (including vertex folding, see for instance [54]), vertices of degree at most two can be removed from the instance in polynomial time. When there is no such vertex, the naive exact algorithm is to pick a vertex, and to branch with two instances: either we add the vertex in the final solution and remove its neighbors from the "child" instance, either we remove this vertex from the instance. In the first case, the size of the instance decreases by at least 4 (one vertex in the solution, at least three vertices removed) ; in the second, the size decreases by 1, and this gives a complexity $O^*(1.38^n)$ (solution of $C(n) \leqslant C(n-4) + C(n-1)$). In this situation we can achieve a $1/2$-ratio by removing an additional vertex in the first branching case. The complexity of this new algorithm will satisfy $C(n) \leqslant C(n-5) + C(n-1)$ giving a complexity $O^*(1.32^n)$. Although in this case, this is not an improvement over some existing approximation algorithms, this is a good illustration of the schema.

We will now introduce some notation to formalize what is a (exact) branching algorithm. We then present what is a diminution rule, and the results on approximate branching algorithms in Section 3.3.1, and finally illustrate this technique on a very detailed example and some others in Section 3.3.3.

Given an optimization problem, we have a size function $I \mapsto |I|$. Note that this can be any measure of the size of the instance (number of vertices in a graph, or number of edges, ...). We require that when $|I| = 0$ the instance is solvable in polynomial time. For the sake of clarity we will suppose that $|I| \in \mathbb{N}$ but similar results can be obtained with $|I| \in \mathbb{R}_+$ (in particular $|I|$ can be the weight associated to $|I|$ in the measure and conquer paradigm).

A branching algorithm solves a given optimization problem by building a search tree. To each node of the tree is associated an instance; a branching rule (such as "take a particular vertex in the solution or do not take it") gives birth to

several subinstances that are the children of the node in the search tree[8]. The leaves of the tree correspond to trivial instances. This might be formalized as follows (see Figure 3.1).

Definition 3.7 (Branching rule)**.** A branching rule of fan-out r is a mapping from an instance I to a set of r subinstances $(I_1, I_2, \cdots, I_r)$ and a set of algorithms $(PB_1, PB_2, \cdots, PB_r)$, that we call payback functions, with the following properties:

a) If S_i is a solution for the subinstance I_i, then $PB_i(I, I_i, S_i)$ is a solution for I (for simplicity, we will denote $PB_i(I, I_i, S_i)$ by $PB_i(S_i)$).

b) The branching and every payback function are computable in polynomial time with respect to the size of the instance.

c) (size reduction) We have a family of positive numbers $(a_i)_{i=1\cdots,r}$ giving a guarantee on the instance size decreasing. This gives **Rule (B1)**: $\forall i$, $|I_i| \leqslant |I| - a_i$.

d) (exhaustiveness) For (at least) one of the subinstances I_i, the payback function increases the value of a feasible solution by at least the difference in the optimal between I and I_i. This, gives **Rule (B2)**: $\exists i, \forall S \in \mathscr{S}(I_i)$, $f(PB_i(S)) - f(S) \geqslant f(I) - f(I_i)$.

Of course, the greater the a_i (in Rule (B1)), the faster the algorithm. Exhaustiveness ensures that an optimal solution for an instance can be found with the branching rule: indeed, exhaustiveness is satisfied in branching algorithms in the branch corresponding to the choice of an optimum solution.

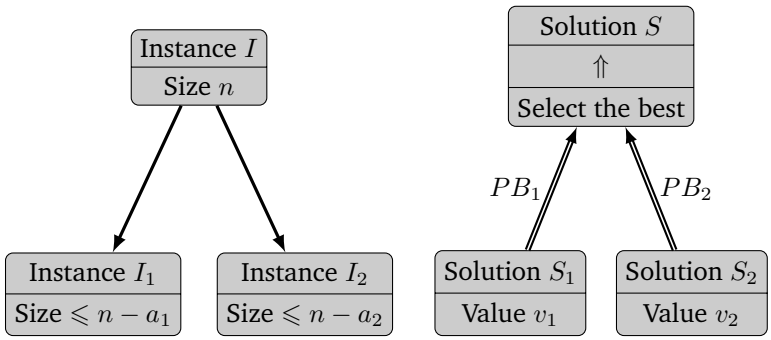
Given an optimization problem P with such a branching rule, we define the branching algorithm 3.1.

The following result is very common in the domain's literature.

Theorem 3.8. Algorithm 3.1 gives an optimal solution in time $O^*(\gamma^{|I_0|})$, with γ the only positive solution of the equation $1 = \sum_{i=1}^{r} \gamma^{-a_i}$.

Proof. Let us call I_0 the instance in the root of the tree. We know that at least one child of I_0 satisfies Rule (B2). We call this child I_1. Then I_1 has a child I_2 that satisfies the same rule. And so on till we reach a leaf, I_k. The instance on

[8]Note that branching algorithms usually use reduction rules that are polynomial time rules applied on each node to simplify the instance. Reduction rules can be formally incorporated in the branching rule, so we do not need to state them explicitly.



(a) An instance is derived into multiple subinstances.

(b) Solutions from these instances give a solution on the main instance.

Figure 3.1: Branching rule with fan-out $r = 2$

ALGORITHM 3.1 – Naive Branching Algorithm

Input : An instance I_0
Output : An optimal solution for I_0

1 We build a tree using the following rule :
a) Each node is labeled with an instance, the root's label being I_0.
b) Each node with a non-empty instance I has r sons, given by the reduction rule.
c) Each node with an empty instance is a leaf.
2 **for** *each leaf in the tree* **do**
3 Compute an optimal solution. We use the payback functions to get a solution for the root of the tree in a bottom up fashion: the solution for a node with instance I is the best solution among the r solutions computed by the payback functions.
4 **end**
5 Output the solution of the tree's root.

the leaf is solvable exactly in polynomial time, giving a solution S_k such that $f(S_k) = f(I_k)$. Then, we use the payback functions to compute solutions for every instance from I_k to I_0, and we call these solutions $S_k, S_{k-1}, \cdots, S_0$. With these solution, we have, using (B2), $f(S_i) - f(S_{i+1}) \geqslant f(I_i) - f(I_{i+1})$, for $i = 0, \cdots, k-1$. Summing these inequalities, we get $f(S_0) - f(S_k) \geqslant f(I_0) - f(I_k)$. So, $f(S_0) = f(I_0)$, which proves the correctness of the algorithm.

The complexity is given by the number of nodes in the tree. For an instance of size n, this number satisfies the equation $C(n) \leqslant C(n-a_1) + C(n-a_2) + \cdots + C(n-a_r)$, which gives the complexity claimed. □

3.3.1 Approximate branching algorithms

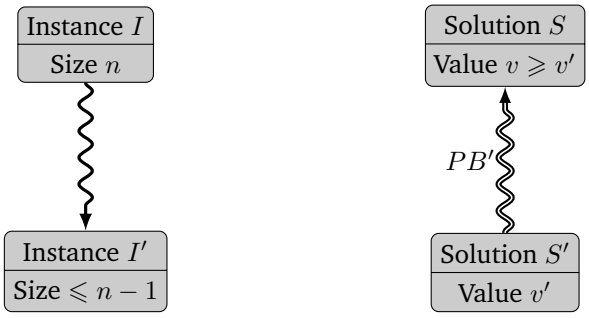
Suppose that we have for a given maximization problem a branching algorithm as given above. We are interested in modifying this algorithm to get an approximation algorithm with a better running time. We show that two properties are sufficient to reach this for any ratio $\rho \neq 1$, i.e., to get an exponential time approximation schema. The first condition, called *strict monotonicity*, deals with the branching rule. Note that it is generally satisfied by most of the branching rules used in the literature for classical optimization problems. The other condition, called *diminution rule*, is independent of the branching rule (it it directly linked to the problem, not to a particular branching algorithm).

Definition 3.9 (Strict monotonicity)**.** A branching rule is said to be strictly monotonic if for at least one subinstance I_i the payback function increases (strictly) the value of feasible solutions (while the other ones do not decrease the value of feasible solutions). Formally, there exists a family of nonnegative integers $(s_i)_{i=1\cdots,r}$ with at least one of them greater than 0 such that the following rule is satisfied: **Rule (B3)**: $\forall i, \forall S \in \mathscr{S}(I_i)$, $f(PB_i(S)) \geqslant f(S) + s_i$.

Of course, the greater the integers s_i, the faster our approximation algorithm. In Example 3.6, the branching rule is strictly monotonic: the payback function will increase by one the value of solutions in the branch where we have taken v (and by 0 in the other branch).

Definition 3.10 (Diminution rule)**.** A diminution rule is a mapping from an instance I to another instance I' together with an polynomial time algorithm (which we call payback function) PB', with the following properties:

1) The new instance's size has decreased by at least one: **Rule (D1)**: $|I'| \leqslant |I| - 1$.



(a) An instance is derived into a smaller subinstance.

(b) A Solution from the subinstance gives a solution on the main instance.

Figure 3.2: Diminution rule

2) The value of optimal solutions has not decreased by more than one. This gives **Rule (D2)**: $f(I') \geqslant f(I) - 1$.

3) The payback function PB' preserves the value of the solutions. This gives **Rule (D3)**: $\forall S' \in \mathscr{S}(I')$, $f(PB'(S')) \geqslant f(S')$.

Example 3.11. For MAX IS (see also Example 3.6), the diminution rule we use is to remove a vertex from the graph: from a graph G we obtain a graph G' with one vertex less (Rule (D1)). Of course the optimum value in G is at most the optimum value in G' plus one (Rule (D2)). The payback function is the identity function (an independent set in G' is an independent set in G) and satisfies Rule (D3).

Given a problem that has both a diminution rule and a strictly monotonic branching rule, we can define the following ρ-approximation algorithm 3.2.

Theorem 3.12. Algorithm 3.2 outputs a ρ-approximate solution in time $O^*(\gamma_\rho^{|I|})$, where γ_ρ is the only positive root of the equation $1 = \sum_{i=1}^{r} \gamma_\rho^{-a_i - \alpha_0 s_i}$. In particular, the problem is in **EAS**[γ] where γ is the only positive root of the equation $1 = \sum_{i=1}^{r} \gamma^{-a_i}$.

Proof. We first analyze the approximation ratio of the algorithm. Let us show that each non-leaf node $[I, \alpha]$ in the tree has at least one child $[I', \alpha']$ with a

ALGORITHM 3.2 – A ρ-approximation pruning algorithm

Input : An instance I_0 and a ratio ρ.
Output : A ρ-approximate solution for I_0.

1 Define α_0 such as $\alpha_0 = \frac{\rho}{1-\rho}$.
2 Build a tree using the following rule :
a) Each node is labeled with an instance I and a number α. We write a node $[I, \alpha]$. The root's label is $[I_0, 0]$.
b) For each node $[I, \alpha]$, with I non-empty and $\alpha < \alpha_0$, we create up to r sons $([I_1, \alpha + s_1], \ldots, [I_r, \alpha + s_r])$, corresponding to the branching rule.
c) For each node $[I, \alpha]$, with I non-empty and $\alpha \geqslant \alpha_0$, we create one son. Its instance is obtained by applying the diminution rule, and its integer is $\alpha - \alpha_0$.
d) The leaves are the empty instances (with any number).
3 **for** *every leaf in the tree* **do**
4 Compute backward an optimal solution in a bottom up fashion: in case b), the solution for a node with instance I is the best solution among the r solutions computed by the payback functions. In case c), the solution for a node with instance I is given by the payback function of the diminution rule.
5 **end**
6 Output the best solutions among those calculated.

payback function PB, and satisfying the following:

$$\forall S' \in \mathscr{S}(I'), f(PB(S')) - f(S') + \frac{1}{1+\alpha_0}(\alpha - \alpha') \geqslant \frac{\alpha_0}{1+\alpha_0}(f(I) - f(I')) \quad (3.1)$$

We examine the following two cases concerning α and α_0.

i) $\alpha \geqslant \alpha_0$. Then I' is derived from I using the diminution rule. For any solution $S' \in \mathscr{S}(I')$, we call $S = PB(S')$. Then using Rule (D3), we have $f(S) - f(S') \geqslant 0$. Also, as we used a diminution, we know that $\alpha' = \alpha - \alpha_0$, and so, $\frac{1}{1+\alpha_0}(\alpha - \alpha') = \frac{1}{1+\alpha_0} \times \alpha_0 = \frac{\alpha_0}{1+\alpha_0}$.

Using Rule (D2), since $f(I) - f(I') \leqslant 1$, $\frac{\alpha_0}{1+\alpha_0}(f(I) - f(I')) \leqslant \frac{1}{1+\alpha_0}(\alpha - \alpha') \leqslant f(S) - f(S') + \frac{1}{1+\alpha_0}(\alpha - \alpha')$.

ii) $\alpha < \alpha_0$. Then, I has up to r children, and using Rule (B2), we know that for one child $[I', \alpha']$, and for any solution $S' \in \mathscr{S}(I')$ $f(S) - f(S') \geqslant f(I) - f(I')$

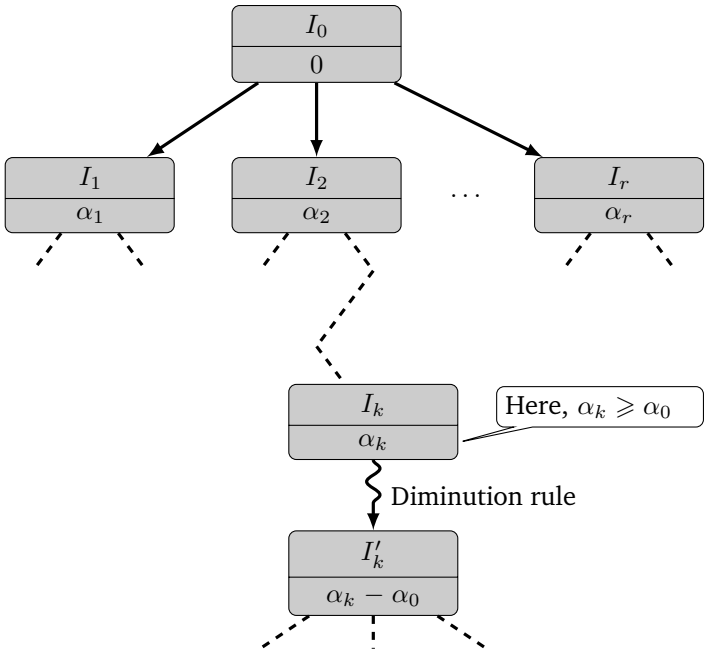


Figure 3.3: Illustration of Algorithm 3.2. In each node, the upper part is the instance, the lower part is the integer (see Algorithm 3.2, step a)).

where $S = PB(S')$. In particular:

$$\frac{\alpha_0}{1+\alpha_0}\left(f(S) - f(S')\right) \geqslant \frac{\alpha_0}{1+\alpha_0}\left(f(I) - f(I')\right) \tag{3.2}$$

Also, we use the definition of α and α', and Rule (B3), and we have $f(S) - f(S') \geqslant \alpha' - \alpha$. So:

$$\frac{1}{1+\alpha_0}(f(S) - f(S')) + \frac{1}{1+\alpha_0}(\alpha - \alpha') \geqslant 0 \tag{3.3}$$

Summing (3.2) and (3.3), we get $\frac{1+\alpha_0}{1+\alpha_0}\left(f(S) - f(S')\right) + \frac{1}{1+\alpha_0}(\alpha - \alpha') \geqslant \frac{\alpha_0}{1+\alpha_0}(f(I) - f(I'))$, which derives (3.1).

>From the above cases, one can see that it is possible to find a branch in the tree, going from the root $[I_0, 0]$ to a leaf $[I_f, \alpha_f]$, satisfying the Proposition (3.1)

at each step. We find in polynomial time a solution S_f for the instance I_f, and then apply the paybacks functions to find a solution S_0 on I_0. If we sum the equations derived from (3.1), we get $f(S_0) - f(S_f) + \frac{1}{1+\alpha_0}(-\alpha_f) \geqslant \frac{\alpha_0}{1+\alpha_0}(f(I_0) - f(I_f))$.

As $\alpha_f \geqslant 0$, and $f(S_f) = f(I_f) \geqslant 0$, we have: $f(S) - f(S_f) \geqslant \frac{\alpha_0}{1+\alpha_0} f(I) - \frac{\alpha_0}{1+\alpha_0} f(I_f)$ i.e., $f(S) \geqslant \frac{\alpha_0}{1+\alpha_0} f(I) + \underbrace{f(S_f) - \frac{\alpha_0}{1+\alpha_0} f(I_f)}_{\geqslant 0} \geqslant \frac{\alpha_0}{1+\alpha_0} f(I) = \rho f(I)$,

that concludes the proof of the ratio.

We now study the running-time of the algorithm. Set, for each node $[I, \alpha]$, $T([I, \alpha]) = |I| - \alpha/\alpha_0$.

When doing a branching rule, a node is replaced by at most r nodes of sizes $|I| - a_1, |I| - a_2, \cdots, |I| - a_r$, and with α values $\alpha + s_1, \alpha + s_2, \cdots, \alpha + s_r$. Note that, for each child, there are at most $|I|$ (i.e., a polynomial number) diminutions (the size decrease by one after a diminution).

In any diminution that transforms $[I, \alpha]$ into $[I', \alpha']$, the quantity T becomes T', and $T' \leqslant T$, so these will not affect the overall computation time of our algorithm. Indeed, we know that $|I'| \leqslant |I| - 1$ because of Rule (D1). Also we have $\alpha' = \alpha - \alpha_0$, due to the definition of the algorithm. Then, $T' = |I'| - \frac{\alpha'}{\alpha_0} \leqslant |I| - 1 - \frac{\alpha - \alpha_0}{\alpha_0} = |I| - \frac{\alpha}{\alpha_0} = T$.

With the branching rule, if the k-th child of $[I, \alpha]$ is $[I_k, \alpha_k]$, and the quantity T becomes T_k, then we have $T_k \leqslant T - a_k - s_k/\alpha_0$. Indeed, we know using Rule (B1) that $|I_k| \leqslant |I| - a_k$. Also, by definition, we have $\alpha_k = \alpha + s_k$. Then, $T_k = |I_k| - \frac{\alpha_k}{\alpha_0} \leqslant |I| - a_k - \frac{\alpha + s_k}{\alpha_0} = T - a_k - \frac{s_k}{\alpha_0}$.

So the complexity according to the quantity T satisfies

$$C(T) \leqslant \sum_{k=1}^{r} C(T - a_k - \frac{s_k}{\alpha_0})$$

For the first node, the quantity T equals the size of the initial instance. Finally, we get a complexity $O^*(\gamma_\rho^{|I|})$, with γ_ρ root of $1 = \sum_{k=1}^{r} \gamma_\rho^{-a_k - s_k(1-\rho)/\rho}$, as claimed. □

Note that γ corresponds to the complexity of the exact branching algorithm (its complexity is $O^*(\gamma^n)$). The fact that $\gamma_\rho < \gamma$ follows from the fact that at least one s_i is strictly positive by hypothesis.

3.3.2 Adaptation for minimization problems

We can adapt this technique to make it work for minimization problems. The necessary condition to use this algorithm is as following: we need a branching rule with nearly the same properties as before: Rule (B2) becomes the following **Rule (B2')**: $\exists i, \forall S_i \in \mathscr{S}(I_i)$, $f(PB_i(S_i)) - f(S_i) \leqslant f(I) - f(I_i)$ (note that Rule (B3) remains the same).

Example 3.13. For MIN VC, the branching rule "corresponding" to the one in Example 3.6 is either to add a vertex v to the vertex cover (and to remove it from the graph), or to take all its neighbors in the vertex cover (and to remove them and v from the graph). This branching rule clearly satisfies Rules (B2') and (B3).

For the diminution rule, we need the same properties with the following modifications:
– Rule (D2) becomes: **Rule (D2')**: $f(I') \leqslant f(I)$;
– Rule (D3) becomes: **Rule (D3')**: $\forall S' \in \mathscr{S}(I')$, $f(PB'(S')) \leqslant 1 + f(S)$, i.e., the PB' function will not increase the size of a solution by more than one.

Take for MIN VC the rule consisting in removing a vertex v in the graph G (thus getting a graph G') and putting it into the vertex cover under construction. Given a vertex cover on G', the payback function simply adds v to G', thus producing a vertex cover of G. Then Rules (D1), (D2') and (D3') are obviously satisfied.

Theorem 3.14. For a minimization problem, with the new set of properties, Algorithm 3.2 has an approximation ratio of $(\alpha_0+1)/\alpha_0$ and the same complexity as before.

Proof. To prove the correctness, we have nearly the same property as in the demonstration above: each non-leaf node $[I, \alpha]$ in the tree has at least one child $[I', \alpha']$ with a payback function PB, and satisfying:

$$\forall S' \in \mathscr{S}(I'), f(PB(S')) - f(S') + \frac{\alpha' - \alpha}{\alpha_0} \leqslant \frac{\alpha_0 + 1}{\alpha_0}(f(I) - f(I')) \qquad (3.1')$$

Indeed, we have two cases with respect to α.

i) $\alpha \geqslant \alpha_0$. In this case, I' is derived from I using the diminution rule. For any solution $S' \in \mathscr{S}(I')$, we call $S = PB'(S')$. Then with Rule (D3'), we have $f(S) - f(S') \leqslant 1$. Also, as we used a diminution, we know that $\alpha' = \alpha - \alpha_0$,

and so $f(S) - f(S') + \frac{\alpha'-\alpha}{\alpha_0} \leqslant 1 - 1 \leqslant 0$. Using Rule (D2'), we know that $f(I) - f(I') \geqslant 0$, which gives (3.1').

ii) $\alpha < \alpha_0$. Then, I has up to r children, and using Rule (B2'), we know that for one child $[I', \alpha']$, and for any solution $S' \in \mathscr{S}(I')$, $f(S)-f(S') \leqslant f(I)-f(I')$ where $S = PB(S')$. Also, $f(S) - f(S') \geqslant \alpha' - \alpha$ (cf. Rule (B3)). So, it holds that $f(S) - f(S') + \frac{\alpha'-\alpha}{\alpha_0} \leqslant \frac{1+\alpha_0}{\alpha_0}(f(S) - f(S')) \leqslant \frac{1+\alpha_0}{\alpha_0}(f(I) - f(I'))$.

The end of the proof is therefore the same as previously. □

Let us conclude this section by the following remark. For clarity, we have fixed constants to 1 in Rules (D1) and (D2). It is worth noticing that similar results would immediately follow from using other constants. In particular (this will be useful for the examples below), suppose that we simply replace Rule (D2) by a new one: *(D2"): There exists a k such that $f(I') \geqslant f(I) - k$.* In this case, all we have to do is dividing the evaluation function by k, which preserves the approximation ratio. Therefore, the parameters s_i will become s_i/k (see Rule (B3)), and the complexity will be $O^*\left(\gamma^{|I|}\right)$, with γ root of $1 = \sum_{i=1}^{r} \gamma^{-a_i} - (s_i/k)(1-\rho)/\rho$.

3.3.3 Getting results with approximate branching

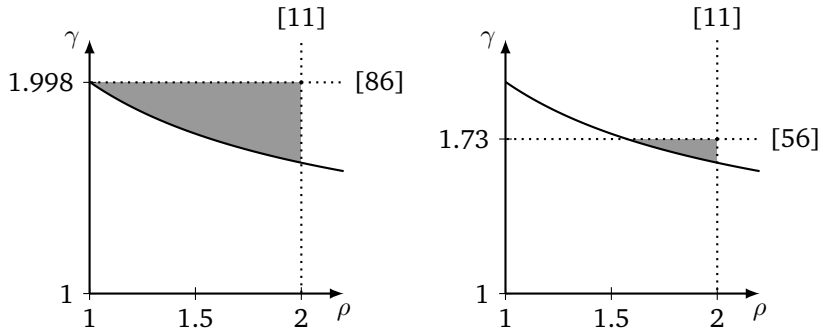
The techniques devised in Section 3.3.1 may be applied to a large variety of optimization problems. We first present in this section a detailed example on the MINIMUM FEEDBACK VERTEX SET (MIN FVS) problem. Then we show that it is in some sense more general than the technique of partitioning the instance [19] since it also works for hereditary problems. Finally, we present three other examples, handling the MIN FVS problem, the Max Cut problem and a non-hereditary problem, the Max k-Coverage problem. Note that we mention here simple applications of this technique: as already mentioned, a more involved implementation of this technique has been used in [50] to derive the best known bounds (until now) for approximating Max Sat for some ratios.

Feedback Vertex Set. Given a graph of order n, MIN FVS consists of finding a minimum-size set of vertices whose deletion makes the graph acyclic (see section 1.3.1, page 22). This problem is **NP**-hard in both directed or undirected graphs ([69]) and cn be optimally solved in time $O^*(1.9977^n)$ in the former class of graphs [86], and in time $O^*(1.7347^n)$ in the latter one [56]. Both cases are approximable in polynomial thime within ratio 2 [11]. We show how our method can be used in order to obtain ratios between 1 and 2 with improved computation times.

To obtain a branching rule on this problem, we formalize an instance of MIN FVS as a graph $G(V, E)$ with a set D of vertices excluded from the solution (we set $D = \emptyset$ at the beginning of the algorithm). The size of an instance is $|V| - |D|$. If we have an instance $I = (G, D)$, we can pick a vertex v from $V - D$ and define $I_1 = (G[V - \{v\}], D)$ and $I_2 = (G, D \cup \{v\})$. Our payback functions are $PB_1(S) = S \cup \{v\}$ and $PB_2(S) = S$. Note that if $D \cup \{v\}$ contains a cycle, then we can skip the I_2 branch as this instance will not have a solution. This branching has the desired properties: the size of the instance decreases by one on each branch, the payback functions give a feasible solution, they can be computed in polynomial time, and the optimal solution can be reached (take the first branch only when the vertex v is in the optimal solution), which ensures exhaustiveness.

We also have a diminution rule: if we have an instance $I = (G, D)$, then we pick any vertex v and we have $I' = (G[V - \{v\}], D)$ and $PB'(S) = S \cup \{v\}$. Also here we see that the size of the instance decreases by one, that the computations are polynomial, and the payback function does not increase the size of a solution by more than one.

Theorem 3.15. MIN FVS can be approximately solved within any ratio ρ in time $O^*(\gamma^n)$, with γ solution of the equation $\gamma \leqslant \gamma^{-1} + \gamma^{-\rho}$ (this result is summarized Figure 3.4).



(a) On undirected graphs (b) On directed graphs

Figure 3.4: Approximating MIN FVS. The grey area represents the improvement over the existing.

Hereditary problems. Our method can also handle hereditary problems. Let P be a maximization problem where, given an instance I, feasible solutions are subsets of a given set S that satisfy a given hereditary property π (if $S' \subseteq S$ satisfies π, then any $S'' \subseteq S'$ satisfies π), the goal being to maximize the size of the subset.

To make hereditary problems fitting the previous framework, we formulate P in a different (equivalent) manner, defining a problem P' where an instance is given by: (i) an instance I for the problem P; (ii) a set of nodes in the solution $T \subseteq S$; (iii) a set of nodes not in the solution $D \subseteq S$ with $D \cap T = \emptyset$. Feasible solutions of an instance (I, T, D) of P' are subsets $S' \subseteq S \setminus (T \cup D)$ such that $T \cup S'$ satisfies π (i.e., these are the feasible solutions of I containing T). The value of the solutions S' is its size $|S'|$.

As already mentioned, for any instance (I, T, D) of this new problem P', we have a natural branching rule: we take an element from S that is neither in T nor in D, and we add it either in T or in D. The branching continue as long as T is a feasible solution. This gives a trivial branching algorithm in time $O^*(2^n)$, where $n = |S|$. This branching rule is strictly monotonic. As a diminution rule, we only add one element to D. In the worst case, the element was in the optimal solution, then its size decreases by one.

So our algorithm applies, and gives an answer with approximation ratio ρ and computation time in $O^*(\gamma_\rho^n)$, γ_ρ being the greatest real solution of the equation $1 = \gamma_\rho^{-1} + \gamma_\rho^{-1-(1-\rho)/\rho}$.

Note that if this indicates that approximate branching algorithms seems to apply to more problems than the technique of partitioning the instance, it is worth noticing that the latter technique derives better running times for hereditary problems (compare $\gamma^{\rho n}$ with the running time of Theorem 3.12).

Note also that analogous results can be obtained for minimization problems where feasible solutions are subsets of a given set S that satisfy some property π such that if $S' \subseteq S$ satisfies π, then any $S'' \supseteq S'$ satisfies π, the goal being to minimize the size of the subset.

Indeed, suppose that P is a minimization problem where given an instance I, feasible solutions are subsets of a given set S that satisfy a given property π such that if $S' \subseteq S$ satisfies π, then any $S'' \supseteq S'$ satisfies π, the goal being to minimize the size of the subset.

Then, we rephrase the problem as in the maximization case by considering the sets T and D of already taken and discarded vertices. A feasible solution is a set $S' \subseteq S \setminus (T \cup D)$ such that $S' \cup T$ satisfies π, and its value is $|S'|$. We can use as branching rule the same as above. The diminution rule is to take an

element that and put it in the solution, which satisfies Rule (D2'). This allow us to find a ρ-approximation in $O^*(\gamma^n)$, γ being the solution of the equation $1 = \gamma^{-1} + \gamma^{-\rho}$.

For example, SET COVER consists, giving a set $E = \{E_1, E_2, \cdots, E_n\}$ of sets, of finding a subset E' of E with as few elements as possible such that $\bigcup_{E_i \in E'} E_i = \bigcup_{E_i \in E} E_i$. This problem is **NP**-hard. We can use the method of approximate branching because this problem is to find a subset of a solution satisfying the property above, and any set containing a solution is a solution.

Max Cut. As another example, consider the Max Cut problem (see section 1.3.1 page 23).

Max Cut consists, given a graph $G(V, E)$, of partitioning V into two subsets V_1 and V_2 such that the number of edges having one endpoint in V_1 and the other one in V_2 is maximal. The set of these crossing edges is called the cut (associated to V_1).

The basic branching schema consisting of fixing, let say, set V_1 and then of considering, for every vertex, that either it belongs, or it does not belongs to V_1, solves the problem in time $O^*(2^n)$ and this is the best worst-case running-time known for this problem in general graphs. However, in sparse graphs, i.e., in graphs with maximum degree bounded by d, the problem is solvable in time $O^*(2^{(1-(2/d))n})$ ([40]). On the other hand, Max Cut is approximable in polynomial time within ratio 0.8785 ([63]), but it is **APX**-hard ([84]).

Consider a graph with maximum degree d. We will show how we can use the techniques developed above in order to get non-trivial moderately exponential ratios.

Branching rule. We need a branching rule that guarantee us that, for each instance of the problem, at last one child will increase the size of any solution by one (cf. Rule (B3)). To achieve this, we first eliminate one pathologic case. If the graph is not connected, then we apply our algorithm on each connected component (which gives, of course, the optimal solution). Therefore, we will suppose in the following that our graph is connected. In the initial instance, we pick one vertex randomly, and we assign it in V_1. Then, the branching rule is to chose a vertex that is adjacent to another one in V_1 or V_2, and to make two children, placing this vertex in V_1 or V_2. If the first node was in V_1, then placing the node in V_2 will guarantee an increasing of the solution of at least one.

Diminution rule. If we now consider a graph G of degree at most d, then we can use the following easy result.

Property 3.16. In a graph of degree at most d, removing a vertex and all its edges decreases the size of the maximum cut by at most d.

Now we have a diminution rule: take a vertex, throw it away and remove every edge adjacent to this vertex in the same time, and Theorem 3.17 follows.

Theorem 3.17. If $G(V, E)$ is a graph of maximum degree d, then we can compute a ρ-approximation for Max Cut in time $O^*(\gamma^{|V|})$, with γ solution of $1 = \gamma^{-1} + \gamma^{-1-(1-\rho)/d\rho}$.

Note that this problem could also be seen as an hereditary problem: if a set S of edges is a cut, then any subset of S is also a cut (or at least a part of a cut). This would lead, using our algorithm, to an approximation algorithm with a complexity of $O^*(\gamma^{|E|}) = O^*(\gamma^{(n^2)})$ (with $\gamma > 1$ a constant depending on the approximation ratio). But such a result is not really interesting.

Max k-Coverage. Let us finally consider a non-hereditary problem, the Max k-Coverage problem in graphs with maximum degree d.

The Max k-Coverage problem, given a graph $G(V, E)$ and an integer k, asks for finding a subset S of V of size k that maximizes the number of edges incident to a vertex in S (see section 1.3.1 page 25). Here again, we will consider graphs of bounded degree d . Note that this problem is solvable in $O^*(2^{(d-1)n/(d+1)})$ by the basic branching schema (take a vertex or do not take it), in graphs with maximum degree d ([34]), and it is approximable in polynomial time within ratio 3/4 ([52, 66]), but it is **APX**-hard as generalization of MIN VC.

Max k-Coverage problem is polynomial if k is bounded by a fixed constant. The branching rule is trivial: take a vertex, add it in the solution or not. Stop branching when k vertex have been added. The diminution rule is simply to take a vertex and eliminate it. As the degree of the graph is bounded, then this elimination can't decrease the solution's size by more than d.

The following similar result holds.

Theorem 3.18. If $G(V, E)$ is an instance of Max k-Coverage of degree at most d, then we can compute a ρ-approximation of the solution with complexity in $O^*(\gamma^{|V|})$, with γ solution of $1 = \gamma^{-1} + \gamma^{-1-(1-\rho)/d\rho}$.

3.4 Around FPT approximation

We have proposed in this article a general framework to devise approximate branching algorithms that achieve polynomial time approximation schemata. We conclude this article by some considerations on similar issues for parameterized complexity. As mentioned in introduction, several recent articles aim at combining parameterized complexity and approximation algorithms (see for

instance [80]). Following these works, we shall define approximation classes in the parameterized framework[9].

Class 3.19 (AFPT). A problem is in **AFPT**[ρ] if there is a function f such that, for any instance of size n, it is possible in time $O^*(f(k,\rho))$ either to find a solution of size at least ρk or to show that there is no solution of size k.

A similar definition can be given for minimization problems.

Class 3.20 (ASFPT). A problem is said to have an **ASFPT** if it is in **AFPT**[ρ] for any $\rho \neq 1$.

Clearly, we know that **FPT** $\subseteq$ **AFPT**[ρ] for any ρ. Though there does not exist to our knowledge a "natural" problem that distinguishes between these classes, the inclusion is indeed strict. Take a $\rho > 1$ and the following problem: given a graph G, feasible solutions are proper colorings of G, and a coloring has value 3 if it uses at most 3 colors and 3ρ otherwise. Clearly, this problem is ρ approximable in polynomial time (hence in FPT time) and is not in **XP** (so not in **FPT**) unless **P** $=$ **NP**.

Property 3.21. $\forall\rho$, **FPT** $\subsetneq$ **AFPT**[ρ], unless **P** $=$ **NP**.

We also know that **APX**[ρ] $\subseteq$ **AFPT**[ρ] (where **APX**[ρ] denotes the set of problems ρ-approximable in polynomial time). Note that MIN VC is in **FPT** but is **NP**-hard to approximate with ratio $\rho < 1.36$ [41]. This shows strict inclusion for ratios $\rho < 1.36$. An amplification of the value of solutions allows to extend the strict inclusion for any ratio ρ: take t such that $1.36^t > \rho$, and define a modified vertex cover problem where the value of a vertex cover V' is $|V'|^t$. Then the problem is still in **FPT** but a polytime approximation ratio smaller than 1.36^t would solve vertex cover within ratio 1.36.

Property 3.22. $\forall\rho$, **APX**[ρ] $\subsetneq$ **AFPT**[ρ] unless **P** $=$ **NP**.

Finally, in the field of approximation parameterized algorithms, strong negative results have been obtained for MINIMUM INDEPENDENT DOMINATING SET (MIN IDS)[10] by [45], leaving as open questions the existence of approximation parameterized algorithms for MAX IS and MINIMUM DOMINATING

[9] Note that we only consider here the standard parameterization by the value of the solution, results dealing with other parameters have also been considered [80].

[10] Given a graph G, find the minimum size vertex subset which is both an independent set and a dominating set.

SET (MIN DS). As a last result, we deduce from the self-improvement property of MAX IS [61] that either MAX IS has a parameterized approximation schema or it does not admit constant approximation parameterized algorithms for any ratio. Note that this result has been used in [48] to obtain conditional negative results for the existence of parameterized approximation algorithms for MAX IS.

Theorem 3.23. If there is a ratio ρ such that MAX IS is in **AFPT**[ρ], then it is also in **AFPT**[$1-\epsilon$] for any $\epsilon > 0$.

Proof. We suppose that MAX IS is **AFPT**[ρ], which means that we have an algorithm with complexity $O(f(k)p(n))$ (with p a polynomial function) that either outputs an independent set of size at least ρk or outputs no (and in this latter case there is no independent set of size k).

Let $G(V,E)$ be a graph of size n. We build a new graph G' [61] with n^2 vertices $V_{i,j}$, with $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant n$. There is an edge between (i_1,j_1) and (i_2,j_2) if:

- $i_1 = i_2$ and $(j_1,j_2) \in E$.
- $i_1 \neq i_2$ and $(i_1,i_2) \in E$.

First, we remark that if we have an independent set of size k in G, then we have an independent set of size k^2 in G'. Reciprocally, if we have an independent set of size k in G', then we can find an independent set of size $\sqrt{k}$ in G:

- either this independent set has vertices in more than $\sqrt{k}$ copy of G, and then the numbers of these copy give an independent set on G.
- either is has vertices in less than $\sqrt{k}$ copy of G, and then there is at least $\sqrt{k}$ vertices in one of them, which gives an independent set on G too.

When given a graph G and a parameter k, we apply our parameterized algorithm to G' with the parameter k^2. If the algorithm outputs no, then we know that there is no independent set of size k in G and we output no too. Else, the algorithm outputs an independent set of size ρk^2 on G'. We know that this leads to an independent set of size $\sqrt{\rho}k$ on G. The complexity of this new algorithm $O(f(k^2)p(n^2))$. In other words, we know that MAX IS is now **AFPT**[$\sqrt{\rho}$]. Iterating this process proves the claimed result. □

4 Greediness for parameterized algorithms on cut problem

MAXIMUM $(k, n-k)$-CUT *(MAX K,N-K CUT) (resp.* MINIMUM $(k, n-k)$-CUT *(MIN K,N-K CUT)) est une version contrainte du problème* MAXIMUM CUT *(MAX CUT) (resp.* MINIMUM CUT *(MIN CUT)) dans lequel on doit partitionner les sommets d'un graphe en deux sous-ensembles de taille contrainte, de manière à maximiser (resp. minimiser) le nombre d'arêtes allant d'un sous-ensemble à l'autre. Pour ces problèmes, la paramétrisation par la taille du sous-ensemble k est déjà montrée* ***W[1]****-difficile. Nous montrons ici que MAX K,N-K CUT est* ***FPT*** *quand il est paramétrisé par la taille de la solution. Nous discutons ensuite d'une technique générale qui permet de montrer que MIN K,N-K CUT est* ***FPT*** *par rapport au paramètre $k + \Delta$, et nous montrons que cette technique peut s'appliquer sur une variété plus larges de problèmes à cardinalité fixée.*

MAX K,N-K CUT (resp. MIN K,N-K CUT) is a constrained version of MAX CUT (resp. MIN CUT) where one has to find a bipartition of the vertex set into two subsets with respectively k and $n-k$ vertices (n being the total number of vertices of the input graph) which maximizes (resp. minimizes) the number of edges going from one subset to the other. In this paper, we show that MAX K,N-K CUT and MIN K,N-K CUT are ***FPT*** *when parameterized by $k+\Delta$. From that, we derive that MAX K,N-K CUT is* ***FPT*** *with p (the number of edges in the cut) as parameter. Although these results could be obtain in a different way, using random separation which is a special color coding, we get better asymptotic running times. It was known that MIN K,N-K CUT and MIN K,N-K CUT are* ***W[1]****-hard. We give some approximation schemata which work in* ***FPT*** *time in k. Finally, we also show that both MAX K,N-K CUT and MIN K,N-K CUT are*

***FPT** parameterized by the size τ of a minimum vertex cover and the treewidth tw of the input graph.*

Contents

4.1 Introduction

Given a graph $G = (V, E)$ and two integers p and k, the decision version of the MAX K,N-K CUT problem consists of deciding whether there exists, or not, a subset $V' \subseteq V$ such that $|V'| = k$ and $|E(V', V \setminus V')| \geqslant p$, where $E(A, B) = \{xy \in E : x \in A \wedge y \in B\}$, while the MIN K,N-K CUT problem consists of deciding whether there exists, or not, a subset $V' \subseteq V$ such that $|V'| = k$ and $|E(V', V \setminus V')| \leqslant p$.

Both problems belong to a large set of problems, sometimes called fixed cardinality problems, where one wishes to optimize (maximize or minimize) the number of edges incident to exactly k vertices and satisfying some property. MAX k-VERTEX COVER, k-DENSEST SUBGRAPH (K-DENSEST), k-SPARSEST SUBGRAPH (K-SPARSEST), MAX K,N-K CUT and MIN K,N-K CUT are the most commonly known fixed cardinality problems.

When dealing with MAX K,N-K CUT and MIN K,N-K CUT, two natural parameters come immediately in mind, k and the value p of the solution $E(V', V \setminus V')$ (frequently called standard parameter). More about parameterized complexity can be found in [44]. The problems handled in this paper have mainly been studied in [23, 43], from a parameterized point of view, and have been proved **W[1]**-hard when parameterized by k, while complexity of standard parameterization remained open.

On the other hand, approximation of MIN K,N-K CUT has been studied in [51] where it is proved 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 $\varepsilon > 0$. Approximation of MAX K,N-K CUT

has been studied in several papers and a ratio 1/2 is achieved in [2] (slightly improved with a randomized algorithm in [52]), for all k.

Here, we first consider the parameterized complexity of MAX and MIN $(k, n-k)$-CUT with respect to the standard parameter p and prove that the former problem is fixed parameter tractable (**FPT**), while, unfortunately, the status of the latter one remains still unclear (Section 4.2). In order to handle MAX $(k, n-k)$-CUT we first show that when $p \leqslant k$ or $p \leqslant \Delta$, where Δ denotes the maximum degree of G, the problem is polynomial. So, the only "non-trivial" case occurs when $p > k$ and $p > \Delta$. We then design an **FPT** algorithm parameterized by $k + \Delta$ which allows us to conclude that MAX $(k, n-k)$-CUT parameterized by p is **FPT**. Then, a refinement of this algorithm allows to show that MIN K,N-K CUT parameterized by $k + \Delta$ is also **FPT**. Unfortunately, this shows inclusion in **FPT** of MIN K,N-K CUT only for some particular cases.

To achieve these **FPT** algorithms, which are the main contributions of this article, we use the following original idea, which might also be useful for other problems. It consists of performing a branching with respect to a vertex chosen upon some greedy criterion. For MAX $(k, n-k)$-CUT this criterion is to consider some vertex v that maximizes the number of edges added to the cut under construction. Without branching, the greedy algorithm is not optimal. However, the principle is that at each step either the greedily chosen vertex v is a good choice (it is in an optimal solution), or some of its neighbors (more precisely, a vertex at bounded distance from v) is a good choice (it is in an optimal solution). This induces a branching rule on neighbors of v which leads to a branching tree whose size is, in our case, bounded by a function of k and Δ.

In Section 4.3, we mainly revisit the parameterization by k but we handle it from an approximation point of view. Given a problem Π parameterized by parameter ℓ and an instance I of Π, a parameterized approximation algorithm for Π is an algorithm running in time $f(\ell)|I|^{O(1)}$ that either finds an approximate solution of size $g(\ell)$ as close as possible to ℓ, or reports that there is no solution of size ℓ. We prove that, although W[1]-hard for the exact computation, MAX $(k, n-k)$-CUT has a parameterized approximation schema with respect to k and MIN $(k, n-k)$-CUT a randomized parameterized approximation schema. These results exhibit two problems which are hard with respect to a given parameter but which become easier when we relax exact computation requirements and seek only (good) approximations. To our knowledge, the only other problem having similar behaviour is another fixed cardinality problem, the MAX k-VERTEX COVER problem, where one has to find the subset of k vertices which cover the greatest number of edges [80]. Note that the existence of problems having this behaviour but with respect to the standard parameter is an open (presumably

very difficult to answer) question in [80].

In Section 4.4, we first handle parameterization of both problems by the treewidth tw of the input graph and show, using a standard dynamic programming technique, that they admit an $O^*(2^{\mathrm{tw}})$-time **FPT** algorithm, when the $O^*(\cdot)$ notation ignores polynomial factors. Indeed, the result is stronger as we prove that a whole class of problems, including fixed cardinality problems but also, for instance MIN BISECTION problem can be solved by the same algorithm. Let us note that the interest of this result, except its structural aspect (many problems for the price of a single algorithm), lies also in the fact that MAX and MIN $(k, n-k)$-CUT does not fit Courcelle's Theorem [32]. Indeed, MAX and MIN BISECTION are not expressible in MSO since the equality of the cardinality of two sets is not MSO-definable. In fact, if one could express that two sets have the same cardinality in MSO, one would be able to express in MSO the fact that a word has the same number of a's and b's, on a two-letter alphabet, which would make that the set $E = \{w : |w|_a = |w|_b\}$ is MSO-definable. But we know that, on words, MSO-definability is equivalent to recognizability; we also know by the standard pumping lemma (see, for instance, [75]) that E is not recognizable [79], a contradiction. Henceforth, MAX and MIN $(k, n-k)$-CUT are not expressible in MSO; consequently, the fact that those two problems, parameterized by tw are **FPT** cannot be obtained by Courcelle's Theorem. Furthermore, even several known extended variants of MSO which capture more problems [88], does not seem to be able to express the equality of two sets either.

4.2 Standard parameterization

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

In the sequel, we denote by $N(v)$ the set of neighbors of v in $G = (V, E)$, namely $\{w \in V\colon \{v, w\} \in E\}$ and define $N[v] = N(v) \cup \{v\}$. We also use the standard notation $G[U]$ for any $U \subseteq V$ to denote the subgraph induced by the vertices of U. In this section, we show that MAX $(k, n-k)$-CUT parameterized by the standard parameter, i.e., by the value p of the solution, is **FPT**. Using an idea of bounding above the value of an optimal solution by a swapping process (see Figure 4.1), we show that the non trivial case satisfies $p > k$. We also show that $p > \Delta$ holds for non trivial instances and get the situation depicted by Figure 4.2. The rest of the proof (see Theorem 4.3) shows that MAX $(k, n-k)$-CUT parameterized by $k+\Delta$ is **FPT**, by designing a particular branching algorithm. This branching algorithm is based on the following intuitive idea.

Consider a vertex v of maximum degree in the graph. If an optimal solution $E(V', V \setminus V')$ is such that no vertex of $N(v)$ is in V', then it is always interesting to take v in V' (this provides Δ edges to the cut, which is the best we can do). This leads to a branching rule with $\Delta + 1$ branches, where in each branch we take in V' one vertex from $N[v]$.

Lemma 4.1. In a graph with minimum degree r, the optimal value OPT of a MAX K,N-K CUT satisfies OPT $\geqslant \min\{n-k, rk\}$.

ALGORITHM 4.1 – A swap algorithm for MAX K,N-K CUT in graphs of minimum degree r

Input : A graph $G(V, E)$ of minimum degree r, a parameter k
Output : A cut of size k and value at least $\min\{n-k, rk\}$

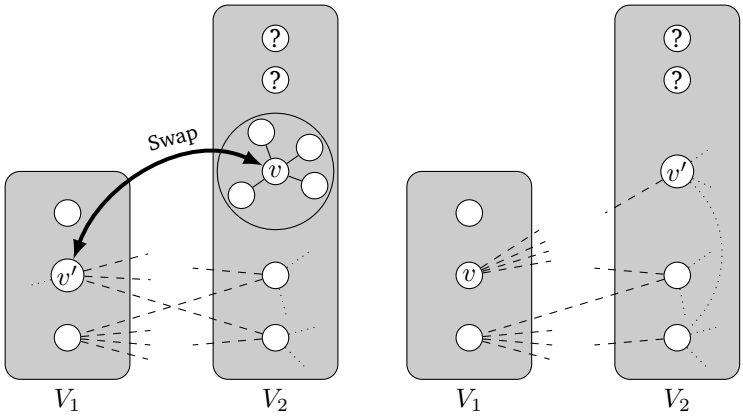
1 Divide arbitrarily the vertices of V into two subsets V_1 and V_2 of size k and $n-k$.
2 **if** *There is a vertex v in V_2 with no neighbor in V_1* **then**
3 | **if** *There is a vertex v' in V_1 with $< r$ neighbors in V_2* **then**
4 | | Swap v and v' in V_1 and V_2 (see Figure 4.1)
5 | | Go to step 2
6 | **else**
7 | | Output V_1 with a value at least rk.
8 | **end**
9 **else**
10 | Output V_1 with a value at least $n-k$.
11 **end**

Proof. We use the Algorithm 4.1. We know that this algorithm will halt because each swap (on step 4) increases strictly the value of the cut V_1/V_2 (moving the vertex v from V_2 to V_1 will increase the cut's value by at least r, and moving v' from V_1 to V_2 will reduce the value of the cut by at most $r-1$).

On step 7, every vertex in V_1 has at least r neighbors in V_2, so the cut's value is at least rk.

On step 10, every vertex in V_2 has a neighbor in V_1, so the cut's value is at least $|V_2| = n-k$. □

Corollary 4.2. In a graph with no isolated vertices, the optimal value for MAX $(k, n-k)$-CUT is at least $\min\{n-k, k\}$.



(a) As v' has at least one neighbor in V_1, it is swapped with v (which has all its neighbors in V_2).

(b) With the swapping the cut size increases, giving an upper bound over the computation time.

Figure 4.1: Illustration of a swapping

Theorem 4.3. The MAX K,N-K CUT problem parameterized by the standard parameter p is **FPT**.

Proof. Considering the symmetry between V' and $V \setminus V'$, one can assume that $k \leqslant \frac{n}{2} \leqslant n - k$. If G contains i isolated vertices, one can run the algorithm we are describing with k' going from $\min\{0, k - i\}$ to k, in G' that is G without the i isolated vertices, with the same value for p, and output YES, if one of the calls outputs YES. Thus, we consider that G contains no isolated vertices.

In addition, if $p > \frac{n}{2}$, then we may exhaustively compute all the $(k, n-k)$-cuts, by computing all the subsets of V on k vertices and the cut any of them induces, in time $O^*(2^n)$. Consequently, in this case, the time needed for the work is **FPT**, $O^*(2^{2p})$. So, in the sequel, we will assume $p \leqslant \frac{n}{2}$. Furthermore, by Corollary 4.2 and the discussion above, the overall assumption for the sequel (in the proof of the theorem) is $k \leqslant p \leqslant \frac{n}{2} \leqslant n - k$, since in a graph with no isolated vertices, the minimum degree is at least 1.

Denote by Δ the maximum degree in the input graph G, and let v_0 be a vertex with maximum degree. If $p \leqslant \Delta$, putting v_0 in V', setting $d = \min\{n - k, \Delta\}$ of its neighbors outside V' and completing arbitrarily (if necessary) the subset $V \setminus V'$, yields a cut of size k and of value at least d. Since $p \leqslant n - k$, we get

$p \leqslant d$, thus this cut is already a solution. So, together with $k \leqslant p \leqslant \frac{n}{2} \leqslant n-k$, we can, in addition, assume $\Delta < p$.

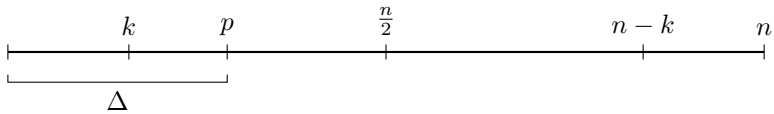


Figure 4.2: Location of parameter p, relatively to k and Δ.

At this point, we can already conclude using the generic algorithm of random separation [25] which deterministically takes time $O^*(2^{k(\Delta+1)}(k(\Delta+1))^{log(k(\Delta+1))})$ thanks to the derandomization presented in [82]. Though, we present the following branching algorithm which will achieve a better running time $O^*(\Delta^k)$.

We consider a set T which is initially empty, and which will intuitively correspond to V' at the end. At each branching step the algorithm puts one more vertex in T. Hence, the depth of the tree is k. To do the branching, we consider a vertex v in $V \setminus T$ which maximizes the quantity:

$$\delta_T(v) = \qquad\qquad |N(v) \cap (V \setminus T)| - |N(v) \cap T|$$

This is a greedy criterion: $\delta_T(v)$ represents the increasing of the value of the cut by taking v in T. For instance, initially $T = \emptyset$ and one can choose the vertex of maximal degree v_0.

Then, in the branching tree, the node has $|N[v] \cap (V \setminus T)|$ children: in each child we take one vertex from $N[v] \cap (V \setminus T)$ and put it in T. We stop branching in nodes where $|T| = k$ and output YES if one of the induced $(k, n-k)$-cuts has value at least p. Note that in branching steps $|N[v]|$ is bounded by $\Delta + 1$, and so is the number of sons of a node (i.e., the arity of the branching tree). Since the depth of the tree is k, the algorithm runs in $O^*(\Delta^k) = O^*(p^p)$.

Let us now show that our algorithm is sound. To prove optimality, we use a classical hybridation technique between some optimal solution and our solution. Consider an optimal solution V'_{opt} (in the sequel, we assume that a solution is represented by the subset V' of size k) different from the solution computed by the algorithm. A node w of the branching tree has two characteristics: the set of taken vertices $T(w)$ (or T if no ambiguity occurs) and the vertex chosen by the greedy criterion $v(w)$ (or simply v). We say that a node w of the branching tree is conform to the optimal solution V'_{opt} if $T(w) \subseteq V'_{\text{opt}}$. A node w deviates from the optimal solution V'_{opt} if none of its sons is conform to V'_{opt}.

We start from the root of our branching tree and we go to a conform son of our current node while it is possible. At some point we stop by reaching a node w which deviates from V'_{opt}. We set $T = T(w)$ and $v = v(w)$. Intuitively, T corresponds to the shared choices between the optimal solution and our algorithm made along the branch from the root to the node w. Given a vertex $z \in V'_{\text{opt}} \setminus T$, we consider the cut induced by the set $V'_{\text{opt}} \cup \{v\} \setminus \{z\}$ obtained from V'_{opt} by adding v and removing z. We show that this cut is also optimal. When removing z and adding v to V'_{opt}, we remove $\delta_{V'_{\text{opt}}}(z)$ edges and add $\delta_{V'_{\text{opt}}}(v)$ edges to the cut, so we only need to show that $\delta_{V'_{\text{opt}}}(v) \geq \delta_{V'_{\text{opt}}}(z)$ (which is then an equality by optimality of V'_{opt}). First note that $\delta_T(z) \leqslant \delta_T(v)$, by the choice of v. Moreover, since $T \subseteq V'_{\text{opt}}$, $\delta_{V'_{\text{opt}}}(z) \leq \delta_T(z)$ (more generally the quantity δ_T never increases since vertices are never removed from T). Furthermore, since the optimal solution does not take any vertex in $N(v) \cap (V \setminus T)$, it holds that $\delta_T(v) = \delta_{V'_{\text{opt}}}(v)$. Therefore, $\delta_{V'_{\text{opt}}}(z) \leqslant \delta_{V'_{\text{opt}}}(v)$, and the cut induced by $V'_{\text{opt}} \cup \{v\} \setminus \{z\}$ is optimal. Thus, by repeating this process at most k times, we can conclude that the solution computed by the algorithm has the same value as the optimal solution considered. □

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

We give in this section a result analogous to that of Theorem 4.3 for MIN $(k, n-k)$-CUT. We devise a more "subtle" **FPT** algorithm in parameter $k+\Delta$, with some refinements of the proof of Theorem 4.3. Unfortunately, in the minimization case of $(k, n-k)$-cut, it enables us to conclude about the parameterized complexity in p, only in some particular cases.

Before presenting the algorithm, let us remark that the same branching algorithm as in the case of MAX K,N-K CUT does not work anymore, since choosing to add a vertex in the solution V' could seem really bad locally, but could turn out to be an optimal choice if all its neighbors are taken in the solution later. In other words, it does not seem possible to design a proper notion of "contribution" of one vertex such as $\delta_T(v)$ which only increases from the moment where it is added in the solution until the end. In order to get through this difficulty, the algorithm we present has to be more involved. In particular, first it does not only assign vertices to V', but it also fixes some vertices to $V \setminus V'$ (definitely). To this purpose, we will maintain a partition of the vertices of V in three sets T (taken vertices), R (rejected vertices) and U (unmarked vertices). More importantly, instead of exploring only the neighborhood of a vertex v when branching, it explores the set of vertices at distance at most k from v.

Theorem 4.4. MIN K,N-K CUT parameterized by $k + \Delta$ is **FPT**.

Proof. Let $(G = (V, E), p, k)$ be a general instance of the MIN K,N-K CUT problem, and Δ be the maximum degree of G. Let us first introduce the following additional notations. The set $N^k[v]$ denotes the set of vertices which are at distance at most k from v in the graph G ($N^1[v] = N[v]$); it is called k-neighborhood of v. In particular, the size of $N^k[v]$ for a given vertex v is bounded by Δ^{k+1}, thus respecting **FPT**-time (with respect to $k + \Delta$).

Besides, one can consider an optimal solution as the union of its (maximal) connected components $V'_1, V'_2, \dots V'_h$ (with $V' = \bigcup_{i=1}^{h} V'_i$). Indeed, the contribution of a connected component V'_i is not varying as long as the solution is built and is definitively equal to $\delta(V'_i)$, where δ counts the number of outgoing edges of a subset of vertices. And the value opt associated to an optimal solution induced by V' is equal to $\Sigma_{i=1}^{h}\delta(V'_i)$. This idea seems to be good for decomposing an optimal solution, but if one also uses it for computing a solution (by adding at each step in the solution not a vertex but a whole connected component), one will be in trouble to prove optimality, since the hybridation has to preserve the size k of the solution.

The Algorithm 4.2 deals with the technical problems presented above.

Obviously, we return the minimum value of the $(k, n - k)$-cut among the solutions at the leaves of the branching tree, and we compare it to p.

Let us now analyze the complexity of the above algorithm. It is divided into two parts: the first part is computation of $v_1, v_2, \dots, v_k$ and the second part is the branching. Since $|N^k[v]| \leqslant \Delta^{k+1}$, computing $\delta_{i,k}(v)$ and determining $v_1, v_2, \dots, v_k$ takes time $O^*((\Delta^{k+1})^k)$. Now, the branching tree has arity at most $k(\Delta^k + 1)$ and depth at most k. So, it has at most $k^k(\Delta^k + 1)^k$ nodes. Then, computing the value of a cut can be done in polynomial time, and our algorithm works in **FPT**-time $O^*(k^k\Delta^{k^2})$.

We prove here that the algorithm computes an optimal solution by using hybridation of the optimal solution and of its solution. Let V'_{opt} be an optimal solution. First, we decompose V'_{opt} in its connected components $V'_{1,\text{opt}}, V'_{2,\text{opt}}, \dots, V'_{h,\text{opt}}$ (as mentioned above). Each node of the branching tree corresponds to some configuration (T, R, U). For instance, the root corresponds to the configuration $(\emptyset, \emptyset, V)$ and any leaf corresponds to a configuration with $|T| = k$. We say that a node of the branching tree is conform to the optimal solution V'_{opt}, if its configuration (T, R, U) satisfies $T \subseteq V'_{\text{opt}}$ and $R \cap V'_{\text{opt}} = \emptyset$. Also, we say that a node of the branching tree deviates from the optimal solution when none of its children is conform to the optimal solution.

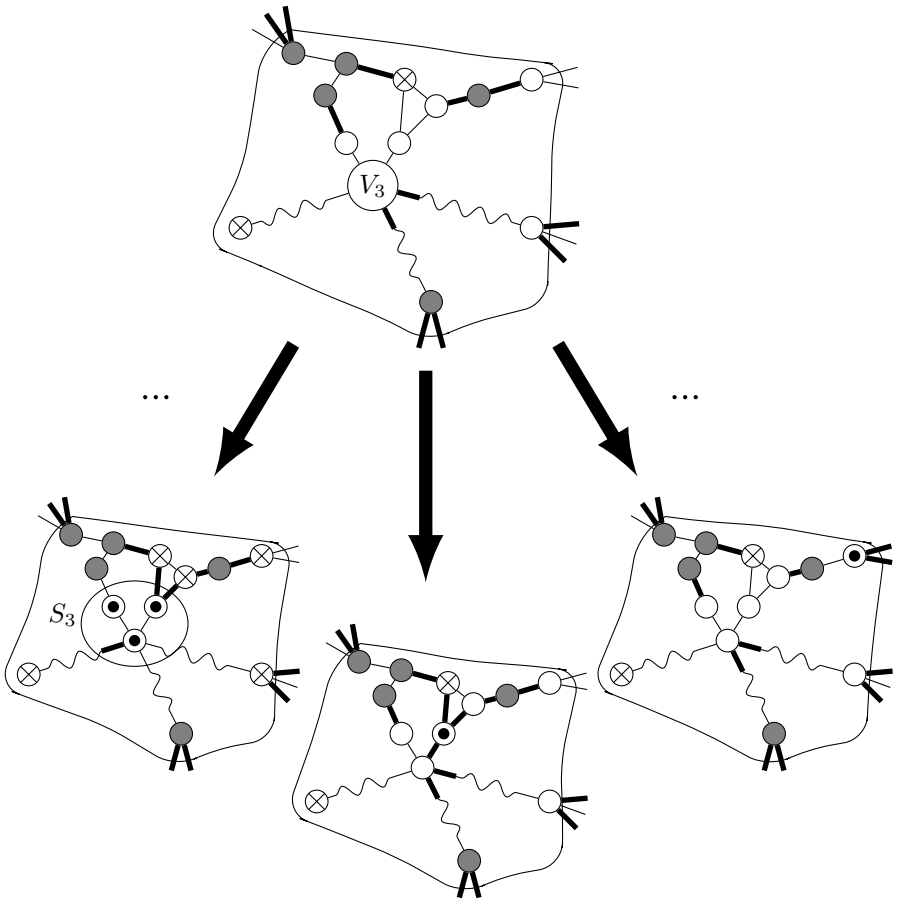


Figure 4.3: The k-neighborhood of v_3 in a fragment of the branching tree. ● are taken vertices, ⊗ are rejected vertices, ○ are unmarked vertices, and ⊙ are the vertices which has just been added to the set of taken vertices. Bold edges are the edges of the current cut. Assume the father node deviates from the optimal solution. Then, the optimal solution takes no unmarked vertices in the k-neighborhood of v_3. Thus, we can substitute S_3 which is the best local choice for adding 3 vertices, to the 3 vertices of the optimal solution which completes one connected component (see Figure 4.4).

ALGORITHM 4.2 – Greedy parametric algorithm to solve MIN K,N-K CUT

Input : An graph G, a parameter k
Output : An optimal solution for the MIN K,N-K CUT problem on G

$\emptyset \to T$
$\emptyset \to R$
if $k > 0$ **then**
  for *each vertex* $v \in U$ *and each* i *from 1 to* k **do**
    Compute the minimum $\delta_{i,k}(v) = \min_S\{\delta(S) - 2|E(S,T)|\}$ where S is any subset of size i of $N^k[v] \cap U$ and determine k vertices $v_1, v_2, \ldots, v_k$ respectively minimizing $\delta_{1,k}, \delta_{2,k}, \ldots, \delta_{k,k}$; let $S_1, S_2, \ldots, S_k$ be the corresponding minimizing subsets
    for *each* i *from 1 to* k **do**
      for *each vertex* v *of* $N^k[v_i] \cap U$ **do**
        Restart from step 3 with $T \cup \{v\} \to T$; $k - 1 \to k$
      end
      Restart from step 3 with $T \cup S_i \to T$; $R \cup (N^k[v_i] \setminus S_i) \to R$; $k - |S_i| \to k$
    end
  end
else
  Output $(T, V \setminus T)$.
end

>From the root, we follow any branch of the branching tree until we reach a node which deviates from V'_{opt}, and we denote by $V'_{\text{shared}} \subset V'_{\text{opt}}$ the set of vertices in the optimal solution V'_{opt} which has been taken in our solution along the followed branch. In other words, V'_{shared} corresponds to shared choices, up to this point, between our branching and the optimal solution (note that V'_{shared} and the crossway node are not necessarily unique). At this node, our algorithm computes $v_1, v_2, \ldots, v_k$ and we know for sure that, for all i, $N^k[v_i] \cap (V'_{\text{opt}} \setminus V'_{\text{shared}}) = \emptyset$. Take a whole connected component $V'_{j,\text{opt}}$ not entirely contained in V'_{shared} and let $c_j = |V'_{j,\text{opt}} \setminus V'_{\text{shared}}|$.

Let us now explain why component S_3 of Figure 4.3 (role played here by S) could substitute V'' of Figure 4.4 (role played here by $V'_{j,\text{opt}} \setminus V'_{\text{shared}}$).

By construction, in $N^k[v_{c_j}] \setminus V'_{\text{shared}}$, and by the previous remark in $N^k[v_{c_j}] \setminus V'_{\text{opt}}$, there exists a subset S with c_j elements which satisfies $\delta(S \cup V'_{\text{shared}}) \leqslant$

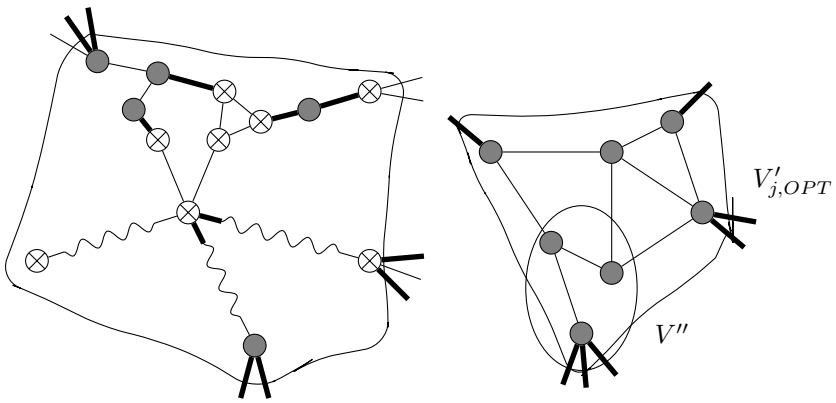


Figure 4.4: Representation of the optimal solution. Filled vertices are in V'_{opt} and the crossed vertices are in the other subset; $V'' = V'_{j,\text{opt}} \setminus V'_{\text{shared}}$; where $V'_{j,\text{opt}}$ is a connected component of V'_{opt} and V'_{shared} denotes the vertices both taken by the optimal solution and by `MINCUT`. As the node of the branching tree of Figure 4.3 is supposed to deviate from V'_{opt}, we can notice that no unmarked vertex of the extended neighborhood of v_3 is taken in V'_{opt}.

$\delta(V'_{j,\text{opt}} \cup V'_{\text{shared}})$. Now, let us consider the set of k vertices $W = (V'_{\text{opt}} \setminus (V'_{j,\text{opt}} \setminus V'_{\text{shared}})) \cup S$. In the sequel, we set $V'' = V'_{j,\text{opt}} \setminus V'_{\text{shared}}$, $X = V'_{\text{shared}} \cup V'_{j,\text{opt}} = V'' \cup V'_{\text{shared}}$ and we repeatedly use that $\delta(A \cup B) = \delta(A) + \delta(B) - 2|E(A,B)|$. It holds that:

$$\begin{aligned}\delta(W) =& \delta\left(S \cup V'_{\text{shared}}\right) + \delta\left(W \setminus (S \cup V'_{\text{shared}})\right) \\ & - 2\left|E\left(S \cup V'_{\text{shared}}, W \setminus (S \cup V'_{\text{shared}})\right)\right|\end{aligned}$$

Besides, $W \setminus (S \cup V'_{\text{shared}}) = V'_{\text{opt}} \setminus X$ and, since S and V'_{shared} are disjoint:

$$\begin{aligned}\left|E\left(S \cup V'_{\text{shared}}, W \setminus (S \cup V'_{\text{shared}})\right)\right| &= \left|E\left(V'_{\text{shared}}, V'_{\text{opt}} \setminus X\right)\right| + \left|E\left(S, V'_{\text{opt}} \setminus X\right)\right| \\ &\geqslant \left|E\left(V'_{\text{shared}}, V'_{\text{opt}} \setminus X\right)\right|\end{aligned}$$

Recall that, by construction of S, $\delta(S \cup V'_{\text{shared}}) \leqslant \delta(X)$. Hence:

$$\delta(W) \leqslant \delta(X) + \delta\left(V'_{\text{opt}} \setminus X\right) - 2\left|E\left(V'_{\text{shared}}, V'_{\text{opt}} \setminus X\right)\right|$$

Also, by construction, $|E(V'', V'_{\text{opt}} \setminus X)| = 0$; so:

$$\begin{aligned}\delta(W) &\leqslant \delta(X) + \delta\left(V'_{\text{opt}} \setminus X\right) - 2\left(\left|E\left(V'_{\text{shared}}, V'_{\text{opt}} \setminus X\right)\right| + \left|E\left(V'', V'_{\text{opt}} \setminus X\right)\right|\right) \\ &= \delta(X) + \delta\left(V'_{\text{opt}} \setminus X\right) - 2\left|E\left(X, V'_{\text{opt}} \setminus X\right)\right| \;=\; \delta\left(V'_{\text{opt}}\right) \qquad (4.1)\end{aligned}$$

where the first equality in (4.1) holds due to the fact V'_{shared} and V'' are disjoint.

By optimality of V'_{opt}, one gets $\delta(W) = \delta(V'_{\text{opt}})$. In the branching tree, we move to the one son taking S and we follow any branch until our algorithm deviates again from the optimal solution. Then we iterate the same hybridation trick. After at most k steps, we get to a leaf of the branching tree with a solution V' which is as good as the optimal solution. □

Let us note that Algorithm 4.2 can be slightly modified in order to work also for another fixed-cardinality problem, the K-DENSEST problem, which consists of determining a subset of k vertices maximizing the number of edges in the subgraph induced by them. Indeed, it suffices to switch from minimization of the number of outgoing edges to maximization of the number of inner edges. The rest of the proof remains the same.

Corollary 4.5. K-DENSEST, parameterized by $k + \Delta$, is **FPT**.

We now return to the standard parameterization of MIN K,N-K CUT. Unfortunately, unlike what have been done for MAX K,N-K CUT, we have not been able to show until now that the case $p < k$ is "trivial". However, we can prove that when $p \geqslant k$, then MIN K,N-K CUT parameterized by the value p of the solution is **FPT**. This is an immediate corollary of the following proposition.

Proposition 4.6. MIN K,N-K CUT parameterized by $p + k$ is **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 4.4 by considering the k-neighborhood of a vertex v not in the whole graph G, but in $G[T \cup U]$. One can easily check that the algorithm still works and since in those subgraphs the degree is bounded by $p + k$ we get an **FPT**algorithm in $p + k$. □

In [51], 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 [51] 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 showing that, when $p \leqslant k$, MIN K,N-K CUT can be solved in time $O^*(n^p)$.

Proposition 4.7. If $p \leqslant k$, then 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. □

4.3 Parameterization by k and approximation

Recall that both MAX and MIN K,N-K CUT parameterized by k are **W[1]**-hard ([43, 23]). In this section, we give some approximation algorithms working in **FPT** time with respect to parameter k.

Proposition 4.8. MAX K,N-K CUT, parameterized by k has a fixed-parameter approximation schema.

Proof. 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 4.3 with time-complexity $O^*(\Delta^k)$ is in this case an $O^*(\frac{k^{2k}}{\varepsilon^k})$-time algorithm. □

Proposition 4.9. MIN K,N-K CUTparameterized by k has a randomized fixed-parameter approximation schema.

Proof. For any $\varepsilon > 0$, if $k < \log n$, then according to the result of [51], there exists a randomized polynomial time $(1+\varepsilon)$-approximation. Else, if $k > \log n$, the exhaustive enumeration of the k-subsets takes time $O^*(n^k) = O^*((2^k)^k) = O^*(2^{k^2})$. □

Finding approximation algorithms that work in **FPT** time with respect to parameter p is an interesting question. Combining the result of [51] and an $O(\log^{1.5}(n))$-approximation algorithm in [52] we can show that the problem is $O(k^{3/5})$ approximable in polynomial time by a randomized algorithm.

We now show that an approximation ratio $\frac{k^2}{f(k)} + 1$ for MIN K,N-K CUTcan be achieved in time $O^*(n^{f(k)})$. This, for instance, concludes a ratio $o(k^2)$ in time $O^*(n^{o(k)})$. 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 subcases, namely $p \leqslant f(k)$ and $k \geqslant p \geqslant f(k)$. In the first of the subcases, using Proposition 4.7, an optimal solution for MIN K,N-K CUT can be found in time at most $O^*(n^{f(k)})$. For the second subcase, 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., $\sigma \leqslant \text{OPT} + k^2$. Hence, if $p < \sigma - k^2$, the algorithm answers "no"; otherwise, some easy algebra leads to an approximation ratio bounded from above by $\frac{k^2}{f(k)} + 1$.

We feel that much better results should be achievable for MIN K,N-K CUT in **FPT** time.

4.4 Other parameterizations

When dealing with parameterization of graph problems, some classical parameters arise naturally. One of them, very frequently used in the fixed parameter literature is the treewidth of the graph.

In [67], an algorithm solving MIN and MAX K,N-K CUT in $O^*(2^{\text{tw}})$ is given. In [21], the same is done for K-DENSEST. We show here that this algorithm could be adapted for a large class of graph problems including those three problems.

Definition 4.10. A local graph partitioning problem is a problem having as input a graph $G = (V, E)$ and two integers k and p. Feasible solutions are

subsets $V' \subseteq V$ of size k. The value of a solution, denoted by $\mathrm{val}(V')$, is a linear combination $\alpha_1 m_1 + \alpha_2 m_2$ where $m_1 = |E(V')|$, $m_2 = |E(V', V \setminus V')|$ and $\alpha_1, \alpha_2 \in \mathbb{R}$. The goal is to determine whether there exists a solution of value at least p (for a maximization problem) or at most p (for a minimization problem). □

Note that $\alpha_1 = 1$, $\alpha_2 = 0$ corresponds to K-DENSEST and K-SPARSEST, while $\alpha_1 = 0$, $\alpha_2 = 1$ corresponds $(k, n-k)$-CUT, and $\alpha_1 = \alpha_2 = 1$ gives MAXIMUM k-COVERAGE (MAX K-COVERAGE). Not only any fixed cardinality problem (as those mentioned above and in [23]) but many other problems fit Definition 4.10. For instance, a very well-known problem, the MINIMUM BISECTION (MIN BISECTION) problem where one wishes to minimize the cut between the sets of an equipartition of the vertices into two sets, is also a local graph partitioning problem.

Proposition 4.11. Any local graph partitioning problem can be solved in time $O^*(2^{\mathrm{tw}})$.

Proof. A tree decomposition of a graph $G(V, E)$ is a pair (X, T) where T is a tree on vertex set $N(T)$ the vertices of which are called nodes and $X = (\{X_i : i \in N(T)\})$ is a collection of subsets of V such that: (i) $\cup_{i \in N(T)} X_i = V$, (ii) for each edge $(v, w) \in E$, there exist an $i \in N(T)$ such that $\{v, w\} \in X_i$, and (iii) for each $v \in V$, the set of nodes $\{i : v \in X_i\}$ forms a subtree of T. The width of a tree decomposition $(\{X_i : i \in N(T)\}, T)$ equals $\max_{i \in N(T)}\{|X_i| - 1\}$. The treewidth of a graph G is the minimum width over all tree decompositions of G. We say that a tree decomposition is nice if any node of its tree that is not the root is one of the following types:

- a leaf that contains a single vertex from the graph;
- an introduce node X_i with one child X_j such that $X_i = X_j \cup \{v\}$ for some vertex $v \in V$;
- a forget node X_i with one child X_j such that $X_j = X_i \cup \{v\}$ for some vertex $v \in V$;
- a join node X_i with two children X_j and X_l such that $X_i = X_j = X_l$.

Assume that the local graph partitioning problem Π is a minimization problem (we want to find V' such that $\mathrm{val}(V') \leqslant p$), the maximization case being similar. An algorithm that transforms in linear time an arbitrary tree decomposition into a nice one with the same treewidth is presented in [72]. Consider a nice tree

decomposition of G and let T_i be the subtree of T rooted at X_i, and $G_i = (V_i, E_i)$ be the subgraph of G induced by the vertices in $\bigcup_{X_j \in T_i} X_j$. For each node $X_i = (v_1, v_2, \ldots, v_{|X_i|})$ of the tree decomposition, define a configuration vector $\vec{c} \in \{0,1\}^{|X_i|}$; $\vec{c}[j] = 1 \iff v_j \in X_i$ belongs to the solution. Moreover, for each node X_i, consider a table A_i of size $2^{|X_i|} \times (k+1)$. Each row of A_i represents a configuration and each column represents the number k', $0 \leq k' \leq k$, of vertices in $V_i \setminus X_i$ included in the solution. The value of an entry of this table equals the value of the best solution respecting both the configuration vector and the number k', and $-\infty$ is used to define an infeasible solution. In the sequel, we set $X_{i,t} = \{v_h \in X_i : \vec{c}(h) = 1\}$ and $X_{i,r} = \{v_h \in X_i : \vec{c}(h) = 0\}$.

The algorithm examines the nodes of T in a bottom-up way and fills in the table A_i for each node X_i. In the initialization step, for each leaf node X_i and each configuration $\vec{c}$, we have $A_i[\vec{c}, k'] = 0$ if $k' = 0$; otherwise $A_i[\vec{c}, k'] = -\infty$.

If X_i is a forget node, then consider a configuration $\vec{c}$ for X_i. In X_j this configuration is extended with the decision whether vertex v is included into the solution or not. Hence, taking into account that $v \in V_i \setminus X_i$ we get:

$$A_i\left[\vec{c}, k'\right] = \min\left\{A_j\left[\vec{c} \times \{0\}, k'\right], A_j\left[\vec{c} \times \{1\}, k' - 1\right]\right\}$$

for each configuration $\vec{c}$ and each k', $0 \leq k' \leq k$.

If X_i is an introduce node, then consider a configuration $\vec{c}$ for X_j. If v is taken in V', its inclusion adds the quantity $\delta_v = \alpha_1 |E(\{v\}, X_{i,t})| + \alpha_2 |E(\{v\}, X_{i,r})|$ to the solution. The crucial point is that δ_v does not depend on the k' vertices of $V_i \setminus X_i$ taken in the solution. Indeed, by construction a vertex in $V_i \setminus X_i$ has its subtree entirely contained in T_i. Besides, the subtree of v intersects T_i only in its root, since v appears in X_i, disappears from X_j and has, by definition, a connected subtree. So, we know that there is no edge in G between v and any vertex of $V_i \setminus X_i$. Hence, $A_i[\vec{c} \times \{1\}, k'] = A_j[\vec{c}, k'] + \delta_v$, since k' counts only the vertices of the current solution in $V_i \setminus X_i$. The case where v is discarded from the solution (not taken in V') is completely similar; we just define δ_v according to the number of edges linking v to vertices of T_i respectively in V' and not in V'.

If X_i is a join node, then for each configuration $\vec{c}$ for X_i and each k', $0 \leq k' \leq k$, we have to find the best solution obtained by k_j, $0 \leq k_j \leq k'$, vertices in A_j plus $k' - k_j$ vertices in A_l. However, the quantity $\delta_{\vec{c}} = \alpha_1 |E(X_{i,t})| + \alpha_2 |E(X_{i,t}, X_{i,r})|$ is counted twice. Note that $\delta_{\vec{c}}$ depends only on $X_{i,t}$ and $X_{i,r}$, since there is no edge between $V_l \setminus X_i$ and $V_j \setminus X_i$. Hence, we get:

$$A_i\left[\vec{c}, k'\right] = \max_{0 \leq k_j \leq k'} \left\{A_j\left[\vec{c}, k_j\right] + A_l\left[\vec{c}, k' - k_j\right]\right\} - \delta_c$$

and the proof of the proposition is completed. □

Corollary 4.12. MAX K,N-K CUT and MIN K,N-K CUT parameterized by the treewidth of the input graph are **FPT**.

Corollary 4.13. Restricted to trees, MAX and MIN K,N-K CUT can be solved in polynomial time.

Corollary 4.14. MIN BISECTION parameterized by the treewidth of the input graph is **FPT**.

It is worth noticing that the result easily extends to the weighted case (edges have weight) and to the case of partitioning V into a constant number of classes (with a higher running time).

Another natural parameter frequently used in the parameterized complexity framework is the size τ of a minimum vertex cover of the input graph. Since it always holds that $\mathrm{tw} \leqslant \tau$, the result of Proposition 4.11 immediately applies to parameterization by τ. However, the algorithm developed there needs exponential space. In what follows, we give a parameterization by τ using polynomial space.

Proposition 4.15. MAX and MIN K,N-K CUTparameterized by τ can be solved in **FPT** $O^*(2^\tau)$ time and in polynomial space.

Proof. Consider the following algorithm:

- compute a minimum vertex cover C of G;
- for every subset X of C of size $|X|$ smaller than k, complete X with the $k - |X|$ vertices of $V \setminus C$ that maximize (resp., minimize) their incidence with $C \setminus X$ (i.e., the number of neighbours in $C \setminus X$);
- output the best solution.

Recall that a minimum size vertex cover can be computed in time $O^*(1.2738^\tau)$ time by means of the fixed-parameter algorithm of [28] and using polynomial space. The operation on every subset is polynomial, so the global computation time is at most $O^*(2^\tau)$.

The soundness follows from the fact that a complement of a vertex cover is an independent set. Denoting by V' the optimal vertex-set (i.e., the k vertices inducing an optimal cut), then $V' \cap C$ will be considered by the above algorithm, and then every vertex of the completion will add exactly to the solution its

number of neighbors in $V' \cap C$, which is maximized (or minimized) in the algorithm. □

4.5 Concluding remarks

Table 4.1 sums up results about parameterizations of MAX and MIN $(k, n-k)$-CUT. The first two lines concern exact algorithms, while the last two deal with approximations. The bold entries show our contribution. Notations fpt(r)AS denote FPT (random) approximation schema. Let us note that our results derive some additional interesting corollaries, for instance, *both problems studied, when parameterized by k are FPT in graphs of bounded degree*.

Table 4.1: The state of the art of MAX and MIN $(k, n-k)$-CUT.

Parameters	k	$k+\Delta$	p	τ	tw
MAX K,N-K CUT	W[1]-hard	**FPT**	**FPT**	**FPT**	**FPT**
MIN K,N-K CUT	W[1]-hard	**FPT**	**FPT for p $>$ k**	**FPT**	**FPT**
MAX K,N-K CUT	**fptAS**				
MIN K,N-K CUT	**fptrAS**				

Settlement of the parameterized complexity of MIN K,N-K CUTwith respect to p is, in our opinion, the major open problem deserving further efforts. Note that, if this problem is FPT, then this result would immediately apply to the case of MIN BISECTION, whose standard parameterization is an open problem. Another interesting open question, in the case where MIN K,N-K CUTis not **FPT**, is to design parameterized (with respect to p) approximation algorithms achieving ratios better than $O(k^2)$ (deterministic algorithm) and $O(k^{3/5})$ (randomized algorithms).

Finally, we hope that applying the idea of devising search tree based on the exploration of the neighborhood of a greedily chosen vertex could be fruitfully applied to other problems.

5 Conclusion

Nous avons présenté dans l'introduction trois approches pour les problèmes **NP**-difficiles :

- L'approche exacte qui vise à obtenir la complexité la plus intéressante pour résoudre un problème. Cela se fait via la conception d'algorithmes plus robustes (études détaillées et parfois fastidieuses de cas), leur analyse plus poussée (les techniques de *Measure-and-conquer* sont un bon exemple d'amélioration générale des analyses). On trouve aussi souvent des améliorations dans des instances particulières (graphe à degré borné, sans cycle, etc.).

- L'approche approchée à ratio garanti qui vise à produit des algorithmes à complexité plus intéressante que celle des algorithmes exacts, mais en interdisant que les solutions soient trop mauvaises par rapport à la solution optimale. Dans ce domaine se pose naturellement la question des compromis possibles entre complexité des algorithmes et qualité des garanties apportées. On se pose également la question d'existence de techniques très générales qui permettent de produire des algorithmes approchés pour une grande variété de problèmes.

- L'approche paramétrique qui vise à produire une solution optimale, mais avec un temps de calcul qui dépend fortement d'un paramètre de l'instance. Sous le prisme du paramétrique, il reste encore de nombreuses choses à trouver. D'abord parce que tous les problèmes n'ont pas été traités. Mais surtout parce que pour un même problème, de nombreux paramètres sont utilisables (la taille de la solution, une donnée entière pour des problèmes à cardinalité fixée, mais aussi le tree-width, ou la taille d'un vertex cover), et donnent parfois des résultats différents.

Cependant, cette thèse ouvre d'intéressantes perspectives de recherches futures, en particulier dans le domaine des algorithmes approchés. En effet, de nouvelles techniques générales sont encore à concevoir pour couvrir encore de nouveaux problèmes. Même sur un problème pour lequel une technique générale s'applique, on peut parfois espérer l'améliorer en tirant parti de caractéristiques du problème. Ainsi, la méthode d'élagage présentée au chapitre 3 bénéficie d'une amélioration particulière quand on l'applique au problème MAX SAT, qui la rend compétitive pour certains ratios d'approximation.

Il est également intéressant de souligner qu'on ne dispose pour le moment d'aucun résultat négatif, ni surtout d'aucun résultat négatif général.

Dans le domaine de l'approximation polynomiale, le théorème **PCP** a permis depuis 1990 d'obtenir de nombreux résultats négatifs, et continue d'être exploité dans ce cadre.

Dans le domaine des algorithmes exacts, on déduit de l'hypothèse **ETH** (qui affirme, informellement, qu'on ne peut résoudre aucun MAX k-SAT avec une complexité sous-exponentielle) des résultats négatifs pour bien des problèmes.

Cependant, en approximation exponentielle, qui se situe au croisement de ces deux domaines, on ne dispose ni d'un tel théorème, ni d'une telle hypothèse (une hypothèse assez générale pour produire des résultats, tout en restant satisfaisante pour l'intuition). Les premiers pas commencent tout juste à être esquissés par des équipes. Il faudra sans doute à l'avenir commencer par produire des résultats négatifs pour des problèmes spécifiques. Il est probable que ce travail passe également par une caractérisation précise de réductions entre problèmes qui conservent les ratios d'approximation, tout en limitant l'explosion de la taille des instances : seules de telles réductions permettent de transposer des résultats (positifs ou négatifs) d'un problème à l'autre.

La tâche ne manquera donc pas pour les futurs chercheurs en approximation exponentielle !

Bibliographie

[1] Scott Aaronson. The complexity zoo. `http://cse.unl.edu/~cbourke/latex/ComplexityZoo.pdf`.

[2] A. A. Ageev and M. Sviridenko. Approximation algorithms for maximum coverage and max cut with given sizes of parts. In G. Cornuéjols, R. E. Burkard, and G. J. Woeginger, editors, *Proc. Conference on Integer Programming and Combinatorial Optimization, IPCO'99*, volume 1610 of *Lecture Notes in Computer Science*, pages 17–30. Springer-Verlag, 1999.

[3] Eric Angel, Romain Campigotto, and Christian Laforest. Analysis and comparison of three algorithms for the vertex cover problem on large graphs with low memory capacities. *Algorithmic Operations Research*, 6(1), 2011.

[4] T. Asano and D. P. Williamson. Improved approximation algorithms for MAX SAT. *J. Algorithms*, 42(1) :173–202, 2002.

[5] Takao Asano. Approximation algorithms for max sat : Yannakakis vs. goemans-williamson. In *Proceedings of the Fifth Israel Symposium on the Theory of Computing Systems (ISTCS '97)*, ISTCS '97, pages 24–, Washington, DC, USA, 1997. IEEE Computer Society.

[6] Takao Asano, Takao Ono, and Tomio Hirata. Approximation algorithms for the maximum satisfiability problem. *Nordic J. of Computing*, 3(4) :388–404, December 1996.

[7] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation. Combinatorial optimization problems and their approximability properties*. Springer-Verlag, Berlin, 1999.

[8] G. Ausiello and V. Th. Paschos. Reductions that preserve approximability. In T. F. Gonzalez, editor, *Handbook of approximation algorithms and metaheuristics*, chapter 15, pages 15–1–15–16. Chapman & Hall, Boca Raton, 2007.

[9] Francisco Barahona, Martin Grötschel, Michael Jünger, and Gerhard Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36(3) :493–513, May/June 1988.

[10] R. Battiti and M. Protasi. Algorithms and heuristics for max-sat. In D. Z. Du and P. M. Pardalos, editors, *Handbook of combinatorial optimization*, volume 1, pages 77–148. Kluwer Academic Publishers, 1998.

[11] Ann Becker and Dan Geiger. Optimization of pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem. *Artificial Intelligence*, 83(1) :167 – 188, 1996.

[12] A. Björklund. Determinant sums for undirected Hamiltonicity. In *Proc. FOCS'10*, pages 173–182. IEEE Computer Society, 2010.

[13] A. Björklund, T. Husfeldt, and M. Koivisto. Set partitioning via inclusion-exclusion. *SIAM J. Comput.*, 39(2) :546–563, 2009.

[14] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8) :423 – 434, 2009.

[15] N. Bourgeois, B. Escoffier, and V. 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`.

[16] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of combinatorial problems by moderately exponential algorithms. In F. Dehne, M. Gavrilova, J.-R. Sack, and C. D. Tóth, editors, *Proc. Algorithms and Data Structures Symposium, WADS'09*, volume 5664 of *Lecture Notes in Computer Science*, pages 507–518. Springer-Verlag, 2009.

[17] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of MIN COLORING by moderately exponential algorithms. *Inform. Process. Lett.*, 109(16) :950–954, 2009.

[18] N. Bourgeois, B. Escoffier, and V. Th. Paschos. Efficient approximation of MIN SET COVER by moderately exponential algorithms. *Theoret. Comput. Sci.*, 410(21-23) :2184–2195, 2009.

[19] N. Bourgeois, B. Escoffier, and V. 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.

[20] N. Bourgeois, B. Escoffier, V. Th. Paschos, and J. M. M. Rooij. A bottom-up method and fast algorithms for max independent set. In Haim Kaplan, editor, *Algorithm Theory - SWAT 2010*, volume 6139 of *Lecture Notes in Computer Science*, pages 62–73. Springer Berlin Heidelberg, 2010.

[21] N. Bourgeois, A. Giannakos, G. Lucarelli, I. Milis, and V. Th. Paschos. Exact and approximation algorithms for DENSEST k-SUBGRAPH. Cahier du LAMSADE 324, LAMSADE, Université Paris-Dauphine, 2012.

[22] L. Brankovic and H. Fernau. Combining two worlds : parameterized approximation for vertex cover. In O. Cheong and K.-Y. Chwa ans K. Park, editors, *Proc. International Symposium on Algorithms and Computation, ISAAC'10*, volume 6506 of *Lecture Notes in Computer Science*, pages 390–402. Spinger-Verlag, 2010.

[23] L. Cai. Parameter complexity of cardinality constrained optimization problems. *The Computer Journal*, 51 :102–121, 2008.

[24] L. Cai and X. Huang. Fixed-parameter approximation : conceptual framework and approximability results. 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 96–108. Springer-Verlag, 2006.

[25] Leizhen Cai, Siu Man Chan, and Siu On Chan. Random separation : a new method for solving fixed-cardinality optimization problems. In *Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006*, pages 239–250. Springer Verlag, 2006.

[26] K.C. Chang and D.H.-C. Du. Efficient algorithms for layer assignment problem. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 6(1) :67 – 78, january 1987.

[27] J. Chen and I. A. Kanj. Improved exact algorithms for MAX SAT. *Discrete Appl. Math.*, 142 :17–27, 2004.

[28] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theoret. Comput. Sci.*, 411(40-42) :3736–3756, 2010.

[29] Jianer Chen, Iyad A. Kanj, and Ge Xia. Labeled search trees and amortized analysis : Improved upper bounds for np-hard problems. *Algorithmica*, 43 :245–273, 2005.

[30] Alan Cobham. The Intrinsic Computational Difficulty of Functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science, proceedings of the second International Congress, held in Jerusalem, 1964*, Amsterdam, 1965. North-Holland.

[31] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[32] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85 :12–75, 1990.

[33] P. Crescenzi, R. Silvestri, and L. Trevisan. To weight or not to weight : where is the question ? In *Proc. Israeli Symposium on Theory of Computing and Systems, ISTCS'96*, pages 68–77. IEEE, 1996.

[34] Federico Della Croce and Vangelis Th. Paschos. Efficient algorithms for the max k -vertex cover problem. In Jos C. M. Baeten, Thomas Ball, and Frank S. de Boer, editors, *IFIP TCS*, volume 7604 of *Lecture Notes in Computer Science*, pages 295–309. Springer, 2012.

[35] M. Cygan, L. Kowalik, and M. Wykurz. Exponential-time approximation of weighted set cover. *Inform. Process. Lett.*, 109(16) :957–961, 2009.

[36] M. Cygan and M. Pilipczuk. Exact and approximate bandwidth. *Theoret. Comput. Sci.*, 411(40–42) :3701–3713, 2010.

[37] E. Dantsin, M. Gavrilovich, E. A. Hirsch, and B. Konev. MAX SAT approximation beyond the limits of polynomial-time approximation. *Ann. Pure and Appl. Logic*, 113 :81–94, 2002.

[38] Evgeny Dantsin and Alexander Wolpert. An improved upper bound for sat. In *Proceedings of the 8th international conference on Theory and Applications of Satisfiability Testing*, SAT'05, pages 400–407, Berlin, Heidelberg, 2005. Springer-Verlag.

[39] George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. John Wiley, New York, 1951.

[40] F. Della Croce, M. J. Kaminski, and V. Th. Paschos. An exact algorithm for MAX CUT in sparse graphs. *Oper. Res. Lett.*, 35 :403–408, 2007.

[41] I. Dinur and M. Safra. The importance of being biased. In *Proc. STOC'02*, pages 33–42, 2002.

[42] Irit Dinur and Samuel Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1) :pp. 439–485, 2005.

[43] R. G. Downey, V. Estivill-Castro, M. R. Fellows, E. Prieto, and F. 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, 2003.

[44] R. G. Downey and M. R. Fellows. *Parameterized complexity*. Monographs in Computer Science. Springer, New York, 1999.

[45] R. G. Downey, M. R. Fellows, and C. 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.

[46] Jack Edmonds. Paths, trees, and flowers. *Canad. J. Math.*, 17 :449–467, 1965.

[47] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2) :248–264, April 1972.

[48] B. Escoffier, E. J. Kim, and V. Th. Paschos. Subexponential and fpt-time inapproximability of independent set and related problems. Cahier du LAMSADE 321, LAMSADE, Université Paris-Dauphine, May 2012.

[49] B. Escoffier and V. Th. Paschos. A survey on the structure of approximation classes. *Computer Science Review*, 4(1) :19–40, 2010.

[50] B. Escoffier, V. Th. Paschos, and É. Tourniaire. Approximating max sat by moderately exponential and parameterized algorithms. In M. Agrawal, S. Barry Cooper, and A. Li, editors, *Proc. Theory and Applications of Models of Computation, TAMC'12*, volume 7287 of *Lecture Notes in Computer Science*, pages 202–213. Springer-Verlag, 2012.

[51] U. Feige, R. Krauthgamer, and K. Nissim. On cutting a few vertices from a graph. *Discrete Appl. Math.*, 127(3) :643–649, 2003.

[52] U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *J. Algorithms*, 41(2) :174–211, 2001.

[53] M. R. Fellows, A. Kulik, F. A. Rosamond, and H. Shachnai. Parameterized approximation via fidelity preserving transformations. In A. Czumaj, K. Mehlhorn, A. Pitts, and R. Wattenhofer, editors, *Proc. ICALP'12*, volume 7391 of *Lecture Notes in Computer Science*, pages 351–362. Springer-Verlag, 2012.

[54] F. V. Fomin, F. Grandoni, and D. Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. Assoc. Comput. Mach.*, 56(5) :1–32, 2009.

[55] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and conquer : a simple o(20.288n) independent set algorithm. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 18–25, New York, NY, USA, 2006. ACM.

[56] Fedor V. Fomin and Yngve Villanger. Finding induced subgraphs via minimal triangulations. In Jean-Yves Marion and Thomas Schwentick, editors, *27th International Symposium on Theoretical Aspects of Computer Science (STACS 2010)*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 383–394, Dagstuhl, Germany, 2010. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[57] FedorV. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and conquer : Domination – a case study. In Luís Caires, GiuseppeF. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata,*

Languages and Programming, volume 3580 of *Lecture Notes in Computer Science*, pages 191–203. Springer Berlin Heidelberg, 2005.

[58] F.V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2011.

[59] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8(0) :399–404, January 1956.

[60] M. Fürer, S. Gaspers, and S. P. Kasiviswanathan. An exponential time 2-approximation algorithm for bandwidth. In *Proc. International Workshop on Parameterized and Exact Computation, IWPEC'09*, volume 5917 of *Lecture Notes in Computer Science*, pages 173–184. Springer, 2009.

[61] M. R. Garey and D. S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness*. W. H. Freeman, San Francisco, 1979.

[62] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical Computer Science*, 1(3) :237 – 267, 1976.

[63] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6) :1115–1145, November 1995.

[64] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4) :798–859, July 2001.

[65] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity ? *J. Comput. System Sci.*, 63(4) :512–530, 2001.

[66] G. Jäger and A. Srivastav. Improved approximation algorithms for maximum graph partitioning problems. In *Proc. Foundations of Software Technology and Theoretical Computer Science, FST&TCS'04*, volume 3328 of *Lecture Notes in Computer Science*, pages 348–359. Springer-Verlag, 2004.

[67] K. Jansen, M. Karpinski, A. Lingas, and E. Seidel. Polynomial time approximation schemes for max-bisection on planar and geometric graphs. *SIAM Journal on Computing*, 35(1) :110–119, 2005.

[68] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3) :256–278, 1974.

[69] R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[70] S. Khot, G. Kindler, E. Mossel, and R. O Donnell. Optimal inapproximability results for maxcut and other 2-variable csp. *SIAM Journal on Computing*, 37(1) :319–357, 2007.

[71] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within. *Journal of Computer and System Sciences*, 74(3) :335 – 349, 2008. <ce :title>Computational Complexity 2003</ce :title>.

[72] T. Kloks. *Treewidth, computations and approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[73] R. Kohli and R. Krishnamurti. Average performance of heuristics for satisfiability. *SIAM J. Discret. Math.*, 2(4) :508–523, November 1989.

[74] VI Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10 :707, 1966.

[75] H. R. Lewis and C. H. Papadimitriou. *Elements of the theory of computation*. Prentice-Hall, 1981.

[76] K. J. Lieberherr and E. Specker. Complexity of partial satisfaction. *J. ACM*, 28(2) :411–421, April 1981.

[77] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5) :960–981, September 1994.

[78] Kazuhisa Makino, Suguru Tamaki, and Masaki Yamamoto. Derandomizing hssw algorithm for 3-sat. In *Proceedings of the 17th annual international conference on Computing and combinatorics*, COCOON'11, pages 1–12, Berlin, Heidelberg, 2011. Springer-Verlag.

[79] S. Maneth. Logic and automata. Lecture 3 : Expressiveness of MSO graph properties. Logic Summer School, December 2006.

[80] D. Marx. Parameterized complexity and approximation algorithms. *The Computer Journal*, 51(1) :60–78, 2008.

[81] Dana Moshkovitz and Ran Raz. Two-query pcp with subconstant error. *J. ACM*, 57(5), 2010.

[82] M. Naor, L.J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 182–191, 1995.

[83] Robert O'Callahan and Jong-Deok Choi. Hybrid dynamic data race detection. *SIGPLAN Not.*, 38(10) :167–178, June 2003.

[84] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. In *Proc. STOC'88*, pages 229–234, 1988.

[85] Panos M. Pardalos and Gregory P. Rodgers. A branch and bound algorithm for the maximum clique problem. *Computers & OR*, 19(5) :363–375, 1992.

[86] Igor Razgon. Computing minimum directed feedback vertex set in $o(1.9977^n)$. In Giuseppe F. Italiano, Eugenio Moggi, and Luigi Laura, editors, *ICTCS*, pages 70–81. World Scientific, 2007.

[87] J.M. Robson. Finding a maximum independent set in time $o(2^{n/4})$. `http://www.labri.fr/perso/robson/mis/techrep.html`.

[88] S. Szeider. Monadic second order logic on graphs with local cardinality constraints. In E. Ochmański and J. Tyszkiewicz, editors, *Proc. Mathematical Foundations of Computer Science, MFCS'08*, volume 5162 of *Lecture Notes in Computer Science*, pages 601–612. Springer-Verlag, 2008.

[89] A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1) :230–265, 1937.

[90] Johan M. M. van Rooij and Hans L. Bodlaender. Design by measure and conquer, a faster exact algorithm for dominating set. *CoRR*, abs/0802.2827, 2008.

[91] V. Vazirani. *Approximation algorithms*. Springer, Berlin, 2001.

[92] Mihalis Yannakakis. On the approximation of maximum satisfiability. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, SODA '92, pages 1–9, Philadelphia, PA, USA, 1992. Society for Industrial and Applied Mathematics.

[93] David Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of computing*, STOC '06, pages 681–690, New York, NY, USA, 2006. ACM.

Index

List of Illustrations

Figures

Tables

Liste des Algorithmes