



**HAL**  
open science

# Approximability, parameterized complexity and solving strategies of some multidimensional assignment problems

Guillaume Duvillié

► **To cite this version:**

Guillaume Duvillié. Approximability, parameterized complexity and solving strategies of some multidimensional assignment problems. Other [cs.OH]. Université Montpellier, 2016. English. NNT : 2016MONTT321 . tel-01816979

**HAL Id: tel-01816979**

**<https://theses.hal.science/tel-01816979>**

Submitted on 15 Jun 2018

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# THÈSE

Pour obtenir le grade de  
Docteur

Délivré par l'Université de Montpellier

Préparée au sein de l'école doctorale **I2S\***  
Et de l'unité de recherche **UMR 5506**

Spécialité: **Informatique**

Présentée par **Guillaume Duvillié**  
guillerme@duvillie.eu

Approximation, Complexité  
Paramétrée et Stratégies de  
Résolution de Problèmes  
d'Affectation Multidimensionnelle

Soutenue le 07/10/2016 devant le jury composé de

M. Rodolphe GIROUDEAU	MdC	Univ. Montpellier	Directeur de thèse
M. Marin BOUGERET	MdC	Univ. Montpellier	Encadrant
M. Dimitrios THILIKOS	DR	CNRS, Lirmm	Examinateur
M. Frédéric HAVET	DR	CNRS, I3S	Examinateur, Président
Mme. Cristina BAZGAN	Professeur	Univ. Paris Dauphine	Rapporteur
M. Christophe PICOULEAU	Professeur	CNAM	Rapporteur



# Contents

---

<b>I</b>	<b>Prolegomena</b>	<b>15</b>
<b>1</b>	<b>Graphs and Hypergraphs</b>	<b>17</b>
1.1	Graphs . . . . .	17
1.2	Hypergraphs . . . . .	19
<b>2</b>	<b>Computational Complexity</b>	<b>21</b>
2.1	Problems, Reductions and Complexity Classes . . . . .	22
2.1.1	Decision Problems . . . . .	22
2.1.2	The <b>NP</b> complexity class . . . . .	23
2.1.3	Optimization problems . . . . .	26
2.2	Approximation Algorithms . . . . .	28
2.3	Approximation Preserving Reductions . . . . .	32
2.4	Gap-reduction . . . . .	37
2.5	Other complexity hypotheses . . . . .	38
2.5.1	The Unique Game Conjecture . . . . .	38
2.5.2	<b>NP</b> $\not\subseteq$ <b>QP</b> . . . . .	39
2.5.3	Exponential Time Hypothesis . . . . .	40
<b>3</b>	<b>Parameterized Complexity</b>	<b>41</b>
3.1	Intuition and definitions . . . . .	41
3.2	Kernelization . . . . .	48
<b>II</b>	<b>Theoretical aspects</b>	<b>53</b>
<b>4</b>	<b>Preliminaries</b>	<b>55</b>
4.1	Modelling . . . . .	55
4.2	Observations . . . . .	61
<b>5</b>	<b>Related Work</b>	<b>63</b>
5.1	MULTI-DIMENSIONAL ASSIGNMENT . . . . .	64
5.2	Maximizing the die yield . . . . .	66
5.3	Minimizing the faults . . . . .	69

<b>6</b>	<b>Intermediate problems</b>	<b>71</b>
6.1	Negative Results	72
6.1.1	$\mathcal{O}\left(\frac{m}{\log(m)}\right)$ -inapproximability for $\max_{\neq 0}$	73
6.1.2	$f(n)$ , $p^{1-\varepsilon}$ and $m^{1-\varepsilon}$ -inapproximability for $\max \max 1$	74
6.1.3	Inapproximability extension to $(\max \max 1)_{\#0 \leq 1}$	76
6.1.4	$(n - 1 - \varepsilon)$ -inapproximability for $(\min \min 0)_{\#0 \leq 1}$	78
6.1.5	$(1 - \varepsilon) \log(m)$ -inapproximability for $\min \min 0$	81
6.2	Positive Results	84
6.3	Conclusion and open questions	85
<b>7</b>	<b>Maximizing the overall number of ones</b>	<b>87</b>
7.1	Negative Results	88
7.1.1	$\mathcal{O}\left(\frac{m}{\log(m)}\right)$ -inapproximability	88
7.1.2	$f(n)$ , $p^{1-\varepsilon}$ , $m^{1-\varepsilon}$ -inapproximability	90
7.2	Considering the vectors dimension	91
7.2.1	$p/r$ -approximation	92
7.2.2	Faster algorithm for fixed $p$ for $\max \sum 1$	94
7.2.3	No polynomial size kernel with parameter $p$	99
7.3	Resolution of sparse instances	101
7.3.1	<b>EPTAS</b> for $(\max \sum 1)_{\#0 \leq r}$	101
7.3.2	A polynomial-time algorithm for $(\max \sum 1)_{\#0 \leq 1}$ when $m$ is fixed	103
7.4	Conclusion and open questions	107
<b>8</b>	<b>Minimizing the overall number of zeros</b>	<b>109</b>
8.1	Approximability of $\min \sum 0$	110
8.1.1	$(n - \varepsilon)$ -inapproximability	110
8.1.2	Inapproximability without UGC	113
8.1.3	Approximation algorithm for $\min \sum 0$	116
8.2	Parameterized complexity of $\min \sum 0$	118
8.2.1	$\mathcal{O}(k^2 m)$ kernel for $(\min \sum 0, k)$	118
8.2.2	Positive results according to $\zeta_\beta$	119
8.2.3	Negative results according to $\zeta_\beta$	120
8.2.4	Positive results according to $\zeta_p$	121
8.2.5	Negative results according to $\zeta_p$	126
8.3	Conclusion and open questions	127
<b>III</b>	<b>Practical Aspects</b>	<b>129</b>
<b>9</b>	<b>Preliminaries</b>	<b>131</b>
9.1	Related work	132
9.2	Instance generation	133
9.2.1	Considered model	133
9.2.2	Model restrictions	134

---

9.2.3	Alternative model . . . . .	134
9.3	Experiment characteristics . . . . .	134
<b>10</b>	<b>ILP formulations</b>	<b>137</b>
10.1	Model description . . . . .	138
10.1.1	MDA inspired formulation . . . . .	138
10.1.2	Hub set formulation . . . . .	142
10.2	ILP comparison . . . . .	143
10.2.1	LP relaxations comparison . . . . .	143
10.2.2	Computational Results . . . . .	148
10.3	ILP based heuristics . . . . .	159
10.3.1	Principle . . . . .	159
10.3.2	Computational Results . . . . .	160
10.4	Sets aggregation technique . . . . .	166
10.4.1	Modus Operandi . . . . .	166
10.4.2	Computational results . . . . .	166
10.5	Conclusion . . . . .	167
<b>11</b>	<b>Matching based heuristics</b>	<b>169</b>
11.1	Set selection strategies . . . . .	170
11.1.1	Iterative Matchings . . . . .	170
11.1.2	Lot Partitioning . . . . .	170
11.1.3	Balanced Binary Tree Strategy . . . . .	172
11.2	Computational results . . . . .	173
11.3	Conclusion . . . . .	175
<b>12</b>	<b>Perspective and conclusion</b>	<b>177</b>
<b>A</b>	<b>Computational results</b>	<b>185</b>
A.1	ILP comparison . . . . .	185



# Remerciements

---

S'il est un exercice qui revêt à mes yeux une importance singulière au cours de la rédaction d'un manuscrit de thèse, c'est bien celui des remerciements. Ces quelques paragraphes sont, en effet, l'occasion de réparer, après trois ans de labeur, une certaine forme d'ingratitude propre au doctorant. Comme absorbé par une tâche dont l'ampleur lui échappe jusqu'à son accomplissement, il en oublie souvent l'abnégation dont peuvent faire preuve les personnes de son entourage. Ainsi, il me semble naturel d'envisager ce manuscrit, non comme une œuvre personnelle, mais comme la résultante des sacrifices et des efforts de l'ensemble des personnes qui m'ont côtoyé, de leur plein gré ou non, durant ces trois années.

Dans un premier temps, je tiens à remercier mes encadrants, Rodolphe Giroudeau et Marin Bougeret. Merci Rodolphe pour avoir cru en moi et ce, suffisamment pour me proposer un sujet. Merci d'avoir vu en moi autre chose qu'un étudiant désinvolte et d'avoir su réveiller une curiosité endormie depuis bien longtemps. Mais surtout un grand merci pour avoir su trouver l'encadrement idéal m'ayant permis de m'épanouir, pour toute cette énergie dépensée, pour toutes ces heures passées à relire parfois dix fois les mêmes écrits, pour avoir su trouver l'équilibre idéal entre autonomie et autorité, ou encore pour le soutien humain dont j'ai eu besoin lorsqu'il m'a fallu concilier ma vie de jeune papa et celle de doctorant.

Merci Marin pour nos échanges spontanés et (presque) toujours productifs et ce à n'importe quelle heure du jour ou de la nuit. Merci de ne jamais m'avoir laissé tomber dans la facilité, pour toutes nos sessions techniques, à la fin desquelles j'avais l'impression d'avoir couru un marathon et auxquelles un grand nombre de résultats de ce manuscrit doivent leur existence, pour cette envie de comprendre qui t'est si particulière et qui m'a obligé à éclaircir chaque zone d'ombre des théories sur lesquelles nous avons travaillé. Merci également d'avoir su me forcer à tout recommencer lorsque c'était nécessaire, ainsi que d'avoir été présent dans mes moments de doute et de lassitude et enfin pour cette envie de partager ce qui t'est cher (y compris les libertés parfois outrancières que prennent les pianistes romantiques avec la pulsation).

Ce fut un véritable plaisir d'avoir évolué à vos côtés tout au long de ces années et j'espère que vous verrez en ces lignes l'aveu d'une mélancolie sincère à l'idée de ne plus travailler ensemble.

Je souhaiterais remercier Cristina Bazgan et Christophe Picouveau pour avoir accepté de prendre de leur temps pour lire et corriger ce manuscrit ainsi que pour leurs encouragements et les échanges que nous avons pu avoir en conférence notamment. Un grand merci également à Dimitrios Thilikos et Frédéric Havet pour avoir accepté de siéger dans mon jury.

Je tiens également à remercier l'ensemble des membres de l'équipe MAORE pour leur accueil et leur sollicitude pendant mon séjour parmi eux. Merci d'avoir supporté



les innombrables présentations de mon sujet, et d'avoir eu la gentillesse de ne jamais pointer du doigt ni le recyclage immodéré de mes transparents, ni mes lubies passagères en terme de style beamer. Je remercie particulièrement Anne-Élisabeth qui m'a fait pleinement confiance en me laissant organiser certains cours et travaux dirigés de réseau, Éric pour m'avoir permis de m'adresser à un public expérimenté en me laissant intervenir au sein de son cours d'optimisation combinatoire et enfin Michaël avec qui je regrette de ne pas avoir eu l'occasion de travailler plus longuement, qui a su m'apporter un peu de la fraîcheur caractéristique de la Belgique et à qui je dois beaucoup aujourd'hui.

Merci à Nicolas Serrurier et Laurie Laverne sans qui les lourdeurs de l'administration auraient eu raison de ma motivation. Merci pour votre gentillesse, votre patience et pour ce don que vous avez de rendre facile et digeste l'ensemble des démarches administratives auxquelles un doctorant peut être confronté.

Je voudrais remercier l'ensemble des doctorants qu'il m'a été donné de côtoyer pendant ces quelques années, avec une mention spéciale pour Rémi qui a directement participé à mon épanouissement personnel en m'ouvrant les portes du conseil des doctorants, pour Sabrina pour sa gentillesse et sa candeur, pour Namrata pour son enthousiasme, pour Swan pour la simplicité de ses points de vue et la force de ses convictions, pour Chloé pour sa connaissance inestimable des méandres administratifs de l'université mais surtout pour sa persévérance et sa détermination à me tirer vers le haut en master et enfin pour Emmanuel, Julien et Adel pour ces innombrables fou-rire, ces longs débats politiques et/ou linguistiques sur l'importance de l'accent circonflexe dans la langue française ainsi que pour toute la bienveillance que vous avez eu à mon égard.

Je continuerai en remerciant Frits Spijksma qui m'a accueilli l'espace de quelques semaines au sein du laboratoire ORSTAT à Leuven, qui a accepté de venir assister à ma soutenance malgré la longue distance séparant Maastricht de Montpellier, mais surtout pour avoir soulevé un grand nombre de questions pertinentes sur le problème considéré qui m'ont tenu en haleine de longs mois durant.

Je finirai cette partie *remerciements recherche* en citant Guillaume Derombise, Jérémie Detrey, Mario Marchetti et Denis Trystram sans qui je n'aurais jamais pu développer autant le goût de la recherche et du partage de connaissances.

Il me tient à présent à cœur de remercier mes proches qui m'ont forcé à sortir de ma caverne pour ne pas oublier qu'il existe une vie en dehors du laboratoire et qui, par leur soutien indéfectible, ont été la base stable et solide m'ayant permis d'affronter sereinement les difficultés et la rudesse d'une telle aventure.

Un merci tout naturel à l'ensemble de mes frères et sœurs, mon oncle, ma tante et ma grand-mère, qui de par leur pluralité et par leurs encouragements, m'ont poussé à me surpasser face aux épreuves et m'ont rasséréiné en période de stress. Merci pour ces parenthèses régulières et toujours un peu déjantées qui m'ont servies de soupape de sécurité.

Un merci à ma belle famille pour tout ce qu'ils ont fait pour moi, pour leurs encouragements, pour m'avoir intégré à leur famille comme ils l'ont fait. Un grand merci de m'avoir ouvert les portes de votre havre de paix aussi souvent qu'il a été nécessaire,

merci d'avoir supporté mes longues journées d'ermite reclus sur mon ordinateur et ce malgré mes manquements évidents aux règles de l'hospitalité et du savoir-vivre et enfin un merci pour votre gentillesse et votre bienveillance à mon égard.

Un merci tout particulier à mes amis du Nord (Victor, Alexis, Jérémie H., Paul-Robert, Isabelle, Elsa et Nicolas) pour avoir été présent et avoir pris le temps de me rendre visite malgré des emplois du temps au moins aussi chargés que le mien. Je regrette juste de ne pas avoir pu vous rendre la pareille.

Je tiens également à remercier singulièrement Jérémie F. et Clément pour les innombrables services que vous avez pu me rendre et surtout pour avoir su donner à ces terres montpelliéraines, si hostiles à mon arrivée, une saveur douce et sincère qui me manque à présent.

Mes derniers remerciements, et non les moindres, sont destinés à Salomé qui m'a accompagné au quotidien dans cette aventure. C'est en effet elle qui a dû savoir gérer mes journées sans fin, mes nuits de travail, ma fatigue, mes sautes d'humeur et mon irritabilité. C'est elle qui a été l'oreille attentive et l'épaule réconfortante dont j'avais besoin après des journées particulièrement difficiles. Merci pour ta sollicitude, ton soutien inconditionnel et réconfortant, merci d'avoir tenu notre petite famille à bout de bras pendant la période de rédaction, d'avoir toléré ces longues journées où, obnubilé par un problème, j'étais présent physiquement mais absent mentalement, d'avoir su prendre soin de notre magnifique petite fille tout en prenant soin de moi et surtout d'avoir supporté pendant trois ans les appels de Marin au milieu de nos rares repas en famille. Sans toi rien de tout cela n'aurait été possible et j'espère avoir l'occasion, un jour, de te rendre la pareille. En attendant, sache que je suis on ne peut plus comblé d'explorer de nouveaux horizons à tes côtés.



# Introduction

---

The development of the integrated circuit (IC) manufacturing since its first prototype on 12 September 1958 built by Jack Kilby, led Gordon Moore to formulate, in 1965, a prediction on the future of semiconductor industry:

The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years.

In 1975, Moore revised its prediction during the IEEE International Electron Devices Meeting. He stated that from 1980, “the rate of increase complexity can be expected to change slope [...]. The new slope might approximate a doubling every two years, rather than every year[...].”

This conjecture has been later popularized as *the second Moore’s Law*, and by extension the 1965 conjecture popularized as *the first Moore’s Law*. This conjecture proved accurate for several decades. However, Moore and number of industry forecasters expect Moore’s Law will end on 2025.

Nowadays, the increase of the microprocessors power computation is partly ensured by an increase of the number of cores. This increase is such that some even predict it to follow Moore’s Law [Kum+05]. Nevertheless, the increase of the number of cores leads to an increase of distance between them and thus an increase of the communication times. This limits the efficiency of such solution. A lead to overcome such issues is to produce three-dimensional microprocessors.

Instead of enclosing several adjacent cores, a three-dimensional microprocessor aims at superimposing the latter. The distance between each die is then greatly reduced to roughly the thickness of the die. The manufacturing process remains quite identical as the one for *classical* microprocessors. In a first time, the dies are engraved on a circular slice of silicon called *wafer*. Depending on the complexity of the engraved die and on the size of the wafer, one can engraved up to several thousands of dies on a single slice. Once this step is performed, dies are tested to spot faulty ones and then cut out. Viable dies are then encapsulated into chips to get microprocessors as we usually know them.

An additional step is required to manufacture three-dimensional microprocessors: the integration. Dies need indeed to be superimposed. This can be achieved by various processes:

**die-to-die** the dies are superimposed among them once they have been cut out.

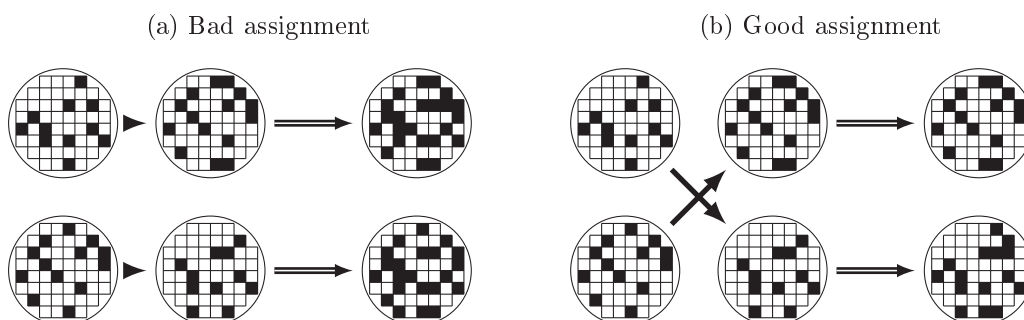


Figure 1: Illustration of the influence of the assignment on the final die yield.

**die-to-wafer** a part of engraved wafers are used as support on which cut out dies are superimposed.

**wafer-to-wafer** the wafers are superimposed before being cut out.

In a nutshell, *die-to-die* and *die-to-wafer* processes offer a good yield but greatly increase manufacturing cost. On the other hand, the *wafer-to-wafer* present several advantages (the integration can be performed at reasonable cost with more accuracy and efficiency, the thickness of the wafers can be reduced, ...) but can lead to very poor yield. Indeed, a stack of dies is considered to be viable if and only if all the dies composing the stack are viable ones. Figure 1 gives clues on the possible yield degradation when superimposing wafers.

This thesis focus on the way to handle such a combinatorics in these problems. The manuscript is split into three main parts. The first part is devoted to the introduction of the theoretical frameworks that have been used to approach the problem. In a first time we present a slight overview on graphs and hypergraphs and then we briefly introduce the ABC of computational complexity and fixed parameter complexity theory.

In the second part, we consider the theoretical aspects of the problem. To do so, the first chapter of this part is devoted to the modelization of the latter and to the introduction of several of its variants, differing mainly on the way to evaluate the quality of an assignment. A chapter is then devoted to what we will call *intermediate problems*, *i.e.* problems presenting mainly theoretical interest as they provide useful tools to handle complexity of the main problems. Last, we consecrate one chapter for each of the both main problems we consider. In each chapter, we try to render as precisely as possible the complexity of the considered variant.

The third part present results obtained from several campaigns of experimentation. The objective of this part is to lay the basis of the practical resolution of the problem by exploring different solving methods such as Integer Linear Programming (ILP) formulations, Constraints Programming (CP) formulations or ad hoc heuristics.

This work led to the following publications:

- 
- [BDG16] Marin Bougeret, Guillaume Duvillié, and Rodolphe Giroudeau. “Approximability and Exact Resolution of the Multidimensional Binary vector Assignment Problem”. In: *Proceedings of the 4th International Symposium on Combinatorial Optimization, ISCO 2016, Vietri-Sul-Mare, Italy, May 17-19, 2016*. 2016, to be published.
- [Bou+15] Marin Bougeret et al. “Multidimensional Binary Vector Assignment Problem: Standard, Structural and Above Guarantee Parameterizations”. In: *Proceedings of the 20th International Symposium on Fundamentals of Computation Theory, FCT 2015, Gdańsk, Poland, August 17-19, 2015*. 2015, pp. 189–201.
- [Bou+16] Marin Bougeret et al. “On the Complexity of Wafer-to-Wafer Integration”. In: *Discrete Optimization* (2016), to be published.
- [Duv+15] Guillaume Duvillié et al. “On the Complexity of Wafer-to-Wafer Integration”. In: *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*. 2015, pp. 208–220.



— PART I —  
**PROLEGOMENA**





# 1

---

## Graphs and Hypergraphs

---

---

### Contents

---

1.1	Graphs . . . . .	17
1.2	Hypergraphs . . . . .	19

---

This short chapter is devoted to the introduction of an important data structure, the graphs. The hypergraphs, a generalization of graphs, are also introduced as they represent a key notion in our work. Let us remark that this chapter only presents trivia on Graph Theory. Therefore, we invite well-versed reader in Graph Theory to start the reading of this manuscript at Chapter 2.

### 1.1 Graphs

**Definition 1.1** (Graph). *A graph  $G$  is a pair  $(U, E)$  where  $U$  is a non empty set of vertices and  $E$  is a set of edges, an edge being a pair of vertices of  $U$ .*

Intuitively, a graph is a convenient way of representing binary relationships among a set of entities. As such graphs are widely used in lots of areas. For instance, they can be used to represent networks:

**social network:** one can think of a person as a vertex. Two vertices are connected by an edge if the person they represent are in contact.

**physical network:** a vertex represents a machine in the network and an edge a physical link between two machines.

**railway network:** during the Cold War, US Army used graphs to represent the Soviet railway network in order to visualize weaknesses in the Soviet railway traffic flow.

...

Naturally, graphs are not restricted to the modelization of networks, they offer a powerful theoretical framework to modelize problems of all sorts.

One can define some interesting and useful notions on a graph. This section is thus devoted to the introduction of some of terms and notions that will be used throughout this manuscript. Obviously, it does not aim at being an exhaustive overview of the graph basics. We refer the reader to the works given by Reinhardt Diestel [Die12] and by Adrian Bondy and Uppaluri Murty [BM11].

In the following, we present the graph notions that will be used throughout this manuscript.

**Definition 1.2** (Complete graph). *A graph  $G = (U, E)$  is said complete if and only if  $\forall u, v \in U, u \neq v, \{u, v\} \in E$ .*

**Definition 1.3** (Induced subgraph [Die12]). *Given a graph  $G = (U, E)$  and a subset of vertices  $S \subseteq U$ , the subgraph  $G[S]$  induced by  $S$  is the graph whose vertex set is  $S$  and whose edge set consists of all the edges in  $E$  that have both endpoints in  $S$ .*

*More formally  $G[S] = (S, \{\{u, v\} \in E : u, v \in S\})$ .*

We can use these definition to define a *clique* in a graph.

**Definition 1.4** (Clique). *Given a graph  $G = (U, E)$ , a clique of  $G$  is a subset of vertices  $C \subseteq U$  such that  $G[C]$  is complete.*

Another interesting and important notion in a graph is the notion of *independent set*.

**Definition 1.5** (Independent Set). *Given a graph  $G = (U, E)$ , an independent set of  $G$  is a subset of vertices  $IS \subseteq U$  such that  $G[IS] = (IS, E')$  satisfies  $E' = \emptyset$ .*

*In other words, there does not exist  $u, v \in IS$  such that  $\{u, v\} \in E$ .*

Such a subset of vertices is also called a *stable set*. A strongly related notion is the *vertex cover*.

**Definition 1.6** (Vertex Cover). *Given a graph  $G = (U, E)$ , a vertex cover of  $G$  is a subset of vertices  $VC \subseteq U$  such that for all  $\{u, v\} \in E$ ,  $u \in VC$  or  $v \in VC$ .*

*In other words,  $VC$  contains at least one endpoint of each edge.*

**Property 1.1.** *Given a graph  $G = (U, E)$  and  $VC$  a vertex cover of  $G$ , the set  $IS = U \setminus VC$  is an independent set.*

**Definition 1.7** (Graph coloring). *A  $k$ -vertex coloring (a  $k$ -coloring for short) of a graph  $G = (U, E)$  is map  $c : U \rightarrow [k]$ .*

*A  $k$ -coloring  $c$  is said to be proper if and only if  $\forall \{u, v\} \in E, c(u) \neq c(v)$ .*

## 1.2 Hypergraphs

**Definition 1.8** (Hypergraph). *A hypergraph  $H$  is a couple  $(U, E)$  where  $U$  is a non empty set of vertices and  $E$  a set of hyperedges, an hyperedge being a non empty subset of  $U$ .*

We also define several notions on these hypergraphs.

**Definition 1.9** ( $k$ -uniform). *Given an integer  $k$ , a hypergraph  $H = (U, E)$  is said to be  $k$ -uniform if and only if  $E \subseteq \binom{U}{k}$ ,  $\binom{U}{k}$  denoting the subsets of  $U$  of size  $k$ .*

In other words, a hypergraph is said  $k$ -uniform if every hyperedge contains exactly  $k$  vertices.

**Definition 1.10** ( $k$ -partite). *Given an integer  $k$ , a hypergraph  $H = (U, E)$  is said to be  $k$ -partite if its vertex set can be partitioned into  $k$  disjoint subsets  $U_1, \dots, U_k$  such that for all  $i \in [k]$  and for all couple of vertices  $u, v \in U_i$ , no hyperedge  $e \in E$  containing both  $u$  and  $v$  exists.*

*In other words, two vertices of a same subset cannot be elements of a same hyperedge.*

We can also naturally extend some graph notions to the hypergraphs.

**Definition 1.11** (Induced subhypergraph). *Given a hypergraph  $H = (U, E)$  and a subset of vertices  $S \subseteq U$ , the subhypergraph  $H[S]$  induced by  $S$  is the hypergraph whose vertex set is  $S$  and whose hyperedge set consists of all the hyperedges in  $E$  that have all endpoints in  $S$ .*

*More formally  $H[S] = (S, \{e \in E : e \cap S = e\})$ .*

**Definition 1.12** (Independent Set). *Given a hypergraph  $H = (U, E)$ , an independent set of  $H$  is a subset of vertices  $IS \subseteq U$  such that  $H[IS] = (IS, E')$  satisfies  $E' = \emptyset$ .*

*In other words, there does not exist  $e \subset IS$  such that  $e \in E$ .*

**Definition 1.13** (Vertex Cover). *Given a hypergraph  $H = (U, E)$ , a vertex cover of  $H$  is a subset of vertices  $VC \subseteq U$  such that  $\forall e \in E, e \cap VC \neq \emptyset$ .*

*In other words,  $VC$  contains at least one element of each hyperedge.*

As for graphs, the following property holds for hypergraphs.

**Property 1.2.** *Given a hypergraph  $H = (U, E)$  and  $VC$  a vertex cover of  $H$ , the set  $IS = U \setminus VC$  is an independent set.*

**Definition 1.14** (Hypergraph proper coloring). *A  $k$ -coloring of the vertices of a hypergraph  $H = (U, E)$  is map  $c : U \rightarrow [k]$ .*

*A  $k$ -coloring is said to be proper if and only if  $\forall e \in E, \exists u, v \in e$  such that  $c(u) \neq c(v)$ .*

**Definition 1.15** (Hypergraph rainbow coloring). *A  $k$ -coloring is said to be a rainbow coloring if and only if  $\forall e \in E, \forall u, v \in e, c(u) \neq c(v)$ .*



# 2

## Computational Complexity

### Contents

<b>2.1</b>	<b>Problems, Reductions and Complexity Classes . . . . .</b>	<b>22</b>
2.1.1	Decision Problems . . . . .	22
2.1.2	The <b>NP</b> complexity class . . . . .	23
2.1.3	Optimization problems . . . . .	26
<b>2.2</b>	<b>Approximation Algorithms . . . . .</b>	<b>28</b>
<b>2.3</b>	<b>Approximation Preserving Reductions . . . . .</b>	<b>32</b>
<b>2.4</b>	<b>Gap-reduction . . . . .</b>	<b>37</b>
<b>2.5</b>	<b>Other complexity hypotheses . . . . .</b>	<b>38</b>
2.5.1	The Unique Game Conjecture . . . . .	38
2.5.2	<b>NP</b> $\not\subseteq$ <b>QP</b> . . . . .	39
2.5.3	Exponential Time Hypothesis . . . . .	40

In this chapter, we introduce the rudiments in complexity theory required for a convenient reading of this manuscript. The definition and results introduced here are considered to be standard and therefore are succinctly presented. However, we aim at providing a bunch of references in which these results are more deeply studied. Furthermore, this section allows us to introduce notations that will be used throughout this manuscript.

In a first time we present the notion of *decision problem*, which is an important notion in complexity theory. We then show how algorithms can be used to highlight relationships between problems, by defining *reductions*. The latter lay the foundations for a classification of decision problems in terms of complexity. We introduce then two *complexity classes* **P** and **NP**.

In a second time, we take *optimization problems* into consideration. The complexity class **NPO** is then defined. The issue of optimization problem solving is then discussed to introduce a new family of algorithms, called *approximation algorithms*. We show how these algorithms are used to design new complexity classes such as **APX**, **PTAS** or **FPTAS**. And finally we present how polynomial reductions can be tuned to design *approximation preserving reductions*, highlighting at the same time, some relationships between optimization problems, in terms of approximability.

## 2.1 Problems, Reductions and Complexity Classes

### 2.1.1 Decision Problems

The *decision problems* represent the most natural problems one can be exposed to. It simply consists in a yes-or-no question. Given a problem  $\Pi$  and an input  $I \in \{0, 1\}^*$ , the objective is to determine whether  $I$  is a positive instance (the answer to the question is *yes*) or a negative instance (the answer to the question is *no*). As such, a decision problem can be simply taken for its set of positive instance. This notion is formally defined in Definition 2.1.

**Definition 2.1** (Decision Problem [Pap94]). *A decision problem  $\Pi$  is a subset  $D_{\Pi}^+ \subseteq \{0, 1\}^*$  of positive instances. Thus the set of negative instances is defined as  $D_{\Pi}^- = \{0, 1\}^* \setminus D_{\Pi}^+$ .*

If we consider the following problem:

---

#### Decision Problem 1 HAMILTONIAN CYCLE

---

<b>Input</b>	A graph $G = (U, E)$ .
<b>Question</b>	Does $G$ admit an HAMILTONIAN CYCLE, <i>i.e.</i> a cycle that includes every vertex of $U$ exactly once?

---

HAMILTONIAN CYCLE is a classical example of decision problem and examples of positive and negative instances are depicted in Figure 2.1

Given an algorithm  $\mathcal{A}$ , we can define several metrics in order to determine its *complexity*. Two main kinds of metrics are widely used, the *time complexity* that aims at giving an idea on how much time will be needed by  $\mathcal{A}$  to reach its end conditions, and the *space complexity* giving some clues on the amount of memory required by  $\mathcal{A}$ . These metrics are often expressed in function of the *size of the instance*.

**Notation 2.1.** *Given  $\Pi$  a problem and a  $\Pi$ -instance  $I \in \{0, 1\}^*$ ,  $|I|$  denotes the size of the instance. It represents the length of the word on the alphabet  $\{0, 1\}$  that encodes the instance.*



Figure 2.1: On the left a graph admitting an Hamiltonian Cycle (positive instance), on the right a graph without Hamiltonian Cycle (negative instance).

Throughout this manuscript, we focus on the time complexity and more precisely on the *worst case time complexity*. Intuitively, this complexity gives an order of magnitude of the maximum number of atomic operations that an algorithm performs before reaching its end conditions.

**Definition 2.2.** *Given a decision problem  $\Pi$  and an algorithm  $\mathcal{A}$ , we say that  $\mathcal{A}$  decides or solves  $\Pi$  in time  $f(n)$  if given any instance  $I \in \{0,1\}^*$ ,  $\mathcal{A}$  determines whether  $I \in D_{\Pi}^+$  or not in at most  $f(n)$  operations, with  $n = |I|$ .*

This order of magnitude is commonly expressed in function of the size of the algorithm input and using the Big-O-Notation.

### 2.1.2 The NP complexity class

**Definition 2.3 (P).** *The class  $\mathbf{P}$  is the set of all decision problems that can be solved with a deterministic algorithm running in polynomial time.*

In the light of the above,  $\mathbf{P}$  contains the *easy* problems. However,  $\mathbf{P}$  is a restrictive class since, for a great number of natural problems no efficient algorithm is known. For instance, HAMILTONIAN CYCLE previously defined is not known to be in  $\mathbf{P}$ . Nevertheless, given  $G = (U, E)$ , an instance of HAMILTONIAN CYCLE and a set of edges  $T \subseteq E$ , determining whether  $T$  defines an Hamiltonian cycle can be done in polynomial time. This motivates the introduction of the class  $\mathbf{NP}$ .

**Definition 2.4 (NP [Wil02]).** *The class  $\mathbf{NP}$  is defined as the set of decision problems  $\Pi$  that admit a polynomial verifier  $\mathcal{A}$ , i.e. an algorithm with the following properties:*

1.  $\mathcal{A}$  runs in polynomial time,
2. given a positive instance  $I \in \{0,1\}^*$  of  $\Pi$ , there exists  $y \in \{0,1\}^*$  such that  $|y| = \mathcal{O}(\text{poly}(|I|))$  and  $\mathcal{A}$  with input  $(I, y)$  returns true,
3. given a negative instance  $I \in \{0,1\}^*$  of  $\Pi$ , for every  $y \in \{0,1\}^*$  such that  $|y| = \mathcal{O}(\text{poly}(|I|))$ ,  $\mathcal{A}$  with input  $(I, y)$  returns false.

Considering a positive instance  $I$  of  $\Pi$ ,  $y$  is a polynomial certificate.



In other words, given a positive instance  $I$  of decision problem  $\Pi \in \mathbf{NP}$ , a polynomial certificate is a solution of this problem ensuring that  $I$  is a positive instance. On the Figure 2.1, a polynomial certificate can be depicted by the order of the vertices in the cycle *i.e.* 1, 3, 5, 2, 4. Note that this certificate is not necessarily unique, indeed, 1, 2, 4, 5, 3 also ensures the existence of a solution and is hence a polynomial certificate.

**Remark 2.1.** *Every problem belonging to  $\mathbf{P}$  also belongs to  $\mathbf{NP}$  and thus  $\mathbf{P} \subseteq \mathbf{NP}$ .*

Note that the strictness of the inclusion is still an open problem, but the correctness of the conjecture stating that  $\mathbf{P} \neq \mathbf{NP}$  (and thus that  $\mathbf{P} \subset \mathbf{NP}$ ) is widely accepted.

Considering the previous hypothesis as true, natural questions arise: How can we identify problems belonging to  $\mathbf{NP}$  but for which a polynomial-time algorithm is not known? Can we characterize them? Is there relationships between problems in  $\mathbf{P}$ , problems in  $\mathbf{NP}$ ?

A first part of the answer has been provided by Karp when he specified what will be called the Karp reduction [Kar72].

**Definition 2.5** (Karp reduction [Kar72]). *Given two decision problems  $\Pi_1$  and  $\Pi_2$ , a Karp reduction is a function  $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$  such that:*

- *$f$  is computable in polynomial time,*
- *for every  $I \in \{0, 1\}^*$ ,  $I \in D_{\Pi_1}^+$  if and only if  $f(I) \in D_{\Pi_2}^+$ .*

*We note then  $\Pi_1 \leq_K \Pi_2$ .*

This reduction is used in the following theorem.

**Theorem 2.1** (Karp reducibility [Kar72]). *Given two decision problems  $\Pi_1$  and  $\Pi_2$ , if  $\Pi_2 \in \mathbf{P}$  and  $\Pi_1 \leq_K \Pi_2$ , thus  $\Pi_1 \in \mathbf{P}$ . We say that Karp reduction preserves membership in  $\mathbf{P}$ .*

The intuition behind this theorem is quite simple: given two problems  $\Pi_1$  and  $\Pi_2$ , the existence of both an efficient algorithm solving  $\Pi_2$  and another efficient algorithm that “transforms  $\Pi_1$  into  $\Pi_2$ ”, implies the existence of an efficient algorithm for  $\Pi_1$ . Indeed, given  $I$  a  $\Pi_1$ -instance, the algorithm that executes the function  $f$  on  $I$  and solves the  $\Pi_2$ -instance  $f(I)$ , solves  $I$  in polynomial time.

**Corollary 2.1.** *Given two decision problems  $\Pi_1$  and  $\Pi_2$  such that  $\Pi_2 \in \mathbf{P}$  and  $\Pi_1 \leq_K \Pi_2$ , thus  $\Pi_1 \in \mathbf{P}$ .*

In scientific literature, a generalization of the Karp reduction is often used. It is called the Turing reduction.

**Definition 2.6** (Turing reduction [Pas04]). *Given problems  $\Pi_1$  and  $\Pi_2$ ,  $\Pi_1$  Turing-reduces to  $\Pi_2$ , noted  $\Pi_1 \leq_T \Pi_2$  if and only if there exists an algorithm  $\mathcal{A}_1$  solving  $\Pi_1$  by calling, a polynomial number of times, an algorithm  $\mathcal{A}_2$  for  $\Pi_2$ , such that if  $\mathcal{A}_2$  runs in polynomial time, thus  $\mathcal{A}_1$  runs in polynomial time.*

The Turing reduction can be understood as follows. Given two decision problems  $\Pi_1$  and  $\Pi_2$ , a Turing reduction is an algorithm that can solve efficiently any  $\Pi_1$ -instance if there exists an efficient subroutine that solves any  $\Pi_2$ -instances. A Karp-reduction is clearly a Turing-reduction solving a single  $\Pi_2$ -instance.

**Notation 2.2** (Polynomial reduction). *Given two problems  $\Pi_1$  and  $\Pi_2$ , a polynomial reduction (Karp or Turing reduction) from  $\Pi_1$  to  $\Pi_2$  is denoted by  $\Pi_1 \leq \Pi_2$ .*

Consider two problems  $\Pi_1, \Pi_2$  such that  $\Pi_1 \leq \Pi_2$ , thus the problem  $\Pi_2$  is said *harder* than  $\Pi_1$  since every instance of  $\Pi_1$  can be solved by solving an instance of  $\Pi_2$ . Note that this reduction is transitive:

**Remark 2.2.** *Given three problems  $\Pi_1, \Pi_2$  and  $\Pi_3$ , if  $\Pi_1 \leq \Pi_2$  and  $\Pi_2 \leq \Pi_3$  thus  $\Pi_1 \leq \Pi_3$ .*

Polynomial reductions are useful tools. They allow us to highlight complexity relationships between problems. A natural question at this point is to determine whether there exists a problem which is harder than every other problem? And, to a lesser extent, to determine whether there exists a problem in **NP** which is harder than every other problem of **NP**? These questions motivate the introduction of the notion of *hardness* and *completeness*.

**Definition 2.7** (**NP**-hardness [Lee90]). *A problem  $\Pi_1$  is said **NP**-hard if and only if for every problem  $\Pi_2 \in \mathbf{NP}$ ,  $\Pi_2 \leq \Pi_1$ .*

Thus an **NP**-hard problem is a problem that is harder than every problem of **NP**. Thus, based on the hypothesis that  $\mathbf{P} \neq \mathbf{NP}$ , these problems do not admit a polynomial-time algorithm. Indeed such an algorithm, composed with the polynomial reduction algorithm, would provide a polynomial-time algorithm for every problem in **NP**. This contradicts the hypothesis  $\mathbf{P} \neq \mathbf{NP}$ .

Note that an **NP**-hard problem does not necessarily belongs to **NP**. We will see in next section that **NP**-hard can describe problem that are not in **NP**, such as some optimization problems. Actually, an **NP**-hard problem that belongs to **NP** is said **NP**-complete:

**Definition 2.8** (**NP**-completeness). *A decision problem  $\Pi_1$  is **NP**-complete if and only if:*

1.  $\Pi_1$  is **NP**-hard,
2.  $\Pi_1 \in \mathbf{NP}$ .

Based on these definitions, the **NP**-completeness of a problem  $\Pi$  can be proved by checking if  $\Pi \in \mathbf{NP}$  and then by providing a polynomial reduction from another **NP**-complete problem. The only thing left, is to find an *initial* **NP**-complete problem. We easily remark that the previous mechanism of proof relies on the existence of a problem already proved **NP**-complete, and thus is of no use to initiate it.

---

**Decision Problem 2 SAT**


---

<b>Input</b>	A set $U$ of binary variables $x_1, x_2, \dots, x_n$ , to whom are associated two literals $x_1, \bar{x}_1, \dots, x_n, \bar{x}_n$ , a set $\mathcal{C}$ of clauses $C_1, C_2, \dots, C_m$ , each clause being a disjunction of literals, and a binary formula $\Phi = C_1 \wedge \dots \wedge C_m$ .
<b>Question</b>	Does an affectation of values 0 or 1 to the variables exist such that $\Phi = 1$ ?

---

In fact, the problem being proved **NP**-complete is called SAT and is defined as follows.

The following theorem has been shown independently by Cook [Coo71] in the United State in 1971 and by Levin [Lev73] in USSR in 1973.

**Theorem 2.2** (Cook-Levin Theorem [Coo71; Lev73]). *SAT is **NP**-complete.*

Cook proved the **NP**-completeness of SAT by using the definition of **NP** based on non-deterministic Turing machines. Indeed, a problem is said in **NP** if it can be solved in polynomial time by a non deterministic Turing machine<sup>1</sup>. Cook proved that any problem that can be solved in polynomial time with such a Turing machine can be reduced to an instance of SAT.

This closes the introduction of tools mainly used to classify decision problems. However, there exist other kinds of problems, and this manuscript mainly focus on the theoretical aspects of a problem of a particular kind: the *optimization problems*.

### 2.1.3 Optimization problems

Optimization problem differ from decision ones. In these problems we consider the best solution. In other words, we are given a cost (or profit) function that allows us to evaluate the quality of a solution. The objective is then to find the solution of the problem that minimizes (resp. maximizes) the cost (resp. the profit) function. Note that this solution is not necessarily unique.

**Definition 2.9** (Optimization Problem [Cre97]). *An optimization problem  $\Pi$  is a four-tuple  $(\mathcal{I}, sol, m, type)$  where:*

- $\mathcal{I}$  is the set of instances of  $\Pi$
- given an instance  $I \in \mathcal{I}$ ,  $sol(I)$  denotes the set of solutions of  $I$ ,

---

<sup>1</sup>For more informations about non deterministic Turing machines and their relation with **NP**, we refer the reader to two excellent books written respectively by Papadimitriou [Pap94] and Garey and Johnson [GJ79].

- given an instance  $I \in \mathcal{I}$  and  $x \in \text{sol}(I)$  a feasible solution of  $I$ ,  $m(I, x)$  denotes the value of the solution  $x$  of  $I$ ,  $m$  is called the objective function.
- $\text{type} \in \{\min, \max\}$  determined whether  $\Pi$  is respectively a minimization problem (in this case, the value of a solution is called the cost, and the objective is to minimize the cost of a solution) or a maximization problem (in this case, the value of a solution is called the profit of a solution, and aim at maximizing it).

**Definition 2.10.** Given an **NPO** problem  $\Pi = (\mathcal{I}, \text{sol}, m, \text{type})$ , and an instance  $I \in \mathcal{I}$ , a solution  $S$  satisfying  $m(I, S) = \text{type} \{m(I, x) : x \in \text{sol}(I)\}$  is called an optimal solution. We denote such a solution as  $\text{opt}(I)$ .

**Definition 2.11.** Given an optimization problem  $\Pi$  and an algorithm  $\mathcal{A}$  we say that  $\mathcal{A}$  solves  $\Pi$  in time  $f(n)$  if, for any  $\Pi$ -instance  $I$ ,  $\mathcal{A}$  returns an optimal solution of  $I$  in at most  $f(n)$  operations, where  $n = |I|$ .

Remark that given an optimization problem  $\Pi$ , we always can define a decision problem, asking whether, given an integer  $k$ , a solution of profit greater than  $k$  or of cost lesser than  $k$  does exist?

**Definition 2.12** (Associated decision problem). Given an optimization problem  $\Pi = (\mathcal{I}, \text{sol}, m, \text{type})$ , we define  $\Pi_d$  the associated decision problem as follows:

- a  $\Pi_d$ -instance can be defined as a couple  $(I, k)$  composed with a  $\Pi$ -instance  $I \in \mathcal{I}$  and a strictly positive integer  $k$ ,
- an instance  $(I, k)$  is positive if and only if there exists a solution  $x \in \text{sol}(I)$  such that  $m(I, x) \geq k$  if  $\text{type} = \max$  or  $m(I, x) \leq k$  otherwise.

We can remark that some optimization problems have **NP**-complete associated decision problem while some others have their associated decision problem belonging to **P**. Does this imply that some optimization problems are harder than others?

This natural question motivates the introduction of the **NPO** and **PO** complexity classes.

**Definition 2.13** (**NPO** [Pas04]). **NPO** is defined as the set of optimization problems  $\Pi = (\mathcal{I}, \text{sol}, m, \text{type})$  verifying the following properties:

1. Given an instance  $I$ , we can define in polynomial time in the size of  $|I|$  whether  $I \in \mathcal{I}$  or not. We say that  $\mathcal{I}$  is recognizable in polynomial time.
2. There exists a polynomial  $p$  such that for every instance  $I \in \mathcal{I}$  and for every feasible solution  $x \in \text{sol}(I)$ ,  $x \in \{0, 1\}^{p(|I|)}$ . In other words, given an instance  $I \in \mathcal{I}$ , every feasible solution of  $I$  can be encoded by a word of size polynomial in  $|I|$ .
3. Given an instance  $I \in \mathcal{I}$  and a solution  $x$ , we can define in polynomial time whether  $x$  belongs to  $\text{sol}(I)$  or not.

4. For every instance  $I \in \mathcal{I}$ , a feasible solution  $x \in \text{sol}(I)$  can be found in polynomial time in  $|I|$ .
5. For every instance  $I \in \mathcal{I}$  and for every solution  $x \in \text{sol}(I)$ ,  $m(I, x)$  is computable in polynomial time in  $|I|$ .

**Proposition 2.1.** *The decision problem  $\Pi_d$  associated to an **NPO** optimization problem  $\Pi$  belongs to **NP**.*

Indeed, the existence of a polynomial verifier for  $\Pi_d$  is ensured by Properties 3 and 5, given in Definition 2.13 of  $\Pi$ .

As for the class **P** on decision problems, we can define the class **PO** on the optimization problems based on definition 2.11.

**Definition 2.14 (PO).** *The class **PO** is the set of all optimization problems that can be solved in polynomial time.*

**Proposition 2.2.** *The decision problem  $\Pi_d$  associated to a **PO** optimization problem  $\Pi$  belongs to **P**.*

Indeed, if there exists an algorithm providing the best solution for  $\Pi$  it is easy to determine whether there exists a solution better than an integer  $k$ . This proposition can be rephrased as follows:

**Proposition 2.3.** *Given an optimization problem  $\Pi$  and its associated problem  $\Pi_d$  such that  $\Pi_d$  is **NP**-complete, thus  $\Pi \notin \mathbf{PO}$  unless  $\mathbf{P} = \mathbf{NP}$ .*

We close this section on the following interesting remark on the classification of optimization problems:

**Remark 2.3.** *An optimization problem with associated **NP**-complete decision problem is **NP**-hard.*

## 2.2 Approximation Algorithms

As for decision problems being **NP**-complete, the most interesting and natural optimization problems do not belong to **PO**. Hence it is very unlikely to find a polynomial-time algorithm that solves optimally at least one of them. How can we then handle such problems? A part of the answer can be determined by analyzing the following sentence: “It is very unlikely to find a polynomial-time algorithm that solves optimally an **NP**-hard optimization problem”. Indeed, this sentence highlight the fact that, if  $\mathbf{P} \neq \mathbf{NP}$ , we can not ensure efficiency and exactness of an algorithm, but we can try to make concessions on the efficiency or on the exactness of it. Based on this observation, tools have be developed either to obtain, in polynomial-time, a solution which is non optimal, or to find an optimal solution allowing non polynomial-time algorithms.

In the first case, the objective is to design polynomial-time algorithm providing solutions without guarantee on the optimality of such solutions. However, in this

manuscript we will focus on a remarkable kind of such algorithms, called the *approximation algorithms*. The approximation algorithms run in polynomial-time but give us a guarantee on the gap between the value of a computed solution and the value of an optimal one.

We can then define the performance ratio of a solution.

**Definition 2.15** (Performance ratio [Cre97]). *Given an NPO problem  $\Pi = (\mathcal{I}, \text{sol}, m, \text{type})$ , an instance  $I \in \mathcal{I}$  and a solution of this instance  $x \in \text{sol}(I)$ , the performance ratio of  $x$  is defined as:*

$$R(I, x) = \max \left\{ \frac{m(I, \text{opt}(I))}{m(I, x)}, \frac{m(I, x)}{m(I, \text{opt}(I))} \right\}$$

*By definition, the performance ratio of any solution  $x$  verifies  $R(I, x) \geq 1$ .*

It can be seen as an intuitive measure of the *gap between a solution of an instance and one of its optima*: better is the solution, closer from one is the ratio. By definition, a performance ratio is greater than one, and a solution with performance ratio equal to one is an optimal solution of the considered instance.

Let us remark that this measure of solution quality is not the only one. Indeed, in opposition to this multiplicative performance ratio we can define an additive one. Given an instance  $I$  of an NPO problem  $\Pi$  and a solution  $x \in \text{sol}(I)$ , the additive performance  $R^+(I, x)$  is defined by:

$$R^+(I, x) = \max \{m(I, \text{opt}(I)) - m(I, x), m(I, x) - m(I, \text{opt}(I)); \}$$

We can, also cite the *differential ratio*, based on the observation that the value of a solution can be seen as a convex combination of the value of an optimal solution, and the value of a worst solution. Hence, given an instance  $I$  of an NPO problem  $\Pi$  and a solution  $x \in \text{sol}(I)$ , if we note *worst*( $I$ ) a worst solution, the ratio is defined by:

$$\delta(I, x) = \frac{|m(I, \text{worst}(I)) - m(I, x)|}{|m(I, \text{worst}(I)) - m(I, \text{opt}(I))|}$$

This ratio takes values in  $[0, 1]$ , but as for performance ratio, an optimal solution has a ratio equal to 1. For more informations on differential ratio and the associated complexity classes we refer the reader to the PhD Thesis of Bruno Escoffier [Esc05]<sup>2</sup>.

From now we only focus on the performance ratio as specified in Definition 2.15. The notion of *approximation algorithms* is deeply related to the notion of performance ratio. Informally, an approximation algorithm is an algorithm that returns, for every instance of a problem, a solution whose performance ratio is bounded by a function. Thus, it offers the guarantee that the returned solution cannot be arbitrarily bad.

**Definition 2.16** (Approximation algorithm [Cre97]). *Given  $\Pi = (\mathcal{I}, \text{sol}, m, \text{type})$  an NPO problem, an algorithm  $\mathcal{A}$  for  $\Pi$  and a function  $\rho : \mathcal{I} \rightarrow [1, +\infty[$ ,  $\mathcal{A}$  is a  $\rho$ -approximation algorithm if and only if it verifies the following properties:*

<sup>2</sup>This manuscript is written in French.

1.  $\forall I \in \mathcal{I}$  the solution returned by  $\mathcal{A}$  for instance  $I$ , denoted  $\mathcal{A}(I)$ , belongs to  $\text{sol}(I)$ ,
2.  $\forall I \in \mathcal{I}, R(I, \mathcal{A}(I)) \leq \rho(I)$ .

$\rho$  is called the ratio of the algorithm.

In other words, given  $\Pi = (\mathcal{I}, \text{sol}, m, \min)$  (resp.  $\Pi = (\mathcal{I}, \text{sol}, m, \max)$ ) an **NPO** problem, a  $\rho$ -approximation algorithm for  $\Pi$  is a polynomial-time algorithm that computes, for every  $\Pi$ -instance  $I$ , a feasible solution  $\mathcal{A}(I) \in \text{sol}(I)$  with cost  $m(I, \mathcal{A}(I))$  satisfying  $m(I, \text{opt}(I)) \leq m(I, \mathcal{A}(I)) \leq \rho \times m(I, \text{opt}(I))$  (resp. with profit  $m(I, \mathcal{A}(I))$  satisfying  $\frac{m(I, \text{opt}(I))}{\rho} \leq m(I, \mathcal{A}(I)) \leq m(I, \text{opt}(I))$ ).

**Definition 2.17** (Approximability). *If an **NPO** problem  $\Pi$  admits a  $\rho$ -approximation algorithm running in time  $f(n)$ , it is said  $\rho$ -approximable in time  $f(n)$ .*

Based on the previous definition, we can make a remark on approximability of a problem.

**Remark 2.4.** *The approximability of an **NPO** problem  $\Pi$  is defined by its polynomial-time approximation algorithm with the best ratio.*

We can then use this remark to define several complexity classes. One of the most natural is the class of **NPO** problems that admits an approximation algorithm with constant ratio. It is called **APX**.

**Definition 2.18** (**APX** [Aus+99]). *An **NPO** problem  $\Pi$  belongs to **APX** if and only if there exists a fixed constant  $r \geq 1$  such that  $\Pi$  is  $r$ -approximable.*

The following class, called **PTAS**, contains every problem that admits an approximation algorithm whose ratio can be as close to 1 as desired. This algorithm achieves this ratio by making concessions on execution time since its running time strongly depends on the desired precision.

**Definition 2.19** (**PTAS** [Aus+99]). *An **NPO** problem  $\Pi$  belongs to **PTAS** if and only if, for every fixed  $\varepsilon > 0$ , it is  $(1 + \varepsilon)$ -approximable.*

In fact, we make a slight abuse by talking about “an approximation algorithm whose ratio can be as close to 1 as desired”. Indeed each **NPO** problem  $\Pi$  belonging to **PTAS** admits an approximation algorithm for every fixed  $\varepsilon$ , in other words  $\Pi$  admits an infinity of approximation algorithms, and the ratio of these algorithms depends on  $\varepsilon$ . This kind of family of algorithms is called an *approximation scheme*.

Note that there exist several approximation schemes defined by their complexity dependency in  $\varepsilon$ . Indeed the time complexity is given in function of the input size but also in function of  $\varepsilon$ . However, since  $\varepsilon$  is fixed for every algorithm of the scheme, even approximation scheme running in  $\mathcal{O}(n^{1/\varepsilon})$  are considered to be polynomial. Indeed, with  $\varepsilon$  fixed,  $1/\varepsilon$  is considered as a constant. An approximation scheme with exponential dependency on  $1/\varepsilon$ , is called *Polynomial-Time Approximation Scheme* (also known as **PTAS**).

Hence the Definition 2.19 of the complexity class **PTAS** can be rephrased using the notion of approximation scheme.

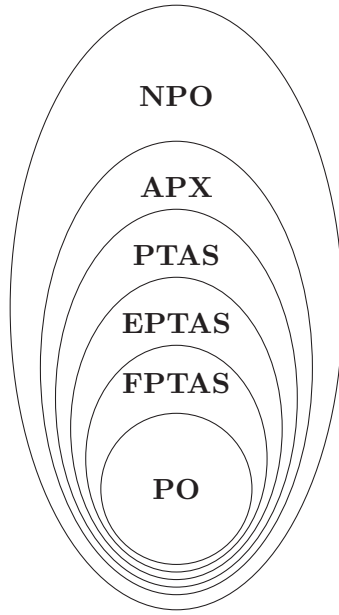


Figure 2.2: Representation of a part of the approximation classes hierarchy if  $\mathbf{P} \neq \mathbf{NP}$ .

**Definition 2.20 (PTAS).** *An NPO problem  $\Pi$  belongs to PTAS if and only if it admits a PTAS.*

Since exponential dependency in  $\varepsilon$  are authorized, a PTAS can quickly be inefficient in practice. This motivates the introduction of more restrictive approximation schemes.

**Definition 2.21 (EPTAS [CT97]).** *An NPO problem belongs to EPTAS if and only if it admits an Efficient Polynomial-Time Approximation Scheme, i.e. an approximation scheme running, for every fixed  $\varepsilon$ , in time  $\mathcal{O}(n^c)$  with  $c$  a constant independent of  $\varepsilon$ .*

In this class can be found approximation running in time  $\mathcal{O}(f(\frac{1}{\varepsilon})\text{poly}(n))$  with  $f$  a function that can be exponential in  $\frac{1}{\varepsilon}$ .

Once again, the exponential dependency in  $\frac{1}{\varepsilon}$  can lead to inefficient algorithm for small  $\varepsilon$ . We introduce then an even more restrictive approximation scheme.

**Definition 2.22 (FPTAS [Aus+99]).** *An NPO problem belongs to FPTAS if and only if it admits a Fully Polynomial-Time Approximation Scheme, i.e. an approximation scheme running in time polynomial in both the size of the input and  $\frac{1}{\varepsilon}$ .*

**Remark 2.5.** *It is clear that  $\mathbf{PO} \subseteq \mathbf{FPTAS} \subseteq \mathbf{EPTAS} \subseteq \mathbf{PTAS} \subseteq \mathbf{APX} \subseteq \mathbf{NPO}$ . The strictness of the inclusion remains an open problem. Indeed  $\mathbf{PO} = \mathbf{FPTAS} = \mathbf{EPTAS} = \mathbf{PTAS} = \mathbf{APX} = \mathbf{NPO}$  if and only if  $\mathbf{P} = \mathbf{NP}$ .*

A graphical representation of this remark is given on Figure 2.2. We point attention of the reader on the fact that the introduced hierarchy is not complete. Some



complexity classes such as **poly** – **APX** or **Log** – **APX**, finding their place in the hierarchy between **APX** and **NPO**, have not been presented. For a more complete description of this hierarchy we recommend the manuscript of Bruno Escoffier [Esc05] and an excellent book written by Giorgio Ausiello et al. [Aus+99].

As for decision problems and related complexity classes, given an optimization problem  $\Pi$ , the objective will be to find clues on its complexity by finding the complexity classes it belongs to. This can be done by inherent properties, *e.g.* the existence of a PTAS to prove the membership in **PTAS**, or by highlighting relations among considered problems. This motivates the introduction of another kind of reductions: the *approximation preserving reductions*.

### 2.3 Approximation Preserving Reductions

This section aims at presenting a few numbers of *approximation preserving reductions*.

Let us first remark that, given two **NPO** problems  $\Pi_1$  and  $\Pi_2$ , the already seen reductions (Karp and Turing reductions) are not appropriate. Indeed, a reduction that given two problems  $\Pi_1$  and  $\Pi_2$ , maps any yes-instance of  $\Pi_1$  to a yes-instance of  $\Pi_2$  is not sufficiently expressive to handle the power of expression of optimization problems. Furthermore, to highlight a mapping between the two problems, we need not only a polynomial-time algorithm transforming any  $\Pi_1$ -instance into a  $\Pi_2$ -instance, but also another polynomial-time algorithm that associates in return the solutions of the  $\Pi_2$ -instance to the solutions of the initial  $\Pi_1$ -instance.

This motivates the following definition of a polynomial-time reduction.

**Definition 2.23** (Polynomial-time reduction [Cre97]). *Given two **NPO** optimization problems  $\Pi_1 = (\mathcal{I}_1, sol_1, m_1, type_1)$  and  $\Pi_2 = (\mathcal{I}_2, sol_2, m_2, type_2)$ , a polynomial-time reduction from  $\Pi_1$  to  $\Pi_2$  is a couple of polynomial-time computable functions  $(f, g)$ :*

$$f : \begin{cases} \mathcal{I}_1 & \rightarrow \mathcal{I}_2 \\ I & \mapsto f(I) \end{cases} \quad g : \begin{cases} \mathcal{I}_1 \times sol_2(f(\mathcal{I}_1)) & \rightarrow sol_1(\mathcal{I}_1) \\ (I, x) & \mapsto g(I, x) \end{cases}$$

*$f$  is such that to any instance of  $I \in \mathcal{I}_1$  of  $\Pi_1$ , it associates an instance  $f(I)$  of  $\Pi_2$ . And given an instance  $I \in \mathcal{I}_1$ , to any solution of constructed instance  $x \in sol_2(f(I))$ ,  $g$  associate a solution of initial instance  $g(I, x) \in sol_1(I)$ .*

The scheme induced by this definition can be depicted by Figure 2.3.

**Notation 2.3.** *In a general way, given two optimization problems  $\Pi_1$  and  $\Pi_2$ , the existence of a polynomial reduction  $(f, g)$  from  $\Pi_1$  to  $\Pi_2$  is noted  $\Pi_1 \leq \Pi_2$ .*

In a first time we extend the notion of *membership preservation*, introduced with the Karp reduction.

**Definition 2.24** (Membership preservation). *Given two **NPO** problems  $\Pi_1, \Pi_2$ , a polynomial-time reduction  $(f, g)$  such that  $\Pi_1 \leq \Pi_2$  and a complexity class  $\mathcal{C}$ ,  $(f, g)$  preserves membership in  $\mathcal{C}$  if and only if  $\Pi_2 \in \mathcal{C} \Rightarrow \Pi_1 \in \mathcal{C}$ .*



Figure 2.3: General reduction scheme.

This notion allows us to extend the notion of completeness to any complexity class.

**Definition 2.25** (Completeness [Pas04; Lee90]). *Given two classes of  $\mathbf{NPO}$ ,  $\mathbf{C}$  and  $\mathbf{C}'$ , and a polynomial-time reduction  $R$  that preserves the membership in  $\mathbf{C}'$ , a problem  $\Pi_1$  is said  $\mathbf{C}$ -complete in regard of  $(\mathbf{C}', R)$  if and only if  $\Pi_1 \in \mathbf{C}$  and for every problem  $\Pi_2$  of  $\mathbf{C}$ ,  $\Pi_2 \leq_R \Pi_1$ .*

Note that this definition generalizes the definition of  $\mathbf{NP}$ -completeness previously defined. In fact, we defined the  $\mathbf{NP}$ -completeness in regard to  $(\mathbf{P}, \leq_T)$ . In the following, in order to avoid heavy notation, we will use the terminology  $\mathbf{C}$ -complete instead of  $\mathbf{C}$ -complete in regard to  $(\mathbf{C}', R)$  when  $\mathbf{C}'$  is included in  $\mathbf{C}$ . For instance, given a polynomial-time reduction  $R$  that preserves membership in  $\mathbf{PTAS}$ , a problem  $\Pi_1 \in \mathbf{APX}$  such that  $\forall \Pi_2 \in \mathbf{APX}, \Pi_2 \leq_R \Pi_1$  will be said  $\mathbf{APX}$ -complete instead of  $\mathbf{APX}$ -complete in regard to  $(\mathbf{PTAS}, R)$ .

We first introduce the easiest, but also the most strict, approximation preserving reduction. Given two  $\mathbf{NPO}$  problems  $\Pi_1$  and  $\Pi_2$ , this polynomial-time reduction maps, to any solution of a  $\Pi_1$ -instance, a solution of the corresponding  $\Pi_2$ -instance of same cost. It follows that this reduction can exist only between  $\mathbf{NPO}$  problems sharing the same type.

**Definition 2.26** (S-reduction [Cre97]). *Given two  $\mathbf{NPO}$  problems  $\Pi_1, \Pi_2$  and a reduction  $(f, g)$  from  $\Pi_1$  to  $\Pi_2$ , if:*

1. for any  $I \in \mathcal{I}_1$ ,  $m_2(f(I), \text{opt}_2(f(I))) = m_1(I, \text{opt}_1(I))$ ,
2. for any  $I \in \mathcal{I}_1$  and any solution  $x \in \text{sol}_2(f(I))$ ,  $m_1(I, g(I, x)) = m_2(f(I), x)$ .

then  $(f, g)$  is an S-reduction.

**Notation 2.4.** *Given two  $\mathbf{NPO}$  problems  $\Pi_1$  and  $\Pi_2$ ,  $\Pi_1$  S-reduces to  $\Pi_2$  is denoted  $\Pi_1 \leq_s \Pi_2$ .*

Several S-reductions are given in this manuscript. They can be found in Sections 6.1.1, 6.1.2, 6.1.4 and 6.1.5.

**Property 2.1.** *An S-reduction preserves the membership in  $\mathbf{FPTAS}$  and  $\mathbf{APX}$ .*

In fact, an S-reduction preserves the membership in  $\mathbf{FPTAS}$ ,  $\mathbf{EPTAS}$ ,  $\mathbf{PTAS}$  and  $\mathbf{APX}$ <sup>3</sup>. This motivates the following remark:

<sup>3</sup>This list is not exhaustive, but we restrict it to the classes we are interested in.

**Property 2.2.** *A polynomial-time reduction (whose execution time only depends on the size of the instance) preserving membership in **FPTAS** also preserves membership in **EPTAS** and **PTAS** but does not necessarily preserve membership in **APX**.*

Indeed, the distinction between **FPTAS**, **EPTAS** and **PTAS** is made on the running time of the algorithm and not on the approximation ratio. Thus a reduction that does not deteriorate the approximation ratio and running time required to define an FPTAS, will not neither deteriorate approximation ratio and running time required to define an EPTAS or a PTAS. Thus, in the following we only specify the preservation of the membership in **PTAS** and/or **APX**.

As we will see hereafter, the  $L$ -reduction, when considered on maximization problems is an illustration of a polynomial-time reduction that preserves membership in **FPTAS**, **EPTAS** and **PTAS** but not in **APX**.

Another pertinent polynomial-time reduction is the strict-reduction. This reduction ensures that any solution of a  $\Pi_2$ -instance is turned into a solution of the corresponding  $\Pi_1$ -instance with a better performance ratio. This reduction is a little bit less restrictive than the  $S$ -reduction since it allows the cost of the solution to be different between the two problems. As such, it allows the reduction from a minimization problem to a maximization one and vice versa.

**Definition 2.27** (Strict-reduction [Cre97]). *Given two NPO problems  $\Pi_1$ ,  $\Pi_2$  and a reduction  $(f, g)$  from  $\Pi_1$  to  $\Pi_2$ , if for any instance of  $\Pi_1$   $I \in \mathcal{I}_1$  and for any solution in the constructed instance of  $\Pi_2$   $x \in \text{sol}_2(f(I))$ :*

$$R_1(I, g(I, x)) \leq R_2(f(I), x)$$

Even though this reduction is more permissive in terms of solution cost, the strict-reduction preserves membership in the same complexity classes as the  $S$ -reduction.

**Property 2.3.** *A strict-reduction preserves the membership in **PTAS** and **APX**.*

**Notation 2.5.** *Given two NPO problems  $\Pi_1$  and  $\Pi_2$ ,  $\Pi_1$  strict-reduces to  $\Pi_2$  is denoted  $\Pi_1 \leq_{\text{strict}} \Pi_2$ .*

Several strict-reductions can be found in this manuscript in Sections 6.1.3, 6.2, 7.1.1 and 7.1.2.

The last approximation preserving reduction we will use in this manuscript is the linear-reduction. This reduction is a little bit more peculiar than the previous ones. Indeed, while the  $S$  and the strict-reductions bound respectively the cost and the performance ratio of the solution of the created instance, leading to intuitive results in terms of approximation classes membership preservation, this reduction bounds the optimal value of the created instance and the cost difference between any solution and the optimal one.

**Definition 2.28** (*L* or linear-reduction [PY91]). Given two **NPO** problems  $\Pi_1, \Pi_2$  and a reduction  $(f, g)$  from  $\Pi_1$  to  $\Pi_2$ , if there exist  $\alpha, \beta \in \mathbb{R}^{+,*}$  such that, for any  $\Pi_1$ -instance of  $I \in \mathcal{I}_1$  and for any solution of the  $\Pi_2$ -instance  $x \in \text{sol}_2(f(I))$ , the following holds:

$$m_2(f(I), \text{opt}_2(f(I))) \leq \alpha m_1(I, \text{opt}_1(I)) \quad (2.1)$$

$$|m_1(I, \text{opt}_1(I)) - m_1(I, g(I, x))| \leq \beta |m_2(f(I), \text{opt}_2(f(I))) - m_2(f(I), x)| \quad (2.2)$$

**Notation 2.6.** Given two **NPO** problems  $\Pi_1$  and  $\Pi_2$ ,  $\Pi_1$  *L*-reduces to  $\Pi_2$  is denoted  $\Pi_1 \leq_L \Pi_2$ .

Another particularity of this reduction is that the conclusions that can be done from the existence of such a polynomial-time reduction differ in function of the type of the initial problem, as explained in Property 2.4.

**Property 2.4.** A linear-reduction from an **NPO** problem  $\Pi_1$  to another **NPO** problem  $\Pi_2$  preserves membership in:

1. **FPTAS** if  $\Pi_1$  is a maximization problem,
2. **FPTAS** and **APX** if  $\Pi_1$  is a minimization problem.

Property 2.4 is not as straightforward as previous ones. Let us explain the previous property. We can identify four cases, but we provide details for only two of them:

*Proof.* **Minimization**  $\rightarrow$  **Minimization**

In this case Equation (2.2) can be written as follows:

$$m_1(I, g(I, x)) - m_1(I, \text{opt}_1(I)) \leq \beta (m_2(f(I), x) - m_2(f(I), \text{opt}_2(f(I))))$$

Using Equation (2.1)

$$\Rightarrow \frac{m_1(I, g(I, x)) - m_1(I, \text{opt}_1(I))}{m_1(I, \text{opt}_1(I))} \leq \frac{\alpha \beta (m_2(f(I), x) - m_2(f(I), \text{opt}_2(f(I))))}{m_2(f(I), \text{opt}_2(f(I)))}$$

$$\Rightarrow R_1(I, g(I, x)) - 1 \leq \alpha \beta (R_2(f(I), x) - 1)$$

$$\Rightarrow R_1(I, g(I, x)) \leq \alpha \beta (R_2(f(I), x) - 1) + 1$$

It is clear that if  $\Pi_2$  admits a PTAS,  $\Pi_1$  admits also a PTAS. Indeed, for any solution  $x$  returned by the algorithm  $R_2(f(I), x) \leq 1 + \varepsilon$  with a small  $\varepsilon$ , thus  $R_1(I, g(I, x)) \leq 1 + \alpha \beta \varepsilon = 1 + \varepsilon'$  with a small  $\varepsilon'$ . It is also easy to see that if  $\Pi_2$  admits an approximation algorithm with a constant ratio,  $\Pi_1$  admits an approximation algorithm with constant ratio. Hence, in this case, the membership in **PTAS** and **APX** is preserved.

A similar proof can be done for a linear-reduction mapping a minimization problem to a maximization one.

**Maximization  $\rightarrow$  Maximization**

In this case Equation (2.2) can be written as follows:

$$\begin{aligned} m_1(I, \text{opt}_1(I)) - m_1(I, g(I, x)) &\leq \beta (m_2(f(I), \text{opt}_2(f(I))) - m_2(f(I), x)) \\ \Rightarrow 1 - \frac{1}{R_1(I, g(I, x))} &\leq \alpha\beta \left(1 - \frac{1}{R_2(f(I), x)}\right) \\ \Rightarrow \frac{1}{R_1(I, g(I, x))} &\geq 1 - \alpha\beta \left(1 - \frac{1}{R_2(f(I), x)}\right) \end{aligned}$$

Suppose now that  $\Pi_2$  admits a PTAS. By definition for a small fixed  $\varepsilon$  and for any solution  $x$  returned by the approximation scheme:

$$\begin{aligned} R_2(f(I), x) &\leq 1 + \varepsilon \\ \Rightarrow 1 - \alpha\beta \left(1 - \frac{1}{R_2(f(I), x)}\right) &\geq \frac{1 + \varepsilon(1 - \alpha\beta)}{1 + \varepsilon} \end{aligned}$$

Since  $\alpha\beta \geq 1$ ,  $1 - \alpha\beta \leq 0$ . If we note  $c = |1 - \alpha\beta|$ , we can write:

$$\begin{aligned} \Rightarrow \frac{1}{R_1(I, g(I, x))} &\geq \frac{1 - c\varepsilon}{1 + \varepsilon} \\ \Rightarrow R_1(I, g(I, x)) &\leq \frac{1 + \varepsilon}{1 - c\varepsilon} \\ \Rightarrow R_1(I, g(I, x)) &\leq 1 + \varepsilon' \quad \text{with } \varepsilon' = \frac{(c+1)\varepsilon}{1 - c\varepsilon} \end{aligned}$$

Hence  $\Pi_1$  admits a PTAS.

Let us now see what happens if  $\Pi_2$  admits an approximation algorithm with constant ratio. In this case:

$$1 - \alpha\beta \left(1 - \frac{1}{R_2(f(I), x)}\right) \leq 0$$

By definition:

$$R_1(I, g(I, x)) \geq 1 > 1 - \alpha\beta \left(1 - \frac{1}{R_2(f(I), x)}\right)$$

Thus  $R_1(I, g(I, x))$  is not constrained by the reduction and can take any value. Similar calculus can be done for a linear-reduction mapping a maximization problem to a minimization one, showing that in case of an initial maximization problem, a linear-reduction gives no clue on the preservation of membership in **APX**. In fact, Crescenzi et al. [Cre+99] highlight strong evidence that in general the linear-reduction cannot be used to prove **APX** membership of a maximization **NPO** problem. □

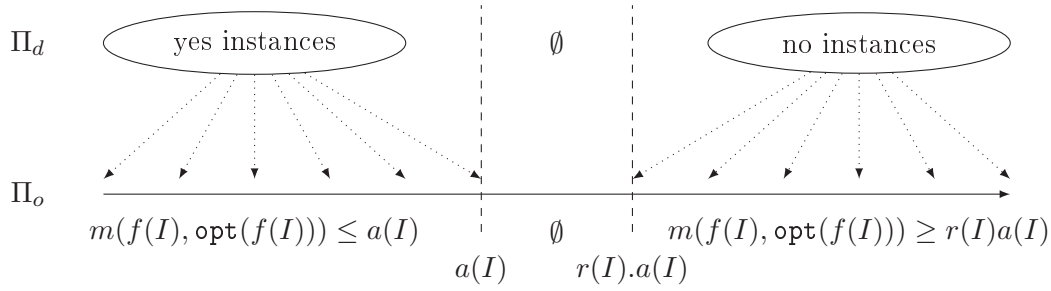


Figure 2.4: Illustration of the principle of a Gap-reduction

## 2.4 Gap-reduction

The next polynomial-time reduction we will define is not an approximation preserving reduction but is mainly used to prove negative results. This reduction, called *Gap Reduction* maps a *promise problem* to an optimization one.

Let us first introduce *promise problems* that are a generalization of decision problems.

**Definition 2.29** (Promise Problem [SV97]). *A promise problem  $\Pi$  is a couple of disjoint subsets  $D_{\Pi}^+ \subseteq \{0, 1\}^*$  and  $D_{\Pi}^- \subseteq \{0, 1\}^*$  of respectively positive and negative instances.*

**Remark 2.6.** *Note that the decision problem can be defined as the set of all promise problems  $\Pi$  such that  $D_{\Pi}^+ \cup D_{\Pi}^- = \{0, 1\}^*$ .*

The notion of decision problem solving can naturally be extended to promise problems as follows:

**Definition 2.30.** *Given a promise problem  $\Pi$  and an algorithm  $\mathcal{A}$ , we say that  $\mathcal{A}$  decides or solves  $\Pi$  in time  $f(n)$  if given any  $\Pi$ -instance  $I \in D_{\Pi}^+ \cup D_{\Pi}^-$ ,  $\mathcal{A}$  determines whether  $I \in D_{\Pi}^+$  or  $I \in D_{\Pi}^-$  in at most  $f(n)$  operations where  $n = |I|$ .*

**Definition 2.31** (Gap reduction [Vaz01]). *Let  $\Pi_d$  be a promise problem and  $\Pi_o$  an NPO minimization problem. A reduction  $(f, g)$  from  $\Pi_d$  to  $\Pi_o$  is said to be a  $r$ -Gap-reduction if there exist two functions  $a$  and  $r$  such that, for every instance of  $\Pi_d$ ,  $I \in D_{\Pi}$ , the following holds:*

1.  $I$  is a yes-instance  $\Rightarrow m(f(I), \text{opt}_o(f(I))) \leq a(I)$
2.  $I$  is a no-instance  $\Rightarrow m(f(I), \text{opt}_o(f(I))) \geq r(I).a(I)$

A graphical representation of this reduction is given on Figure 2.4. We can see that yes-instances are mapped to  $\Pi_o$  instances with the optimal value being less than  $a(I)$ . On the other hand, no-instances of  $\Pi_d$  are mapped to instances of  $\Pi_o$  with optimal value larger than  $r(I)a(I)$ . It follows that, for any  $\varepsilon > 0$ , a  $(r - \varepsilon)$ -approximation algorithm

for  $\Pi_o$  would solve  $\Pi_d$  in polynomial time. Indeed, given  $I$  a yes-instance for  $\Pi_d$ , the cost of the optimal solution in  $f(I)$  verify:  $m(\text{opt}(I)) \leq a(I)$  and thus the solution  $S$  computed by approximation algorithm verify:  $m(S) \leq (r(I) - \varepsilon)a(I) < r(I)a(I)$ . On the other hand, given  $I$  an no-instance of  $\Pi_d$ ,  $f(I) \geq r(I)a(I)$  by construction. Hence, such an algorithm implies the existence of a polynomial-time algorithm that solves  $\Pi_d$ .

From this, we deduce the following lemma.

**Lemma 2.1.** *If there exists a  $r$ -Gap-reduction from an NP-hard promise problem  $\Pi_d$  to an NPO problem  $\Pi_o$ , thus, for any  $\varepsilon > 0$ ,  $\Pi_o$  does not admit a  $(r - \varepsilon)$ -approximation algorithm unless  $\mathbf{P} = \mathbf{NP}$ .*

## 2.5 Other complexity hypotheses

### 2.5.1 The Unique Game Conjecture

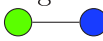
Previously, in Subsection 2.1.2, we introduced a widely accepted conjecture stating that  $\mathbf{P}$  and  $\mathbf{NP}$  are distinct complexity classes. Naturally, other hypotheses are used in computer science. This section aims at introducing one of them that will be used in this manuscript.

This conjecture has been introduced in 2002, by Subhash Khot in [Kho02] and formulated in terms of some particular *two-provers one-round games* called *Unique games*. However, this formulation requires the introduction of a certain number of notions that will not be useful in this manuscript. This motivates the formulation of this conjecture under an equivalent but purely combinatorial way.

Let us first introduce UNIQUE LABEL COVER problem. This problem can be described as follows. We are given  $k$  colors and a complete bipartite graph  $G = (V, W, E)$ . Each edge has a weight  $p_e$ , such that the sum of the weights of all edges is equal to one. Each edge  $e$  comes with a label, where a label is a permutation  $\pi_e : [k] \rightarrow [k]$ .

The objective is to find a coloring  $c : V \cup W \rightarrow [k]$  of the vertices of the graph that maximizes the weight of satisfied edges, an edge  $e = \{u_1, u_2\}$  being said satisfied if and only if  $c(u_2) = \pi_e(c(u_1))$ .

This problem is formalized in Problem 1.

Examples of instances are given in Figures 2.5 and 2.6. On Figure 2.6, the bold edge is unsatisfied, the pair  being not part of its label.

The Unique Game Conjecture can be stated as follows.

**Conjecture 2.1.** UNIQUE LABEL COVER is NP-hard.

**Remark 2.7.** *Instances of UNIQUE LABEL COVER admitting an assignment  $\mathcal{A}$  such that  $c(\mathcal{A}) = 1$  can be detected in polynomial time.*

A possible algorithm consists in selecting a node of the graph and for each of the available color *propagating* the coloration with respect to the edge labels. Since the

---

**Optimization Problem 1** UNIQUE LABEL COVER
 

---

<b>Input</b>	A complete bipartite graph $G = (V, W, E)$ , an integer $k$ , for each $e \in E$ a weight $p_e$ such that $\sum_{e \in E} p_e = 1$ and a collection of permutations $\pi_e : [k] \rightarrow [k]$ , called labels.
<b>Output</b>	An assignment $\mathcal{A}$ of the labels to the vertices of $V, W$ .
<b>Objective</b>	Maximize $c(\mathcal{A})$ , the total weight of satisfied edges.

---

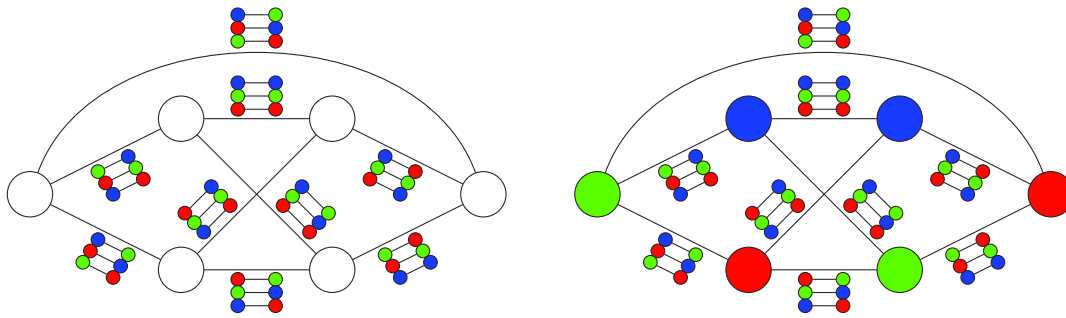


Figure 2.5: On the left, an instance of UNIQUE LABEL COVER with equal weights  $p_e = 1/e, \forall e \in E$  and, on the right, an assignment  $\mathcal{A}$  of the labels to the vertices such that  $c(\mathcal{A}) = 1$ .

instance is such that every edge of the graph can be satisfied, there exists a coloring of the selected vertex such that the propagation does not lead to a conflict.

This hypothesis is used in Sections 6.1.4 and 8.1.1.

### 2.5.2 NP $\not\subseteq$ QP

We briefly introduce the class of decision problems called **QP**.

**Definition 2.32 (QP).** *The class **QP** contains every decision problem that admits a quasi-polynomial time algorithm, i.e. algorithms running in time  $2^{\mathcal{O}(\log n)^c}$  for some fixed  $c$  and where  $n$  is the size of the input instance.*

Note that the relationship between **P** and **QP** is obvious. Indeed, the existence of a polynomial-time algorithm for a problem naturally implies the existence of a quasi-polynomial-time algorithm for the latter. It follows that  $\mathbf{P} \subseteq \mathbf{QP}$ . However its relationship with **NP** is not as clear. Note that **QP** make no mention of a polynomial certificate in its definition, this implies that there may exist problem being elements of **QP** while not being part of **NP**. On the other side, the inclusion of **NP** in **QP** is unlikely as this would imply that every problem of **NP** admit a quasi-polynomial-time algorithm.



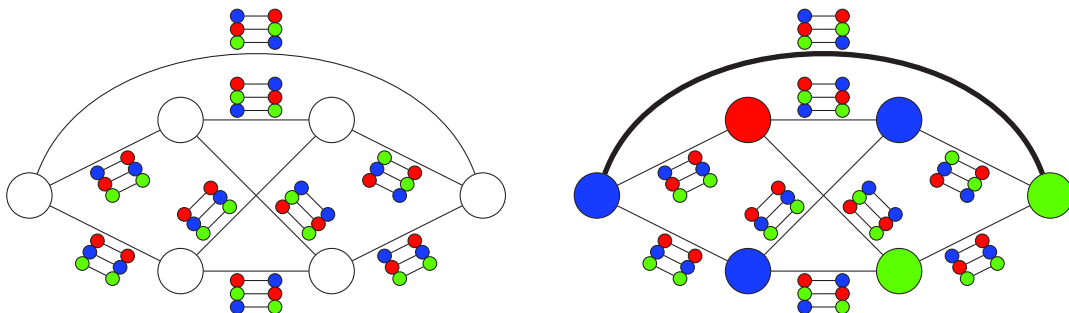


Figure 2.6: On the left, an instance of UNIQUE LABEL COVER with equal weights  $p_e = 1/e, \forall e \in E$  and, on the right, an assignment  $\mathcal{A}$  of the labels to the vertices such that  $c(\mathcal{A}) = 8/9$ .

This leads us to introduce the following conjecture.

**Conjecture 2.2.**  $\mathbf{NP} \not\subseteq \mathbf{QP}$

Note that this statement is stronger than  $\mathbf{P} \neq \mathbf{NP}$ . Indeed, the correctness of this conjecture implies the correctness of the  $\mathbf{P} \neq \mathbf{NP}$  conjecture, since  $\mathbf{P} \subseteq \mathbf{QP}$ . On the other hand, proving that  $\mathbf{P}$  is not equal to  $\mathbf{NP}$  gives no clue on the correctness of this statement.

This hypothesis is used in Section 6.1.5.

### 2.5.3 Exponential Time Hypothesis

The previously introduced complexity conjecture states that there exist problems of  $\mathbf{NP}$  that cannot be solved in quasi-polynomial time. Impagliazzo et al. [IP01] introduced an even stronger conjecture called the *Exponential Time Hypothesis (ETH)*. This conjecture can be written as follows.

**Conjecture 2.3** ([IP01]).  $\forall k \geq 3$ ,  $k$ -SAT does not have a subexponential algorithm, i.e. an algorithm running in time  $2^{o(n)}$  where  $n$  is the size of the input instance.

The correctness of this conjecture would highlight a problem of  $\mathbf{NP}$  that does not have subexponential algorithm and thus does not have a quasi-polynomial time algorithm, proving at the same time the correctness of Conjecture 2.2. However ETH can be proved to be false without implying anything for Conjecture 2.2. Indeed an algorithm running in time  $2^{n^c}$  with  $c < 1$  is a subexponential algorithm but is not a quasi-polynomial time algorithm.

We can then order the complexity hypothesis used in this manuscript based on their strength:

$$\mathbf{P} \neq \mathbf{NP} \leq \mathbf{NP} \not\subseteq \mathbf{QP} \leq \text{ETH}$$

This hypothesis is used in Sections 6.1.5 and 8.2.3.

# 3

## Parameterized Complexity

---

### Contents

---

<b>3.1</b>	<b>Intuition and definitions</b>	<b>41</b>
<b>3.2</b>	<b>Kernelization</b>	<b>48</b>

---

This section is a light introduction to Fixed Parameter Complexity Theory and some of its common tools that will be used in this manuscript. For a complete overview of this discipline, we refer the user to the reference works of Downey and Fellows [DF99] and Flum and Grohe [FG06].

### 3.1 Intuition and definitions

Before getting into the heart of the matter, let us come back on the sentence we introduced in Section 2.2: “It is very unlikely to find a polynomial algorithm that solves optimally an **NP**-hard optimization problem”. We saw that a first solution to handle these **NP**-hard optimization problems is to design approximation algorithms. The latter allow us to find solutions to these problems at the cost of optimality of these solutions. Another possibility is to make concession on the speed of resolution. Of course, it does not simply consist in designing brute force algorithms but to use structural properties of the problems to design algorithms whose exponential complexity mainly depends on a parameter of the problem that turns out to be small.

A well-known example of this idea is the **VERTEX COVER** problem, formalized in Problem 2. This problem is known to be one of the six **NP**-hard problems introduced by Garey and Johnson [GJ79]. As such, it seems hopeless to find a polynomial-time algorithm solving it optimally. However, given a graph  $G = (U, E)$ , if we note  $k$  the

desired size of the vertex cover of  $G$ , Chen et al. [WLC10] provided an algorithm solving VERTEX COVER in time  $\mathcal{O}(1.2738^k n^{\mathcal{O}(1)})$ , with  $n = |U|$ .

Remark that the naive enumeration algorithm solves the problem in  $\mathcal{O}(n^k)$ . However, even though both of the previous algorithms run in polynomial time when  $k$  is fixed, the algorithm of Chen et al. [WLC10] is of more interest. Indeed, its dependence in  $k$  is separated from its dependence in  $n$ .

---

### Optimization Problem 2 VERTEX COVER

---

<b>Input</b>	A graph $G = (U, E)$ .
<b>Output</b>	A subset of vertices $VC \subset U$ s.t. $\forall \{u, v\} \in E, u \in VC$ or $v \in VC$ .
<b>Objective</b>	Minimize $ VC $ .

---

Another idea behind fixed parameterized complexity theory is to identify which parameters, in this case  $k$ , make the problem hard to solve. Indeed, the existence of an algorithm admitting a time complexity such as the time complexity of the Chen et al. algorithm shows that a slight increase of the value of  $k$  drastically increases the computation time of the algorithm and thus of the problem resolution.

Remark that, given an **NPO** optimization problem, one can imagine a wide range of parameters more or less suitable to analyze it.

We can then define a parameterized problem.

**Definition 3.1** (Parameterized problem [DF13]). *A parameterized problem is a language  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$ . The integer is called the parameter.*

**Notation 3.1.** *Given a parameterized problem  $Q$  and a  $Q$ -instance  $\langle I, k \rangle$ ,  $|\langle I, k \rangle|$  denotes the size of the instance.*

VERTEX COVER parameterized by the size of the solution can be written as follows:

---

### Parameterized Problem 1 VERTEX COVER parameterized by standard parameter

---

<b>Input</b>	A graph $G = (U, E)$ .
<b>Parameter</b>	A positive integer $k$
<b>Question</b>	Does $G$ admits a vertex cover $VC$ of size $ VC  \leq k$ ?

---

Note that, in the latter, the value of the desired solution is called the *standard parameter*.

**Definition 3.2.** *Given a parameterized problem  $Q$  and an algorithm  $\mathcal{A}$ , we say that  $\mathcal{A}$  decides  $Q$  in time  $f(n)$  if and only if for any instance  $\langle I, k \rangle \in \{0, 1\}^* \times \mathbb{N}$ ,  $\mathcal{A}$  determines in time  $f(n)$  whether  $\langle I, k \rangle \in Q$  or not.*

In parameterized complexity theory, the emphasis is on the complexity dependence with regard of a given parameter. This justifies the use of  $\mathcal{O}^*$  to express the complexity of an algorithm. Intuitively, this notation is similar to the  $\mathcal{O}$  notation except that the terms polynomial in the size of the input are also hidden.

Now that parameterized problems are introduced, let us present a first interesting complexity class called **XP** and defined as follows.

**Definition 3.3 (XP class [FG06]).** *A parameterized problem  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$  belongs to **XP** if and only if there exists an XP-algorithm that decides  $Q$ , i.e. an algorithm deciding  $Q$  in time  $\mathcal{O}(f(k)|\langle I, k \rangle|^{g(k)})$ , where  $f$  and  $g$  are two functions computable in polynomial time.*

Intuitively, such an algorithm runs in polynomial time for every fixed  $k$ . Indeed, once  $k$  is fixed,  $f(k)$  and  $g(k)$  turn out to be constants, leading to an algorithm running in time  $\mathcal{O}(\text{poly}(|\langle I, k \rangle|))$ . An interesting consequence of such an algorithm is that, for every fixed  $k$ , the increase of the computation time depending on the instance size is similar to the increase observed for polynomial-time algorithm. Naturally, these algorithms prove to be efficient in practice only for (very) small values of  $k$ .

This motivates the introduction of another kind of algorithms: the FPT-algorithms.

**Definition 3.4 (Fixed Parameter Tractable [FG06]).** *A parameterized problem  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$  is fixed parameter tractable (FPT) if there exists an algorithm that decides  $Q$  in time  $\mathcal{O}(f(k)\text{poly}(|\langle I, k \rangle|))$  where  $f$  is a function computable in polynomial time.*

*Such an algorithm is called an FPT algorithm and is said to run in time FPT with respect to  $k$ .*

Contrary to XP-algorithms that allow the exponent of  $|\langle I, k \rangle|$  to be dependent of  $k$ , FPT-algorithms constrain the exponential dependence in  $k$  within the function  $f$ . This implies an efficient resolution of problems admitting this kind of algorithm for greater values of  $k$ . Obviously one can imagine examples of function  $f$  leading to inefficient algorithm even for small value of  $k$ . However, given a parameterized problem  $Q$ , the existence of an FPT-algorithm for  $Q$  is of great interest. Indeed:

- if  $f$  has small dependence according to  $k$ , then the FPT-algorithm can be an efficient algorithm, even though  $k$  is quite large,
- no matter the behaviour of  $f$  in function of  $k$ , an FPT-algorithm gives strong clues on the parameters that are *responsible* on the difficulty of the considered problem.

Let us make few remarks on the notations that will be used throughout this paper.

**Remark 3.1.** *A decision problem  $\Pi_d$  is said fixed parameter tractable when parameterized by a function  $\kappa : \{0, 1\}^* \rightarrow \mathbb{N}$  (noted  $FPT/\kappa$ ) if its parameterized version  $(\Pi_d, \kappa)$  is FPT.*

**Remark 3.2.** *By abusing the notation and to avoid multiplicity of formalisms, we will say that a decision problem  $\Pi$  is  $FPT/k$  instead of  $FPT/\kappa$  with  $k = \kappa(I)$  for any  $\Pi$ -instance  $I$ .*

**Remark 3.3.** *By misuse of language, an optimization problem  $\Pi = (\mathcal{I}, sol, m, type)$  is said  $FPT$  when parameterized by  $k$  if and only if its decision version is  $FPT$  when parameterized by  $k$ .*

$FPT$  algorithms help us to define a new complexity class, logically called **FPT**.

**Definition 3.5 (FPT class).** *A parameterized problem  $Q \subseteq \{0,1\}^* \times \mathbb{N}$  belongs to **FPT** if and only if it admits an  $FPT$  algorithm.*

For instance, VERTEX COVER when parameterized by the standard parameter is in **FPT**. Indeed, we will see in next section that any instance  $\langle G = (U, E), k \rangle$  of VERTEX COVER can be reduced in polynomial time to an instance of size upper bounded by  $k^2$ . Thus the algorithm consisting in applying this preprocessing and enumerating every subset of the newly created instance runs in time  $\mathcal{O}(2^{k^2} poly(|G|))$ , which clearly is an  $FPT$ -algorithm. VERTEX COVER parameterized by the standard parameter belongs to **FPT**.

From the above definitions, we formulate the following remark.

**Remark 3.4.  $FPT \subseteq XP$ .**

In fact, **FPT**  $\neq$  **XP** unless **P** = **NP**. In the light of section 2.1, natural questions arise:

1. Are there mechanisms, similar to polynomial-time reductions, that preserve membership in **FPT**?
2. Are there problems that are known to be fixed parameter intractable? More broadly, can we define a hierarchy of complexity classes such as the polynomial hierarchy formerly superficially introduced?

The following definition answers positively to the first one.

**Definition 3.6 (Parameterized reduction [FG06]).** *Given two parameterized problems  $Q_1, Q_2 \subseteq \{0,1\}^* \times \mathbb{N}$ , a parameterized reduction is a pair of functions  $(f : \{0,1\}^* \times \mathbb{N} \rightarrow \{0,1\}^* \times \mathbb{N}, g : \mathbb{N} \rightarrow \mathbb{N})$ , such that:*

1.  *$f$  maps any  $Q_1$ -instance  $\langle I, k \rangle$  to a  $Q_2$ -instance  $f(\langle I, k \rangle) = \langle I', k' \rangle$  and is computable in  $FPT$  time with respect to  $k$ ,*
2. *for every  $Q_1$ -instance  $\langle I, k \rangle$ ,  $\langle I, k \rangle$  is a yes instance if and only if  $f(\langle I, k \rangle)$  is a yes-instance.*
3.  *$g$  is computable and is such that for any  $Q_1$ -instance  $\langle I, k \rangle$ ,  $k \leq g(k')$ .*

*An  $FPT$  reduction from  $Q_1$  to  $Q_2$  is noted  $Q_1 \leq_{FPT} Q_2$ .*

The principle behind this reduction is the same as the one behind Turing or Karp reductions. Indeed, given two parameterized problems  $Q_1$  and  $Q_2$ , such that  $Q_1 \leq_{\text{FPT}} Q_2$ , the existence of an FPT algorithm for  $Q_2$  implies an FPT algorithm for  $Q_1$ . Since we want to keep an overall algorithm running in time  $FPT/k$ , the function  $f$  can be computed in time  $FPT/k$ . Furthermore the second constraint, ensuring that  $k$  is bounded by a function of  $k'$ , avoid explosion of  $k$  through the transformation. This leads to the following property.

**Property 3.1** ([FG06]). *The parameterized reduction preserves the membership in FPT.*

Let us focus on the second stated question. As for **NP** next to **P**, there indeed exist problems for which the existence of an FPT algorithm is very unlikely. Let us introduce the notions that will be necessary for the definition of such a problem, and at the same time the prerequisite to introduce the **W**-hierarchy.

In Section 2.1, we introduced the satisfiability problem, SAT. This problem generalizes a certain number of more particular satisfiability problems. These problems are often defined thanks to a determined form applied to clauses. The most famous of this form is probably the Conjunctive Normal Form (CNF), defined as follows:

**Definition 3.7** (Conjunctive Normal Form). *A Boolean formula is in conjunctive normal form if it contains only conjunctions of disjunctions.*

In other words, a logical formula is in CNF if and only if it contains only logical and  $\wedge$ , logical or  $\vee$  and unary negation  $\neg$  (the negation can only be used on a single literal). For instance, when considering three Boolean variables  $x, y, z$ , the formula  $(x \vee y) \wedge (x \vee \neg y \vee z)$  is in CNF while the formula  $(x \vee y) \wedge \neg(x \vee y \vee z)$  is not.

We can now introduce the following problem.

---

**Parameterized Problem 2** WEIGHTED 2-CNF-SATISFIABILITY (W-2-CNF-SAT)

---

<b>Input</b>	A boolean formula $\Phi$ in Conjunctive Normal Form each clause containing at most 2 literals.
<b>Parameter</b>	A positive integer $k$ .
<b>Question</b>	Does an affectation of 0 and 1 to the variables of $\Phi$ exists such that $\Phi = 1$ and such that the number of variables set to 1 is at most equal to $k$ ?

---

Remark that, when there is no bound on the number of variables that can be set to 1, the problem can be solved in polynomial time. However, this version turns out to be **NP**-complete. Indeed, there exists a reduction from VERTEX COVER to W-2-CNF-SAT.

*Proof.* Consider an instance of vertex cover defined by a graph  $G = (U, E)$ . To each vertex  $v \in U$ , we associate the Boolean variable  $x_v$  that will be set to 1 if and only if  $v \in VC$ . To each edge  $(uv) \in E$ , we associate the clause  $x_u \vee x_v$ . The formula  $\Phi$  is defined as:

$$\Phi = \bigwedge_{(uv) \in E} x_u \vee x_v$$

Clearly  $\Phi$  is written in CNF, and each clause contains exactly two literals. The constructed instance is then an instance of W-2-CNF-SAT. Furthermore,  $\Phi$  is satisfied if and only if for each clause at least one of the variables in the clause is set to one (since  $\Phi$  does not contain negation), this implies that for each edge at least one of its end vertices has been selected. By definition, such a set of vertices is a vertex cover. Note also that if  $\Phi$  is satisfied with at most  $k$  variables set to one,  $VC$  has size  $|VC| \leq k$ . Lastly, W-2-CNF-SAT is in **NP**, thus, it is **NP**-complete.  $\square$

This **NP**-complete problem will be used to define a class of likely intractable problems.

**Definition 3.8 (W[1] [DF13]).** *A parameterized problem belongs to **W[1]** if and only if it FPT-reduces to W-2-CNF-SAT.*

Note that the classical definition of this class is based on the structure of the logical circuit used to solve the problems of the class. However such a definition requires the introduction of notions that will be of no interest in this manuscript.

From Definition 3.8, we deduce the following property.

**Property 3.2.** *Every parameterized problem belonging to **FPT** belongs to **W[1]**.*

The proof can be stated as follows. The FPT-reduction can be executed in time FPT in the considered parameter. Thus, given a parameterized problem belonging in **FPT**, the algorithm consisting in solving the problem and returning a trivially yes or no instance of W-2-CNF-SAT in function of the result of the FPT algorithm, is an FPT-reduction.

As for SAT being the first problem proved **NP**-complete, W-2-CNF-SAT is the first problem to be proved **W[1]**-complete.

**Theorem 3.1** (Analog of Cook Theorem [DF99]). *W-2-CNF-SAT is complete for **W[1]**.*

By combining the notion of completeness previously defined and the FPT-reduction that preserves membership in **FPT**, a parameterized problem can be proved **W[1]**-complete by using same mechanisms as those used to prove **NP**-completeness in previous chapter. Indeed, to prove **W[1]**-completeness of a parameterized problem, two things are necessary: the membership of this problem in **W[1]** and an FPT reduction from a **W[1]**-complete problem.

**Corollary 3.1.** *A parameterized problem  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$  is  $\mathbf{W}[1]$ -complete if and only if there exists a FPT-reduction from W-2-CNF-SAT to  $Q$ .*

The reasoning used to define  $\mathbf{W}[1]$  can also be used to define a more general complexity class called  $\mathbf{W}[2]$  defined with use of a generalization of W-2-CNF-SAT. In fact, it is possible to define the whole  $\mathbf{W}$ -hierarchy with use of some particular cases of the parameterized version of SAT. Nevertheless, a complete description of the whole  $\mathbf{W}$ -hierarchy is of no interest within the framework of this thesis. For such a characterization of this hierarchy we refer the reader to the reference work of Flum and Grohe [FG06].

Let us characterize  $\mathbf{W}[2]$ .

---

**Parameterized Problem 3** WEIGHTED CNF-SATISFIABILITY (W-CNF-SAT)

---

<b>Input</b>	A boolean formula $\Phi$ in Conjunctive Normal Form.
<b>Parameter</b>	A positive integer $k$ .
<b>Question</b>	Does an affectation of 0 and 1 to the variables of $\Phi$ exists such that $\Phi = 1$ and such that the number of variables set to 1 is at most equal to $k$ ?

---

**Definition 3.9** ( $\mathbf{W}[2]$  [DF13]). *A parameterized problem belongs to  $\mathbf{W}[2]$  if and only if it FPT-reduces to W-CNF-SAT.*

W-2-CNF-SAT being a particular case of W-CNF-SAT, every parameterized problem that FPT-reduces to W-2-CNF-SAT, also FPT-reduces to W-CNF-SAT. It follows that:

$$\mathbf{FPT} \subseteq \mathbf{W}[1] \subseteq \mathbf{W}[2]$$

This inclusion are supposed to be strict unless  $\mathbf{P} = \mathbf{NP}$ . As for other complexity classes, a parameterized problem  $Q = (\Pi, k)$  is said  $\mathbf{W}[2]$ -hard if and only if every problem of  $\mathbf{W}[2]$  FPT-reduces to  $Q$ , and  $\mathbf{W}[2]$ -complete if and only if it is  $\mathbf{W}[2]$ -hard and it belongs to  $\mathbf{W}[2]$ .

Finally, we superficially introduce the top-level complexity class of the  $\mathbf{W}$ -hierarchy, allowing us to give an large overview of this hierarchy without describing intermediate classes. This class relies on the parameterized version of SAT.



---

**Parameterized Problem 4** WEIGHTED SATISFIABILITY (W-SAT)
 

---

<b>Input</b>	A boolean formula $\Phi$ .
<b>Parameter</b>	A positive integer $k$ .
<b>Question</b>	Does an affectation of 0 and 1 to the variables of $\Phi$ exists such that $\Phi = 1$ and such that the number of variables set to 1 is at most equal to $k$ ?

---

W-SAT defines the class  $\mathbf{W}[SAT]$  and as a generalization of W-CNF-SAT and thus W-2-CNF-SAT we can write:

$$\mathbf{FPT} \subseteq \mathbf{W}[1] \subseteq \mathbf{W}[2] \subseteq \dots \subseteq \mathbf{W}[SAT]$$

As previously, the inclusion is supposed to be strict. Furthermore, given an instance of W-SAT defined by a formula  $\Phi$  and an integer  $k$ , it is possible to determine whether  $\Phi$  can be satisfied with at most  $k$  variables set to 1 in time  $\mathcal{O}(n^k)$  where  $n$  is the number of variables. This can be done by enumerating every  $k$ -tuples of variables and verifying whether  $\Phi$  is satisfied. This implies that W-SAT is polynomial when  $k$  is fixed, and thus:

$$\mathbf{FPT} \subset \mathbf{W}[1] \subset \mathbf{W}[2] \subset \dots \subset \mathbf{W}[SAT] \subset \mathbf{XP}$$

## 3.2 Kernelization

Now that basics of fixed parameterized theory have been introduced, we present few techniques developed and formalized that became folklore of the FPT community. The first presented technique is quite intuitive but nevertheless powerful. It is called *kernelization* and can be formalized as follows.

**Definition 3.10** (Kernelization [FG06]). *Given a parameterized problem  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$ , a function  $K : \{0, 1\}^* \times \mathbb{N} \rightarrow \{0, 1\}^* \times \mathbb{N}$  is a kernelization if there exists a computable function  $h : \mathbb{N} \rightarrow \mathbb{N}$  such that for any instance  $\langle I, k \rangle$  of  $Q$ ,  $\langle I, k \rangle$  is a yes-instance if and only if  $K(\langle I, k \rangle)$  is a yes-instance and such that  $|K(\langle I, k \rangle)| \leq h(k)$ .*

*The image  $K(\langle I, k \rangle)$  of an instance  $\langle I, k \rangle$  is called the kernel and  $h$  is called the size of the kernel.*

A kernelization can be seen as a preprocessing algorithm applying on input instance a certain number of rules. In general these rules reduce the input instance by identifying structures that can be proved to be part of the solution, or by deleting others than can be proved to not belong to the solution.

In previous section, we mention a transformation that can be applied on any instance of VERTEX COVER parameterized by the standard parameter  $k$  bounding the

size of the new instance by  $\mathcal{O}(k^2)$ . Such a reduction is, in fact, a kernelization. Let us explicit it.

Consider an instance  $\langle G = (U, E), l \rangle$  of VERTEX COVER parameterized by standard parameter. We define the following rules:

1. If  $k \geq 0$  and there exists a vertex  $v \in U$  of degree greater than  $k$ , thus  $v$  is part of the solution. Indeed, since every edge has to be covered by the vertex cover  $VC$ , if  $v \notin VC$ , thus  $VC$  contains every neighbours of  $v$  and has size greater than  $k$ . Hence, we add  $v$  to  $VC$  and decrease  $k$  by one.
2. If  $v$  is an isolated vertex,  $v$  can be removed since it has no adjacent edges.
3. If there exists an edge  $\{u, v\} \in E$  such that one of the end points, let us say  $u$  *w.l.o.g.*, has degree equal to one thus we can add  $v$  to  $VC$ . Indeed, if  $u$  is part of the solution, then we can select  $v$  instead without increasing the size of  $VC$ . Since  $u$  is only adjacent to  $\{u, v\}$ , thus  $VC$  is still a vertex cover. We can then add  $v$  to  $VC$  remove every adjacent edges and decrease  $k$  by one.

Once this rules can not be applied anymore, we claim that if  $\langle G = (U, E), k \rangle$  is a yes-instance thus the constructed instance  $\langle G' = (U', E'), k' \rangle$  has at most  $k'^2$  edges and  $2k'^2$  vertices. If computed instance contains more than  $k^2$  edges,  $\langle G', k' \rangle$  (and thus  $\langle G, k \rangle$ ) is a no instance. Indeed, each vertex in  $U'$  has at most  $k'$  neighbours and can then cover at most  $k'$  edges, it follows that no more than  $k'^2$  edges can be covered by a set of  $k'$  vertices. Since  $k' \leq k$ , this algorithm is a kernelization.

Note that every fixed parameter tractable parameterized problem admits a kernelization depicted on Algorithm 1.

---

**Algorithm 1:** Kernelization from FPT-algorithm

---

**Data:** An instance  $\langle I, k \rangle$  of a parameterized problem  $Q$ , an FPT-algorithm  $\mathcal{A}$  for  $Q$  running in time  $\mathcal{O}(f(k) \cdot |I|^c)$

**Result:** A kernel of  $\langle I, k \rangle$

```

if  $|I| \leq f(k)$  then
    return  $\langle I, k \rangle$ ;
Solve  $\langle I, k \rangle$  with algorithm  $\mathcal{A}$ ;
if  $\langle I, k \rangle$  is a yes-instance then
    return  $T$  a trivial yes instance of size  $|T| \leq f(k)$ ;
else
    return  $F$  a trivial no instance of size  $|F| \leq f(k)$ ;

```

---

**Lemma 3.1** ([FG06]). *The Algorithm 1 is a kernelization for every parameterized problem belonging to **FPT**.*

*Proof.* Consider a parameterized problem  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$  admitting an FPT-algorithm  $\mathcal{A}$ . Let us first remark that by construction, given an instance  $\langle I, k \rangle$ , the instance computed by Algorithm 1 is a kernel:

1. the instance is a yes instance if and only if  $\langle I, k \rangle$  is a yes instance,
2. its size is upper bound by  $f(k)$ .

Furthermore, Algorithm 1 runs in polynomial time. Indeed, if  $|\langle I, k \rangle| > f(k)$ ,  $\mathcal{A}$  runs in time  $\mathcal{O}(f(k) \cdot |\langle I, k \rangle|^c) = \mathcal{O}(|\langle I, k \rangle|^{c+1})$ , with  $c$  being a positive constant.

Thus Algorithm 1 is a kernelization.  $\square$

We use Lemma 3.1 to prove Theorem 3.2.

**Theorem 3.2** ([FG06]). *A parameterized problem  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$  is FPT if and only if it admits a kernelization.*

*Proof.* The existence of a kernelization for an FPT parameterized problem is ensured by Lemma 3.1.

Let us consider a parameterized problem  $Q$  and a kernelization  $K$ . By kernelization definition, for every instance  $\langle I, k \rangle$  of  $Q$ ,  $|K(\langle I, k \rangle)| \leq h(k)$  and  $K(\langle I, k \rangle)$  is a yes instance if and only if  $\langle I, k \rangle$  is a yes instance. Thus an exponential exact algorithm on  $K(\langle I, k \rangle)$  runs in time  $\mathcal{O}(f(K(\langle I, k \rangle))) \leq \mathcal{O}(f(h(k)))$  and thus any exact algorithm is an FPT-algorithm for  $Q$ .  $\square$

Remark that we can distinguish two kinds of kernels: polynomial and non polynomial ones. Given a parameterized problem  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$ , a polynomial kernel for  $Q$  is a kernel with size in  $\mathcal{O}(\text{poly}(k))$ . Theorem 3.2 proves that,  $Q$  is fixed parameter tractable if and only if it admits a kernel. No indication is given on the size of such a kernel. It seems natural to ask whether every FPT problem admits a polynomial kernel. The framework introduced hereafter aims at answering negatively at this question.

**Definition 3.11** (Polynomial equivalence relation according to [BJK14]). *An equivalence relation  $\mathcal{R}$  on  $\{0, 1\}^*$  is called a polynomial equivalence relation if the two following conditions hold:*

- *There exists an algorithm that given two strings  $I_1, I_2 \in \{0, 1\}^*$ , decides whether  $I_1$  and  $I_2$  belong to the same equivalence class in  $(|I_1| + |I_2|)^{O(1)}$  time.*
- *For any finite set  $S \subseteq \{0, 1\}^*$ , the equivalence relation  $\mathcal{R}$  partitions the elements of  $S$  into at most  $(\max_{I_1 \in S} |I_1|)^{O(1)}$  classes.*

**Definition 3.12** (OR-cross-composition according to [BJK14]). *Let  $\Pi_1 \subseteq \{0, 1\}^*$  be a decision problem and let  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$  be a parameterized problem. We say that  $\Pi_1$  OR-cross-composes into  $Q$  if there exists a polynomial equivalence relation  $\mathcal{R}$  and an algorithm which, given  $t$  strings belonging to the same equivalence class of  $\mathcal{R}$ , computes an instance  $\langle I^*, k^* \rangle \in \{0, 1\}^* \times \mathbb{N}$  in time polynomial in  $\sum_{i=1}^t |I_i|$  such that:*

- $\langle I^*, k^* \rangle$  is a yes instance  $\Leftrightarrow I_i \in D_{\Pi_1}^+$  for some  $i \in \{1, \dots, t\}$
- $k^*$  is bounded by a polynomial in  $\max_{i=1}^t |I_i| + \log t$

**Theorem 3.3** ([BJK14]). *If some decision problem  $\Pi_1 \subseteq \{0, 1\}^*$  is **NP**-hard and  $\Pi_1$  OR-cross-composes into a parameterized problem  $Q \subseteq \{0, 1\}^* \times \mathbb{N}$ , then there is no polynomial kernel for  $Q$  unless  $\mathbf{NP} \subseteq \mathbf{coNP}/\mathbf{poly}$ .*

A similar cross-composition, called the AND-cross-composition can be defined in the same way as the OR-cross-composition except that an instance  $\langle I^*, k^* \rangle$  is a yes instance  $\Leftrightarrow I_i \in D_{\Pi_1}^+$  for every  $i \in \{1, \dots, t\}$ . The Theorem 3.3 has been later extended to AND-cross-composition by Drucker et al. in [Dru15].

An example of usage of this framework can be found in Section 8.2.5.

We refer the reader to a book of Cygan et al. [Cyg+15] offering an intuitive but nevertheless complete overview of the main lines and details of the kernel lower bounds theory. Let us however formulate some remarks on the intuition behind such a result.

Note that the existence of a cross composition from a decision problem  $\Pi_1$  to a parameterized problem  $Q$  allows us to *store* a possibly huge amount of informations (contained in the equivalent instances of  $\Pi_1$ ) into a single instance of  $Q$ . Note that this instance can be huge, but the AND-cross-composition ensures that the value the parameter is bounded by a function that does not depend much on the number of  $\Pi_1$ -instances. The intuition of Theorem 3.3, is that if  $\Pi_1$  is a hard problem, it seems highly implausible that so much informations can be stored in a small amount of space.

But what does “highly implausible” stand for? The hypothesis  $\mathbf{NP} \not\subseteq \mathbf{coNP}/\mathbf{poly}$  can be seen as a strengthened version of  $\mathbf{NP} \neq \mathbf{coNP}$ , where  $\mathbf{coNP}$  is the set of decidable problems such that negative instances admit a polynomial certificate. In other words, for a decision problem in  $\mathbf{coNP}$ , every negative instance can be verified in polynomial time. It follows that, assuming  $\mathbf{NP} \not\subseteq \mathbf{coNP}/\mathbf{poly}$ , remains to assume that it does not exist a general algorithm that solves every problem in  $\mathbf{NP}$  and every problem in  $\mathbf{coNP}$ , even if this algorithm has access to a polynomial-size advice.

## Conclusion

We end this introductory part on this notion of cross-composition. In the following we enter the heart of the matter: the study of *wafer-to-wafer integration* problems. This study will be performed along two main lines. On a first side, we focus on what we call the *theoretical aspects* of the problem. We aim at *classifying* the problems and some of their variants. In other words, we try to determine, in function of parameters, the complexity of the problems and their main variants. This part is called *theoretical* since it aims at giving clues on the underlying complexity of the problems. However, as we will see in the following chapters, we highlight some positive results (membership of some variants in  $\mathbf{P}$ , subexponential of FPT-algorithms, ...) that can hardly be used in practice.

On the other side, we focus on what we call the *practical aspects* of the problem. This part aims at tackling *real world* instances. We present several resolution techniques based on approximation algorithms, Integer Linear Programming formulations or Constraints Programming formulations. We also try to highlight the advantages and the drawbacks of each method and try to compare them when possible. However,

the main work of this part, is the generation of a benchmark and computational results of the techniques when applied to the benchmark.

— PART II —  
**THEORETICAL  
ASPECTS**



# 4

## Preliminaries

---

### Contents

---

<b>4.1</b>	<b>Modelling</b> . . . . .	<b>55</b>
<b>4.2</b>	<b>Observations</b> . . . . .	<b>61</b>

---

The previous part briefly introduced the complexity framework we will use hereafter. It also informally introduced the context of this thesis and thus the considered problems. However, before trying to take advantage of the presented tools, we need to modelize the problems. This is the main goal of this first chapter. It also aims at introducing notations, definitions and assumptions that will be used throughout this manuscript.

### 4.1 Modelling

In this section, we present the framework used to modelize and study the *wafer-to-wafer integration problems* presented in the introduction. We consider several variants of these problems presenting either a practical or theoretical interest. All these variants share the same input. Let us define the way we modelize it.

As presented in the introduction, we consider  $m$  sets each containing  $n$  wafers. On each wafer,  $p$  dies are engraved. Due to engraving techniques, some of the dies are faulty. The position of the faulty dies on the wafers is known. An example of input is given on Figure 4.1.

All the wafers of the instance contain exactly the same number of dies. We can define an order on the latter. A wafer can thus be represented by a  $p$ -dimensional binary vector such that the  $l^{th}$  coordinate is set to zero if and only if the  $l^{th}$  die in the predefined order is a faulty one. The  $l^{th}$  coordinate is set to one otherwise. The



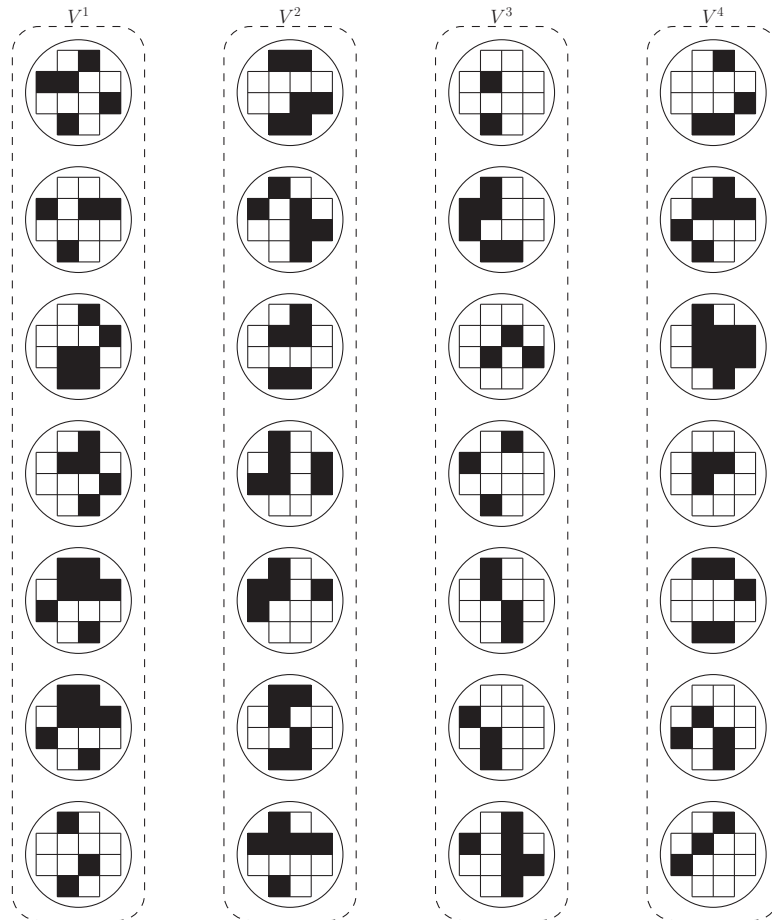


Figure 4.1: Example of input of a wafer-to-wafer integration problems, with  $m = 4$ ,  $n = 7$  and  $p = 12$ . The faulty dies are represented by black squares while viable ones are represented by white squares.

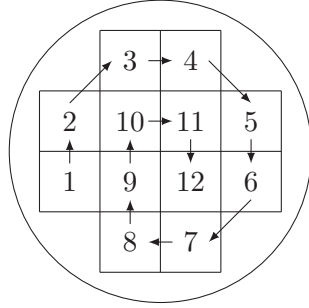


Figure 4.2: Example of order on the dies being used in to modelize wafers in the practical study of wafer-to-wafer integration problems.

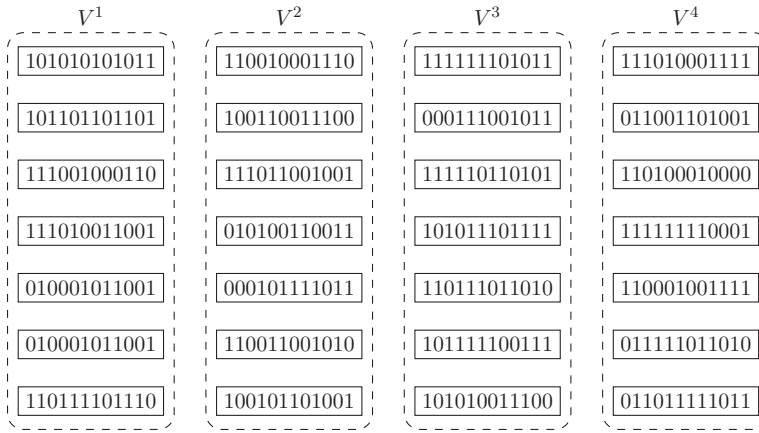


Figure 4.3: Wafer-to-wafer integration problem input of Figure 4.1 represented with binary vectors.

Figure 4.2 presents a possible order on the dies. Figure 4.3 depicts the representation with binary vectors of input represented in Figure 4.1.

It follows that an input of wafer-to-wafer integration problems can be modeled by  $m$  multisets  $V^1, \dots, V^m$ . Each multiset  $V^i$  contains  $n$  vectors  $v_1^i, \dots, v_n^i$  being  $p$ -dimensional. Every vector  $v_j^i$  is such that its component  $v_j^i[l] = 0$  if and only if the  $l^{th}$  die on its corresponding wafer is a faulty one.

Note that we use multisets modelize the sets of wafers since two different wafers of a same set can share the same representative vector as depicted on Figures 4.1 and 4.3 (fifth and sixth vector of  $V^1$ ).

The objective of these problems is to create  $n$  stacks containing exactly one wafer of each set. A solution  $S$  is therefore a set of  $n$  stacks  $S = \{s_1, s_2, \dots, s_n\}$  such that every wafer of the input appears in exactly one created stack. Remember that, given a stack, a die at position  $l$  is considered as non faulty if and only if all the dies at position  $l$  are non faulty in every wafer composing the stack.

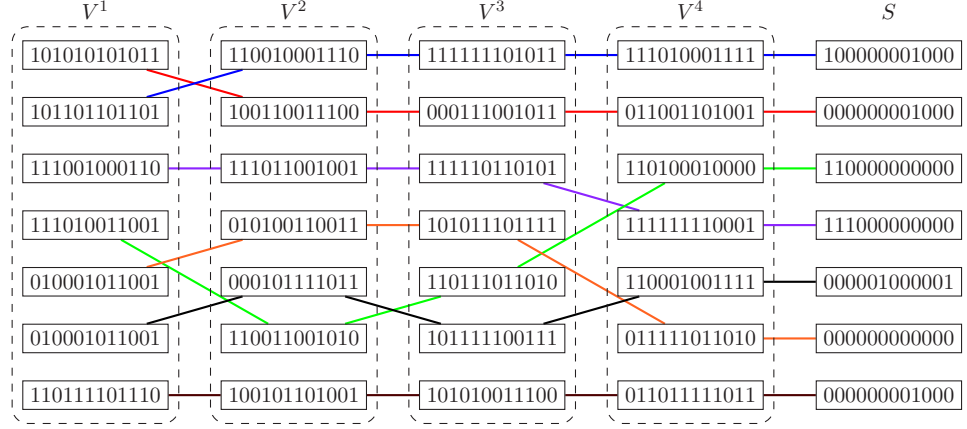


Figure 4.4: Example of solution  $S$  for the instance of wafer-to-wafer integration problem depicted on Figure 4.3.

Hence, we consider a stack  $s$  as an  $m$ -tuple of vectors  $(v_1^s, \dots, v_m^s)$  such that  $v_i^s \in V^i$ . A stack  $s$  can then be represented by a  $p$ -dimensional binary vector  $v_s$  computed by performing the bitwise AND between every vector composing the stack. An example of such an output is depicted on Figure 4.4. It follows that:

$$v_s[l] = \bigwedge_{i=1}^m v_i^s[l]$$

These problem input and output will be common to every wafer-to-wafer integration problem that we will consider in this thesis. Therefore we define the following generic problem called BINARY MULTIDIMENSIONAL VECTOR ASSIGNMENT.

---

**Problem 1** BINARY MULTIDIMENSIONAL VECTOR ASSIGNMENT (BMVA)

---

**Input**  $m$  multisets of  $n$   $p$ -dimensional binary vectors.

**Output** A set  $S$  of  $n$  disjoint stacks.

---

We now introduce different ways to evaluate a solution  $S$ . Each objective function defines a distinct problem. Let us first introduce both problems presented in introduction. In the first of them, the objective is to maximize the overall number of non faulty dies. In other words, the objective is to maximize the number of ones in the vectors representing the stacks. The profit of a solution  $S$  is given by  $c_{\max \sum 1}(S) = \sum_{s \in S} c(v_s)$  where  $c(v_s) = \sum_{l=1}^p v_s[l]$ , leading to the following definition of  $\max \sum 1$ -BMVA:

The Chapter 7 is entirely dedicated to its theoretical study.

---

**Optimization Problem 3**  $\max \sum 1$ -BMVA ( $\max \sum 1$ )

---

<b>Input</b>	BMVA input.
<b>Output</b>	BMVA output.
<b>Objective</b>	Maximize $c_{\max \sum 1}(S) = \sum_{s \in S} c(v_s)$ , the overall number of ones.

---

The second problem considered aims at minimizing the overall number of zeros. We refer at it as  $\min \sum 0$ -BMVA. The associated cost function of a solution  $S$  is described by:  $c_{\min \sum 0}(S) = \sum_{s \in S} p - c(v_s)$ . This problem is the dual version of  $\max \sum 1$  and can be formally defined as follows:

---

**Optimization Problem 4**  $\min \sum 0$ -BMVA ( $\min \sum 0$ )

---

<b>Input</b>	BMVA input.
<b>Output</b>	BMVA output.
<b>Objective</b>	Minimize $c_{\min \sum 0}(S) = \sum_{s \in S} (p - c(v_s)) = np - c_{\max \sum 1}(S)$ , the overall number of zeros.

---

The complexity study of this problem is handled in Chapter 8.

Both of these problems present obvious practical interest as being possible mod-elizations of IC manufacturing issues and are the main considered problem in this manuscript. Other problems present a more theoretical interest since they do not have straightforward applications but the study of their complexity gives several clues on the complexity of  $\max \sum 1$  and  $\min \sum 0$  as we will see in Chapters 7 and 8.

The first one, denoted as  $\max_{\neq 0}$ -BMVA, aims at maximizing the number of stacks represented by a vector with at least one coordinate set to one.

---

**Optimization Problem 5**  $\max_{\neq 0}$ -BMVA ( $\max_{\neq 0}$ )

---

<b>Input</b>	BMVA input.
<b>Output</b>	BMVA output.
<b>Objective</b>	Maximize $c_{\max_{\neq 0}}(S) =  \{s \in S : c(v_s) > 0\} $ , the number of non zero stacks.

---

The second is denoted  $\max \max 1\text{-BMVA}$ . In this problem, given an input of wafer-to-wafer integration problems and a solution  $S$ , the objective is to maximize the profit of the best stack in  $S$ :

---

**Optimization Problem 6**  $\max \max 1\text{-BMVA}$  ( $\max \max 1$ )

---

<b>Input</b>	BMVA input.
<b>Output</b>	BMVA output.
<b>Objective</b>	Maximize $c_{\max \max 1}(S) = \max_{s \in S} c(v_s)$ , the profit of the maximum stack.

---

We will also consider its dual version,  $\min \min 0\text{-BMVA}$  in which the objective is to minimize the cost of the stack containing the minimum number of zeros.

---

**Optimization Problem 7**  $\min \min 0\text{-BMVA}$  ( $\min \min 0$ )

---

<b>Input</b>	BMVA input.
<b>Output</b>	BMVA output.
<b>Objective</b>	Minimize $c_{\min \min 0}(S) = \min_{s \in S} p - c(v_s)$ , the cost of the minimum stack.

---

All of these three problems are tackled in Chapter 6.

In the following, we shall study the influence of the number of zeros on the complexity of the problems. We will then take into consideration special cases of the previously presented problems where the number of coordinates set to zero in any vector will be bounded by a constant  $c$ . These problems will be noted  $(\cdot)_{\#0 \leq c}$ , thus  $\min \sum 0$  restricted to instances where each input vector has at most one component set to zero will be denoted as  $(\min \sum 0)_{\#0 \leq 1}$ .

For ease of reading, whenever the context is unambiguous, the notation  $c(S)$  will be used instead of  $c_{\max \sum 1}(S)$ ,  $c_{\min \sum 0}(S)$ ,  $c_{\max \neq 0}(S)$ ,  $c_{\max \max 1}(S)$  or  $c_{\min \min 0}(S)$ . Furthermore, instances of  $\max \sum 1$ ,  $\min \sum 0$ ,  $\max \max 1$ ,  $\min \min 0$  and  $\max \neq 0$  will be denoted by  $I[m, n, p]$ .

## 4.2 Observations

This section aims at listing the observations and assumptions made on input instances of the considered problems. These observations and assumptions are considered to be true throughout the whole manuscript unless explicitly specified.

**Hypothesis 4.1.** *We suppose that for all set  $V^i$ , there always exist  $j \in [n]$ ,  $l \in [p]$  such that  $v_j^i[l] = 0$ .*

In other words, we consider instances that do not contain a set consisting only in *perfect vectors*, *i.e.* vectors with no coordinate set to zero. Given a BMVA-instance, if it contains at least one of those *perfect* sets, the latter can be removed without altering the cost of the optimal solution. Indeed, perfect vectors can be seen as neutral element for the bitwise AND operation.

The same hypothesis can be done for the *zero sets*.

**Hypothesis 4.2.** *We suppose that for all set  $V^i$ , there always exist  $j \in [n]$ ,  $l \in [p]$  such that  $v_j^i[l] = 1$ .*

In other words, we consider instances with no set containing only *zero vectors*, *i.e.* vectors with all its coordinates set to zero. Such a set in an BMVA-instance would *nullify* every stack representative vector, since a zero vector can be seen as an absorbing element for the bitwise AND operation. It follows that any assignment is an optimal solution and thus that such BMVA-instances can be solved in polynomial time.

Based on the same remark, we can define a similar hypothesis at the coordinate level.

**Hypothesis 4.3.** *We suppose that for all set  $V^i$  and for every coordinate  $l \in [p]$ , there always exists  $j \in [n]$  such that  $v_j^i[l] = 1$ .*

Let us consider an instance that violates Hypothesis 4.3 and thus such that given a set  $V^i$  and a coordinate  $l \in [p]$ ,  $v_j^i[l] = 0$ ,  $\forall j \in [n]$ . In such a configuration, any solution  $S$  satisfies  $v_s[l] = 0$  for every stack  $s \in S$ . The coordinate  $l$  can thus be removed from every vector of the instance. Note that this can lead to an alteration of the solution cost depending on the objective function, and thus legitimates questions about approximability results for these problems<sup>1</sup>. However, these problems are minimization problems. Hence any approximation algorithm for BMVA-instances satisfying 4.3 would provide an approximation algorithm with at least the same ratio for BMVA-instances containing one or more zero coordinates. It is sufficient to delete the zero-coordinates, applying the algorithm on the created instance, and adding back the deleted coordinates. When adding back the zero-coordinates, the cost of the returned solution and of the optimal one are increased by the same amount and the performance ratio of the returned solution is improved.

A last hypothesis can be done at the coordinate level.

---

<sup>1</sup>Incriminated problems are  $\min \min 0$  and  $\min \sum 0$ .

**Hypothesis 4.4.** *We suppose that for every coordinate  $l \in [p]$ , there always exist  $i \in [m]$  and  $j \in [n]$  such that  $v_j^i[l] = 0$ .*

Roughly speaking, for any coordinate  $l \in [p]$ , there exists at least one vector in the whole instance that has the  $l^{\text{th}}$  coordinate set to zero. Otherwise, any solution  $S$  would satisfy  $v_s[l] = 1$  for every stack  $s \in S$ . Such coordinates can then be removed from the instance. We use same argument, but for maximization problems, to show that such an operation is safe from the approximability point of view.

# 5

## Related Work

---

### Contents

---

5.1	MULTI-DIMENSIONAL ASSIGNMENT . . . . .	64
5.2	Maximizing the die yield . . . . .	66
5.3	Minimizing the faults . . . . .	69

---

As pointed out by their names, these problems have strong relationships with assignment problems. In fact, these are particular cases of the AXIAL MULTI-INDEX ASSIGNMENT (AXIAL MIA for short) also known as MULTI-DIMENSIONAL ASSIGNMENT (or shortly MDA). In this problem we are given  $m$  sets  $A_1, A_2, \dots, A_m$  of  $n$  elements each. For each  $m$ -tuple  $t \in A_1 \times \dots \times A_m$ , a cost  $c_t$  is known. The problem is now to find exactly  $n$   $m$ -tuples such that each element of  $\bigcup_{i \in [m]} A_i$  is in exactly one  $m$ -tuple. This problem can be formally defined as follows:

---

### Optimization Problem 8 MULTI-DIMENSIONAL ASSIGNMENT (MDA)

---

<b>Input</b>	$m$ sets $A_1, \dots, A_m$ such that $ A_1  = \dots =  A_m  = n$ and a cost (or a profit) function $c : A_1 \times \dots \times A_m \rightarrow \mathbb{N}$ .
<b>Output</b>	$n$ disjoint $m$ -tuples.
<b>Objective</b>	Minimize the total cost (or maximize the total profit) of the $m$ -tuples.

---



## 5.1 MULTI-DIMENSIONAL ASSIGNMENT

Problem 8 has been first introduced by Pierskalla [Pie68] in 1968 as an extension of the well-known linear assignment problems. This paper mainly focus on the THREE-DIMENSIONAL ASSIGNMENT (denoted as 3DA) being the particular case of MDA with  $m = 3$ . The author provides an integer linear programming formulation depicted hereafter. Note that it can be easily generalized for greater values of  $m$ .

---

### Integer Linear Program I Generic formulation for 3DA

---

$$\text{Minimize } \sum_{i \in [n]} \sum_{j \in [n]} \sum_{l \in [n]} c_{ijl} x_{ijl} \quad (\text{I.1})$$

$$\text{Subject To } \sum_{i \in [n]} \sum_{j \in [n]} x_{ijl} = 1 \quad \forall l = 1, \dots, n \quad (\text{I.2})$$

$$\sum_{i \in [n]} \sum_{l \in [n]} x_{ijl} = 1 \quad \forall j = 1, \dots, n \quad (\text{I.3})$$

$$\sum_{j \in [n]} \sum_{l \in [n]} x_{ijl} = 1 \quad \forall i = 1, \dots, n \quad (\text{I.4})$$

$$x_{ijl} \in \{0, 1\} \quad \forall i, j, l \quad (\text{I.5})$$


---

Unlike the classical two dimensional assignment problems, MDA turns out to be one of the 21 **NP**-hard problems introduced by Karp [Kar72] even for  $m = 3$ . In fact, Crama et al. proved in [CS92] that 3DA cannot be approximated within a constant ratio unless  $\mathbf{P} = \mathbf{NP}$ .

On the other hand, when restricted to some particular objective functions, the problem admits approximation algorithms with constant ratio. In [CS92], Crama et al. investigated 3DA with triangle inequalities. This problem is based on the graph theoretic version of 3DA, formalized hereafter.

---

### Optimization Problem 9 MULTI-DIMENSIONAL ASSIGNMENT (graph version)

---

**Input** A tripartite complete graph  $G = (I, J, L)$  such that  $|I| = |J| = |L| = n$  and a cost (or profit) function  $c$  that maps to any triangle ( $i \in I, j \in J, l \in L$ ) a cost  $c(i, j, l)$ .

**Output**  $n$  disjoint triangles.

**Objective** Minimize the total cost (or maximize the total profit) of the triangles.

---

Instead of defining arbitrary costs for each triangle, the authors consider a distance  $d(i, j)$  (resp.  $d(j, l)$ ,  $d(i, l)$ ) for every edge  $\{i, j\}$  (resp.  $\{j, l\}$ ,  $\{i, l\}$ ) and they assume that any triangle  $(i, j, l)$  is such that the triangle inequality is satisfied. Based on this model, they define two cost functions. The first cost function  $c_1$  is defined as the sum of the distance, in other words:

$$c_1(i, j, l) = d(i, j) + d(i, l) + d(j, l)$$

The second cost function  $c_2$  is defined as:

$$c_2(i, j, l) = d(i, j) + d(i, l) + d(j, l) - \max\{d(i, j), d(i, l), d(j, l)\}$$

Roughly speaking  $c_2$  is the sum of the two shortest distances in the considered triangle. Even though this model is **NP**-hard, authors of [CS92] provides a  $3/2$ -approximation algorithm (resp.  $4/3$ -approximation algorithm) for the first (resp. the second) model.

This article paved the way toward similar costs models: *decomposable costs* based models. We say that a cost model is based on decomposable costs if, given a cost function  $c$  and values  $d_{u,v}$  associated to each edge  $\{u, v\}$ ,  $c$  is such that the cost of a triangle depend on the  $d$ 's associated to the edges of the triangles. The notion of decomposable costs can be extended to cost model based on integers associated to vertices instead of edges.

One can for instance cite the work of Burkard et al. [BRW96] where integers are associated to vertices and where  $c_{i,j,l} = d_i \times d_j \times d_l$ . The authors prove that, with the  $d$  values being non-negative real numbers, the problem is **NP**-hard for minimization but is in **P** when considering the maximization version.

When considering values of  $m > 3$ , we can cite the work of Bandelt et al. [BCS94]. In this article, authors focus on MDA with decomposable costs. Given a  $m$ -partite complete graph, the notion of decomposable costs is extended in the intuitive way: the cost function  $c$  is based on decomposable costs if the cost of a clique depends on the edge related coefficients. They consider several cost functions:

- the sum of the lengths of the edges,
- the minimum length of a spanning star,
- the traveling salesman tour,
- the minimum length of a spanning tree.

For each of these cost functions, they provide heuristics and show that according to some parameters<sup>1</sup>, these heuristics are approximation algorithms.

We can remark that the problems we consider have model costs based on decomposable costs. Indeed, given a vector  $v$  in the complete  $m$ -partite graph representing an instance of  $\max \sum 1$  (resp.  $\min \sum 0$ ),  $v$  can be seen as the binary representation of

<sup>1</sup>Including what they call the distance to triangle inequality which, given a clique, gives a measure of how far is this clique from satisfying triangle inequality.

an integer (resp. as the one's complement of the binary representation of an integer) that we call  $d_v$ . The profit (resp. the cost) of an  $m$ -tuple is defined as the Hamming weight of the vector resulting in the bitwise AND of the binary representations of vectors in the  $m$ -tuple.

Other works have been performed on 3DA and more generally on MDA but their scope tends to be far from what we will consider in this manuscript. For instance, works have been done on the polyhedral aspects or about the asymptotic behaviour of the problem. We refer the keen reader to surveys of Burkard et al. [BÇ99] and of Spieksma [Spi00] and to the excellent book of Burkard et al. [BDM09].

## 5.2 Maximizing the die yield

One of the problems we consider,  $\max \sum 1$ -BMVA, has been first introduced by Reda et al. in [RSS09] as the FUNCTIONAL YIELD MAXIMIZATION problem. Authors of this paper point out that the problem is **NP**-hard. They indeed mention that the classical **NP**-hard problem 3-DM reduces to the FUNCTIONAL YIELD MAXIMIZATION problem. They also provide two heuristics based respectively on greedy and iterative matchings strategies.

Given an instance of the FUNCTIONAL YIELD MAXIMIZATION problem with  $m$  sets of  $n$  wafers. The greedy heuristic consists in computing every  $n^m$  possible stacks and constructing a feasible solution by selecting first the stacks with the best yield. Note that this heuristic runs in polynomial time only when  $m$  is fixed.

The iterative matching based heuristic proceeds as follows: it maintains a set of partial stacks, that we will call the *hub set*. At each iteration, the optimal matching between the hub set and every unprocessed set is computed. The heuristic selects the unprocessed set that provides the maximum profit matching, marks this set as processed and update the wafers of the hub set according to the optimal matching.

Authors of [RSS09] also consider the classical ILP formulation depicted in ILP (I). They provide computational results based on the comparison of their proposed resolution techniques with a random assignment strategy. These computational results will be discussed in the Chapter 9.1 in which we present a state of the art of the practical aspects of the problem.

To our knowledge, only exact resolution of the problem has been studied. It has been broached in two articles of Dokka et al. [Dok+12; DCS14], both dealing with its dual version  $\min \sum 0$ -BMVA. In these articles, the authors provide two alternative ILP formulations depicted in Formulations (II) and (III).

The idea of the Formulation (II) is to consider the set  $V^1$  as a hub set on which vectors of the other set will be *plugged*. The formulation is based on  $z_{u,v}$  variables with  $u \in V^1$  and  $v \in \bigcup_{i=2}^m$ . A variable  $z_{u,v}$  is set to one, if vector  $v$  is assigned to vector  $u$ . Other variables are introduced to compute the profit of an assignment. These are the  $y_{u,l}$  variables with  $u \in V^1$  and  $l \in [p]$ . A  $y_{u,l}$  variable is set to one if and only if the  $l^{th}$  coordinate of the vector representing the stack containing  $u$  is set to one. The objective is then to maximize the sum over the  $y_{u,l}$  variables.

---

**Integer Linear Program II** Alternative ILP formulation for  $\max \sum 1$ 


---

$$\text{Maximize } \sum_{u \in V^1} \sum_{l \in [p]} y_{u,l} \quad (\text{II.1})$$

$$\text{Subject To } \sum_{u \in V^1} z_{u,v} = 1 \quad \forall v \in \bigcup_{i=2}^m V^i \quad (\text{II.2})$$

$$\sum_{v \in \bigcup_{i=2}^m V^i} z_{u,v} = 1 \quad \forall u \in V^1 \quad (\text{II.3})$$

$$y_{u,l} \leq \min(u[l], v[l]) \cdot z_{u,v} \quad \forall l \in [p], u \in V^1, v \in \bigcup_{i=2}^m V^i \quad (\text{II.4})$$

$$z_{u,v} \in \{0, 1\} \quad \forall u \in V^1, v \in \bigcup_{i=2}^m V^i \quad (\text{II.5})$$


---

The Formulation (III) has been introduced to prove that  $\min \sum 0$  (and thus  $\max \sum 1$ ) can be solved in polynomial time when  $p$  is fixed. In this formulation, they consider the type of an  $m$ -tuple, defined as follows: given an  $m$ -tuple  $s$  and its representative vector  $v_s$ , the type  $t$  of  $s$  is the integer binary representation  $b_t$  equal to  $v_s$ . Thus given an instance with vectors of size  $p$  we can count  $2^p$  different types each type  $t$  having a cost  $c(b_t)$ .

The ILP formulation aims at determining the number  $x_t$  of  $m$ -tuples of type  $t$ . They also introduce  $z_{jt}^i$  variables equal to 1 if and only if the vector  $v_j^i$  is assigned to an  $m$ -tuple of type  $t$ . The binary representation  $b_t$  of an integer  $t$  being said greater than a vector  $v$  if every coordinate set to zero in  $v$  is a zero coordinate in  $b_t$ , the Equations III.2 ensure that every vector is assigned to a *compatible*  $m$ -tuple while the Equations III.3 ensure that every vector is assigned to exactly one  $m$ -tuple. This leads to a correct formulation.

Furthermore, remark that only  $x_t$  variables are integer variables (the integrality of  $z_{jt}^i$  variables being a consequence of the integrality of  $x_t$  variables) and that there are exactly  $2^p$  of them. Each of the  $x_t$  variables can take at most  $n + 1$  distinct values. The authors prove that to find an optimal solution, it is enough to check the feasibility of ILP constraints for the  $\mathcal{O}(n^{2^p})$  possible values for the  $x_t$  variables, leading to an  $XP$ -algorithm for  $\max \sum 1$  when parameterized by  $p$ .

This ends the presentation of existing works for  $\max \sum 1$ . In this manuscript, we focus on the approximability of the problem with regard to its three natural parameters  $m$ ,  $n$  and  $p$ . Several approximation preserving reductions are provided in Section 7.1 leading to inapproximability results for  $\max \sum 1$ . A cartography of these reductions is depicted on the Figure 5.1.

We further the study of the exact resolution of the problem with regard to pa-

---

**Integer Linear Program III** XP ILP formulation for  $\max \sum 1$ 


---

$$\text{Maximize } \sum_{t=0}^{2^p-1} c(b_t)x_t \quad (\text{III.1})$$

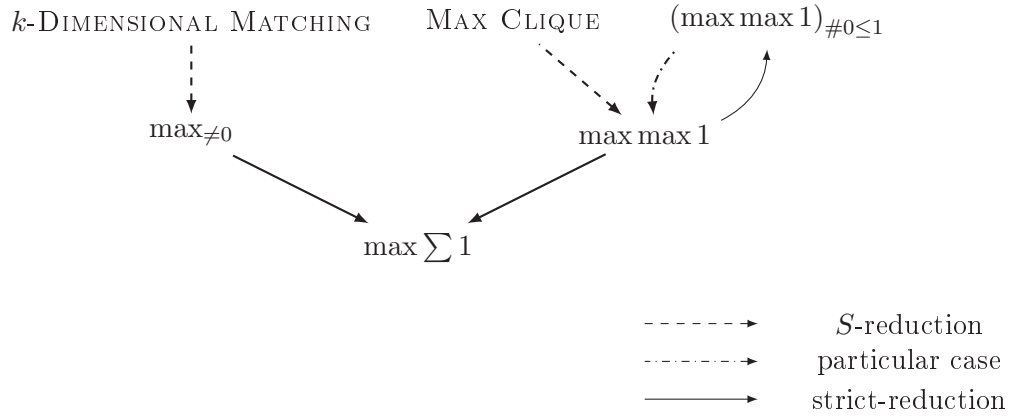
$$\text{Subject To } \sum_{t:b_t \geq v_j^i} z_{jt}^i = x_t \quad \forall t \in \{0, \dots, 2^p-1\}, i \in [m] \quad (\text{III.2})$$

$$\sum_{j:b_t \geq v_j^i} z_{jt}^i = 1 \quad \forall j \in [n], i \in [m] \quad (\text{III.3})$$

$$x_t \text{ integer} \quad \forall t = 0, \dots, 2^p - 1 \quad (\text{III.4})$$

$$z_{jt}^i \leq 0 \quad \forall t = 0, \dots, 2^p - 1, j \in [n], i \in [m] \quad (\text{III.5})$$


---

Figure 5.1: Cartography of polynomial reductions for  $\max \sum 1$ 

parameter  $p$ . Based on a similar approach, we improve, in Section 7.2.2 the result of Dokka et al. [DCS14] by proving the problem membership in **FPT** when parameterized by  $p$ . However we give, in Section 7.2.3, strong evidences on the non existence of a polynomial-size kernel for the latter. Then, in Section 7.3, we focus on sparse instances, *i.e.* instances with vectors having a bounded number of coordinates set to zero. We give in Section 7.3.1 an EPTAS for  $(\max \sum 1)_{\#0 \leq r}$ , with  $r$  being a constant, based on FPT resolution of  $\max \sum 1$  and a polynomial-time algorithm for  $(\max \sum 1)_{\#0 \leq 1}$  when  $m$  is fixed in Section 7.3.2.

### 5.3 Minimizing the faults

The dual version of  $\max \sum 1, \min \sum 0$ -BMVA, has been first introduced by Dokka et al. [Dok+12] as a natural way to tackle the  $\max \sum 1$  problem. The related work is limited to the two previously cited articles [Dok+12] and [DCS14]. In both of them, authors focus on approximability and exact resolution of the problem.

As mentioned earlier, authors of [Dok+12] first provide two ILP formulations, the first one being the application of the classical MULTI-DIMENSIONAL ASSIGNMENT problem formulation depicted by the Formulation (I) when  $m = 3$  and the second being described by Formulation (II). However the main contribution of the paper is the design of an approximation algorithm for  $\min \sum 0$  when  $m = 3$ . The authors propose a heuristic based on matchings that processes as follows. Given an instance of  $\min \sum 0$  with  $m = 3$ , the optimal assignment between vectors of  $V^1$  and vectors of  $V^2$  is computed. The resulting matching being called  $M$ , the algorithm solves optimally another assignment problem between resulting vectors of  $M$  and vectors of  $V^3$ . They prove that this heuristic, called the *sequential* heuristic, is a  $3/2$ -approximation algorithm that can be improved to a ratio  $4/3$  if the sets are sorted by decreasing costs, *i.e.* by decreasing overall number of zeros in the vectors of the set. The heuristic performing the sort of the sets and the sequential heuristic on sorted set is called *heaviest first* heuristic.

These heuristics are generalized to arbitrary  $m$ <sup>2</sup> in [DCS14]. They prove that, if we consider other costs functions  $c$ , then subadditivity and monotonicity of the cost function  $c$  is a necessary condition for the *heaviest set* and *sequential* heuristics to be approximation algorithm. On the other hand, if  $c$  turns out to be monotone and subadditive then every heuristic is an  $m$ -approximation algorithm. Finally, if  $c$  appears to be submodular, hence the *sequential* heuristic is a  $m/2$ -approximation algorithm while the *heaviest first* heuristic is a  $(\frac{1}{2}(m+1) - \frac{1}{4} \ln(m-1))$ -approximation algorithm.

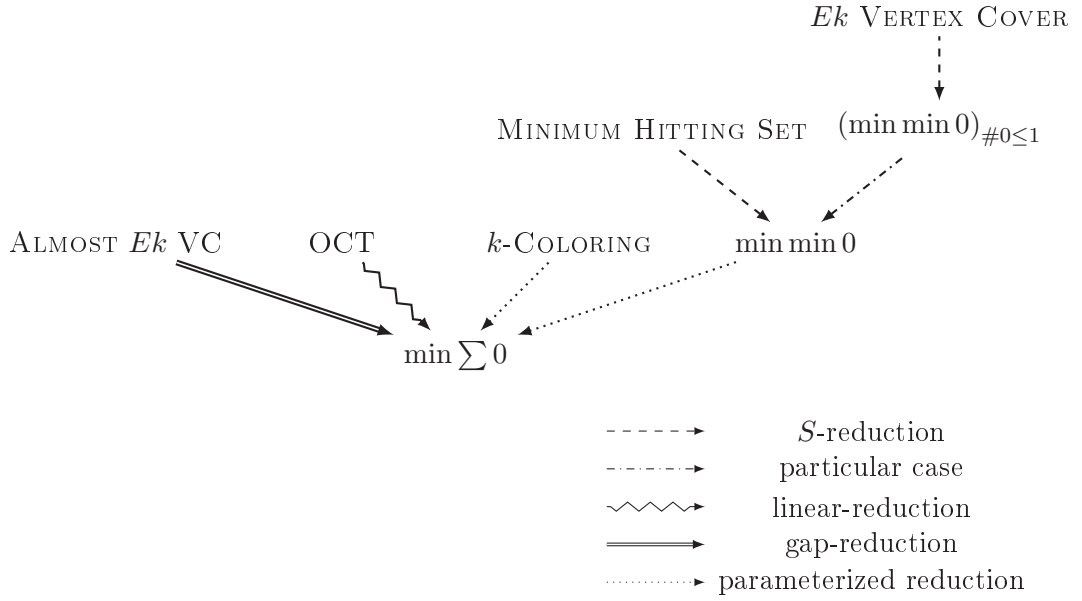
Concerning the negative results, they provide a reduction from a particular case of 3-DIMENSIONAL MATCHING called 3-BOUNDED MAXIMUM DIMENSIONAL MATCHING where each element appears in at most three triples. They show that the existence of a *PTAS* for  $(\min \sum 0)_{\#0 < 2}$  where  $m = 3$ , would imply the existence of a *PTAS* for MAX-3DM-3 contradicting a result of Kann [Kan91].

In this manuscript we further the investigation of the approximability of the problem by providing  $(n - \varepsilon)$ -inapproximability result for  $(\min \sum 0)_{\#0 < 1}$  under UGC in Section 8.1.1. We also complete the result of Dokka et al. [DCS14] by proving, in Section 8.1.2, that  $(\min \sum 0)_{\#0 < 1}$  is **APX**-hard even for  $n = 2$  unless **P** = **NP**. A cartography of used reductions is given in Figure 5.2

We initiate the parameterized analysis of  $\min \sum 0$  as well. First, Section 8.2.1 the standard parameter is considered. We prove that the problem is in **FPT** by providing a  $\mathcal{O}(k^2m)$  kernel. We also consider two *above guarantee parameters*. Roughly speaking,

---

<sup>2</sup>In fact they generalize these heuristics to a more general problem called MULTIDIMENSIONAL VECTOR ASSIGNMENT problem in which the vectors are not required to be binary vectors.

Figure 5.2: Cartography of polynomial reductions for  $\min \Sigma 0$ 

these are equal to the gap between a known lower bound and the value of the standard parameter. The first bound we consider is the overall number of coordinates set to zero in the vectors of the heaviest set, denoted  $\beta$ . It leads to the introduction of the parameter  $\zeta_\beta = k - \beta$ . We show that  $\min \Sigma 0$  is in **FPT** when parameterized by  $\zeta_\beta$  and  $n$  in Section 8.2.2. On the other hand we show, in Section 8.2.3, that when parameterized only by  $\zeta_\beta$ , the problem is **W[2]**-hard, and that the existence of a subexponential algorithm with dependency only in  $\zeta_\beta$  and  $n$  contradicts ETH.

The second considered bound is  $p$ , leading to the introduction of  $\zeta_p = k - p$ . In Section 8.2.4, we show that  $\min \Sigma 0$  where  $n = 2$  is **FPT** when parameterized by  $\zeta_p$ . Finally, in Section 8.2.5, we show that for any fixed  $n \geq 3$ ,  $\min \Sigma 0$  with fixed  $n$  is not even in **XP**.

---

# 6

---

## Intermediate problems

---

---

### Contents

---

<b>6.1</b>	<b>Negative Results</b> . . . . .	<b>72</b>
6.1.1	$\mathcal{O}\left(\frac{m}{\log(m)}\right)$ -inapproximability for $\max_{\neq 0}$ . . . . .	73
6.1.2	$f(n)$ , $p^{1-\varepsilon}$ and $m^{1-\varepsilon}$ -inapproximability for $\max \max 1$ . . . . .	74
6.1.3	Inapproximability extension to $(\max \max 1)_{\#0 \leq 1}$ . . . . .	76
6.1.4	$(n - 1 - \varepsilon)$ -inapproximability for $(\min \min 0)_{\#0 \leq 1}$ . . . . .	78
6.1.5	$(1 - \varepsilon) \log(m)$ -inapproximability for $\min \min 0$ . . . . .	81
<b>6.2</b>	<b>Positive Results</b> . . . . .	<b>84</b>
<b>6.3</b>	<b>Conclusion and open questions</b> . . . . .	<b>85</b>

---

As mentioned earlier, the study of the *wafer-to-wafer* integration problems is performed in two steps. In a first time, the focus is made on some intermediate problems providing a convenient framework to encode several graphs problem. This chapter is devoted to present the results concerning the latter:  $\max_{\neq 0}$ ,  $\max \max 1$  and its dual version  $\min \min 0$ . All of these problems present theoretical interest. Indeed, there exist several reductions giving strong negative results for them. However, their main purpose in this thesis is to provide negative results for  $\max \sum 1$  in Chapter 7 and  $\min \sum 0$  in Chapter 8.

In a first time we prove negative results on these problems and in a second time we show that using a reduction to INDEPENDENT SET,  $\max \max 1$  and  $\min \min 0$  can be solved quite efficiently when  $n = 2$  and when  $p$  is small.



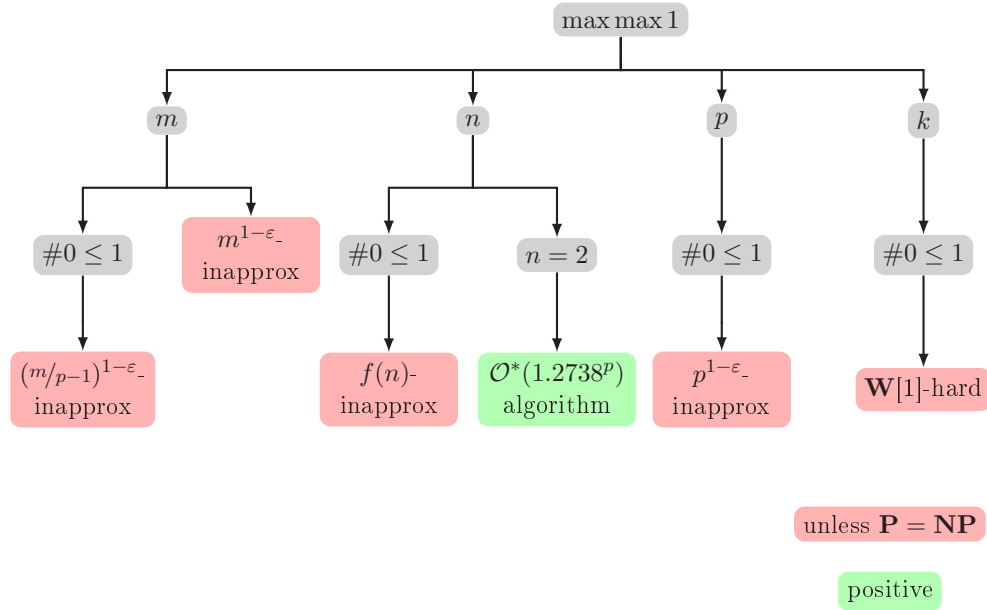


Figure 6.1: Overview of the results for max max 1

## 6.1 Negative Results

This section can be divided into three main work leads. In a first time, we focus on the  $\max_{\neq 0}$  problem. The reduction presented in the first section is an implicit reduction from  $k$ -DIMENSIONAL MATCHING first mentioned in [RSS09]. We make this reduction explicit and prove, in Section 6.1.1, that it happens to be an  $S$ -reduction proving  $\mathcal{O}\left(\frac{m}{\log(m)}\right)$ -inapproximability for  $\max_{\neq 0}$ .

In a second time we focus on the complexity of max max 1. A first  $S$ -reduction from MAX CLIQUE is depicted in Section 6.1.2 which proves  $p^{1-\varepsilon}$ ,  $m^{1-\varepsilon}$  and  $f(n)$ -inapproximability for any polynomial-time computable function  $f$  under the classical complexity hypothesis  $\mathbf{P} \neq \mathbf{NP}$ . This reduction also proves the  $\mathbf{W}[1]$ -hardness of max max 1 when parameterized by the standard parameter  $k$ . Then, in Section 6.1.3, we extend these negative results to  $(\max \max 1)_{\#0 \leq 1}$  by providing a strict reduction from max max 1 defined on instances with fixed  $n$  to  $(\max \max 1)_{\#0 \leq 1}$ . A graphical overview of these results is given on Figure 6.1

In a third and last time, we focus on min min 0. We highlight in Section 6.1.4 an  $S$ -reduction from  $Ek$  VERTEX COVER to min min 0 proving  $(n-1-\varepsilon)$ -inapproximability unless  $\mathbf{P} = \mathbf{NP}$  but also  $(n-\varepsilon)$ -inapproximability under UGC. At last, in Section 6.1.5 we provide another  $S$ -reduction from MINIMUM HITTING SET proving the  $\mathbf{W}[2]$ -hardness of min min 0 when parameterized by  $k$ , but also proving that the latter does not admit algorithm with subexponential dependency in  $k$  unless ETH fails. A graphical overview of these results is given on Figure 6.2.

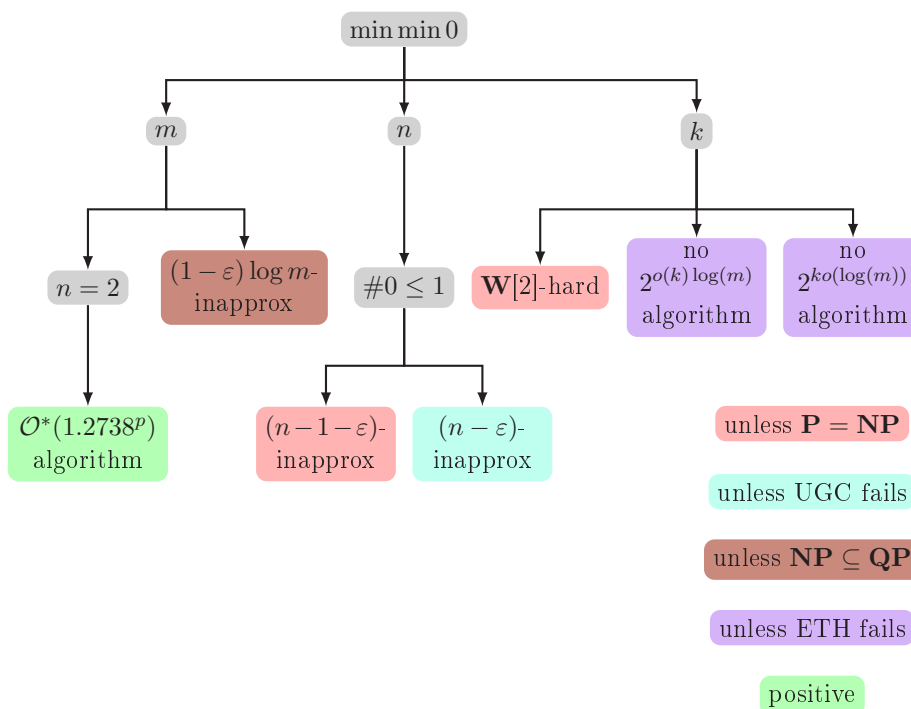


Figure 6.2: Overview of the results for min min 0

### 6.1.1 $\mathcal{O}\left(\frac{m}{\log(m)}\right)$ -inapproximability for $\max_{\neq 0}$

As stated in introduction, we aim at making explicit a reduction first mentioned in [Dok+12]. However, we also highlight that this reduction is an  $S$ -reduction. It provides therefore negative results for  $\max_{\neq 0}$ .

**Lemma 6.1** (Implicit in [Dok+12]).  $k$ -DIMENSIONAL MATCHING  $S$ -reduces to  $\max_{\neq 0}$ .

*Proof.* Let us highlight function  $f$  and  $g$  as depicted in Definition 2.26. We consider an instance  $I$  of  $k$ -DIMENSIONAL MATCHING described by  $k$  pairwise disjoint sets  $X_i$ ,  $i \in [k]$  such that  $|X_i| = n$ , and a set  $T$  of  $x$  distinct  $k$ -tuples  $t_l \in X_1 \times \dots \times X_k$ ,  $l \in [x]$ . We denote by  $a_j^i$ ,  $j \in [n]$  the elements of set  $X_i$ .

From this instance, we construct an instance  $f(I)$  of  $\max_{\neq 0}$ . To each set  $X_i$ , we associate a set  $V^i$  containing  $n$  vectors. To each element  $a_j^i \in X_i$  of  $X_i$  we associate an  $x$ -dimensional vector  $v_j^i$  such that  $v_j^i[l] = 1$  if and only if  $a_j^i \in t_l$ .

Let us remark that given a solution  $S$  of  $f(I)$ , a stack  $s \in S$  has cost  $c(s) = 1$  if and only if it represents a tuple in  $T$ . Indeed,  $v_s[l] = 1$  if and only if  $v_i^s[l] = 1$ ,  $\forall v_i^s \in s$ . Thus  $s$  contains only vectors representing a vertex belonging to  $t_l$  by construction.

Remark also that there does not exist stack with cost at least 2 as it would imply that two coordinates represents a same set.

Based on these remarks, we define the function  $g$  as the function that, given a solution  $S$  of  $f(I)$ , selects the tuples  $t \in T$  represented by non zero stacks in  $S$ .

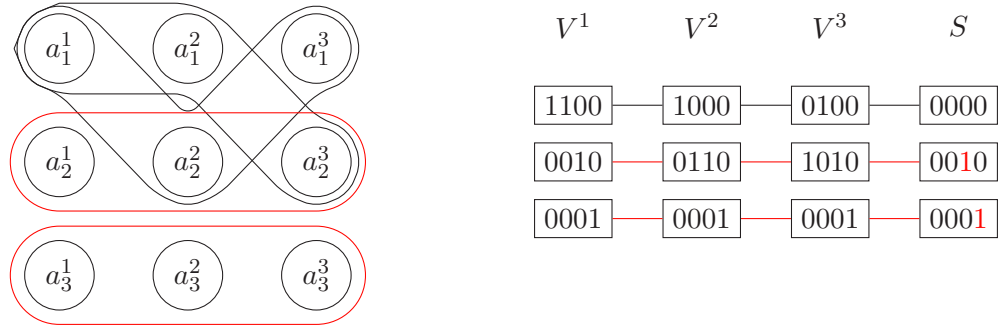


Figure 6.3: Example of reduction from an instance  $I$  of  $k$ -DIMENSIONAL MATCHING with  $k = 3$ ,  $X_1 = \{a_1^1, a_2^1, a_3^1\}$ ,  $X_2 = \{a_1^2, a_2^2, a_3^2\}$ ,  $X_3 = \{a_1^3, a_2^3, a_3^3\}$ ,  $T = \{(a_1^1, a_2^1, a_3^1), (a_1^1, a_2^2, a_3^2), (a_1^2, a_2^2, a_3^2), (a_1^2, a_2^3, a_3^3), (a_1^3, a_2^3, a_3^3)\}$  and a solution of profit 2 to an instance of  $\max_{\neq 0}$ ,  $m = k = 3$ ,  $n = |X_1| = 4$ ,  $p = |T| = 4$  and a solution  $S$  of profit 2.

Note that these tuples are disjoint, since stacks are disjoint. It follows that for any solution  $S$  of  $f(I)$  of cost  $c(S)$  can be turned into a solution  $g(I, S)$  of  $k$ -DIMENSIONAL MATCHING of size  $c(S)$ .

An example of the reduction is depicted on Figure 6.3. □

As it is **NP**-hard to approximate  $k$ -DIMENSIONAL MATCHING within a factor  $\mathcal{O}(\frac{k}{\log(k)})$  [HSS03], and as the previous reduction satisfies  $k = m$ , we get the following corollary:

**Theorem 6.1.** *It is **NP**-hard to approximate  $\max_{\neq 0}$  within a factor  $\mathcal{O}(\frac{m}{\log(m)})$ .*

### 6.1.2 $f(n)$ , $p^{1-\varepsilon}$ and $m^{1-\varepsilon}$ -inapproximability for $\max \max 1$

We consider the MAX CLIQUE problem. In this problem, given a graph  $G = (U, E)$ , the objective is to find the maximum clique<sup>1</sup> of  $G$ .

---

#### Optimization Problem 10 MAX CLIQUE

---

**Input** A graph  $G = (U, E)$ .

**Output** A clique  $\mathcal{C} \subseteq U$ .

**Objective** Maximize  $|\mathcal{C}|$ .

---

**Lemma 6.2.** MAX CLIQUE  $S$ -reduces to  $\max \max 1$  with  $n = 2$ .

<sup>1</sup>We refer the reader to the Chapter 1 for an overview of the graph notions used in this manuscript.

*Proof.* Let us construct  $(f, g)$  as in Definition 2.26. Let us consider an instance  $G = (U, E)$  of MAX CLIQUE. The corresponding instance of  $\max \max 1$  is constructed as follows. We consider  $m = |U|$  sets, each having two vectors. All the vectors have  $p = |U|$  coordinates. For each vertex  $i$  of  $U$ , we create the set  $V^i = (v_1^i, v_2^i)$ . For any  $i$ , we define  $v_1^i$  as follows. The  $l^{\text{th}}$  coordinate  $v_1^i[l]$  is set to one if and only if  $\{i, l\} \in E$  or  $l = i$ . Otherwise  $v_1^i[l]$  is set to zero. On the other side,  $v_2^i[l]$  is set to zero if and only if  $l = i$ . Every other coordinate is set to one. In other words,  $v_1^i$  corresponds to the  $i^{\text{th}}$  row of the adjacency matrix of  $G$ , including self loop.

The idea is that selecting  $v_1^i$  corresponds to selecting vertex  $i$  in graph, and selecting  $v_2^i$  will turn the  $i^{\text{th}}$  coordinate to zero, which corresponds to a penalty for not choosing vertex  $i$ .

We first need to state an intermediate lemma. For any stack  $s = \{v_1^s, \dots, v_m^s\}$ , let  $X_s = \{i | v_i^s = v_1^i\}$  be the associated set of vertices in  $G$ . Recall that  $v_s$  is the  $p$  dimensional vector representing  $s$ .

**Lemma 6.3.**  $\forall i \in [p], v_s[i] = 1 \Leftrightarrow ((i \in X_s) \text{ and } (\forall x \in X_s \setminus i, \{x, i\} \in E))$ .

$\Delta$  Let us first prove Lemma 6.3. Suppose that the  $i^{\text{th}}$  coordinate of  $v_s$  is equal to one. This implies that  $v_1^i \in s$ , and thus  $i \in X_s$ . Now, suppose by contradiction that  $\exists x \in X_s \setminus i$  such that  $\{x, i\} \notin E$ .  $x \in X_s$  implies that  $v_x^x \in s$ . Moreover,  $v_s[i] = 1$  implies that  $v_x^x[i] = 1$ , and thus  $\{x, i\} \in E$ , which leads to a contradiction.

Suppose now that  $i \in X_s$ , and  $\forall x \in X_s \setminus i, \{x, i\} \in E$ . Let us prove that  $\forall i', v_{i'}^s[i] = 1$ . Notice first that for  $i' = i$  we have  $v_i^s[i] = v_1^i[i] = 1$ . Moreover,  $\forall i' \neq i$  such that  $i' \notin X_s$  we have  $v_{i'}^s[i] = v_2^{i'}[i] = 1$ .

Finally,  $\forall i' \neq i$  such that  $i' \in X_s$ , we have  $v_{i'}^s[i] = v_1^{i'}[i] = 1$  as  $\{i', i\} \in E$ .  $\Delta$

It is now straightforward to prove that  $\forall x \in \mathbb{N}$ , “ $\exists$  solution  $S$  for  $\max \max 1$  of value  $c(S) = x$  if and only if a clique  $X$  of size  $x$  exists in  $G$ ”. Indeed, suppose first that we have a solution  $S$  such that  $c(S) = x$ . Let  $s = \{v_1^s, \dots, v_m^s\}$  be the stack in  $S$  of value  $x$ , and let  $G_s = \{l | v_s[l] = 1\}$  be the set of coordinates set to one in  $s$ . We immediately get that the vertices corresponding to  $G_s$  define a clique in  $G$ , as  $\forall i$  and  $j \in G_s$  the previous property implies that  $i \in X_s, j \in X_s$ , and thus  $\{i, j\} \in E$ .

Suppose now that there is a clique  $X^*$  in  $G$ , and let  $s$  be a stack such that  $X_s = X^*$ . The previous property implies that  $\forall i \in X_s, v_s[i] = 1$ .

Thus, the profit of  $\text{opt}_{\max \max 1}(f(I))$  is equal to the size of the maximum clique in  $G$ . As the previous reduction is polynomial, and as any solution  $S$  of  $f(I)$  can be translated back in polynomial time into a corresponding clique of size  $c(S)$  in  $G$ , we get the desired result.

An example of this reduction is depicted on Figure 6.4

□

The MAX CLIQUE problem is known to be hard to approximate.

**Theorem 6.2** (Zuckerman [Zuc07]). MAX CLIQUE defined on a graph  $G = (U, E)$  is hard to approximate within a ratio  $|U|^{1-\varepsilon}$  unless  $\mathbf{P} = \mathbf{NP}$ .

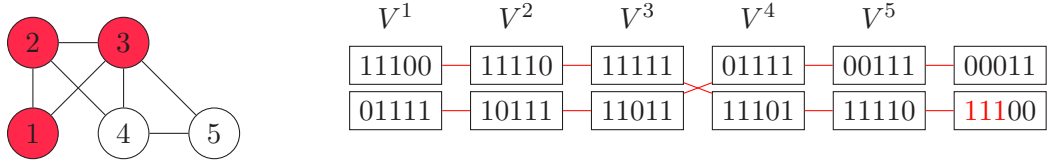


Figure 6.4: Illustration of the reduction from an instance of MAX CLIQUE defined by graph  $G = (\{1, 2, 3, 4, 5\}, \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)\})$  admitting a solution of profit 3 to an instance of max max 1 with  $m = p = |V| = 5$ ,  $n = 2$  admitting a solution  $S$  of cost  $c(S) = 3$ .

It follows that max max 1, is also hard to approximate, since in the reduction  $m = p = |U|$ .

**Theorem 6.3.** *Even for  $n = 2$ , for any  $\epsilon > 0$ , there is no  $\rho(m, p)$ -approximation such that  $\rho(x, x) = x^{1-\epsilon}$  for max max 1.*

Notice that in particular,  $p^{1-\epsilon}$  and  $m^{1-\epsilon}$  are not possible, but for example  $(pm)^{\frac{1}{2}}$  is not excluded. Furthermore, this reduction maps any instance of MAX CLIQUE to an instance of max max 1 where  $n = 2$ . It follows that any  $f(n)$ -approximation algorithm for max max 1 would provide an algorithm with ratio better than  $|U|^{1-\epsilon}$  for sufficiently large instances of MAX CLIQUE. We can hence also deduce inapproximability of the problem with regard to  $n$ .

**Theorem 6.4.** *There is no  $f(n)$ -approximation algorithm for max max 1, unless  $\mathbf{P} = \mathbf{NP}$ .*

Note that the cost of optimal solutions is preserved via the  $S$ -reduction, it follows that this reduction can be seen as an FPT-reduction from MAX CLIQUE when parameterized by the standard parameter to max max 1 when parameterized by the standard parameter.

**Theorem 6.5** (Downey and Fellows [DF95]). *MAX CLIQUE parameterized by the standard parameter is  $\mathbf{W}[1]$ -hard.*

**Theorem 6.6.** *max max 1 parameterized by the standard parameter is  $\mathbf{W}[1]$ -hard.*

### 6.1.3 Inapproximability extension to $(\max \max 1)_{\#0 \leq 1}$

We can extend the previous results to  $(\max \max 1)_{\#0 \leq 1}$ . Indeed, we design in this section a strict-reduction from max max 1 when  $n$  is fixed to  $(\max \max 1)_{\#0 \leq 1}$ . Since the previous reduction proves inapproximability for max max 1 with  $n = 2$ , these results can then be extended to  $(\max \max 1)_{\#0 \leq 1}$ .

**Lemma 6.4.** *max max 1 on instances with fixed  $n$  strict-reduces to  $(\max \max 1)_{\#0 \leq 1}$ .*

*Proof.* Let us construct  $(f, g)$  as defined in Definition 2.27. We consider an instance  $I$  of  $\max \max 1$  with fixed  $n$ . Note that any vector  $v$  of the instance can be seen as a set of positive integers representing the coordinates set to zero in  $v$ . For instance the vector 01101 can be seen as the set  $\{1, 4\}$ . Based on this representation we can then define the Cartesian product of two vectors as the Cartesian product of their representative sets. Furthermore, given a set  $V^i = \{v_1^i, \dots, v_n^i\}$ , we can define the  $n$ -ary Cartesian product of the vectors of  $V^i$  as  $\Pi_{V^i} = v_1^i \times \dots \times v_n^i$ . We now define  $f$  as follows. For every set  $V^i$  of  $I$ , we first compute the Cartesian product of the vectors of  $V^i$ . To each  $n$ -tuple  $\tau_i = (t_1, \dots, t_n)$  of  $\Pi_{V^i}$  we construct a set  $V^{\tau_i}$  in  $f(I)$ . To each integer  $t_j$  of  $\tau_i$  we associate a vector  $v_j^{\tau_i}$  having coordinate  $t_j$  set to zero. In other words  $v_j^{\tau_i}[l] = 0$  if  $l = t_j$ ,  $v_j^{\tau_i}[l] = 1$  otherwise. Intuitively, the transformation splits every vector into vectors containing only one zero. The Cartesian product aims at penalizing every stacks in  $f(I)$  that would *mixes* vectors coming from different vectors of  $I$ . The Figure 6.5 gives an example of this transformation.

Before defining the function  $g$ , we formulate the following lemma:

**Lemma 6.5.** *Given a solution  $S$  of  $f(I)$ , if we call  $s_{\max}$  the stack such that  $c(v_{s_{\max}}) = c(S)$ , thus for every set  $V^i$  of  $I$  there exists a vector  $v_j^i$  such that  $\forall l \in [p]$ ,  $v_j^i[l] = 0 \Rightarrow v_{s_{\max}}[l] = 0$ .*

$\Delta$  Let us reason by contradiction. Suppose that there exists a set  $V^i$  of  $I$  such that  $\forall v_j^i, \exists l_j$  such that  $v_j^i[l_j] = 0$  and  $v_{s_{\max}}[l_j] = 1$ . By definition of the Cartesian product of the vectors of the set  $V^i$ ,  $\Pi_{V^i}$  contains the tuple  $(l_1, \dots, l_m)$ , the tuple of each position of each vector not covered by  $v_{s_{\max}}$ . Hence, by construction, there exists a set  $V^{(l_1, \dots, l_n)_i}$ . It follows that  $s_{\max}$  contains one of the vectors of  $V^{(l_1, \dots, l_n)_i}$  and thus that its representative vector has one of the *forbidden* component set to zero. This leads to a contradiction.  $\Delta$

We define the function  $g$  as follows. Given a solution  $S$  of  $f(I)$  and  $s_{\max}$  the stack such that  $c(v_{s_{\max}}) = c(S)$ , for every set  $V^i$  of  $I$  we add to a stack a vector  $v_j^i$  of  $V^i$  that satisfies the Lemma 6.5. The remaining stacks of  $g(I, S)$  are greedily created.

It is straightforward to see that given a solution  $S$  of  $f(I)$  of cost  $c(S)$ ,  $g(I, S)$  contains a stack  $s$  of cost  $c(s) \leq c(v_{s_{\max}})$  and thus  $c(g(I, S)) \leq c(S)$ . Indeed, each of the selected vectors have their zeros at the same coordinates as  $v_{s_{\max}}$ .

On the other hand, if  $I$  admits a solution  $S$  of cost  $c(S)$ , thus selecting the vectors in  $f(I)$  corresponding to the vectors contained in the best stack in  $S$  leads to a stack  $s$ , and thus a solution for  $f(I)$ , that has the same cost as  $S$ .

The reduction is thus a strict reduction.  $\square$

**Theorem 6.7.** *For every  $\varepsilon > 0$ ,  $(\max \max 1)_{\#0 \leq 1}$  is hard to approximate within a factor  $f(n)$ ,  $p^{1-\varepsilon}$  or  $(m/p-1)^{1-\varepsilon}$  unless  $\mathbf{P} = \mathbf{NP}$ .*

*Proof.* Given an instance  $I$  of  $\max \max 1$  with fixed  $n$ , the designed reduction leaves parameters  $n$  and  $p$  unchanged in  $f(I)$ , inapproximability with regard to these parameters is thus straightforward. However, the number of sets in  $f(I)$  is at most equal to

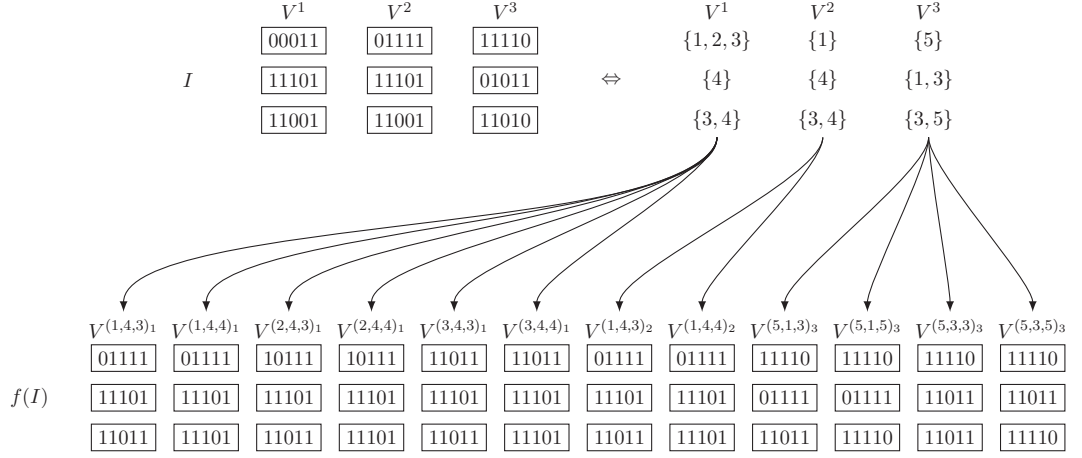


Figure 6.5: Example of reduction from an instance  $I$  of  $\max \max 1$  with  $m = 3$ ,  $p = 5$  and fixed  $n = 3$  to an instance  $f(I)$  of  $(\max \max 1)_{\#0 \leq 1}$  with  $m' = 12$ ,  $n' = 3$  and  $p' = 5$ .

$mp^n$  due to the Cartesian product. If we restrict the initial problem to the instance constructed by the reduction from MAX CLIQUE, depicted in Section 6.1.2, the number of sets in  $f(I)$  is even bounded by  $mp$ . Indeed, in each set there exists one vector that has exactly one coordinate set to zero by construction. Thus the Cartesian product generates at most  $p - 1$  sets in  $f(I)$  for each set in  $I$ .

Since  $\max \max 1$  cannot be approximated within a ration  $m^{1-\varepsilon}$ ,  $(\max \max 1)_{\#0 \leq 1}$  cannot be approximated within a ratio  $(m/p-1)^{1-\varepsilon}$ .  $\square$

#### 6.1.4 $(n - 1 - \varepsilon)$ -inapproximability for $(\min \min 0)_{\#0 \leq 1}$

The reduction provided in the previous section gives strong inapproximability results with regard to the natural parameters  $m$ ,  $n$  and  $p$  for  $(\max \max 1)_{\#0 \leq 1}$ . It seems natural to ask about the approximability of its dual version:  $\min \min 0$ .

In a first time we provide a generic polynomial-time computable function, that encodes, given an integer  $k$ , a  $k$ -uniform hypergraph with an input of a BMVA problem where every vector has at most one component set to zero. In a second time, we show how this function can be used to design an  $S$ -reduction from  $Ek$  VERTEX COVER to  $(\min \min 0)_{\#0 \leq 1}$ .

#### Construct BMVA instance from $k$ -uniform hypergraph

In this section, we present a polynomial-time computable function  $f$  that, given an integer  $k$ , encodes a  $k$ -uniform hypergraph with an input of wafer-to-wafer integration problems where every vector has exactly one component set to zero. The reduction of Subsection 6.1.5, transforming a hypergraph, not necessarily uniform, to an instance of BMVA with vectors containing more than only one coordinate set to zero is based

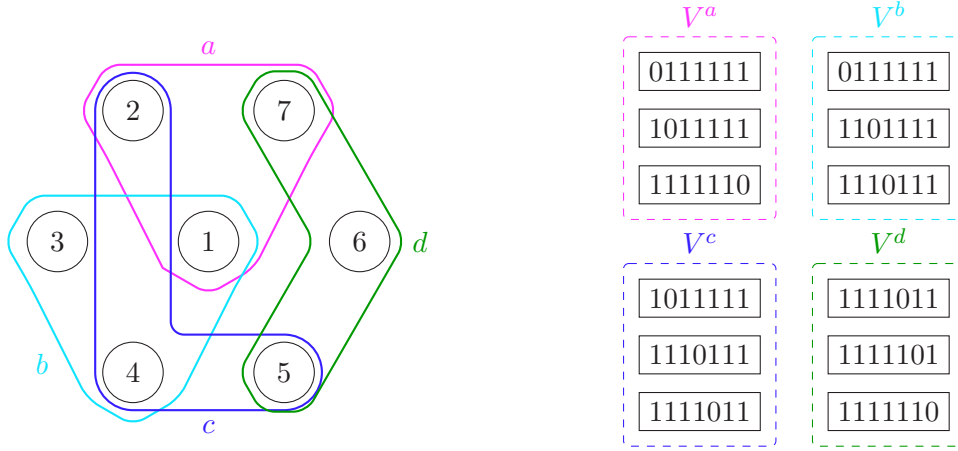


Figure 6.6: Illustration of the construction of a  $(\text{BMVA})_{\#0 \leq 1}$  instance from a 3-uniform hypergraph  $H = (U = \{1, 2, 3, 4, 5, 6, 7\}, E = \{\{1, 2, 7\}, \{1, 3, 4\}, \{2, 4, 5\}, \{5, 6, 7\}\})$ .

on the same principle.

Let us consider a  $k$ -uniform hypergraph  $H = (U, E)$ . The polynomial-time computable function  $f$  that creates an instance of  $(\text{BMVA})_{\#0 \leq 1}$  from  $H$  can be described as follows.

1. We set  $m = |E|$ ,  $n = k$  and  $p = |U|$ .
2. For each hyperedge  $e = \{u_1, u_2, \dots, u_k\} \in E$ , we create the set  $V^e$  containing  $k$  vectors  $\{v_j^e, j \in [k]\}$ , where for all  $j \in [k]$ ,  $v_j^e[u_j] = 0$  and  $v_j^e[l] = 1$  for  $l \neq u_j$ . We say that a vector  $v$  represents  $u \in U$  if and only if  $v[u] = 0$  and  $v[l \neq u] = 1$  (and thus vector  $v_j^e$  represents  $u_j$ ).

An example of this construction is given in Figure 6.6.

### **$S$ -reduction from $Ek$ VERTEX COVER to $(\min \min 0)_{\#0 \leq 1}$**

We consider  $Ek$  VERTEX COVER, the generalization to hypergraphs of the well-known VERTEX COVER problem. Remember that, given a hypergraph, a subset of vertices  $VC$  is considered to be a vertex cover if and only if  $VC$  contains at least one vertex of each hyperedge.

**Lemma 6.6.**  $Ek$  VERTEX COVER  $S$ -reduces to  $(\min \min 0)_{\#0 \leq 1}$ .

*Proof.* We use function  $f$  previously introduced. Thus given an instance  $I$  of  $Ek$  VERTEX COVER defined by a  $k$ -uniform hypergraph  $H = (U, E)$ , we are able to construct an instance  $f(I)$  of  $(\min \min 0)_{\#0 \leq 1}$  with  $m = |E|$ ,  $n = k$  and  $p = |U|$ .

Let us show that  $H$  admits a vertex cover  $VC \subseteq U$  of size  $|VC|$  if and only if  $f(I)$  admits a solution  $S$  of cost  $c(S) \leq |VC|$ .



---

**Optimization Problem 11** *Ek* VERTEX COVER
 

---

**Input**            A  $k$ -uniform hypergraph  $H = (U, E)$  and an integer  $k \geq 2$ .

**Output**          A vertex cover  $VC \subseteq U$ .

**Objective**       Minimize  $|VC|$ .

---

$\Rightarrow$  Given a vertex cover  $VC \subseteq U$ , we show how to construct a stack  $s$  in  $f(I)$  of cost  $c(s) \leq |VC|$  proving at the same time the existence of a solution  $S$  of  $f(I)$  of cost  $c(S) \leq c(s) \leq |VC|$ .

By definition of a vertex cover, for every hyperedge  $e \in E$ ,  $e \cap VC \neq \emptyset$ . We construct  $s$  as follows. For each hyperedge  $e \in E$ , we add to  $s$  a vector of  $V^e$  representing a vertex of  $VC$ . If more than one vertex of  $e$  belongs to  $VC$ , we chose the vector arbitrarily among them.

We claim that the number of zeros in this stack is at most the size of  $VC$ . Indeed, for every set, only vectors representing vertices of  $VC$  have been selected. Thus  $v_s$  has at most  $|VC|$  component set to zeros.

$\Leftarrow$  Given a solution  $S$  for  $f(I)$ , we highlight the polynomial-time computable function  $g$  computing a solution  $g(I, S)$  for  $I$ . We then prove that the computed solution is a vertex cover of size  $c(S)$ .

We consider  $s_{\max} \in S$  a stack of  $S$  of profit  $c(s_{\max}) = c(S)$ . For every coordinate equal to zero in  $v_{s_{\max}}$ , we add the corresponding vertex in  $g(I, S)$ .

We claim that  $g(I, S)$  is a vertex cover. By construction, for every set  $V^e$ ,  $s_{\max}$  indeed contains a vector representing a vertex  $u \in e$ , *i.e.* a vector  $v \in V^e$  such that  $v[u] = 0$ . This implies that  $v_{s_{\max}}[u] = 0$  and that a vertex has been selected for each hyperedge.

Furthermore,  $g(I, S)$  is such that  $|g(I, S)| = c(s_{\max}) = c(S)$ . This reduction  $(f, g)$  is thus an  $S$ -reduction.

□

**Theorem 6.8** (Dinur et al. [Din+05]). *Ek* VERTEX COVER is hard to approximate within a ratio  $(k - 1 - \varepsilon)$  for arbitrary constant  $k \geq 3$  and  $\varepsilon > 0$  unless  $\mathbf{P} = \mathbf{NP}$ .

**Theorem 6.9.**  $\forall n \geq 3, \forall \varepsilon > 0$ ,  $(\min \min 0)_{\#0 \leq 1}$  is hard to approximate within ratios  $(n - 1 - \varepsilon)$  unless  $\mathbf{P} = \mathbf{NP}$ .

This inapproximability threshold can be strengthened by assuming UGC.

**Theorem 6.10** (Bansal et al. [BK10]). *Assuming UGC, for every  $k \geq 2$ ,  $\forall \varepsilon > 0$ , Ek* VERTEX COVER *is hard to approximate within a ratio  $k - \varepsilon$ .*

**Corollary 6.1.** *Assuming UGC, for every  $n \geq 2$ ,  $\forall \varepsilon > 0$ ,  $\text{min min 0}$  is hard to approximate within a ratio  $n - \varepsilon$ .*

### 6.1.5 $(1 - \varepsilon) \log(m)$ -inapproximability for $\text{min min 0}$

The strong inapproximability result of  $(\text{min min 0})_{\#0 \leq 1}$  legitimates the question of approximability with regard to  $m$  and  $p$ . This section aims at providing inapproximability result for  $\text{min min 0}$  with regard to  $m$  with help of an  $S$ -reduction from MINIMUM HITTING SET (MIN-HS) to  $\text{min min 0}$ . This reduction also provides fixed parameterized intractability of  $\text{min min 0}$  when parameterized by  $k$ . We first present the definition of MIN-HS.

---

#### Optimization Problem 12 MINIMUM HITTING SET

---

<b>Input</b>	$m$ subsets $R_1, R_2, \dots, R_m$ of $[n]$ .
<b>Output</b>	A set $HS \subseteq [n]$ such that $HS \cap R_i \neq \emptyset$ , $\forall i \in [m]$ .
<b>Objective</b>	Minimize $ HS $ .

---

**Lemma 6.7.** *MIN-HS  $S$ -reduces to  $\text{min min 0}$ .*

*Proof.* Let us first remark that MIN-HS can be formulated in term of hypergraph by associating to each element of  $[n]$  a vertex and to each set  $R_i$  an hyperedge. Based on this observation, the reduction is essentially the same as the one presented in Section 6.1.4. The main difference is due to the fact that MIN-HS is defined on general hypergraphs instead of  $k$ -regular ones.

We described function  $f$  as follows. Given a MIN-HS instance  $I$ , for each set  $R_i$ , we create a set  $V^i$ . Each set contains  $n$  binary vectors of dimension  $n$ .

Given a set  $R_i$ , the set  $V^i$  is constructed as follows. For each element  $j \in R_i$ , we create a vector that represents  $j$ , *i.e.* a vector  $v_j^i$  such that  $v_j^i[j] = 0$  and  $\forall l \neq j$ ,  $v_j^i[l] = 1$ . The  $n - |R_i|$  others vectors of  $V^i$  are zero vectors.

An example of such a polynomial-time computable function  $f$  is depicted on Figure 6.7.

Given an integer  $k$ , we will actually show that  $\text{min min 0}$  instance  $f(I)$  has a solution  $S$  of cost  $c(S) = k$  if and only if  $R_1, \dots, R_m$  has a hitting set of size  $k$ . By the foregoing, we only need to focus on the only vector of each set which is assigned to  $s_{min}$ , the stack with the minimum number of coordinates set to zero.

$\Rightarrow$  Let  $HS \subseteq [n]$  be a Hitting Set of size  $k$ . By the definition of a hitting set, for all  $i \in [m]$ , there exists  $j_i \in HS \cap R_i$ . Thus, for all  $i \in [m]$ , we select the vector  $v_{j_i}^i$  from the set  $V^i$  to be assigned to  $s_{min}$ . By construction, this vector has only one zero at the  $j_i^{th}$  coordinate, which implies that the conjunction of all such vectors

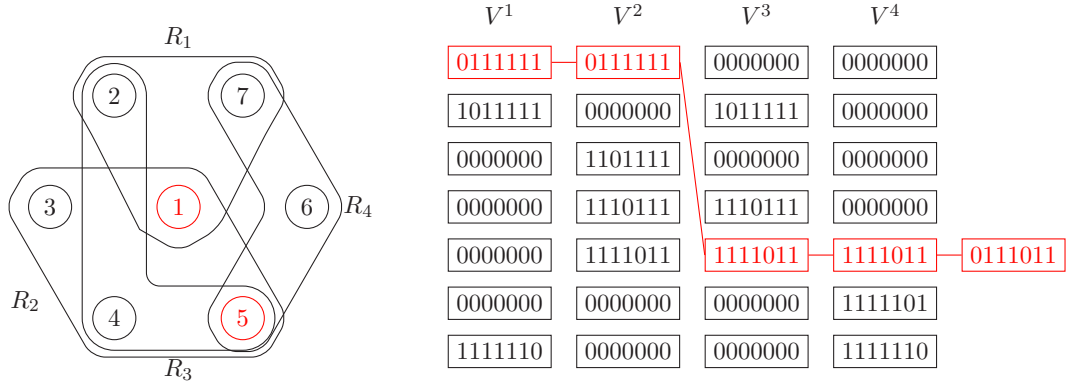


Figure 6.7: Example of reduction from an instance of MINIMUM HITTING SET consisting of four subsets of  $[n = 7]$  :  $R_1 = \{1, 2, 7\}$ ,  $R_2 = \{1, 3, 4, 5\}$ ,  $R_3 = \{2, 4, 5\}$ ,  $R_4 = \{5, 6, 7\}$ , and an integer  $k = 2$ , to an instance of  $\min \min 0$  with  $m' = 5$ ,  $n' = n$ ,  $p = n$ .

$\bigwedge_{i=1}^m v_{j_i}^i$  will have a 1 everywhere except at the coordinates corresponding to  $HS$ . We thus have the desired number of zeros in our solution.

$\Leftarrow$  Conversely, for each  $i \in [m]$ , let  $j_i \in [n]$  be the vector from  $V^i$  which is assigned to  $s_{min}$ . Since the resulting conjunction of all these vectors has only  $k$  zeros,  $v_{j_i}^i$  cannot be a 0-vector, and we thus have  $j_i \in R_i$ . Using the same arguments as previously,  $\{u_{j_i}\}_{i \in [m]}$  corresponds to a hitting set of  $R_1, \dots, R_m$  of size  $k$ .

□

We make the following observation that will be useful in Section 8.2.3.

**Observation 6.1.** *The overall number of zeros in each set is bounded by  $p(n-2)+2$ . Indeed, each set encodes an hyperedge that has at least two endpoints.*

Inapproximability results are known for MIN-HS in function of the number of sets  $m$ .

**Theorem 6.11** (Trevisan [Tre01]). *MINIMUM HITTING SET does not admit any  $(1-\varepsilon)\log(m)$  approximation algorithm with  $\varepsilon > 0$ , unless  $\mathbf{NP} \subseteq \mathbf{QP}$ .*

Since the reduction preserves the value of the parameter  $m$ , it follows that:

**Theorem 6.12.**  *$\min \min 0$  does not admit any  $(1-\varepsilon)\log(m)$  approximation algorithm with  $\varepsilon > 0$ , unless  $\mathbf{NP} \subseteq \mathbf{QP}$ .*

The previous reduction being an  $S$ -reduction, it can also be seen as an  $FPT$ -reduction from MIN-HS parameterized by the standard parameter to  $\min \min 0$  parameterized by the standard parameter.

**Corollary 6.2.** *There exists an  $FPT$ -reduction from MIN-HS parameterized by the standard parameter  $(\text{MIN-HS}, k)$  to  $\min \min 0$  parameterized by the standard parameter  $(\min \min 0, k)$ .*

Well, negative results are known about MIN-HS when parameterized by the standard parameter.

**Theorem 6.13** (Downey and Fellows [DF13]). (MIN-HS,  $k$ ) is  $\mathbf{W}[2]$ -hard.

**Theorem 6.14.** (min min 0,  $k$ ) is  $\mathbf{W}[2]$ -hard.

We can use this reduction to show further negative results. Let us consider a constrained version of the MINIMUM HITTING SET problem obtained by [LMS11], where the element set is  $[d] \times [d]$  and can thus be seen as a table with  $d$  rows and  $d$  columns:

---

**Decision Problem 3**  $d \times d$ -HITTING SET

---

<b>Input</b>	An integer $d$ and $R_1, R_2, \dots, R_t \subseteq [d] \times [d]$
<b>Question</b>	Is there a set $R$ containing exactly one element from each row such that $R \cap R_i \neq \emptyset$ for any $i \in [t]$ ?

---

[LMS11] show the following theorem:

**Theorem 6.15** ([LMS11]).  $d \times d$ -HITTING SET cannot be solved in time  $2^{o(d \log(d))} n^{O(1)}$  unless ETH fails.

We can then deduce from Lemma 6.7 the following theorem.

**Theorem 6.16.** (min min 0,  $k$ ) cannot be solved in  $O^*(2^{o(k) \log(n)})$  nor  $O^*(2^{ko(\log(n))})$ , unless ETH fails.

*Proof.* Notice that we can modify the question of this problem by dropping the constraint that  $S$  contains at least one element from each row. Indeed, let us add to the instance a set of  $d$  sets  $\{R'_1, \dots, R'_d\}$ , where  $R'_i$  contains all elements of row  $i$  for  $i \in [d]$ . Now, finding a (classical) hitting set of size  $d$  on this modified instance is equivalent to finding a solution of size  $d$  for the original instance of  $d \times d$ -HITTING SET. Moreover, it is easy to check that a  $2^{o(d \log(d))} n^{O(1)}$  algorithm for this relaxed problem would also contradict ETH. To summarize, we know that unless ETH fails, there is no  $2^{o(d \log(d))} n^{O(1)}$  algorithm for the classical MINIMUM HITTING SET problem, even when the ground set has size  $d^2$ . This allows us to perform the reduction of Lemma 6.7 on these special instances, leading to an instance  $f(I)[m', n', p']$  with associated standard parameter  $k$  such that  $k = d$  and  $n' = d^2$ . Suppose now that there exists an algorithm for min  $\sum 0$  running in  $2^{o(k) \log(n)} (k + m + n + p)^{O(1)}$ . Using the reduction above, we would be able to solve the instance of  $d \times d$ -HITTING SET in  $2^{o(d) \log(d^2)} n^{O(1)}$ , and thus in  $2^{o(d \log(d))} n^{O(1)}$ , which would violate ETH. A similar idea also rules out any algorithm running in  $2^{ko(\log(n))}$  under ETH.  $\square$

## 6.2 Positive Results

Before getting further, we need to introduce some useful notions. First, we extend the classical  $[n]$  notation standing for the set  $\{1, 2, \dots, n\}$ , to the following notation.

**Notation 6.1.** *Given two integers  $k$  and  $n$  such that  $k \leq n$ ,  $[n]_k$  stands for the set  $\{k, k+1, \dots, n\}$ .*

*As such  $[n]$  and  $[n]_1$  are equivalent.*

We then introduce the configuration of a vector as follows:

**Definition 6.1.** *For any  $t \in [2^p - 1]_0$ , we define configuration  $t$  as  $B_t$ : the  $p$ -dimensional binary vector that represents  $t$  in binary. We say that a  $p$ -dimensional vector  $v$  is in configuration  $t$  if and only if  $v = B_t$ .*

In light of these definitions, we remark that  $\max \max 1$  can be trivially solved in  $\mathcal{O}^*(2^p)$ . This can be done by testing, for all configuration  $t$ , if there exists a feasible stack  $s$  with representative vector in configuration  $t_s$  such that  $t_s$  dominates  $t$  and by keeping the best created stack. Informally, we say that a configuration  $t_1$  dominates another configuration  $t_2$  if the set of coordinates set to zero in  $B_{t_2}$  is included in the set of coordinates set to zeros in  $B_{t_1}$ .

A natural question arising is to know whether improving the running time of the algorithm to  $\mathcal{O}^*(r^p)$  with  $r < 2$  is possible. We show now that, when considering the special case where  $n = 2$ , a simple reduction from  $\max \max 1$  to INDEPENDENT SET provides an algorithm with improved running time.

**Theorem 6.17.** *For  $n = 2$ ,  $\max \max 1$  can be solved in  $\phi(r, p)$  using any algorithm for INDEPENDENT SET running in  $\phi(r, n)$ .*

*Proof.* To prove the previous theorem, we show that there exists an  $S$ -reduction  $(f, g)$  from  $\max \max 1$ , when  $n = 2$ , to INDEPENDENT SET.

Let us consider a  $\max \max 1$  instance  $I$  with  $n = 2$ , we show how to construct an instance  $f(I)$  of INDEPENDENT SET defined by a graph  $G = (U, E)$ .

Remember that according to Hypothesis 4.3, two vectors of a same set  $v_1^i, v_2^i$  cannot have both the same coordinate  $l$  set to zero.

We construct the graph  $G$  as follows:

- we set  $U = p$ ,
- for each couple  $(l_1, l_2) \in [p]^2$ , we create an edge  $(l_1, l_2) \in E$  if and only if there exists a set  $V^i$  such that  $v_1^i[l_1] = 0$  and  $v_2^i[l_2] = 0$ .

We claim now that finding a solution  $S$  for  $I$  of profit  $c(S) = k$  is equivalent to finding an independent set  $IS$  in  $G = (U, E)$  of profit  $|IS| = k$ .

$\Leftarrow$  We consider an independent set  $IS$  in  $G = (U, E)$ , we show how to construct a solution  $g(I, IS) = s_1, s_2$  for  $\max \max 1$ . For each  $l \in IS$ , we assign to the

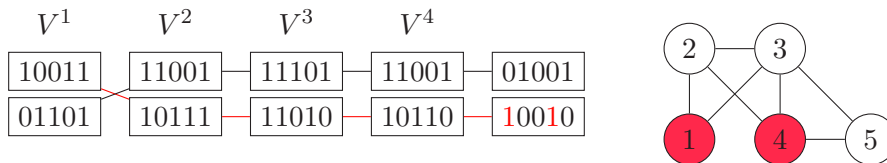


Figure 6.8: Illustration of the reduction from a max max 1 instance  $I$  with  $m = 4$ ,  $n = 2$ ,  $p = 5$  admitting a solution  $S$  of profit  $c(S) = 2$  to an instance  $G = (\{1, 2, 3, 4, 5\}, \{(1, 2), (1, 3), (2, 3), (2, 4), (3, 4), (3, 5), (4, 5)\})$  of INDEPENDENT SET admitting a solution  $IS = \{1, 4\}$  of profit two.

stack  $s_1$ , every vector of any set  $V^i$  that verifies  $v_j^i[l] = 0$ , if  $s_1$  isn't complete, we assign greedily vectors of remaining sets to  $s_1$ .

Note that two vectors of a same set cannot be assigned to  $s_1$ . Let us indeed suppose that there exists a set  $V^i$  such that  $v_1^i$  and  $v_2^i$  are assigned to  $s_1$ . By construction, there exists  $l_1 \in v_1^i$  and  $l_2 \neq l_1 \in v_2^i$  such that  $l_1, l_2 \in IS$ . By construction, such a pair of coordinates implies an edge in  $G$ , thus  $IS$  is not an independent set.

By construction, the second stack  $s_2 \in g(I, IS)$  is such that  $\forall l \in IS, v_2^s[l] = 1$ . Thus  $c(g(I, IS) = \{s_1, s_2\}) \geq |IS|$ .

$\Rightarrow$  We consider a solution  $S$  for  $I$  of profit  $c(S)$ . There exists a stack, let us say  $s_1$ , such that  $v_{s_1}$  contains  $c(S)$  components equal to one. We construct an independent set  $IS$  as follows,  $\forall l/v_{s_1}[l] = 1, k \in IS$ .

If we call  $Z = \{l/v_{s_1}[l] = 1\}$ , we remark that  $\forall (l_1, l_2) \in Z^2$ , there doesn't exist a set  $V^i$  such that  $v_1^i[l_1] = 0$  and  $v_2^i[l_2] = 0$ . Such a set would imply that either  $v_{s_1}[l_1] = 0$  and  $v_{s_1}[l_2] = 1$  or that  $v_{s_1}[l_1] = 1$  and  $v_{s_1}[l_2] = 0$ .

Hence, by construction,  $\forall (l_1, l_2) \in Z^2, (l_1, l_2) \notin E$  and  $Z$  is a independent set of size  $|IS| = c(S)$ .

An illustration of this reduction is given on Figure 6.8. □

Since INDEPENDENT SET can be, for instance, solve in time  $\mathcal{O}^*(1.2738^p)$  [CKX10], we can write the following Corollary.

**Corollary 6.3.** *For  $n = 2$ , max max 1 can be solved in  $\mathcal{O}^*(1.2738^p)$ .*

## 6.3 Conclusion and open questions

This positive result brings to a close our study of these intermediate problems. We can see that even though the power of expression of these problems is in a way reduced due to the simplicity of the cost function <sup>2</sup>, these problems encode a large variety of graph

<sup>2</sup>Keep in mind that only one  $m$ -tuple is taken into consideration in max max 1 and min min 0, and only non zero  $m$ -tuple are counted in max <sub>$\neq 0$</sub> .

and hypergraph problems. This expressiveness implies a strong inherent hardness for the latter but provides on the other hand a convenient and intuitive framework to tackle many graph problems. As such, furthering the study of positive results even in constrained instances can be an interesting area of research.

Furthermore as we will see in the both next chapters, the complexity of the masters problems we consider ( $\max \sum 1$  and  $\min \sum 0$ ) is intrinsically linked to the complexity of these intermediate problems. It follows that on one hand, negative results for the intermediate problems can probably be extended to the masters problems and on another hand, positive results for the intermediate problems give intuition on the way to tackle the masters one. In this respect, the main open question to our opinion is about the existence of an  $f(m)$ -approximation algorithm for  $\max \sum 1$ . It seems to be a really hard but interesting question. Indeed, we can easily see that no greedy strategy can lead to an approximation algorithm. Such strategies can in a way be seen as online strategies. Once this remark formulated, one can see that, no matter the choices that have been made on the  $m - 1$  first sets, it is always possible to design an adversary forcing this partial solution to be arbitrarily bad.

As we will see in next chapter, this question naturally holds for  $\max \sum 1$ .

# 7

## Maximizing the overall number of ones

### Contents

<b>7.1</b>	<b>Negative Results</b>	<b>88</b>
7.1.1	$\mathcal{O}(\frac{m}{\log(m)})$ -inapproximability	88
7.1.2	$f(n)$ , $p^{1-\varepsilon}$ , $m^{1-\varepsilon}$ -inapproximability	90
<b>7.2</b>	<b>Considering the vectors dimension</b>	<b>91</b>
7.2.1	$p/r$ -approximation	92
7.2.2	Faster algorithm for fixed $p$ for $\max \sum 1$	94
7.2.3	No polynomial size kernel with parameter $p$	99
<b>7.3</b>	<b>Resolution of sparse instances</b>	<b>101</b>
7.3.1	<b>EPTAS</b> for $(\max \sum 1)_{\#0 \leq r}$	101
7.3.2	A polynomial-time algorithm for $(\max \sum 1)_{\#0 \leq 1}$ when $m$ is fixed	103
<b>7.4</b>	<b>Conclusion and open questions</b>	<b>107</b>

This chapter is devoted to the study of the  $\max \sum 1$ -BMVA ( $\max \sum 1$ ) problem, presented in Introduction. It is, with  $\min \sum 0$ -BMVA, one of the two considered problems that has direct application to IC manufacturing. Its practical resolution, based on Integer Linear Programming or Constraints Programming formulations is discussed in the next part of this manuscript. This chapter presents theoretical results on the complexity of the problems.

In a first time we present negative results, mainly based on reduction from intermediate problems previously presented. We first prove, in section 7.1.1,  $\mathcal{O}(m/\log m)$ -inapproximability for  $m \geq 3$  based on a reduction from  $\max_{\neq 0}$ . The following section,



Section 7.1.2, aims at extending the inapproximability results of  $\max \sum 1$  to  $\max \sum 1$  with help of an  $S$ -reduction. In a second time we focus on the dependence of the complexity in  $p$ . We establish, in section 7.2.1, that  $\max \sum 1$  admits a  $p/r$ -approximation algorithm based on a routine that solves optimally smallest instances of  $\max \sum 1$  with  $r$ -dimensional vectors. We then demonstrate, in the Section 7.2.2, that  $(\max \sum 1, p)$  is FPT implying that previous algorithm runs in polynomial time when  $r$  is fixed. However the Section 7.2.3 provides AND-cross-composition proving that the problem does not admit a polynomial kernel with size depending only in  $p$ .

Finally we focus on the impact of the number of coordinates set to zeros per vector on the complexity of the problem. Contrary to its general version  $(\max \sum 1, m)$ , we show that  $((\max \sum 1)_{\#0 \leq 1}, m)$  is in  $\mathbf{XP}$ .

An overview of the known results on  $\max \sum 1$  is given in Figure 7.1.

Excluding the AND-cross-composition, these results are published in [Bou+16] and [BDG16].

## 7.1 Negative Results

### 7.1.1 $\mathcal{O}(\frac{m}{\log(m)})$ -inapproximability

In this section, we provide a strict-reduction from the  $\max_{\neq 0}$  intermediate problem, proving inapproximability results for  $\max \sum 1$ .

**Lemma 7.1.**  $\max_{\neq 0}$  *strict-reduces* to  $\max \sum 1$ .

*Proof.* We consider a  $\max_{\neq 0}$  instance  $I[m, n, p]$ . We introduce  $f$  that maps  $I$  to a  $\max \sum 1$  instance  $f(I)[m' = m + 1, n' = np, p' = p]$ . For every set  $V^i$  we create a set  $V'^i$ . These sets are constructed as follows. For every vector  $v_j^i \in V^i$ , we create a copy  $v_j'^i \in V'^i$ . We complete  $V'^i$  with  $n' - n$  zero vectors;

The set  $V'^{m+1}$  contains  $n$  occurrences of the following set of vectors:

$$\{\underbrace{1000 \dots 000}_{p=p'}, 0100 \dots 000, 0010 \dots 000, \dots, 0000 \dots 010, 0000 \dots 001\}$$

As an example, the following instance  $I$  of  $\max_{\neq 0}$

$$\begin{array}{ccc} v_1^1 = 1010 & v_1^2 = 0001 & v_1^3 = 1111 \\ v_2^1 = 1001 & v_2^2 = 0100 & v_2^3 = 1000 \\ \underbrace{\hspace{1.5cm}}_{V^1} & \underbrace{\hspace{1.5cm}}_{V^2} & \underbrace{\hspace{1.5cm}}_{V^3} \end{array}$$

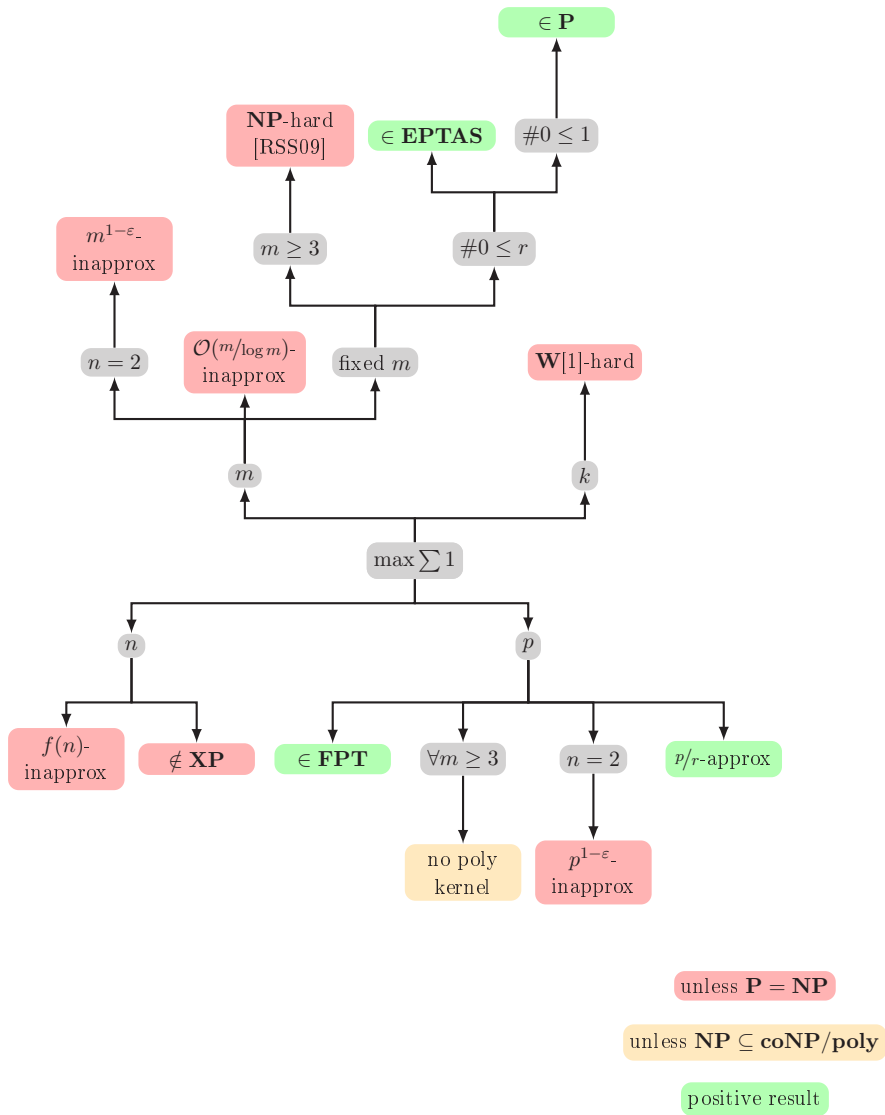


Figure 7.1: Overview of the results for  $\max \sum 1$

is turned into the following one  $I$  of  $\max \sum 1$ :

$$\begin{array}{cccc}
 v_1^1 = 1010 & v_1^2 = 0001 & v_1^3 = 1111 & v_1^4 = 1000 \\
 v_2^1 = 1001 & v_2^2 = 0100 & v_2^3 = 1000 & v_2^4 = 0100 \\
 v_3^1 = 0000 & v_3^2 = 0000 & v_3^3 = 0000 & v_3^4 = 0010 \\
 v_4^1 = 0000 & v_4^2 = 0000 & v_4^3 = 0000 & v_4^4 = 0001 \\
 v_5^1 = 0000 & v_5^2 = 0000 & v_5^3 = 0000 & v_5^4 = 1000 \\
 v_6^1 = 0000 & v_6^2 = 0000 & v_6^3 = 0000 & v_6^4 = 0100 \\
 v_7^1 = 0000 & v_7^2 = 0000 & v_7^3 = 0000 & v_7^4 = 0010 \\
 v_8^1 = 0000 & v_8^2 = 0000 & v_8^3 = 0000 & v_8^4 = 0001 \\
 \underbrace{\hspace{1.5cm}}_{V^1} & \underbrace{\hspace{1.5cm}}_{V^2} & \underbrace{\hspace{1.5cm}}_{V^3} & \underbrace{\hspace{1.5cm}}_{V^4}
 \end{array}$$

Informally, the last set in  $f(I)$  acts like a mask transforming any non zero stack into a stack of profit equal to one.

Let us now prove that this transformation is an strict-reduction.

$\Rightarrow$  Suppose that  $I$  admits a solution  $S$  of profit  $c_{\max \neq 0}(S)$ . We construct a solution  $S'$  of  $f(I)$  as follows. For each non zero stack  $s \in S$ , we construct a stack  $s'$  containing the created of each vectors in  $s$  and associate a vector of  $V^{m+1}$  that does not nullify  $s'$ . Such a vector always exists since  $S$  contains at most  $n$  non zero stacks and  $V^{m+1}$  contains  $n$  vectors with coordinate  $l$  set to one  $\forall l \in [p]$ . The other stack are greedily constructed.

It follows that  $c_{\max \sum 1}(S') \geq c_{\max \neq 0}(S)$ .

$\Leftarrow$  Consider a solution  $S$  of  $f(I)$ , we construct a solution  $g(I, S)$  of  $I$  as follows.

For every non zero stack  $s$  of  $S$  we add to  $g(I, S)$  a stack  $s'$  containing the copy of the vectors in  $s$  selected in sets  $V^1$  to  $V^m$ . Note that these vectors exist since  $s$  cannot contains one of the created zero vectors.  $s$  being a non zero stack,  $s'$  is also a non zero stack.

The other stacks of  $g(I, S)$  are greedily constructed. It follows that for any solution  $S$  of  $f(I)$   $c_{\max \neq 0}(g(I, S)) \geq c_{\max \sum 1}(S)$ .

Therefore, the reduction  $(f, g)$  is a strict-reduction.  $\square$

Using Theorem 6.1, we get the following result:

**Theorem 7.1.**  $\max \sum 1$  is hard to approximate within a factor  $\mathcal{O}(\frac{m}{\log(m)})$  unless  $\mathbf{P} = \mathbf{NP}$ .

### 7.1.2 $f(n)$ , $p^{1-\varepsilon}$ , $m^{1-\varepsilon}$ -inapproximability

We can also highlight a strict-reduction from  $\max \sum 1$ . This reduction provides negative results with regard to parameters  $n$  and  $p$ . But it also provides inapproximability in function of  $m$ . Although the  $m^{1-\varepsilon}$ -inapproximability result is weaker than the previous  $m/\log(m)$ -inapproximability, this one holds even for  $m = 2$ .

**Lemma 7.2.**  $\max \max 1$  *strict-reduces to*  $\max \sum 1$ .

*Proof.* Let us construct  $(f, g)$  as in Definition 2.27. Consider an instance  $I$  of  $\max \max 1$ . We construct an instance  $f(I)$  of  $\max \sum 1$  as follows: we set  $p' = p, n' = n, m' = m + 1$ . The  $m$  sets of  $I$  remain unchanged in  $f(I)$ :  $\forall i \in [m]_1, V^i = V^i$  and the last set  $V^{m+1}$  contains  $(n - 1)$  zero vectors and a unique one vector.

Informally, the set  $V^{m+1}$  of  $f(I)$  behaves like a selecting mask: since all stacks except one are turned into zero stacks when assigning the vectors of last set, the unique one vector of set  $V^{m+1}$  must be assigned to the best stack, and maximizing the sum of the stacks is equivalent to maximizing the best stack.

More precisely, given a solution  $S$  of  $f(I)$ , if we define  $g(I, S)$  as the solution of  $I$  containing a stack  $s$  created with copies of vectors of sets  $V^1, V^2, \dots, V^m$  composing the non zero stack  $s'$  in  $S$ , it is straightforward to see that the following statement is true:  $\forall x, \exists \text{ solution } g(I, S) \text{ of } \max \max 1 \text{ of value } c_{\max \max 1}(g(I, S)) = x \Leftrightarrow \exists \text{ solution } S \text{ of } \max \sum 1 \text{ of value } c_{\max \sum 1}(S) = x$ . Thus, we get  $c(\text{opt}_{\max \max 1}(I)) = c(\text{opt}_{\max \sum 1}(f(I)))$ . As  $f, g$  are polynomial-time computable functions, we get the desired result. □

Using Theorems 6.3 and 6.4, we get the following results.

**Theorem 7.2.**  $\max \sum 1$  *is hard to approximate within a factor*  $f(n)$  *for any computable function*  $f$ , *unless*  $\mathbf{P} = \mathbf{NP}$ .

**Theorem 7.3.** *Even for*  $n = 2$ , *for any*  $\varepsilon > 0$ ,  $\max \sum 1$  *is hard to approximate within factors*  $\mathcal{O}(m^{1-\varepsilon})$  *and*  $\mathcal{O}(p^{1-\varepsilon})$ , *unless*  $\mathbf{P} = \mathbf{NP}$ .

## 7.2 Considering the vectors dimension

The strict-reductions from  $\max_{\neq 0}$  and  $\max \max 1$  prove strong inapproximability results for  $\max \sum 1$  with regard to the three natural parameters  $m, n$  and  $p$ . These results motivate a further study of the dependence of the  $\max \sum 1$  complexity in function of its parameters. However, the reduction from MAX CLIQUE with  $n = 2$  shows that  $\max \max 1$  (and thus  $\max \sum 1$ ) is not even in  $\mathbf{XP}$  when parameterized by  $n$ . Furthermore the reduction of Dokka et al. [DCS14] from a particular case of 3-DIMENSIONAL MATCHING shows that  $\min \sum 0$  (and thus  $\max \sum 1$ ) is neither in  $\mathbf{XP}$  when parameterized by  $m$ . This legitimates the study of  $\max \sum 1$  with regard to  $p$ .

In this section, we first focus on approximability of  $\max \sum 1$  in function of  $p$ . A trivial polynomial-time algorithm with approximation ratio of  $p$  is given raising the question of the existence of a  $p/r$ -approximation algorithm. We provide then a  $p/r$ -approximation algorithm relying on a routine solving efficiently instances of  $\max \sum 1$  when  $p$  is small.

Naturally the parameterized complexity of  $\max \sum 1$  when parameterized by  $p$  is taken into consideration. We show that  $(\max \sum 1, p)$  is FPT and use this result to

justify that the previous  $p/r$ -approximability algorithm runs in polynomial time when  $r$  is fixed. The membership of  $\max \sum 1/p$  in **FPT** naturally leads us to ask about the size of its kernel. We prove that the existence of a polynomial size kernel is very unlikely.

### 7.2.1 $p/r$ -approximation

When considering Theorem 7.3, it seems natural to look for ratio  $p/r$ , where  $r$  is a constant. Let us first see how to achieve a ratio  $p$  with Algorithm 2.

---

**Algorithm 2:**  $p$ -approximation for  $\max \sum 1$

---

```

 $x = 0;$ 
while  $\exists l$  such that it is possible to create a stack  $s$  with  $v_s[l] = 1$  do
    Add  $s$  to the solution;
     $x = x + 1;$ 
if  $x < n$  then
    Add  $n - x$  arbitrary (null) stacks to the solution;

```

---

**Property 7.1.** *Algorithm 2 is a  $p$ -approximation algorithm for  $\max \sum 1$ .*

*Proof.* Given an instance  $i$ , let  $S = S_{\neq 0} \cup S_0$  be the solution computed by the algorithm, where  $S_{\neq 0}$  is the set of non zero stacks, and  $S_0$  is the set of the remaining null stacks. Since  $S_{\neq 0}$  and  $S_0$  are disjoint, we have  $S_0 = S \setminus S_{\neq 0}$ . Let  $n_1 = |S_{\neq 0}|$ , and  $\forall i$ , let  $V^i = V^i \cap S_0$ . Let  $n_2 = |S_0| = |V^i|$  (all the  $V^i$  have the same size). Notice that  $n = n_1 + n_2$ .

As the algorithm cannot create any non null stack at the end of the loop, we know that for any position  $l \in [p]$ , there is a set  $i_l$  such for any vector  $v \in V^{i_l}$ ,  $v[l] = 0$ . In other words, we can say that there is a column of  $n_2$  zeros in set  $V^{i_l}$ . Notice there may be several columns of zeros in a given set. Thus, we deduce that there are at least  $p$  columns (of  $n_2$  zeros) in the vectors of  $V^{i_l}$ . Moreover, as none of these zeros can be matched in a solution, we know that these  $n_2 p$  zeros appear in any solution.

Thus, given  $\text{opt}(i)$  an optimal solution, we have  $c(\text{opt}(i)) \leq np - n_2 p = n_1 p$ . As  $c(S) \geq n_1$ , we get the desired result.  $\square$

Given a fixed integer  $r$  (and targeting a ratio  $p/r$ ), a natural way to extend Algorithm 2 is to first try to create stacks of profit exactly  $r$ , *i.e.* find  $(l_1, \dots, l_r)$  such that it is possible to create  $s$  such that  $v_s[l_1] = \dots = v_s[l_r] = 1$ , then stacks of profit exactly  $(r - 1)$ , and so on until only null stacks can be created. However, even for  $r = 2$  this algorithm is not sufficient to get a ratio  $p/2$ . Intuitively, when no stack of profit 2 can be created in the instance, this algorithm runs in the exact same way as Algorithm 2. The latter being tight, as shown by the example depicted in Figure 7.2, this algorithm cannot achieve a better ratio.

In this example it is not possible to create any stack of value strictly greater than one since set  $V^1$  *kills* positions  $\{1, 2\}$  (we say that a set kills positions  $\{l_1, l_2\}$  if and

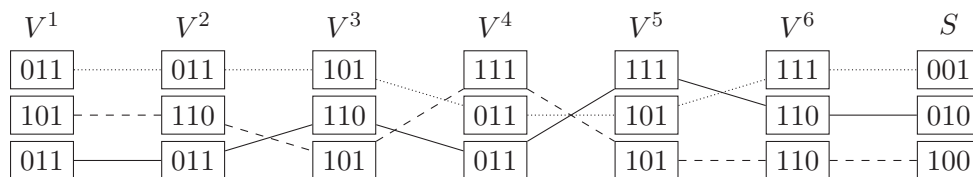


Figure 7.2: Counter-example showing that Algorithm 2 for  $r = 2$  remains a  $p$ -approximation. The depicted stacks correspond to an optimal solution of profit 3 whereas the algorithm outputs a solution of profit 1.

only if there is no vector in the set such that  $v[l_1] = v[l_2] = 1$ ), set  $V^2$  kills positions  $\{1, 3\}$ , and set  $V^3$  kills positions  $\{2, 3\}$ .

Thus, as previously mentioned, in this case (and more generally when no stack of value greater than 1 can be created), the solution computed by the algorithm for  $r = 2$  is the same as one computed by Algorithm 2. In the worst case, the algorithm creates only one stack of value 1 (by choosing the first vector of each set). However, as depicted in Figure 7.2, the optimal value is 3, and thus the ratio  $p/2$  is not verified. In other words, knowing that no stack of profit 2 can be created does not provide better results for Algorithm 2. This motivates the different approach we follow hereafter.

**Property 7.2.** *Suppose that there exists an exact algorithm for  $\max \sum 1$  running in  $f(n, m, p)$ . Then, for any  $r \in [p]$  we have a  $p/r$ -approximation running in  $\mathcal{O}(p \times f(n, m, r))$ .*

*Proof.* The idea is to use a classical *shifting technique* by guessing the subset of the  $r$  most valuable consecutive positions in the optimal solution, and running the exact algorithm on these  $r$  positions.

Given an instance  $I[m, n, p]$  of  $\max \sum 1$ , we consider an optimal solution  $\text{opt}(I)$ . The cost of  $\text{opt}(I)$  can be written as follows:  $c(\text{opt}(I)) = \sum_{l=1}^p a_l$ , where  $a_l = |\{s \in \text{opt}(I) : v_s[l] = 1\}|$  is the number of stacks in  $\text{opt}(I)$  that save position  $l$ .  $\forall l \in [p]$ , let  $X_l = \{l, \dots, 1 + ((l + r - 1) \bmod p)\}$ , and  $\sigma_l = \sum_{t \in X_l} a_t$ . Notice that we have  $\sum_{l=1}^p \sigma_l = r \sum_{l=1}^p a_l = rc(\text{opt}(I))$ , as each value  $a_l$  appears exactly  $r$  times in  $\sum_{l=1}^p \sigma_l$ . This implies  $\max_{l \in [p]} \sigma_l \geq \frac{r}{p} c(\text{opt}(I))$ .

For any  $l$ , let  $I_l$  be the restricted instance where all the vectors are truncated to only keep positions in  $X_l$  (there are still  $nm$  vectors in  $I_l$ , but each vector is now a  $r$  dimensional vector). By running the exact algorithm on all the  $I_l$  and keeping the best solution, we get a  $p/r$ -approximation running in  $\mathcal{O}(pf(n, m, r))$ .  $\square$

The previous lemma motivates the exact resolution of  $\max \sum 1$  in polynomial-time for fixed  $p$ .

The next section is devoted to the design of an FPT-algorithm for  $\max \sum 1$  (and thus  $\min \sum 0$ ) when parameterized by  $p$ . Note that these parameterized problems have already been proven in  $\mathbf{XP}$  by Dokka et al. [DCS14]. They indeed prove that  $\min \sum 0$  can be solved in  $\mathcal{O}(m(n^{2^p}))$ . As this result also apply to  $\max \sum 1$ , we get a  $p/r$ -approximation running in  $\mathcal{O}(pm(n^{2^r}))$ , for any  $r \in [p]$ .

We now show that this running time can be improved by showing that  $\min \sum 0$  and  $\max \sum 1$  are FPT when parameterized by  $p$ .

### 7.2.2 Faster algorithm for fixed $p$ for $\max \sum 1$

#### First ideas to get an FPT algorithm

In a first time, we present the already known algorithm presented in [DCS14] for fixed  $p$ . This result is obtained using an integer linear programming formulation of the following form. In this paper, they consider  $\min \sum 0$ , as such the objective function is  $\min \sum_{t=0}^{2^p-1} x_t \bar{c}_t$  where  $x_t \in [n]_0$  is an integer variable representing the number of stacks in configuration<sup>1</sup>  $t$ , and  $\bar{c}_t \in [p]_0$  is the number of 0 in configuration  $t$ .

This is a good starting point to get an **FPT** algorithm. Indeed, if we note  $n_{var}$  (resp.  $m_{ctr}$ ) the number of variables (resp. number of constraints) of an ILP, for any  $A \in \mathbb{Q}^{n_{var} \times m_{ctr}}, b \in \mathbb{Q}^{m_{ctr}}$ , the famous algorithm of Kannan [Kan83] allows us to decide the feasibility of an ILP, under the form  $\exists?x \in \mathbb{Z}^{n_{var}} | Ax \leq b$ , in time  $\mathcal{O}(n_{var}^{9n_{var} m_{ctr}} \ln m_{ctr})$ . Thus, to get an **FPT** algorithm parameterized by  $p$ , it is sufficient to write  $\min \sum 0$  (and  $\max \sum 1$ ) as an ILP using  $f(p)$  variables.

However, it remains now to add constraints that represent the  $\min \sum 0$  problem. In [DCS14], these constraints are added using  $z_{jt}^i$  variables (for  $i \in [m], j \in [n], t \in [2^p - 1]_0$ ), where  $z_{jt}^i = 1$  if and only if  $v_j^i$  is assigned to a stack of type  $t$ , a stack  $s$  being of type  $t$  if and only if its representative vector  $v_s$  is in configuration  $t$ . Nevertheless these new  $\mathcal{O}(mn2^p)$  variables prevent us to use [Kan83]. Thus, we now come back to the  $\max \sum 1$  problem, and our objective is to express the constraints using only the  $\{x_t\}$  variables.

#### Presentation of the new ILP for $\max \sum 1$

For any  $t \in [2^p - 1]_0$ , we define an integer variable  $x_t \in [n]_0$  representing the number of stacks in configuration  $t$ . Let also  $c_t \in [p]_0 = c(B_t)$  be the number of coordinates set to one in a vector in configuration  $t$ ,  $B_t$  being roughly the binary representation of the integer  $t$ , cf Definition 6.1.

**Definition 7.1.** A profile is a tuple  $P = \{x_0, \dots, x_{2^p-1}\}$  such that  $\sum_{t=0}^{2^p-1} x_t = n$ .

**Definition 7.2.** The profile  $Pr(S) = \{x_0, \dots, x_{2^p-1}\}$  of a solution  $S = \{s_1, \dots, s_n\}$  is defined by  $x_t = |\{i : v_{s_i} \text{ is in configuration } t\}|$ , for  $t \in [2^p - 1]_0$ .

**Definition 7.3.** Given a profile  $P$ , an associated solution  $S$  is a solution such that  $Pr(S) = P$ . We say that a profile  $P$  is feasible if and only if there exists an associated solution  $S$  that is feasible.

Notice that the definition of associated solutions also applies to a non feasible profile. In this case, any associated solution will also be non feasible.

Obviously, the  $\max \sum 1$  problem can be formulated using the following ILP.

<sup>1</sup>Recall that, intuitively, a vector  $v$  is in configuration  $t$  if  $v$  is the binary representation of  $t$ , cf Definition 6.1 for formal definition.

---

**Integer Linear Program IV** FPT-formulation sketch
 

---

$$\text{Maximize } \sum_{t=0}^{2^p-1} x_t c_t \quad (\text{IV.1})$$

$$\text{subject to } \sum_{t=0}^{2^p-1} x_t = n \quad (\text{IV.2})$$

$$x_t \in \mathbb{N} \quad \forall 0 \leq t < 2^p \quad (\text{IV.3})$$

$$P = \{x_t\} \text{ is feasible} \quad (\text{IV.4})$$

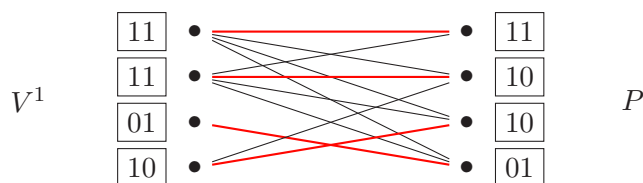


Figure 7.3: Example showing that satisfying demands of profile  $P$  with set  $V^1$  requires to find a perfect matching. The edges represent the domination between configurations.

Our objective is now to express the feasibility of a profile by using only these  $2^p$  variables. Roughly speaking, the idea to ensure the feasibility is the following. Let us suppose (with  $p = 2$  and  $n = 4$  for example) that there exists a feasible solution of fixed profile  $x_0 = 0, x_1 = 1, x_2 = 2, x_3 = 1$ . Suppose also that the first set is as depicted in Figure 7.3. To create a feasible solution with this profile, we have to “satisfy” (for each set  $V^i$ ) the demands  $x_t$  for all configurations  $t$ . For example in set 1, the demand  $x_2$  can be satisfied by using one vector in configuration 2 and one vector of configuration 3, and the demand 3 can be satisfied using the remaining vector of 3 (the demand  $x_0$  is clearly satisfied). Notice that a demand of a given configuration (*e.g.* configuration 2 here) can be satisfied using a vector that “dominates” this configuration (*e.g.* configuration 3 here). The notion of domination is introduced in Definition 7.4. Thus, a feasible profile implies that for any set  $i$  there exists a perfect matching between the vectors of  $V^i$  and the profile  $\{x_t\}$ .

Let us now define more formally the previous ideas.

**Definition 7.4** (Domination). *A  $p$ -dimensional vector  $v_1$  dominates a  $p$ -dimensional vector  $v_2$  (denoted by  $v_1 \gg v_2$ ) if and only if  $\forall l \in [p], v_2[l] = 1 \Rightarrow v_1[l] = 1$ .*

*A configuration  $t_1 \in [2^p - 1]_0$  dominates a configuration  $t_2 \in [2^p - 1]_0$  (denoted by  $t_1 \gg t_2$ ) if and only if  $B_{t_1} \gg B_{t_2}$  (recall that  $B_t$  is the  $p$ -dimensional binary representation of  $t$ ).*

*A solution  $S'$  dominates a solution  $S$  (denoted by  $S' \gg S$ ) if and only if  $\exists$  a bijection  $\phi : [n] \rightarrow [n]$  such that for any  $i \in [n], v_{s'_i} \gg v_{s_{\phi(i)}}$  (in other word, there is a*



one to one domination between stacks of  $S'$  and stacks of  $S$ ).

A profile  $P'$  dominates a profile  $P$  (denoted by  $P' \gg P$ ) if and only if there exists solutions  $S'$  and  $S$  such that  $Pr(S') = P'$ ,  $Pr(S) = P$  and  $S' \gg S$ .

**Definition 7.5.** For any  $i \in [m]$  and any  $t \in [2^p - 1]_0$ , let  $b_t^i$  be the number of vectors of set  $V^i$  in configuration  $t$ .

**Definition 7.6** (Graph  $G_P^i$ ). Let  $P$  be a profile not necessarily feasible. Let us denote as  $G_P^i = ((\Delta_P, \Lambda^i), E_{\gg})$ , the bipartite graph composed with set of vertices  $\Lambda^i = \{\lambda_t^{i,l}, 0 \leq t \leq 2^p - 1, 1 \leq l \leq b_t^i\}$ , being the set containing one vertex per vector in  $V^i$  and labeled according to their configuration, and set of vertices  $\Delta_P = \{\delta_t^l, 0 \leq t \leq 2^p - 1, 1 \leq l \leq x_t\}$ , containing one vertex per demand in profile  $P$  also labeled according to their configuration.

Let us fix a bijection  $f : \Delta_P \cup \Lambda^i \mapsto [2^p - 1]_0$ , that associates to each vertex  $\lambda_t^{i,l}$  and to each vertex  $\delta_t^l$  the vector in configuration  $t$ . Notice that  $|\Lambda^i| = |\Delta_P| = n$ . Finally, we set  $E_{\gg} = \{\{a, b\} | a \in \Delta_P, b \in \Lambda^i, f(a) \ll f(b)\}$ .

Intuitively, on one hand the vertices of  $\Lambda^i$  represent the vectors of  $V^i$ , on the other hand the vertices of  $\Delta_P$  represent the demands of profile  $P$  and an edge exists between a vertex of  $\Lambda^i$  and a vertex of  $\Delta_P$  if and only if the corresponding vector satisfies the corresponding demand. Thus the graph  $G_P^i$  aims at anonymizing the vectors of  $V^i$ . Indeed vectors of a same set sharing the same configuration will be represented in  $G_P^i$  by two vertices sharing the same neighborhood. Thus these vertices can be used indistinctly in the perfect matching we are trying to highlight.

We are now ready to show the following proposition.

**Proposition 7.1.** For any profile  $P = \{x_0, \dots, x_{2^p-1}\}$ ,

$$(\exists P' \text{ feasible, with } P' \gg P) \Leftrightarrow \forall i \in [m], \exists a \text{ matching of size } n \text{ in } G_P^i$$

Before starting the proof, notice that the simpler proposition “for any  $P$ ,  $P$  feasible  $\Leftrightarrow \forall i \in [m]$ , there is a matching of size  $n$  in  $G_P^i$ ” does not hold. Indeed,  $\Rightarrow$  is correct, but  $\Leftarrow$  is not: consider  $P$  with  $x_0 = n$  (recall that configuration 0 is the null vector), and an instance with  $nm$  "1 vectors" (containing only 1). In this case, there is a matching of size  $n$  in all the  $G_P^i$ , but  $P$  is not feasible. This explains the formulation of Proposition 7.1. An example of the correct formulation is depicted Figure 7.4.

*Proof.* Let  $P$  be a profile.

( $\Rightarrow$ ) Let  $P'$  be a feasible profile that dominates  $P$ . Let  $S = \{s_1, \dots, s_n\}$  and  $S' = \{s'_1, \dots, s'_n\}$  two solutions such that  $S'$  is feasible,  $Pr(S) = P$ ,  $Pr(S') = P'$  (notice that  $S$  and  $P$  are not necessarily feasible), and  $S' \gg S$ . *W.l.o.g.*, let us assume that  $\forall j, s'_j \gg s_j$  (*i.e.* the bijection  $\phi$  of Definition 7.4 is the identity), and let us assume that for any  $j$ ,  $s_j = (v_j^1, \dots, v_j^m)$ . Since  $v_j^i \in s'_j$ , then for any  $i$ , we know that  $v_j^i \gg s'_j \gg s_j, \forall j \in [n]$ . This implies a matching of size  $n$  in all the graphs  $G_P^i$ .

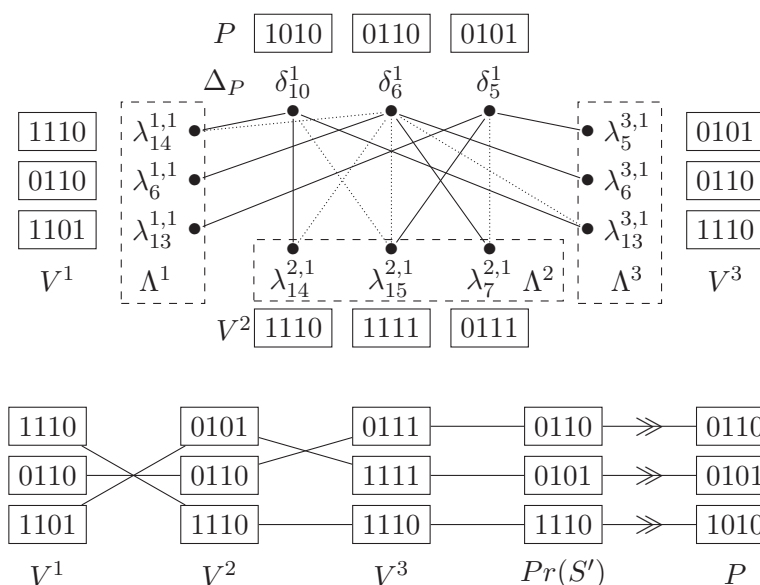


Figure 7.4: Illustration of Proposition 7.1 with  $m = n = 3$  and  $p = 4$ . Above: The three  $G_P^i$  graphs (edges are depicted by solid and dotted lines), and three matchings (in solid lines) corresponding to  $S'$ . Below: Solution  $S'$  s.t.  $Pr(S') \gg P$ .

( $\Leftarrow$ ) Let us suppose that  $\forall i \in [m]$ , there is a matching  $\mathcal{M}^i$  of size  $n$  in  $G_P^i$ .

*W.l.o.g.* let us rename  $\{\delta_1, \dots, \delta_n\}$  the vertices of  $\Delta_P$ , and  $\{\lambda_1^i, \dots, \lambda_n^i\}$  the vertices of  $\Lambda^i$  such that for any  $i$ ,  $\mathcal{M}^i = \{\{\lambda_1^i, \delta_1\}, \dots, \{\lambda_n^i, \delta_n\}\}$ . This implies  $f(\lambda_1^i) \gg f(\delta_1), \dots, f(\lambda_n^i) \gg f(\delta_n)$ . Let us define  $S = \{s_1, \dots, s_n\}$ , where  $\forall j \in [n]$ ,  $s_j = (f(\lambda_j^1), \dots, f(\lambda_j^m))$ . Notice that for any  $j$ ,  $s_j \gg f(\delta_j)$ , as all the  $f(\lambda_j^i) \gg f(\delta_j)$ , and combining two vectors  $f(\lambda_j^{i_1}) \gg f(\delta_j)$  and  $f(\lambda_j^{i_2}) \gg f(\delta_j)$  creates another vector that dominates  $f(\delta_j)$ . Thus,  $S$  is feasible, and  $Pr(S) \gg P$ , and we set  $P' = Pr(S)$ .  $\square$

Now, we can use the famous Hall's Theorem to express the existence of a matching in every set.

**Theorem 7.4** (Hall's Theorem). *Let  $G = ((V^1, V^2), E)$  be a bipartite graph with  $|V^1| = |V^2| = n$ . There is a matching of size  $n$  in  $G$  if and only if  $\forall \sigma \subseteq V^1$ ,  $|\sigma| \leq |\Gamma(\sigma)|$ , where  $\Gamma(\sigma) = \{v_2 \in V^2 : \exists v_1 \in \sigma \text{ such that } \{v_1, v_2\} \in E\}$ .*

**Remark 7.1.** *Notice that we cannot use Hall's Theorem directly on graphs  $G_P^i$ , as we would have to add the  $2^n$  constraints of the form  $\forall S \subseteq V^i$ . However, we will reduce the number of constraints to a function  $f(p)$  by exploiting the particular structure of  $G_P^i$ .*

**Proposition 7.2** (Matching in  $G_P^i$ ).  $\forall i \in [m]$ ,  $\forall P = \{x_0, \dots, x_{2^p-1}\}$ :  $(\forall \sigma \subseteq \Delta_P, |\sigma| \leq |\Gamma(\sigma)|) \Leftrightarrow (\forall \sigma_{cfcg} \subseteq [2^p - 1]_0, \sum_{t \in \sigma_{cfcg}} x_t \leq \sum_{t \in \text{dom}(\sigma_{cfcg})} b_t^i)$  where  $\text{dom}(\sigma_{cfcg}) = \{t' : \exists t \in \sigma_{cfcg} \text{ such that } t' \gg t\}$  is the set of configurations that dominate  $\sigma_{cfcg}$ .

*Proof.* ( $\Rightarrow$ ) Let  $\sigma_{cfg} = \{t_1, \dots, t_\alpha\}$ . Let  $\sigma = \{\delta_{t_i}^l, 1 \leq i \leq \alpha, 1 \leq l \leq x_{t_i}\}$  be the vertices of  $\Delta_P$  corresponding to the demands in  $\sigma_{cfg}$ . Observe that  $\sum_{t \in \sigma_{cfg}} x_t = |\sigma|$ . Notice also that  $\Gamma(\sigma) = \{\lambda_t^{i,l}, t \in \text{dom}(\sigma), 1 \leq l \leq b_t^i\}$  by construction. Thus,  $|\sigma| \leq |\Gamma(\sigma)|$  implies  $\sum_{t \in \sigma_{cfg}} x_t \leq \sum_{t \in \text{dom}(\sigma_{cfg})} b_t^i$ .

( $\Leftarrow$ ) Let  $\sigma \subseteq \Delta_P$ .  $\forall t \in [2^p - 1]_0$ , let  $X_t = \{\delta_t^l, 1 \leq l \leq x_t\}$ , let  $\sigma_t = \sigma \cap X_t$ . Let  $\sigma_{cfg} = \{t_1, \dots, t_\alpha\} = \{t : \sigma_t \neq \emptyset\}$ . Let  $X = \bigcup_{t \in \sigma_{cfg}} X_t$ . Notice that  $|\sigma| \leq |X| = \sum_{t \in \sigma_{cfg}} x_t$ .

Let us first prove that  $\Gamma(\sigma) = \Gamma(X)$ .  $\Gamma(\sigma) \subseteq \Gamma(X)$  is obvious. Now, if there is a  $\lambda_{t'}^{i,l'} \in \Gamma(X)$ , it means that there is a  $t \in \sigma_{cfg}$  such that  $\lambda_{t'}^{i,l'} \in \Gamma(X_t)$ , and thus there exists  $l$  such that  $\{\delta_t^l, \lambda_{t'}^{i,l'}\} \in E$  (which implies that  $t' \gg t$ ). As  $\sigma_t \neq \emptyset$ , there exists  $l'$  such that  $\delta_{t'}^{l'} \in \sigma_t$ , and  $\{\delta_{t'}^{l'}, \lambda_{t'}^{i,l'}\} \in E$  as  $t' \gg t$ .

Finally, the hypothesis with our set  $\sigma_{cfg}$  leads to

$$|\sigma| \leq |X| = \sum_{t \in \sigma_{cfg}} x_t \leq \sum_{t \in \text{dom}(\sigma_{cfg})} b_t^i = |\Gamma(X)| = |\Gamma(\sigma)|$$

□

Propositions 7.1 and 7.2 imply that for any profile  $P = \{x_0, \dots, x_{2^p-1}\}$ :

$$\exists p' \text{ feasible, with } P' \gg P \Leftrightarrow \forall i, \forall \sigma_{cfg} \subseteq [2^p - 1]_0, \sum_{t \in \sigma_{cfg}} x_t \leq \sum_{t \in \text{dom}(\sigma_{cfg})} b_t^i.$$

Thus, we use now the following ILP to describe the  $\max \sum 1$  problem:

---

**Integer Linear Program V** FPT-formulation with Hall constraints

---

$$\text{Maximize } \sum_{t=0}^{2^p-1} x_t c_t \tag{V.1}$$

$$\text{subject to } \sum_{t=0}^{2^p-1} x_t = n \tag{V.2}$$

$$x_t \in \mathbb{N} \quad \forall 0 \leq t < 2^p \tag{V.3}$$

$$\sum_{t \in \sigma_{cfg}} x_t \leq \sum_{t \in \text{dom}(\sigma_{cfg})} b_t^i \quad \forall i \in [m], \forall \sigma_{cfg} \subseteq [2^p - 1]_0 \tag{V.4}$$


---

This linear program has  $2^p$  variables and  $(m2^{2^p} + 2^p)$  constraints. Thus, we can solve it using [Kan83] in time  $f(p)\text{poly}(m)$ , we get that  $\max \sum 1$  and  $\min \sum 0$  are **FPT** parameterized by  $p$ . Using Property 7.2 this ILP leads to a  $\frac{p}{r}$ -approximation algorithm for  $\max \sum 1$  running in time  $f(r)\text{poly}(n + m + p)$ .

### 7.2.3 No polynomial size kernel with parameter $p$

We further the study of  $\max \sum 1$  when parameterized by  $p$ . In this section we show that even though the problem is in **FPT** when parameterized by  $p$ , it does not admit a polynomial-size kernel even if  $m = 3$ . Such a result is achieved by using an AND-cross-composition defined in Section 3.2.

**Theorem 7.5.** *Even for  $m = 3$ ,  $\max \sum 1$  when parameterized by  $p$  does not admit a polynomial kernel unless  $\mathbf{NP} \subseteq \mathbf{coNP}/\text{poly}$ .*

*Proof.* The proof is an AND-cross-composition inspired by the **NP**-hardness reduction for  $\max \sum 1$  provided by [Dok+12] and depicted in Section 6.1.1. More precisely, we cross-compose from a sequence of instances of 3-DIMENSIONAL PERFECT MATCHING. According to Definition 3.12 and Theorem 3.3, this will imply the desired result. 3-DIMENSIONAL PERFECT MATCHING is formally defined as follows:

---

#### Decision Problem 4 3-DIMENSIONAL PERFECT MATCHING (3-DPM)

---

**Input** Three sets  $X, Y$  and  $Z$  of size  $n$ , a set of hyperedges  $S \subseteq X \times Y \times Z$

**Question** Does there exist a subset  $S' \subseteq S$  such that:

- for all  $e, e' \in S'$  with  $e = (x, y, z)$  and  $e' = (x', y', z')$ , we have  $x \neq x', y \neq y'$  and  $z \neq z'$  (that is,  $S'$  is a matching)
  - $|S'| = n$  (that is,  $S'$  is perfect)
- 

Let  $(X_1, Y_1, Z_1, S_1), \dots, (X_t, Y_t, Z_t, S_t)$  be a sequence of  $t$  equivalent instances of 3-DPM, with respect to the following polynomial equivalence relation:  $(X, Y, Z, S)$  and  $(X', Y', Z', S')$  are equivalent if  $|X| = |X'|$  (and thus  $|Y| = |Y'| = |Z| = |Z'| = |X|$ ), and  $|S| = |S'|$ . In the following we denote by  $n$  the cardinality of the sets  $X_i$  (and equivalently the sets  $Y_i$  and  $Z_i$ ), and by  $m$  the cardinality of the sets  $S_i$ . Moreover, for all  $i \in \{1, \dots, t\}$  we define  $X_i = \{x_{i,1}, \dots, x_{i,n}\}$ ,  $Y_i = \{y_{i,1}, \dots, y_{i,n}\}$ ,  $Z_i = \{z_{i,1}, \dots, z_{i,n}\}$ , and  $S_i = \{s_{i,1}, \dots, s_{i,m}\}$ . We also assume that  $t = 2^q$  for some  $q \in \mathbb{N}$  (if it is not the case, we add a sufficiently number of dummy yes-instances).

In the following we construct three sets  $(X^*, Y^*, Z^*)$  of  $nt$  vectors each:  $X^* = \{x_{i,j}^* : i \in [t], j \in [n]\}$ ,  $Y^* = \{y_{i,j}^* : i \in [t], j \in [n]\}$  and  $Z^* = \{z_{i,j}^* : i \in [t], j \in [n]\}$ , where each vector is composed of  $p^* = m + 2mq$  components. Let us first describe the first  $m$  components of each vector. For all  $i \in [t]$ ,  $j \in [n]$  and  $k \in [m]$  we set:

$$x_{i,j}^*[k] = \begin{cases} 1 & \text{if the hyperedge } s_{i,k} \text{ contains } x_{i,j} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{i,j}^*[k] = \begin{cases} 1 & \text{if the hyperedge } s_{i,k} \text{ contains } y_{i,j} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{i,j}^*[k] = \begin{cases} 1 & \text{if the hyperedge } s_{i,k} \text{ contains } z_{i,j} \\ 0 & \text{otherwise} \end{cases}$$

Intuitively, if we restrict the transformation to the first  $m$  coordinates, the reduction is exactly the same as the one provided in [Dok+12] and depicted in Section 6.1.1.

However this only transformation does not ensure that the constructed instance of  $\max \sum 1$  is positive if and only if every instance of 3-DPM is positive. Indeed, as such vectors of  $\max \sum 1$  instance encoding different 3-DPM instances can be matched to get a better solution. Thus, for all  $i \in [t]$ , and for all  $j \in [j]$ , we append two vectors  $b_i$  and  $\bar{b}_i$  to all vectors  $x_{i,j}^*, y_{i,j}^*$  and  $z_{i,j}^*$ . The vector  $b_i$  is composed of  $mq$  coordinates, and is defined as the binary representation of the integer  $i$ , where each bit is duplicated  $m$  times. Finally,  $\bar{b}_i$  is obtained by taking the complement of  $b_i$  (*i.e.* replacing all zeros by ones, and conversely) as depicted in Figure 7.5. These suffixes artificially increase of  $m \log(t) = mq$  units the number of ones in representative vectors of stacks containing only vectors encoding a same 3-DPM instance, preventing the combination into stacks of vectors encoding different 3-DPM instances. Indeed, we will see that the cost of the stacks, defined by the number of zeros in its representative vector, greatly increase when mixing the vectors.

It is now clear that each vector  $x_{i,j}^*$  (*resp.*  $y_{i,j}^*, z_{i,j}^*$ ) is composed of  $p^* = m + 2mq$  coordinates. Thus, the parameter of the input instance is a polynomial in  $n, m$  and  $\log t$  whereas the total size of the instance is a polynomial in the size of the sequence of inputs, as required in cross-compositions. It now remains to prove that  $(X^*, Y^*, Z^*)$  contains an assignment of cost  $k^* = nt(mq + m - 1)$  if and only if for all  $i \in \{1, \dots, t\}$ ,  $S_i$  contains a perfect matching  $S'_i$ .

$\Leftarrow$  Suppose that for all  $i \in [t]$  we have a perfect matching  $S'_i \subseteq S_i$ . W.l.o.g. suppose that  $S'_i = \{s_{i,1}, \dots, s_{i,n}\}$ . Then, for each  $j \in [n]$ , we have  $s_{i,j} = (x_{i,j_1}, y_{i,j_2}, z_{i,j_3})$  for some  $j_1, j_2, j_3 \in [n]$ . We assign  $x_{i,j_1}^*$  with  $y_{i,j_2}^*$  and  $z_{i,j_3}^*$ . It is easy to see that the cost of this triple is  $m - 1 + mq$ . Indeed, they all have a one at the  $j^{\text{th}}$  coordinate, corresponding to the  $j^{\text{th}}$  hyperedge of  $S_i$  (and this is the only shared one, since we can suppose that all hyperedges are pairwise distinct), and they all contain the same vectors  $b_i$  and  $\bar{b}_i$ . Summing up for all instances, we get the desired solution value.

$\Rightarrow$  Conversely, first remark that in any assignment, the cost of every triple  $(x_{i_1,j_1}^*, y_{i_2,j_2}^*, z_{i_3,j_3}^*)$  is at least  $m - 1 + mq$ , and let us prove that this bound is tight when (1) all elements are chosen within the same instance, *i.e.*  $i_1 = i_2 = i_3 = i$ , and (2) this triple corresponds to an element of  $S_i$ , *i.e.*  $(x_{i,j_1}, y_{i,j_2}, z_{i,j_3}) \in S_i$ . Indeed, suppose first that  $i_1 \neq i_2$ . Then, since the binary representation of  $i_1$  and  $i_2$  differs on at least one bit, it is clear that the resulting vector is of cost

at least  $m(q + 1) > mq + m - 1$ . Now if  $i_1 = i_2 = i_3 = i$ , then the result is straightforward, since at most one hyperedge of  $S_i$  can contain  $x_{i_1, j_1}^*$ ,  $y_{i_2, j_2}^*$  and  $z_{i_3, j_3}^*$ . Finally, using the same arguments as previously, we can easily deduce a perfect matching  $S'_i \subseteq S_i$  for each  $i \in [t]$ , and the result follows.  $\square$

This result closes our study of the problem parameterized by  $p$ . It also ends the study of the problem with regard to the three natural parameters  $m$ ,  $n$  and  $p$ . In our opinion, the only important question left opened by this study is about the possibility to efficiently approximate  $\max \sum 1$  within a ratio  $f(m)$  with  $f$  being a polynomial-time computable function. Indeed, such an algorithm exists for  $\min \sum 0$  [Dok+12; DCS14], however the latter is unknown even for  $\max \sum 1$ .

It seems then natural to look for other structural parameters that could lead to efficient algorithms. This is the aim of the next section.

### 7.3 Resolution of sparse instances

In view of the previous study, the complexity of  $\max \sum 1$  with regard to its three natural parameters tends to be clarified. The membership of  $(\max \sum 1, p)$  in **FPT** gives us the intuition about the influence of the parameter  $p$  on the problem complexity. This intuition is strengthened by the **NP**-hardness of the problem even when  $n = 2$  or  $m = 3$ . It seems thus interesting to focus on structural parameters linked to the number of dies. We thus focus on the maximum number of coordinates set to zeros in a vector.

#### 7.3.1 EPTAS for $(\max \sum 1)_{\#0 \leq r}$

We take into consideration instances of  $\max \sum 1$  having a limited number of coordinates set to zero in each vector. The instances are called *sparse*. On these sparse instances, we get the following results:

**Corollary 7.1.** *For any fixed  $m$ ,  $(\max \sum 1)_{\#0 \leq r}$  admits an **EPTAS**.*

*Proof.* Let us consider  $I[m, n, p]$  an instance of  $(\max \sum 1)_{\#0 \leq r}$ , and a constant  $l > 1$ . We make out two possibilities:

- $p \leq lrm$ . In this case, the problem can be optimally solved in time  $f(l)poly(n + m)$  by using ILP (V).
- $p > lrm$ . In this case, the profit of each possible stack  $s$  is lower bounded by  $c(s) \geq p - rm > 0$ . Indeed, the number of coordinates set to zero is upper bounded by  $r$ . It follows that the profit of each vector  $v$  verifies  $c(v) \geq p - r$ . Since every stack  $s$  is an  $m$ -tuple of vectors, we get that  $c(s) \geq p - mr$ . Therefore any greedy algorithm  $A$  returns a solution  $S$  verifying  $c(S) \geq m(p - mr)$ .

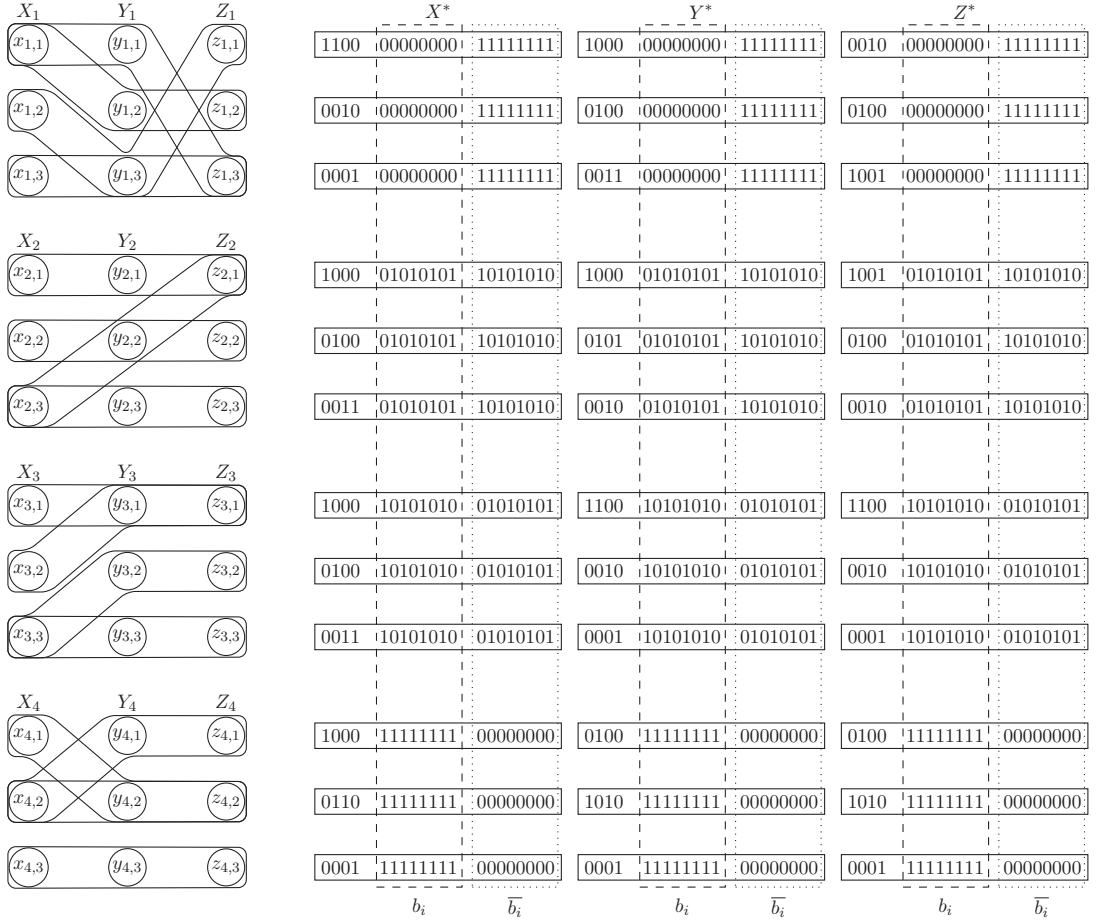


Figure 7.5: Example of construction of the AND-cross composition from four equivalent instances of 3-DPM:  $I_1 = (X_1, Y_1, Z_1, \{(x_{1,1}, y_{1,1}, z_{1,3}), (x_{1,1}, y_{1,2}, z_{1,2}), (x_{1,2}, y_{1,3}, z_{1,3}), (x_{1,3}, y_{1,3}, z_{1,3})\})$ ,  $I_2 = (X_2, Y_2, Z_2, \{(x_{2,1}, y_{2,1}, z_{2,1}), (x_{2,2}, y_{2,2}, z_{2,2}), (x_{2,3}, y_{2,3}, z_{2,3}), (x_{2,3}, y_{2,2}, z_{2,1})\})$ ,  $I_3 = (X_3, Y_3, Z_3, \{(x_{3,1}, y_{3,1}, z_{3,1}), (x_{3,2}, y_{3,1}, z_{3,1}), (x_{3,3}, y_{3,2}, z_{3,2}), (x_{3,3}, y_{3,3}, z_{3,3})\})$ ,  $I_4 = (X_4, Y_4, Z_4, \{(x_{4,1}, y_{4,2}, z_{4,2}), (x_{4,2}, y_{4,1}, z_{4,1}), (x_{4,2}, y_{4,2}, z_{4,2}), (x_{4,3}, y_{4,3}, z_{4,3})\})$  into an instance of BMVA with  $m = 3$ ,  $n = 12$  and  $p = 20$ .

On the other hand, the profit of every optimal solution  $S^*$  is upper bounded by  $np$ .  $(\max \sum 1)_{\#0 \leq r}$  being a maximization problem the performance ratio  $\rho$  of a solution  $S$  is defined as :

$$\rho = \frac{c(S^*)}{c(S)} \leq \frac{p}{p - rm} = 1 + \frac{rm}{p - rm} < 1 + \frac{rm}{(l-1)rm} = 1 + \frac{1}{l-1}$$

In this configuration, every polynomial time algorithm is a  $1 + \frac{1}{l-1}$ -approximation algorithm.

□

### 7.3.2 A polynomial-time algorithm for $(\max \sum 1)_{\#0 \leq 1}$ when $m$ is fixed

Previous section shows that the number of coordinates set to zero has a real impact on the approximability of the problem. However, as we will see in Section 8.1.2,  $(\min \sum 0)_{\#0 \leq 1}$  (and thus  $(\max \sum 1)_{\#0 \leq 1}$ ) remains **NP**-hard (even for  $n = 2$ ). It is thus natural to ask if  $(\max \sum 1)_{\#0 \leq 1}$  is polynomial-time solvable for fixed  $m$  but for general  $n$ . This section is devoted to answer positively to this question. Notice that we cannot extend this result to a more general notion of sparsity as  $(\min \sum 0)_{\#0 \leq 2}$  is **APX**-complete for  $m = 3$  [DCS14]. However, the question if  $(\max \sum 1)_{\#0 \leq 1}$  is fixed parameter tractable when parameterized by  $m$  is left open.

We first need some definitions, and refer the reader to Figure 7.6 where an example is depicted.

#### Definition 7.7.

- For any  $l \in [p], i \in [m]$ , we define  $B^{(l,i)} = \{v_j^i : v_j^i[l] = 0\}$  to be the set of vectors of set  $i$  that have their (unique) zero at position  $l$ . For the sake of homogeneous notation, we define  $B^{(p+1,i)} = \{v_j^i : v_j^i \text{ is a 1 vector}\}$ . Notice that the  $B^{(l,i)}$  form a partition of all the vectors of the input, and thus an input of  $(\max \sum 1)_{\#0 \leq 1}$  is completely characterized by the  $B^{(l,i)}$ .
- For any  $l \in [p+1]$ , the **block**  $B^l = \bigcup_{i \in [m]} B^{(l,i)}$ .

Informally, the idea to solve  $(\max \sum 1)_{\#0 \leq 1}$  in polynomial time for fixed  $m$  is to parse the input block after block using a dynamic programming algorithm. When arriving at block  $B^l$  we only need to remember for each  $c \subseteq [m]$  the number  $x_c$  of “partial stacks” that have only one vector for each  $V^i, i \in c$ . Indeed, we do not need to remember what is “inside” these partial stacks as all the remaining vectors from  $B^{l'}, l' \geq l$  cannot “match” (*i.e.* have their zero in the same position) the vectors in these partial stacks.

#### Definition 7.8.



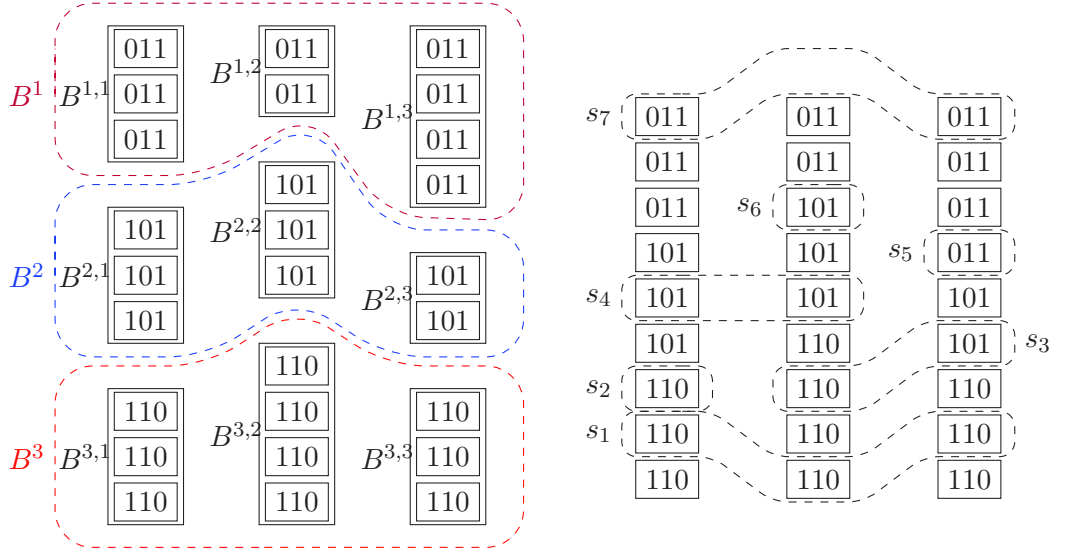


Figure 7.6: Left: An instance  $I$  of  $(\max \sum 1)_{\#0 \leq 1}$  partitioned into blocks. Right: A shape  $Sh = \{x_{\{0\}} = 2, x_{\{1\}} = 1, x_{\{2\}} = 1, x_{\{3\}} = 1, x_{\{1,2\}} = 1, x_{\{1,3\}} = 1, x_{\{2,3\}} = 1, x_{\{1,2,3\}} = 1\}$  encoding a set  $S$  of partial stacks of  $I$  containing two empty stacks. The support of  $s_7$  is  $\text{sup}(s_7) = \{1, 3\}$  and has cost  $c(s_7) = 1$ .

- A **partial stack**  $s = \{v_{i_1}^s, \dots, v_{i_k}^s\}$  of  $I$  is such that  $\{i_x \in [m], x \in [k]\}$  are pairwise disjoint, and for any  $x \in [k]$ ,  $v_{i_x}^s \in V^{i_x}$ . The **support** of a partial stack  $s$  is  $\text{sup}(s) = \{i_x, x \in [k]\}$ . Notice that a stack  $s$  (i.e. non partial) has  $\text{sup}(s) = [m]$ .
- The cost is extended in the natural way: the cost of a partial stack  $c(s) = c\left(\bigwedge_{x \in [k]} v_{i_x}^s\right)$  is the number of zeros of the bitwise AND of the vectors of  $s$ .

We define the notion of shape as follows:

**Definition 7.9.** A **shape**  $Sh = \{x_c, c \subseteq [m]\}$  is a set of  $2^m$  positive integers such that  $\sum_{c \subseteq [m]} x_c = n$ .

In the following, a shape will be used to encode a set  $S$  of  $n$  partial stacks by keeping a record of their support. In other words,  $x_c, c \subseteq [m]$  will denote the number of partial stacks in  $S$  of support  $c$ . This leads us to introduce the notion of reachable shape as follows:

**Definition 7.10.** Given two shapes  $Sh = \{x_c : c \subseteq [m]\}$  and  $Sh' = \{x'_c : c' \subseteq [m]\}$  and a set  $S = \{s_1, \dots, s_n\}$  of  $n$  partial stacks,  $Sh'$  is said reachable from  $Sh$  through  $S$  if and only if there exist  $n$  couples  $(s_1, c_1), (s_2, c_2), \dots, (s_n, c_n)$  such that:

- For each couple  $(s, c)$ ,  $\text{sup}(s) \cap c = \emptyset$ .
- For each  $c \subseteq [m]$ ,  $|\{(s_j, c_j) : c_j = c, j = 1, \dots, n\}| = x_c$ . Intuitively, the configuration  $c$  appears in exactly  $x_c$  couples.

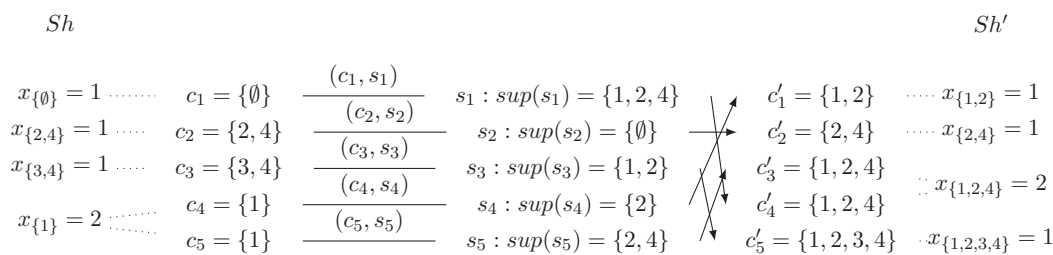


Figure 7.7: Example of a shape  $Sh' = \{x_{\{1,2\}} = 1, x_{\{2,4\}} = 1, x_{\{1,2,4\}} = 2, x_{\{1,2,3,4\}} = 1\}$  being reachable from  $Sh = \{x_{\{\emptyset\}} = 1, x_{\{1\}} = 2, x_{\{2,4\}} = 1, x_{\{3,4\}} = 1\}$  through  $S = \{s_1 : \text{sup}(s_1) = \{1, 2, 4\}, s_2 : \text{sup}(s_2) = \{\emptyset\}, s_3 : \text{sup}(s_3) = \{1, 2\}, s_4 : \text{sup}(s_4) = \{2\}, s_5 : \text{sup}(s_5) = \{2, 4\}\}$ .

- For each  $c' \subseteq [m]$ ,  $|\{(s_j, c_j) : \text{sup}(s_j) \cup c_j = c', j = 1, \dots, n\}| = x'_{c'}$ . Intuitively, there exist exactly  $x'_{c'}$  couples that, when associated, create a partial of shape  $c'$ .

Given two shapes  $Sh$  and  $Sh'$ ,  $Sh'$  is said reachable from  $Sh$ , if there exists a set  $S$  of  $n$  partial stacks such that  $Sh'$  is reachable from  $Sh$  through  $S$ .

Intuitively, a shape  $Sh'$  is reachable from  $Sh$  through  $S$  if every partial stack of the set encoded by  $Sh$  can be assigned to a unique partial stack from  $S$  to obtain a set of new partial stacks encoded by  $Sh'$ .

Remark that, given a set of partial stacks  $S$  only their shape is used to determine whether a shape is reachable or not. An example of a reachable shape is given on Figure 7.7.

We introduce now the following problem  $\Pi$ . We then show that this problem can be used to solve  $(\max \sum_{\#0 \leq 1} 1)$  problem, and we present a dynamic programming algorithm that solves  $\Pi$  in polynomial time when  $m$  is fixed.

---

### Optimization Problem 13 $\Pi$

---

**Input**  $(l, Sh)$  with  $l \in [p + 1]$ ,  $Sh$  a shape.

**Output** A set of  $n$  partial stacks  $S = \{s_1, s_2, \dots, s_n\}$  such that  $S$  is a partition of  $\mathcal{B} = \bigcup_{l' \geq l} B^{l'}$  and for every  $c \subseteq [m]$ ,  $|\{s \in S | \text{sup}(s) = [m] \setminus c\}| = x_c$ .

**Objective** Minimize  $c(S) = \sum_{j=1}^n c(s_j)$ .

---

Remark that an instance  $I$  of  $(\max \sum_{\#0 \leq 1} 1)$  can be solved optimally by solving optimally the instance  $I' = (1, Sh = \{x_{\{\emptyset\}} = n, x_c = 0, \forall c \neq \emptyset\})$  of  $\Pi$ . The optimal solution of  $I'$  is indeed a set of  $n$  partial disjoint stacks of support  $[m]$  of minimum cost.

We are now ready to define the following dynamic programming algorithm that solves any instance  $(l, Sh)$  of  $\Pi$  by parsing the instance block after block and branching for each of these blocks on every reachable shape.

---

**Function**  $\text{MinSumZeroDP}(l, Sh)$

---

**if**  $l == p + 1$  **then**  
     **return** 0;  
**return**  $\min(c(S') + \text{MinSumZeroDP}(l + 1, Sh'))$ , with  $Sh'$  reachable from  $Sh$   
     through  $S'$ , where  $S'$  partition of  $B^l$ ;

---

Note that this dynamic programming assumes the existence of a procedure that enumerates *efficiently* all the shapes  $Sh'$  that are reachable from  $Sh$ . The existence of such a procedure will be shown thereafter.

**Lemma 7.3.** *For any instance of  $\Pi$   $(l, Sh)$ ,  $\text{MinSumZeroDP}(l, Sh) = \text{Opt}(l, Sh)$ .*

*Proof.* Lemma 7.3 is true as in a given block  $l$ , the algorithm tries every reachable shape, and the zeros of vectors in blocks  $\mathcal{B} = \bigcup_{l' < l} B^{l'}$  cannot be matched with those of vectors in block  $\mathcal{B}' = \bigcup_{l' \geq l} B^{l'}$ . This is the reason why the support of the already created partial stacks (stored in shape  $Sh$ ) is sufficient to keep a record of what have been done (the positions of the zeros in the partial stacks corresponding to  $P$  is not relevant).  $\square$

Let us focus now on the procedure in charge of the enumeration of the reachable shape. A first and intuitive way to perform this operation is by guessing, for all  $c, c' \subseteq [m]$ ,  $y_{c,c'}$  the number of partial stacks in configuration  $c$  that will be turned into configuration  $c'$  with vectors of current block  $B^l$ . For each such guess it is possible to greedily verify that each  $y_{c,c'}$  can be satisfied with the vectors of the current block. As each of the  $y_{c,c'}$  can take values from 0 to  $n$  and  $c$  and  $c'$  can be both enumerated in  $\mathcal{O}^*(n^{2^m})$ , the previous algorithm runs in  $\mathcal{O}^*(n^{2^{2m}})$ .

This complexity can be improved as follows. The idea is to enumerate every possible shape  $Sh'$  and to verify using another dynamic programming algorithm if such a  $Sh'$  is reachable from  $Sh$ . We define  $\text{Aux}_{Sh'}(Sh, X)$ , that verifies if  $Sh'$  is reachable from  $Sh$  by using all vectors of  $X$ . If  $X = \emptyset$ , then the algorithm returns whether  $Sh$  is equal to  $Sh'$  or not. Otherwise, we consider the first vector  $v$  of  $X$  (we fix any arbitrary order) for which a branching is done on every possible assignment of  $v$ . More formally, the algorithm returns  $\bigvee_{c \subseteq [m], x_c > 0, c \cap \text{sup}(v) = \emptyset} \text{Aux}_{Sh'}(Sh_2 = \{x'_l\}, X \setminus \{v\})$ , where  $x'_l = x_l - 1$  if  $l = c$ ,  $x'_l = x_l + 1$  if  $l = c \cup \text{sup}(v)$ , and  $x'_l = x_l$  otherwise.

Using  $\text{Aux}$  in  $\text{MinSumZeroDP}$ , we get the following theorem.

**Theorem 7.6.**  $(\max \sum 1)_{\#0 \leq 1}$  can be solved in  $\mathcal{O}^*(n^{2^{m+2}})$ .

We compute the overall complexity as follows: for each of the  $pn^{2^m}$  possible values of the parameters of  $\text{MinSumZeroDP}$ , the algorithm tries the  $n^{2^m}$  shapes  $Sh'$ , and run for each one  $\text{Aux}_{Sh'}$  in  $\mathcal{O}^*(n^{2^m} nm)$  (the first parameter of  $\text{Aux}$  can take  $n^{2^m}$  values, and the second  $nm$  as we just encode how many vectors left in  $X$ ).

## 7.4 Conclusion and open questions

The problem has been mainly studied in regard to four important parameters:  $m$ ,  $n$ ,  $p$  and  $\#0$ . From the approximability point of view, the study of  $\max \sum 1$  is almost closed. Indeed, concerning  $n$ , we prove  $f(n)$ -inapproximability. Concerning  $p$ , we prove  $p^{1-\varepsilon}$ -inapproximability being somehow the best inapproximability we can hope for since we provide  $p$ -approximation algorithm and  $p/r$ -approximation algorithm that runs in polynomial time when  $r$  is fixed. Concerning  $m$ , we prove  $m^{1-\varepsilon}$ -inapproximability. However, the existence of an  $m$  (and even an  $f(m)$ ) approximation algorithm is left open.

If we consider the exact resolution of the problem, we show that the hardness of the problem was closely linked to the parameter  $p$  since we design an *FPT*-algorithm for  $\max \sum 1$  when parameterized by  $p$ . To a lesser extent, we show that  $\#0$  has also a certain influence on the complexity of  $\max \sum 1$  since when fixed, the problem can be efficiently approximated and with  $\#0$  fixed to 1,  $\max \sum 1$  belongs to **XP** when parameterized by  $m$ . Leading to another open question: does this problem belongs to **FPT**?

In practice, the positive results we highlight for  $\max \sum 1$  seems to be hardly tractable. Real life instances are composed of wafers on which up to 7000 dies are engraved. This leads to a parameters  $p$  being at most equal to 7000. Hence, we need to explore other trails to tackle these real instances.

In the next chapter, we explore one of this trail by studying the dual version of  $\max \sum 1$ :  $\min \sum 0$ .



# 8

## Minimizing the overall number of zeros

### Contents

<b>8.1</b>	<b>Approximability of <math>\min \sum 0</math></b>	<b>110</b>
8.1.1	$(n - \varepsilon)$ -inapproximability	110
8.1.2	Inapproximability without UGC	113
8.1.3	Approximation algorithm for $\min \sum 0$	116
<b>8.2</b>	<b>Parameterized complexity of <math>\min \sum 0</math></b>	<b>118</b>
8.2.1	$\mathcal{O}(k^2m)$ kernel for $(\min \sum 0, k)$	118
8.2.2	Positive results according to $\zeta_\beta$	119
8.2.3	Negative results according to $\zeta_\beta$	120
8.2.4	Positive results according to $\zeta_p$	121
8.2.5	Negative results according to $\zeta_p$	126
<b>8.3</b>	<b>Conclusion and open questions</b>	<b>127</b>

This chapter is devoted to the study of  $\min \sum 0$ , the dual version of  $\max \sum 1$ . The development of the problem analysis is quite similar to what has been previously done for  $\max \sum 1$ . In a first time we focus on the approximability of the problem.

We first highlight a Gap-reduction constructing from any instance of ALMOST  $Ek$  VERTEX COVER an instance of  $(\min \sum 0)_{\#0 \leq 1}$  proving that,  $(\min \sum 0)_{\#0 \leq 1}$  is **NP**-hard to approximate within a ratio  $(n - \varepsilon)$  unless UGC fails. We then provide a Linear reduction from a particular case of ODD CYCLE TRANSVERSAL proving that the problem is **APX**-hard even if  $n = 2$  and  $\#0 \leq 1$  unless **P** = **NP**.

Eventually we focus on the complexity of the problem. Contrary to  $\max \sum 1$  there is no reduction, like the one from MAX CLIQUE, proving that  $\min \sum 0$  when param-

eterized by the standard parameter is  $\mathbf{W}[1]$ -hard. We indeed show that it admits a kernel of size  $\mathcal{O}(k^2m)$ , and thus that it is in  $\mathbf{FPT}$ . Based on this result, it is possible to obtain interesting results by subtracting to the objective function a known lower bound of it. For instance, if one can prove that any solution of a given minimization problem is of cost at least  $\beta$ , then one can ask for a solution of cost  $\beta + c$  and parameterize by  $c$ . This idea, called *above guarantee parameterization* was introduced by [MR99] and first applied to MAX SAT and MAX CUT problems. It then became a fruitful line of research with similar results obtained for many other problems (among others, see [Gut+13; MRS09; Cyt+13; GY12]).

The parameterized complexity of  $\min \sum 0$  is hence divided in two parts in function of the type of considered parameters:

1. the standard parameter  $k$ ,
2. two above guarantee parameters.

As showed in Section 7.2.2,  $\min \sum 0$  is *FPT* parameterized by  $p$ . As we will notice in Lemma 8.4 that we can obtain  $p \leq k$  after a polynomial pre-processing step, this implies that  $\min \sum 0$  is also *FPT* with its standard parameter. Our idea here is to use this previous inequality in order to obtain smaller parameters. Thus, we define our first above guarantee parameter  $\zeta_p = k - p$ .

Finally, in order to define our last parameter, we first need to describe the corresponding lower bound  $\beta$ , that will represent the maximum, over all sets of vectors, of the total number of zeros for each set. More formally, we define  $\beta = \max_{i \in [m]} c(V^i)$  where  $c(V^i) = \sum_{j=1}^m c(v_j^i)$ . Since we perform a bit-wise AND over each  $m$ -tuple, it is easily seen that any solution will be of cost at least  $\beta$ . Thus, we define our last parameter  $\zeta_\beta = k - \beta$ .

We prove that  $\min \sum 0$  can be solved in  $O^*(4^{\zeta_\beta \log(n)})$ , while it is  $\mathbf{W}[2]$ -hard when parameterized by  $\zeta_\beta$  only, and cannot be solved in  $O^*(2^{\zeta_\beta \log(n)})$  nor in  $O^*(2^{\zeta_\beta o(\log(n))})$  assuming *ETH*. We then focus on the parameterization by  $\zeta_p$ : we show that when  $n = 2$ , the problem can be solved in single exponential time with this parameter, but is not in  $\mathbf{XP}$  for any fixed  $n \geq 3$  (unless  $\mathbf{P} = \mathbf{NP}$ ). The reduction we use also shows that for fixed  $n \in \mathbb{N}$ , the problem cannot be solved in  $2^{o(k)}$  (and thus in  $2^{o(\zeta_\beta)}$ ) unless *ETH* fails.

An overview of these results is given on Figure 8.1.

These results are published in [Bou+15; BDG16].

## 8.1 Approximability of $\min \sum 0$

### 8.1.1 $(n - \varepsilon)$ -inapproximability

In this section we consider the ALMOST  $Ek$  VERTEX COVER problem. Recall that we call a vertex cover in a  $k$ -regular hypergraph  $H = (U, E)$  a set  $U' \subseteq U$  such that for any hyperedge  $e \in E$ ,  $U' \cap e \neq \emptyset$ .

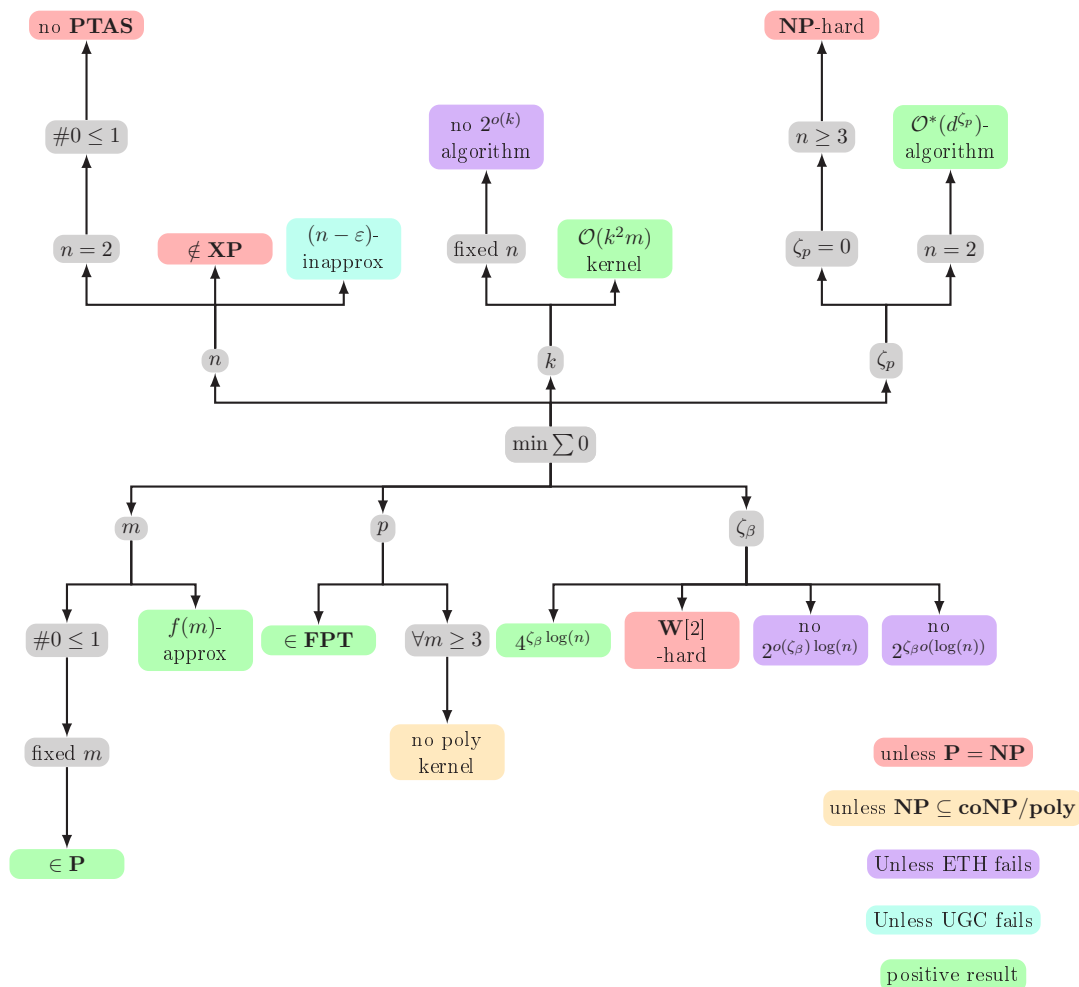


Figure 8.1: Overview of the results for  $\min \sum 0$



---

**Decision Problem 5** ALMOST  $Ek$  VERTEX COVER
 

---

**Input** We are given an integer  $k \geq 2$ , two arbitrary positive constants  $\varepsilon$  and  $\delta$  and a  $k$ -uniform hypergraph  $G = (U, E)$ .

**Question** Distinguish between the following cases:

**YES Case** there exist  $k$  disjoint subsets  $U^1, U^2, \dots, U^k \subseteq U$ , satisfying  $|U^i| \geq \frac{1-\varepsilon}{k}|U|$  and such that every hyperedge contains at most one vertex from each  $U^i$ .

**NO Case** every vertex cover has size at least  $(1 - \delta)|U|$ .

---

It is shown in [BK10] that, assuming UGC, this problem is **NP**-complete.

**Theorem 8.1.** *For any fixed  $n \geq 2$ , for any constants  $\varepsilon, \delta > 0$ , there exists a  $\frac{n-n\delta}{1+n\varepsilon}$ -Gap reduction from ALMOST  $Ek$  VERTEX COVER to  $(\min \sum 0)_{\#0 \leq 1}$ . Consequently, under UGC, for any fixed  $n$ ,  $(\min \sum 0)_{\#0 \leq 1}$  is hard to approximate within a factor  $(n - \varepsilon')$  for any  $\varepsilon' > 0$ .*

*Proof.* We consider an instance  $I$  of ALMOST  $Ek$  VERTEX COVER defined by two positive constants  $\delta$  and  $\varepsilon$ , an integer  $k$  and a  $k$ -regular hypergraph  $H = (U, E)$ .

We use the function  $f$  defined in Section 6.1.4 to construct an instance  $f(I)$  of  $(\min \sum 0)_{\#0 \leq 1}$ . Recall that this reduction associates to each hyperedge  $e$  of  $H$  a set  $V^e$ . Each set contains  $k$  vectors, one encoding each endpoint of  $e$ . A vector  $v$  encodes a vertex  $u$  if and only if  $v[l] = 0$  if  $l = u$  and  $v[l] = 1$  otherwise. Let us now prove that if  $I$  is a positive instance,  $f(I)$  admits a solution  $S$  of cost  $c(S) < (1 + n\varepsilon)|U|$ , and otherwise any solution  $S$  of  $f(I)$  has cost  $c(S) \geq n(1 - \delta)|U|$ .

**NO Case** Let  $S$  be a solution of  $f(I)$ . Let us first remark that for any stack  $s \in S$ , the set  $\{l : v_s[l] = 0\}$  defines a vertex cover in  $H$ . Indeed,  $s$  contains exactly one vector per set, and thus by construction  $s$  selects one vertex per hyperedge in  $H$ . Remark also that the cost of  $s$  is equal to the size of the corresponding vertex cover.

Now, suppose that  $I$  is a negative instance. Hence each vertex cover has a size at least equal to  $(1 - \delta)|U|$ . Since any solution  $S$  of  $f(I)$  consists of exactly  $n$  stacks,  $S$  verifies  $c(S) \geq n(1 - \delta)|U|$ .

**YES Case** If  $I$  is a positive instance, there exists  $k$  disjoint sets  $U^1, U^2, \dots, U^k \subseteq U$  such that,  $\forall i \in [k]$ ,  $|U^i| \geq \frac{1-\varepsilon}{k}|U|$  and such that every hyperedge contains at most one vertex from each  $U^i$ .

We introduce the subset  $X = U \setminus \bigcup_{i=1}^k U^i$ . By definition  $\{U^1, U^2, \dots, U^k, X\}$  is a partition of  $U$  and  $X \leq \varepsilon|U|$ . Furthermore,  $U^i \cup X$  is a vertex cover  $\forall i = 1, \dots, k$ .

Indeed, each hyperedge  $e \in E$  that contains no vertex of  $U^i$ , contains at least one vertex of  $X$  since  $e$  contains  $k$  vertices.

We now construct a solution  $S$  of  $f(I)$ . Our objective is to construct stacks  $\{s_i\}$  such that for any  $i$ , the zeros of  $s_i$  are included in  $U^i \cup X$  (i.e.  $\{l : v_{s_i}[l] = 0\} \subseteq U^i \cup X$ ). For each  $e = \{u_1, \dots, u_k\} \in E$ , we show how to assign exactly one vector of  $V^e$  to each stack  $s_1, \dots, s_k$ . For all  $i \in [k]$ , if  $v_j^e$  represents a vertex  $u$  with  $u \in U^i$ , then we assign  $v_j^e$  to  $s_i$ . *W.l.o.g.*, let  $S'_e = \{s_1, \dots, s_{k'}\}$  (for  $k' \leq k$ ) be the set of stacks that received a vertex during this process. Notice that as every hyperedge contains at most one vertex from each  $U^i$ , we only assigned one vector to each stack of  $S'_e$ . After this, every unassigned vector  $v \in V^e$  represents a vertex of  $X$  (otherwise, such a vector  $v$  would belong to a set  $U^i$ ,  $i \in k'$ , a contradiction). We assign arbitrarily these vectors to the remaining stacks that are not in  $S'_e$ . As by construction  $\forall i \in [k]$ ,  $v_{s_i}$  contains only vectors representing vertices from  $U^i \cup X$ , we get  $c(s_i) \leq |U^i| + |X|$ .

Thus, we obtain a feasible solution  $S$  of cost  $c(S) = \sum_{i=1}^k c(s_i) \leq k|X| + \sum_{i=1}^k |U^i|$ . As by definition we have  $|X| + \sum_{i=1}^k |U^i| = |U|$ , it follows that  $c(S) \leq |U| + (k-1)\varepsilon|U|$  and since  $k = n$ ,  $c(S) < |U|(1 + n\varepsilon)$ .

According to the Definition 2.31 of the Gap-reduction, if we define  $a(n) = (1 + n\varepsilon)|U|$  and  $r(n) = \frac{n(1-\delta)}{(1+n\varepsilon)}$ , the previous reduction is a  $r(n)$ -Gap reduction. Furthermore,  $\lim_{\delta, \varepsilon \rightarrow 0} r(n) = n$ , thus it is **NP**-hard to approximate  $(\min \sum 0)_{\#0 \leq 1}$  within a ratio  $(n - \varepsilon')$  for any  $\varepsilon' > 0$ .  $\square$   $\square$

Notice that, as a function of  $n$ , this inapproximability result is optimal. Indeed, we observe that any feasible solution  $S$  is an  $n$ -approximation as, for any instance  $I$  of  $\min \sum 0$ . Recall indeed that, according to Hypothesis 4.4, we assume  $\forall l \in [p], \exists i, \exists j$  such that  $v_j^i[l] = 0$ . It follows that  $c(\text{opt}(I)) \geq p$  and for any solution  $S$ ,  $c(S) \leq pn$ .

### 8.1.2 Inapproximability without UGC

Let us now study the negative results we can get when only assuming  $\mathbf{P} \neq \mathbf{NP}$ . Our objective is to prove that  $(\min \sum 0)_{\#0 \leq 1}$  is **APX**-hard, even for  $n = 2$ . To do so, we present a reduction from **ODD CYCLE TRANSVERSAL**, which is defined as follows. Given an input graph  $G = (U, E)$ , the objective is to find an odd cycle transversal of minimum size, i.e. a subset  $T \subseteq U$  of minimum size such that  $G[U \setminus T]$  is bipartite. The problem can be formally defined as follows.

For any integer  $\gamma \geq 2$ , we denote  $\mathcal{G}_\gamma$  the class of graphs  $G = (U, E)$  such that any optimal odd cycle transversal  $T$  has size  $|T| \geq \frac{|U|}{\gamma}$ . Given  $\mathcal{G}$  a class of graphs, we denote  $OCT_{\mathcal{G}}$  the **ODD CYCLE TRANSVERSAL** problem restricted to  $\mathcal{G}$ .

**Lemma 8.1.** *For any constant  $\gamma \geq 2$ ,  $OCT_{\mathcal{G}_\gamma}$   $L$ -reduces to  $(\min \sum 0)_{\#0 \leq 1}$  with  $n = 2$ .*

---

**Optimization Problem 14** ODD CYCLE TRANSVERSAL
 

---

<b>Input</b>	A graph $G = (U, E)$ .
<b>Output</b>	An odd cycle transversal $T$ .
<b>Objective</b>	Minimize $ T $ .

---

*Proof.* Let us consider an integer  $\gamma$ , an instance  $I$  of  $OCT_{\mathcal{G}_\gamma}$ , defined by a graph  $G = (U, E)$  such that  $G \in \mathcal{G}_\gamma$ . *W.l.o.g.*, we can consider that  $G$  contains no isolated vertex.

Remark that any graph can be seen as a 2-uniform hypergraph. Thus, we use the function  $f$  defined in Section 6.1.4 to construct an instance  $f(I)$  of  $(\min \sum 0)_{\#0 \leq 1}$  such that  $n = 2$ . Since,  $G$  contains no isolated vertex,  $f(I)$  contains no position  $l$  such that  $\forall i \in [m], \forall j \in [n], v_j^i[l] = 1$ .

Let us now prove that  $I$  admits an odd cycle transversal of size  $t$  if and only if  $f(I)$  admits a solution of cost  $p + t$ .

$\Leftarrow$  We consider an instance  $f(I)$  of  $(\min \sum 0)_{\#0 \leq 1}$  with  $n = 2$  admitting a solution  $S = \{s_A, s_B\}$  with cost  $c(S) = p + t$ . Let us specify a function  $g$  which produces from  $S$  a solution  $T = g(I, S)$  of  $OCT_{\mathcal{G}_\gamma}$ , *i.e.* a set of vertices of  $U$  such that  $G[U \setminus T]$  is bipartite.

We define  $T = \{u \in U : v_{s_A}[u] = v_{s_B}[u] = 0\}$ , the set of coordinates equal to zero in both  $s_A$  and  $s_B$ . We also define  $A = \{u \in V : v_{s_A}[u] = 0 \text{ and } v_{s_B}[u] = 1\}$  (resp.  $B = \{u \in V : v_{s_B}[u] = 0 \text{ and } v_{s_A}[u] = 1\}$ ), the set of coordinates set to zero only in  $s_A$  (resp.  $s_B$ ). Notice that  $\{T, A, B\}$  is a partition of  $U$ .

Remark that  $A$  and  $B$  are independent sets. Indeed, suppose that  $\exists \{u, v\} \in E$  such that  $u, v \in A$ . As  $\{u, v\} \in E$  there exists a set  $V^{(u,v)}$  containing a vector that represents  $u$  and another vector that represents  $v$ , these vectors are assigned to different stacks. This leads to a contradiction. It follows that  $G[U \setminus T]$  is bipartite and  $T$  is an odd cycle transversal.

Since  $c(S) = |A| + |B| + 2|T| = p + |T| = p + t$ , we get  $|T| = t$ .

$\Rightarrow$  We consider an instance  $I$  of  $OCT_{\mathcal{G}_\gamma}$  and a solution  $T$  of size  $t$ . We now construct a solution  $S = \{s_A, s_B\}$  of  $f(I)$  from  $T$ .

By definition,  $G[U \setminus T]$  is a bipartite graph, thus the vertices in  $U \setminus T$  may be split into two disjoint independent sets  $A$  and  $B$ . For each edge  $e \in E$ , the following cases can occur:

- if  $\exists u \in e$  such that  $u \in A$ , then the vector corresponding to  $u$  is assigned to  $s_A$ , and the vector corresponding to  $e \setminus \{u\}$  is assigned to  $s_B$  (and the same rule holds by exchanging  $A$  and  $B$ )
- otherwise,  $u$  and  $v \in T$ , and we assign arbitrarily  $v_u^e$  to  $s_A$  and the other to  $s_B$ .

We claim that the stacks  $s_A$  and  $s_B$  describe a feasible solution  $S$  of cost at most  $p + t$ .

Since, for each set, only one vector is assigned to  $s_A$  and the other to  $s_B$ , the two stacks  $s_A$  and  $s_B$  are disjoint and contain exactly  $m$  vectors.  $S$  is therefore a feasible solution.

Remark that  $v_{s_A}$  (resp.  $v_{s_B}$ ) contains only vectors  $v$  such that  $v[l] = 0 \Rightarrow l \in A \cup T$  (resp.  $l \in B \cup T$ ), and thus  $c(v_A) \leq |A| + |T|$  (resp.  $c(v_B) \leq |B| + |T|$ ). Hence  $c(S) \leq |A| + |B| + 2|T| = p + t$ .

Let us now prove that this reduction is an  $L$ -reduction.

1. By definition, any instance  $I$  of  $OCT_{\mathcal{G}_\gamma}$  verifies  $|\text{opt}(I)| \geq |U|/\gamma$ . Thus,

$$c(\text{opt}(f(I))) \leq |U| + |\text{opt}(I)| \leq (\gamma + 1)|\text{opt}(I)|$$

2. We consider an arbitrary instance  $I$  of  $OCT_{\mathcal{G}_\gamma}$ ,  $f(I)$  the corresponding instance of  $(\min \sum 0)_{\#0 \leq 1}$ ,  $S$  a solution of  $f(I)$  and  $T = g(I, S)$  the corresponding solution of  $I$ .

$$\text{We proved } |T| - |\text{opt}(I)| = c(S) - |U| - c(\text{opt}(f(I))) - |U| = c(S) - c(\text{opt}(f(I))).$$

Therefore, we get an  $L$ -reduction for  $\alpha = \gamma + 1$  and  $\beta = 1$ . □

**Lemma 8.2.** *There exists a constant  $\gamma$  and  $\mathcal{G} \subset \mathcal{G}_\gamma$  such that  $OCT_{\mathcal{G}}$  is **APX**-hard.*

*Proof.* We present an  $S$ -reduction from VC-3, the VERTEX COVER problem on graph with maximum degree 3, to  $OCT_{\mathcal{G}_{VC}}$  for an appropriate  $\mathcal{G}_{VC}$ . VC-3 is known to be **APX**-complete [AK97].

Let us define the functions  $f$  and  $g$  depicted in the Definition 2.26. Given an instance  $G = (U, E)$  of VC-3, we construct an instance  $f(G) = (U', E')$  as follows:

1. For each  $\{u, v\} \in E$ , create a vertex  $z_{u,v}$ . These  $z$ -vertices form the set  $Z$ .
2.  $U' = U \cup Z$ .
3.  $E' = E \cup \{(u, z_{u,v}), (v, z_{u,v}) : \{u, v\} \in E\}$ . In other words, for each  $\{u, v\} \in E$ , we create the triangle  $\{u, v, z_{u,v}\}$ .

Let us prove that  $G = (U, E)$  admits a solution  $VC$  of size  $|VC| = t$  if and only if  $f(G)$  admits a solution  $T$  of size  $|T| = t$ .

$\Rightarrow$  Consider a vertex cover  $VC$  of size  $|VC| = t$ , for each  $u \in VC$ , we add the vertex  $u'$  to  $T$ . By definition,  $VC$  covers all the edges of  $G$  and then all its (odd) cycles. Furthermore, it also covers all the created triangles in  $f(G)$  since each of these cycles contains exactly one edge in common with  $f(G)[U' \setminus Z]$ . Thus  $T$  is an odd cycle transversal and  $|T| = |VC|$ .

⇐ Let us construct a function  $g$  that, given any solution  $T$  of  $f(G)$ , computes a solution  $VC = g(G, T)$  of  $G$ . Notice first that we can suppose that  $T$  contains no  $z$ -vertex. Otherwise every triangle  $\{u, v, z_{u,v}\}$  covered by a  $z_{u,v} \in T$ , can instead be covered by either  $u$  or  $v$  without increasing the size of  $T$ . Thus, we set  $VC = T$ .

By definition of an odd cycle transversal,  $T$  covers all the odd cycles of  $f(G)$  and especially the created triangles. Thus, the triangle  $\{u, v, z_{u,v}\}$  corresponding to any edge  $\{u, v\} \in E$  is covered by  $VC$ . As  $VC \cap Z = \emptyset$ ,  $VC$  is a vertex cover of  $G$ .

The reduction  $(f, g)$  is an  $S$ -reduction. Let us call  $\mathcal{G}_{VC}$  the class of graph generated in this reduction. The previous reduction shows that  $OCT_{\mathcal{G}_{VC}}$  is **APX**-hard. It remains to check that  $\mathcal{G}_{VC} \subseteq \mathcal{G}_\gamma$  for a constant  $\gamma$ .

Remark that VC-3 is only defined on 3-regular graphs, it implies that for any instance  $G = (U, E)$  of VC-3,  $Opt(G) \geq \frac{|U|}{3}$ . As  $|U'| = |U| + |E| \leq \frac{5|U|}{2}$ , it follows that:

$$|opt(f(G))| = |opt(G)| \geq \frac{|U|}{3} \geq \frac{2|U'|}{15}$$

Hence,  $\mathcal{G}_{VC} \subset \mathcal{G}_\gamma$  with  $\gamma = \frac{15}{2}$ . □

The following result is now immediate.

**Theorem 8.2.**  $(\min \sum 0)_{\#0 \leq 1}$  is **APX**-hard, even for  $n = 2$ .

### 8.1.3 Approximation algorithm for $\min \sum 0$

Let us now show an example of an algorithm achieving a  $n - f(n, m)$  ratio. Notice that the  $(n - \epsilon)$ -inapproximability result holds for fixed  $n$  and  $\#0 = 1$ , while the following algorithm is polynomial-time computable when  $n$  is part of the input and  $\#0$  is arbitrary.

**Proposition 8.1.** *There is a polynomial-time  $n - \frac{n-1}{n\rho(n,m)}$  approximation algorithm for  $\min \sum 0$ , where  $\rho(n, m) > 1$  is the approximation ratio for independent set in graphs that are the union of  $m$  complete  $n$ -partite graphs.*

*Proof.* Let  $I$  be an instance of  $\min \sum 0$ . Let us now consider an optimal solution  $opt(I) = \{s_1^*, \dots, s_n^*\}$  of  $I$ . For any  $i \in [n]$ , let  $Z_i^* = \{l \in [p] : v_{s_i^*}[l] = 0 \text{ and } v_{s_t^*}[l] = 1, \forall t \neq i\}$  be the set of coordinates equal to zero only in stack  $s_i^*$ . Let  $\Delta = \sum_{i=1}^n |Z_i^*|$ . Notice that we have  $c(opt(I)) \geq \Delta + 2(p - \Delta)$ , as for any coordinate  $l$  outside  $\bigcup_i Z_i^*$ , there are at least two stacks with the  $l^{th}$  coordinate set to zero. *W.l.o.g.*, let us suppose that  $Z_1^*$  is the largest set among  $\{Z_i^*\}$ , implying  $|Z_1^*| \geq \frac{\Delta}{n}$ .

Given a subset  $Z \subset [p]$ , we will construct a solution  $S = \{s_1, \dots, s_n\}$  such that for any  $l \in Z$ ,  $v_{s_1}[l] = 0$ , and for any  $i \neq 1$ ,  $v_{s_i}[l] = 1$ . Informally, the zero at coordinates  $Z$  will appear only in  $s_1$ , which behaves as a "trash" stack. The cost of such a solution

is  $c(S) \leq c(s_1) + \sum_{i=2}^n c(s_i) \leq p + (n-1)(p - |Z|)$ . Our objective is now to compute such a set  $Z$ , and to lower bound  $|Z|$  according to  $|Z_1^*|$ .

Let us define how we compute  $Z$ . Let  $P = \{l \in [p] : \forall i \in [m], |\{j : v_j^i[l] = 0\}| \leq 1\}$  be the subset of coordinates that are never nullified in two different vectors of the same set. We will construct a simple undirected graph  $G = (P, E)$ , and thus it remains to define  $E$ . For vector  $v_j^i$ , let  $Z_j^i = Z(v_j^i) \cap P$ , where  $Z(v) \subseteq [p]$  denotes the set of null coordinates of vector  $v$ . For any  $i \in [m]$ , we add to  $G$  the edges of the complete  $n$ -partite graph  $G^i = (\{Z_1^i \times \dots \times Z_n^i\})$  (i.e. for any  $j_1, j_2, v_1 \in Z_{j_1}^i, v_2 \in Z_{j_2}^i$ , we add edge  $\{v_1, v_2\}$  to  $G$ ). This concludes the description of  $G$ , which can be seen as the union of  $m$  complete  $n$ -partite graphs.

Let us now see the link between independent set in  $G$  and our problem. Let us first see why  $Z_1^*$  is a independent set in  $G$ . Recall that by definition of  $Z_1^*$ , for any  $l \in Z_1^*, v_{s_1^*}[l] = 0$ , but  $v_{s_j^*}[l] = 1, j \geq 2$ . Thus, it is immediate that  $Z_1^* \subseteq P$ . Moreover, assume by contradiction that there exists an edge in  $G$  between to vertices  $l_1$  and  $l_2$  of  $Z_1^*$ . This implies that there exists  $i \in [m], j_1$  and  $j_2 \neq j_1$  such that  $v_{j_1}^i[l_1] = 0$  and  $v_{j_2}^i[l_2] = 0$ . As by definition of  $Z_1^*$  we must have  $v_{s_j^*}[l_1] = 1$  and  $v_{s_j^*}[l_2] = 1$  for  $j \geq 2$ , this implies that  $s_1^*$  must contains both  $v_{j_1}^i$  and  $v_{j_2}^i$ , a contradiction. Thus, we get  $\text{opt}(G) \geq |Z_1^*|$ , where  $\text{opt}(G)$  is the size of a maximum independent set in  $G$ .

Now, let us check that for any independent set  $Z \subseteq P$  in  $G$ , we can construct a solution  $S = \{s_1, \dots, s_n\}$  such that for any  $l \in Z, v_{s_1}[l] = 0$ , and for any  $i \neq 1, v_{s_i}[l] = 1$ . To construct such a solution, we have to prove that we can add in  $s_1$  all the vectors  $v$  such that  $\exists l \in Z$  such that  $v[l] = 0$ . However, this last statement is clearly true as for any  $i \in [m]$ , there is at most one vector  $v_j^i$  with  $Z(v_j^i) \subseteq Z$ .

Thus, any  $\rho(n, m)$  approximation algorithm gives us a set  $Z$  with:

$$|Z| \geq \frac{|Z_1^*|}{\rho(n, m)} \geq \frac{\Delta}{n\rho(n, m)}$$

Therefore, we get a ratio of:

$$\frac{p + (n-1)(p - \frac{\Delta}{n\rho(n, m)})}{2p - \Delta} \leq n - \frac{n-1}{n\rho(n, m)} \quad \text{for } \Delta = p$$

□

**Remark 8.1.** We can get, for example,  $\rho(n, m) = mn^{m-1}$  using the following algorithm. Given an instance  $I$ , for any  $i \in [m]$ , let  $G^i = (A_1^i, \dots, A_n^i)$  be the  $i$ -th complete  $n$ -partite graph. W.l.o.g., suppose that  $A_1^1$  is the largest set among  $\{A_j^i\}$ . Notice that  $|A_1^1| \geq \frac{\text{opt}(I)}{m}$ . The algorithm starts by setting  $S_1 = A_1^1$  ( $S_1$  may not be an independent set). Then, for any  $i$  from 2 to  $m$ , the algorithm set  $S_i = S_{i-1} \setminus (\cup_{j \neq j_0} A_j^i)$ , where  $j_0 = \arg \max_j \{|S_{i-1} \cap A_j^i|\}$ . Thus, for any  $i$  we have  $|S_i| \geq \frac{|S_{i-1}|}{n}$ , and  $S_i$  is an independent set when considering only edges from  $\cup_{l=1}^i G^l$ . Finally, we get an independent set of  $G$  of size  $|S_m| \geq \frac{S_1}{n^{m-1}} \geq \frac{\text{opt}(I)}{mn^{m-1}}$ .

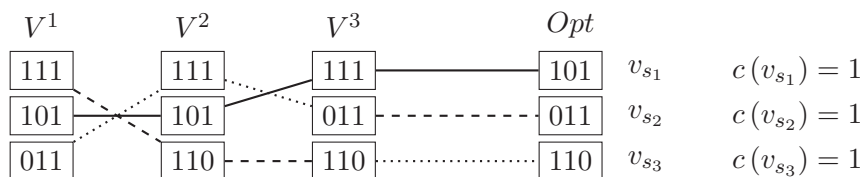


Figure 8.2: Example of  $\min \sum 0$  instance such that no optimal solution creates a 1-stack.

## 8.2 Parameterized complexity of $\min \sum 0$

### 8.2.1 $\mathcal{O}(k^2m)$ kernel for $(\min \sum 0, k)$

Let us start with two simple lemmas allowing us to bound the size of the input. Notice first that, creating a perfect stack<sup>1</sup> when possible, is not always safe. Indeed, in instance  $V^1 = \{\langle 111 \rangle, \langle 101 \rangle, \langle 011 \rangle\}$ ,  $V^2 = \{\langle 111 \rangle, \langle 101 \rangle, \langle 110 \rangle\}$ ,  $V^3 = \{\langle 111 \rangle, \langle 011 \rangle, \langle 110 \rangle\}$ , depicted in Figure 8.2, no optimal solution creates a 1-stack. However, as we will see in Lemma 8.3, creating perfect stacks becomes safe if  $n > k$ .

**Lemma 8.3.** *There exists a polynomial algorithm which, given any instance  $I[m, n, p, k]$  of  $(\min \sum 0, k)$ , either detects that  $I$  is a negative instance, or outputs an equivalent instance  $I'[m, n', p, k]$  such that  $n' \leq k$ .*

*Proof.* Let  $I[m, n, p, k]$  be an instance of  $\min \sum 0$ , and suppose that  $n > k$ . Let us write a polynomial pre-processing rule that either detects that  $I$  is a no instance, or compute an equivalent instance  $I''[m, n'', p, k]$  with  $n'' = n - 1$ .

Notice first that there exists at least a perfect vector in every set  $V^i$ . If not,  $I$  is a no instance as any solution would be of cost at least  $n > k$ . It is now safe to create a perfect stack, obtaining a remaining instance  $I''$  with  $n'' = n - 1$ . Indeed, if  $I$  is a yes instance, then there must exist at least one perfect stack in the solution (otherwise the cost would be at least  $n > k$ ), and thus the remaining instance is also a yes instance. As the converse is trivially true, the rule is safe. Applying it at most  $n - k$  times finally leads to the desired upper bound.  $\square$

**Lemma 8.4.** *There exists a polynomial algorithm which, given any instance  $I[m, n, p, k]$  of  $(\min \sum 0, k)$ , either detects that  $I$  is a negative instance, or outputs an equivalent instance  $I'[m, n, p', k]$  such that  $p' \leq k$ .*

*Proof.* Let  $I[m, n, p, k]$  be an instance of the problem, and suppose that there exists  $l \in [p]$  such that for all  $(i, j) \in [m] \times [n]$  we have  $v_j^i[l] = 1$ . In other words, the  $r^{\text{th}}$  component of all vectors of all sets is a 1. In this case, it is clear that all vectors of any set of  $n$  stacks obtained from  $I$  will also contain a 1 at the  $r^{\text{th}}$  component. Hence, we

<sup>1</sup>Remind that a perfect stack is a stack consisting of only perfect vectors, *i.e.* vectors with all their coordinates set to one.

can modify  $I[m, n, p, k]$  into  $I'[m, n, p', k]$  with  $p' < p$  by dropping all such components for all vectors. It is clear that this rule is safe since the cost of any solution remains unchanged, and it can be applied in polynomial time. After applying this rule, for all  $r \in [p']$  there exists  $(i, j) \in [m] \times [n]$  such that  $v_j^i[r] = 0$ . This immediately implies that the cost of any solution is at least  $p'$ , and thus if  $p' > k$  the algorithm detected a no instance.  $\square$

Given the two previous lemmas, we can suppose from now on that for any instance of  $\min \sum 0$  we have  $n \leq k$  and  $p \leq k$ . This immediately implies a polynomial kernel parameterized by  $k$  and  $m$ .

**Theorem 8.3.**  $\min \sum 0$  admits a kernel with  $O(k^2 m)$  bits.

### 8.2.2 Positive results according to $\zeta_\beta$

In this section, we present an **FPT** algorithm when parameterized by  $\zeta_\beta$  and  $n$  (recall that both  $\zeta_\beta$  and  $n$  are smaller parameters than the standard one  $k$ , since  $k = \beta + \zeta_\beta$  and  $n \leq k$  in any reduced instance). Notice first that it is easy to get a  $\mathcal{O}^*(2^{\zeta_\beta(\log(n) + \log(p))})$  algorithm. Indeed, by considering a set  $i \in [m]$  where  $c(V^i) = \beta$ , and guessing the positions of the  $\zeta_\beta$  new zeros (among  $np$  possible positions) that will appear in an optimal solution, we can actually guess in  $\mathcal{O}^*((np)^{\zeta_\beta})$  the vectors  $\{v_{s_j}^*\}$  of an optimal solution, and it remains to check in polynomial time that every  $V^j$  can be “matched” to  $\{v_{s_j}^*\}$ . Now we show how to get rid of the  $\log(p)$  term in the exponent.

**Theorem 8.4.**  $\min \sum 0$  can be solved in  $\mathcal{O}^*(4^{\zeta_\beta \log(n)})$ .

*Proof.* Let  $I[m, n, p, \zeta_\beta]$  be an instance of our problem and, *w.l.o.g.*, suppose that  $V^1$  is a set whose number of zeros reaches the upper bound  $\beta$ , *i.e.*  $c(V^1) = \beta$ . The algorithm consists in constructing a solution by finding an optimal assignment between  $V^1$  and  $V^2, \dots, V^m$ , successively.

We first claim that we can decide in polynomial time whether there is an assignment between  $V^1$  and  $V^2$  which does not create any additional zero.

To that end, we create a bipartite graph  $G$  with bipartization  $(A, B)$ , with  $A = \{a_1, \dots, a_n\}$ ,  $B = \{b_1, \dots, b_n\}$ , and link  $a_{j_1}$  and  $b_{j_2}$  for all  $(j_1, j_2) \in [n] \times [n]$  if and only if assigning vector  $v_{j_1}^1 \gg v_{j_2}^2$ . In other words, an edge  $(v_{j_1}^1, v_{j_2}^2)$  exists if and only if  $v_{j_1}^1 \wedge v_{j_2}^2 = v_{j_1}^1$ .

If a perfect matching can be found in  $G$ , then we can safely delete the set  $V^2$  and continue. In order to avoid heavy notations, we consider this first step as a polynomial pre-processing, and we re-label  $V^i$  into  $V^{i-1}$  for all  $i \in [m]_3$  (and  $m$  is implicitly decreased by one).

In the following, we suppose that the previous pre-processing step cannot apply (*i.e.* there is no perfect matching in  $G$ ). Intuitively, in this case any assignment (including an optimal one) between  $V^1$  and  $V^2$  must lead to at least one additional zero in  $V^1$ . In this case, we perform a branching to guess one couple of vectors from  $V^1 \times V^2$  which will induce such an additional zero. More formally, we branch on every



couple  $(j_1, j_2) \in [n] \times [n]$ , and create a new instance as a copy of  $I$  in which  $v_{j_1}^1$  is replaced by  $v_{j_1}^1 \wedge v_{j_2}^2$ . This operation increases  $c(V^1)$  by at least one, and thus  $\beta$  by at least one as well. If we denote by  $I'$  this new instance, we can check that a solution of cost at most  $k$  for  $I'$  will immediately imply a solution of cost at most  $k$  for  $I$ , as  $I'$  is constructed from  $I[m, n, p]$  by adding some zeros. The converse is also true as one assignment we enumerate corresponds to one from an optimal solution.

As the value of  $\beta$  in this branching increases by at least one while we still look for a solution of cost  $k$ , this implies that this branching will be applied at most  $\zeta_\beta$  times. Summing up, we have one polynomial pre-processing and one branching of size  $n^2$  which will be applied at most  $\zeta_\beta$  times. The total running time of this algorithm is thus bounded by  $\mathcal{O}^*(4^{\zeta_\beta \log(n)})$ . □

### 8.2.3 Negative results according to $\zeta_\beta$

Despite its simplicity, we now show that, when considering each parameter ( $n$  and  $\zeta_\beta$ ) separately, this algorithm is the best we can hope for (whereas the existence of an  $\mathcal{O}^*(2^k)$  algorithm is still open). Indeed, we first show in Theorem 8.2 that the linear dependence in  $\zeta_\beta$  and  $\log(n)$  in the exponent is necessary (unless ETH fails), and also that we cannot hope for an **FPT** algorithm parameterized by  $\zeta_\beta$  only unless **FPT** = **W**[2] (Theorem 8.1). Finally, as we will see in the next section (Theorem 8.6), this result is matched by a  $2^{\mathcal{O}(\zeta_\beta)}$  lower bound when  $n \in \mathbb{N}$  is fixed. We now present an *FPT*-reduction from  $(\min \min 0, k)$  problem which produces an instance of  $\min \sum 0$  in which parameters  $n$  is preserved and  $\zeta_\beta = k$ .

**Lemma 8.5.** *There exists a polynomial reduction from  $(\min \min 0, k)$  restricted to the instances where  $\beta < p(n-1)$  to  $\min \sum 0 / \zeta_\beta$  that given an instance  $I[m, n, p]$  and an integer  $k$ , constructs an instance of  $\min \sum 0 / \zeta_\beta$   $f(I), m' = m+1, n' = n, p' = p[m, n, p]$  such that  $\zeta_\beta = k$ .*

*Proof.* The reduction is the same reduction as the one presented in Section 7.1.2: we add to the instance  $I$  of  $\min \min 0$  a single set  $V^{m+1}$  containing  $n-1$  zero vectors and exactly one perfect vector.

However, since the objective function focus on the overall number of zeros, this reduction is not an *S*-reduction:  $c(\text{opt}(f(I))) = c(\text{opt}(I)) + p(n-1)$ . We claim that this reduction is an *FPT*-reduction. By construction,  $\beta = c(V^{m'}) = p(n-1)$ , it follows that  $c(\text{opt}(f(I))) = c(\text{opt}(I)) + p(n-1) = k + \beta$ , and thus  $\zeta_\beta = k$ . □

Based on Observation 6.1 and Theorem 6.14, we know that  $(\min \min 0, k)$  restricted to the instances where  $\beta < p(n-1)$  is **W**[2]-hard. We deduce the following corollary.

**Corollary 8.1.**  $(\min \sum 0, \zeta_\beta)$  is **W**[2]-hard.

**Corollary 8.2.**  $\min \sum 0$  cannot be solved in  $\mathcal{O}^*(2^{\mathcal{O}(\zeta_\beta \log(n))})$  nor  $\mathcal{O}^*(2^{\zeta_\beta \mathcal{O}(\log(n))})$ , unless ETH fails.

### 8.2.4 Positive results according to $\zeta_p$

We now consider the problem parameterized by  $\zeta_p = k - p$  (recall that  $p \leq k$ ). Notice that one motivation of this parameterization is the previous reduction of Lemma 8.5 from MINIMUM HITTING SET. Indeed, when applied for  $n = 2$ , it reduces an instance of VERTEX COVER to an instance of  $\min \sum 0$  with  $k = p + \zeta_p$  where  $\zeta_p$  is equal to the size of the vertex cover. Our intuition is confirmed by the following result: we show that when parameterized by  $\zeta_p$ , the problem is indeed in **FPT** when  $n = 2$  (Theorem 8.5). We complement this by showing that for any  $n \geq 3$ , it becomes **NP**-hard when  $\zeta_p = 0$  (Theorem 8.6), and is thus even not in **XP**. The reduction we use even proves that for any fixed  $n \geq 3$ , the problem cannot be solved in  $2^{o(k)}$  (and thus in  $2^{o(\zeta_p)}$ ) unless ETH fails, while the algorithm of Theorem 8.4 runs in  $O^*(2^{O(\zeta_p)})$ . In the following,  $n$ - $\min \sum 0$  denotes the problem  $\min \sum 0$  where the size of all sets is fixed to some constant  $n \in \mathbb{N}$ .

In this subsection, we prove that 2- $\min \sum 0$  is **FPT** parameterized by  $\zeta_p$ . To do so, we reduce to the ODD CYCLE TRANSVERSAL problem. As a reminder, OCT consists, given a graph  $G = (U, E)$  and an integer  $c \in \mathbb{N}$ , in deciding whether there exists a partition  $(T, S_1, S_2)$  of  $U$  with  $|T| \leq c$  such that  $G[U \setminus T]$  is bipartite.

We first introduce a generalized version of OCT, called BIP-OCT. In this problem, we are given a set of vertices  $U$ , an integer  $c$ , and a set of  $m$  pairs  $(A_1, B_1), \dots, (A_m, B_m)$  with  $A_i, B_i \subseteq U$  for all  $i \in [m]$  and  $A_i \cap B_i = \emptyset$ . Informally, each pair  $(A_i, B_i)$  can be seen as a complete bipartite subgraph. The output of BIP-OCT is described by a partition  $(T, S_1, S_2)$  of  $U$  such that for any  $i \in [m]$ , either  $(A_i \setminus T \subseteq S_1$  and  $B_i \setminus T \subseteq S_2)$  or  $(A_i \setminus T \subseteq S_2$  and  $B_i \setminus T \subseteq S_1)$ . The objective is to determine whether there exists such a partition with  $|T| \leq c$ . As we can see, if all  $A_i$  and  $B_i$  are singletons (and thus form edges), then BIP-OCT corresponds to OCT. Notice that in the following, the considered parameter of OCT and BIP-OCT will always be the standard parameter, *i.e.*  $c$ . We first show that there is a linear parameter-preserving reduction from 2- $\min \sum 0$  parameterized by  $\zeta_p$  to BIP-OCT, and then that there is also a linear parameter-preserving transformation from BIP-OCT to OCT.

**Lemma 8.6.** *There is a linear parameter-preserving reduction from 2- $\min \sum 0$  parameterized by  $\zeta_p$  to BIP-OCT.*

*Proof.* Let  $I[m, 2, p, p + \zeta_p]$  be an instance of 2- $\min \sum 0$  (*i.e.* in which every set contains only two vectors), and let us construct an instance  $I'$  of BIP-OCT, such that  $I$  has a solution of cost  $p + \zeta_p$  if and only if  $I'$  has a solution of size  $\zeta_p$ . Notice first that, according to Hypothesis 4.3, we can suppose that for any  $i \in [m]$  and any  $l \in [p]$ , we cannot have both  $v_1^i[l] = 0$  and  $v_2^i[l] = 0$  as otherwise any stack  $s$  from any solution would have  $v_s[l] = 0$ , and thus we could safely remove such a component  $l$  from the instance (and decrease  $k$  and  $p$  by one).

Let the vertex set of  $I'$  be  $[p]$ . Then, for all  $i \in [m]$ , let us define  $A_i = \{l : v_1^i[l] = 0\}$ , and  $B_i = \{l : v_2^i[l] = 0\}$  as depicted in Figure 8.3. By the foregoing, and as required in an instance of BIP-OCT, we have  $A_i \cap B_i = \emptyset$ . Let us prove that  $I$  has a solution of cost  $p + \zeta_p$  if and only if  $I'$  has a solution  $(T, S_1, S_2)$  with  $|T| \leq \zeta_p$ .

$\Rightarrow$  Let  $S = \{s_1, s_2\}$  be a solution of  $I$  of cost  $p + \zeta_p$ . Let  $T = \{l : v_{s_1}[l] = v_{s_2}[l] = 0\}$ ,  $S_1 = \{l : v_{s_1}[l] = 0 \text{ and } v_{s_2}[l] = 1\}$ , and  $S_2 = \{l : v_{s_1}[l] = 1 \text{ and } v_{s_2}[l] = 0\}$ . Notice that  $(T, S_1, S_2)$  forms a partition of  $[p]$  (as we cannot have a coordinate  $l$  such that  $v_{s_1}[l] = v_{s_2}[l] = 1$ , as this would imply that all the  $nm$  vectors have  $v[l] = 1$ , and such coordinates have been removed from the instance in Lemma 8.4), and  $|T| = \zeta_p$ . It remains to prove that  $(T, S_1, S_2)$  is a feasible solution of  $I'$ . Let  $i \in [m]$ . Without loss of generality, let us suppose that  $v_1^i$  has been added to  $s_1$  and  $v_2^i$  has been added to  $s_2$ . Let  $l \in A_i \setminus T$ . Since  $l \in A_i$ , we have  $v_1^i[l] = 0$ , and thus  $v_{s_1}[l] = 0$ . Since  $l \notin T$ , we have  $v_{s_2}[l] = 1$ . Thus,  $l \in S_1$ , which proves  $A_i \setminus T \subseteq S_1$ . Similarly, we can prove that  $B_i \setminus T \subseteq S_2$ .

$\Leftarrow$  Let  $(T, S_1, S_2)$  be a solution of  $I'$  with  $|T| \leq \zeta_p$ . Let  $s_1$  be such that  $v_{s_1}[l] = 0$  if and only if  $l \in T$  or  $l \in S_1$ , and let  $s_2$  be such that  $v_{s_2}[l] = 0$  if and only if  $l \in T$  or  $l \in S_2$ . It remains to prove that the solution  $S = \{s_1, s_2\}$  is feasible, which immediately implies that its cost is  $p + \zeta_p$ . Let  $i \in [m]$ . Without loss of generality, let us suppose that  $A_i \setminus T \subseteq S_1$  and  $B_i \setminus T \subseteq S_2$ . We now claim that  $v_1^i$  can be assigned to  $s_1$  and  $v_2^i$  can be assigned to  $s_2$  without creating any new zero. To do so, let us show that for all  $l \in [p]$ , we have  $v_1^i[l] = 0 \Rightarrow v_{s_1}[p] = 0$  (resp.  $v_2^i[l] = 0 \Rightarrow v_{s_2}[p] = 0$ ). Indeed, let  $l \in [p]$  such that  $v_1^i[l] = 0$ . Then by construction, it means that  $l \in A_i$ . Thus, by definition of the solution  $(T, S_1, S_2)$ , it means that either  $l \in T$  or  $l \in S_1$ , which implies  $v_{s_1}[l] = 0$  as desired. Similar arguments show that  $v_2^i[l] = 0 \Rightarrow v_{s_2}[p] = 0$  for all  $l \in [p]$ .  $\square$

**Lemma 8.7.** *There is a linear parameterized reduction from BIP-OCT to OCT.*

*Proof.* Let  $I = (U, \{A_i, B_i\}_{i \in [m]}, c)$  be an instance of BIP-OCT. Let us construct a graph  $G' = (U', E')$  which contains an odd cycle transversal of size  $c$  if and only if  $I$  has a solution of size  $c$  for BIP-OCT. Observe first that we cannot simply set  $U' = U$  and  $E' = \bigcup_{i \in [m], a \in A_i, b \in B_i} \{a, b\}$ . Indeed, if for example  $A_1 = \{2, \dots, n\}$ ,  $B_1 = \{1\}$ ,  $A_2 = \{2, \dots, \frac{n}{2}\}$  and  $B_2 = \{\frac{n}{2} + 1, \dots, n\}$ , defining  $G'$  as above would lead to an odd cycle transversal of size one, as removing only vertex  $\{1\}$  makes the graph bipartite with bipartization  $(A_2, B_2)$ . However, this solution is not feasible for BIP-OCT as  $A_1 \setminus T = A_1$ , and  $A_1 \not\subseteq A_2$  and  $A_1 \not\subseteq B_2$ . Intuitively, we have to prevent solutions of  $G'$  from splitting sets  $A_i \setminus T$  (and  $B_i \setminus T$ ) between the two parts of the bipartization. To do so, we will construct  $G'$  as described above, and then we "enlarge" each bipartite graph by adding  $c + 1$  new vertices on each side. More formally, we start by setting  $U' = U$  as said before, and for all  $i \in [m]$ , we create two sets of  $c + 1$  new vertices  $A'_i, B'_i$ . We then set  $E' = \bigcup_{i \in [m], a \in A_i \cup A'_i, b \in B_i \cup B'_i} \{a, b\}$ .

Let us now prove that  $I$  contains a solution of size  $c$  for BIP-OCT if and only if  $G'$  contains an odd cycle transversal of size  $c$ .

$\Rightarrow$  Let  $(T, S_1, S_2)$  be an optimal solution of  $I$ . We define a partial solution  $T', S'_1, S'_2$  of  $G'$  by setting  $T' = T$  and  $S'_l = S_l$  for  $l \in \{1, 2\}$  (the solution is partial in the sense that it remains to assign vertices of  $A'_i \cup B'_i$ , for all  $i \in [m]$ ). Let  $i \in [m]$ . If  $A_i \setminus T = \emptyset$  and  $B_i \setminus T = \emptyset$ , then we add (arbitrarily)  $A'_i$  to  $S'_1$  and  $B'_i$  to  $S'_2$ .

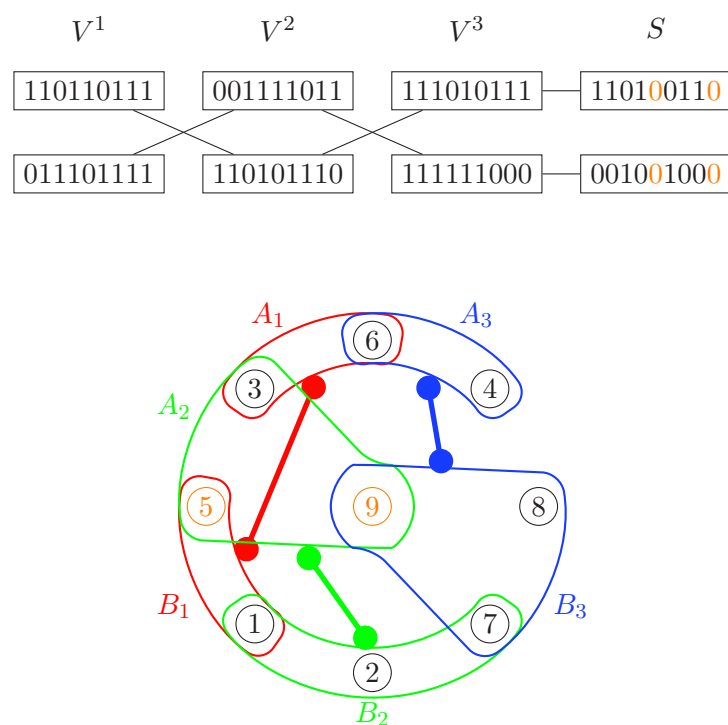


Figure 8.3: Example of reduction from an instance of  $2\text{-min } \Sigma 0$ , with  $n = 2, m = 3, p = 9$ , admitting a solution of cost  $p + 2$ , to an instance of BIP-ODD CYCLE TRANSVERSAL, with  $|U| = p$ ,  $A_1 = \{3, 6\}, B_1 = \{1, 5\}, A_2 = \{2, 3, 7\}, B_2 = \{3, 5, 9\}, A_3 = \{4, 6\}, B_3 = \{7, 8, 9\}$ , admitting the partition  $(\{5, 9\}, \{3, 4, 6\}, \{1, 2, 7, 8\})$  as solution of cost 2.

Otherwise, if  $A_i \setminus T \neq \emptyset$  and is added to  $S_l$ , we add  $A'_i$  to  $S'_l$  and  $B'_i$  to  $S'_{l'}$  with  $l, l' \in \{1, 2\}, l' \neq l$ , and if  $B_i \setminus T \neq \emptyset$  and is added to  $S_l$ , we add  $B'_i$  to  $S'_l$  and  $A'_i$  to  $S'_{l'}$  with  $l, l' \in \{1, 2\}, l' \neq l$ .

This new solution has the same size ( $|T'| = |T|$ ) and we claim that it is an odd cycle transversal of  $G'$ . Indeed, let us check that any edge  $\{u, v\} \in E'$  such that  $\{u, v\} \cap T' = \emptyset$  is not entirely contained in a  $S'_l$ . If  $\{u, v\}$  is an edge of a complete bipartite of  $I$ , *i.e.* if there exists  $i$  such that  $u \in A_i$  and  $v \in B_i$ , then by definition of the solution  $(T, S_1, S_2)$  it is straightforward that  $u$  and  $v$  are not both in  $S'_1$  nor in  $S'_2$ . Otherwise, if  $\{u, v\}$  is adjacent to one or two of the new vertices, let  $i$  be such that  $u \in A'_i$ . If  $v \in B'_i$ , then the solution is valid as  $A'_i$  and  $B'_i$  are never added to the same set  $S'_l, l \in \{1, 2\}$ . Otherwise, we necessarily have  $v \in B_i$ . Let  $l \in \{1, 2\}$  be such that  $B_i \setminus T$  (which is not empty) has been added to  $S'_l$ . In this case  $A_i$  (and thus  $u$ ) has been added to  $S'_{l'}$ , with  $l' \neq l$ .

$\Leftarrow$  Let  $(T', S'_1, S'_2)$  be an optimal solution of  $G'$ . For any  $i \in [m]$ , let  $\tilde{A}_i = (A_i \cup A'_i) \setminus T'$  and  $\tilde{B}_i = (B_i \cup B'_i) \setminus T'$ . A first observation is that  $\tilde{A}_i \neq \emptyset$  and  $\tilde{B}_i \neq \emptyset$  as  $|A_i \cup A'_i| = |B_i \cup B'_i| > c$  and  $|T'| \leq c$ . A second observation is that for any  $u$  and  $v \in \tilde{A}_i, u$  and  $v$  are in the same set  $S'_l$  for some  $l \in \{1, 2\}$ . Indeed, suppose by contradiction that  $u \in S'_1$  and  $v \in S'_2$ . As  $\tilde{B}_i \neq \emptyset$ , there exists  $b \in \tilde{B}_i$  and  $l \in \{1, 2\}$  such that  $b \in S'_l$ . As all the edges of the complete bipartite subgraph on  $(A_i \cup A'_i, B_i \cup B'_i)$  belong to  $E'$ , we have  $\{u, b\} \in E'$  and  $\{v, b\} \in E'$ , and thus  $S'_l$  contains both endpoints of an edge of  $E'$ , which is a contradiction. In the same way, we can prove that for any  $i \in [m]$ , and any  $u$  and  $v \in \tilde{B}_i, u$  and  $v$  are in the same set  $S'_l$  for some  $l \in \{1, 2\}$ .

Thus, according to the two previous observations, for any  $i \in [m]$  we can define  $\lambda_{\tilde{A}_i} \in \{1, 2\}$  and  $\lambda_{\tilde{B}_i} \in \{1, 2\}$  such that  $\tilde{A}_i \subseteq S'_{\lambda_{\tilde{A}_i}}$  and  $\tilde{B}_i \subseteq S'_{\lambda_{\tilde{B}_i}}$ , with  $\lambda_{\tilde{A}_i} \neq \lambda_{\tilde{B}_i}$ .

Let us now define  $T = T' \cap V$ ,  $S_1 = S'_1 \cap V$ , and  $S_2 = S'_2 \cap V$ , and check that this is a valid solution of  $I$ . Let  $i \in [m]$ . Observe first that  $A_i \setminus T \subseteq \tilde{A}_i$ , and thus either  $A_i \setminus T = \emptyset$ , or  $A_i \setminus T \subseteq S_{\lambda_{\tilde{A}_i}}$ . As the same fact also holds for  $B_i \setminus T$ , and as  $\lambda_{\tilde{A}_i} \neq \lambda_{\tilde{B}_i}$ , the constraint  $(A_i \setminus T \subseteq S_1$  and  $B_i \setminus T \subseteq S_2)$  or  $(A_i \setminus T \subseteq S_2$  and  $B_i \setminus T \subseteq S_1)$  is respected, and the solution is feasible, which concludes the proof.

An example of this reduction is given on Figure 8.4. □

As ODD CYCLE TRANSVERSAL can be solved in  $O^*(2.3146^c)$ , proved by [Lok+14], and since our parameters are exactly preserved in our two reductions, we obtain the following result:

**Theorem 8.5.** *2-min  $\sum 0$  can be solved in  $O^*(d^{c^p})$  where  $d \leq 2.3146$  is such that OCT can be solved in  $O^*(d^c)$ .*

(a) Example of BIP-OCT instance  $I$  with  $U = \{1, 2, 3, 4\}$ ,  $(A_1 = \{1\}, B_1 = \{2, 3, 4\})$  and  $(A_2 = \{1, 3\}, B_2 = \{2\})$  with solution  $T = \{3\}$  of size  $c = 1$ .



(b) Example of reduction from the previous BIP-OCT instance  $I$  admitting a solution of size  $c = 1$ , to an instance of OCT admitting a solution  $T = \{3\}$  of size  $c = 1$ .

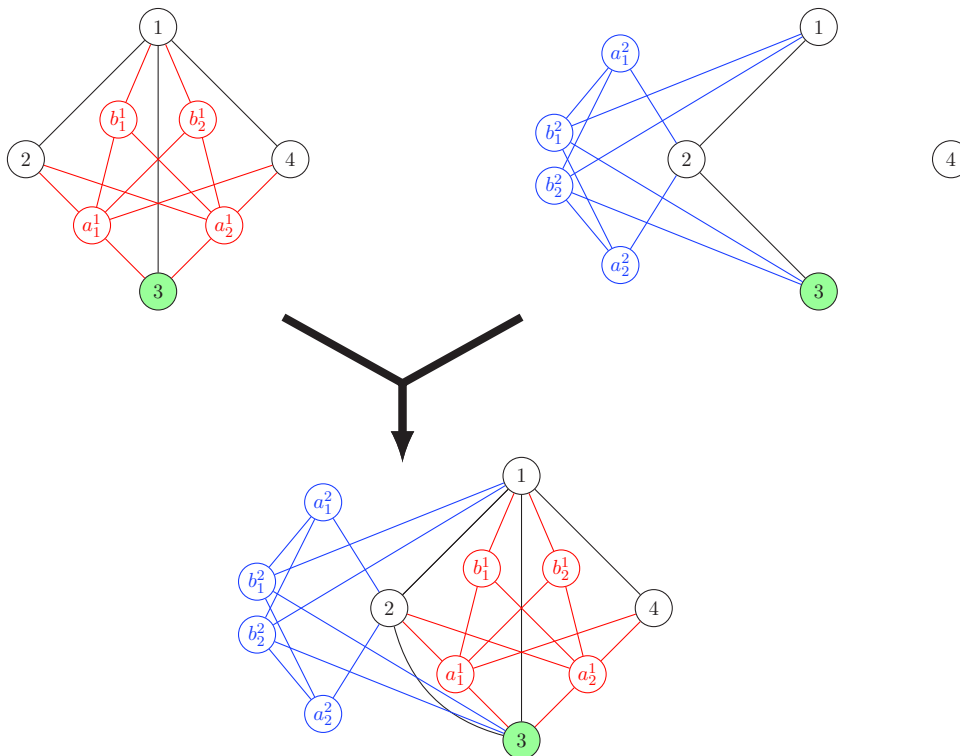


Figure 8.4: Example of reduction from BIP-OCT to OCT.

### 8.2.5 Negative results according to $\zeta_p$

We now complement the previous result by proving that the problem is intractable with respect to the parameter  $\zeta_p$  for larger values of  $n$ .

**Theorem 8.6.** *For any fixed  $n \geq 3$ ,  $n\text{-min } \sum 0$  is not in  $\mathbf{XP}$  when parameterized by  $\zeta_p$  (unless  $\mathbf{P} = \mathbf{NP}$ ), and cannot be solved in  $2^{o(k)}$  (unless  $\text{ETH}$  fails).*

*Proof.* Let  $\chi \geq 3$ . We present a reduction from  $\chi\text{-COLORING}$ , which consists in, given a graph  $G = (U, E)$ , to ask for a mapping  $f : U \rightarrow [\chi]$  such that for all  $\{u, v\} \in E$  we have  $f(u) \neq f(v)$ . Let  $E = \{e_1, \dots, e_{m_G}\}$  and  $U = [n_G]$ . Let us construct an instance  $I$  of  $n\text{-min } \sum 0$  with  $n = \chi$ ,  $p = n_G$ ,  $m = m_G$  and such that  $G$  admits a  $\chi$ -coloring if and only if  $I$  has a solution of cost  $p$  (i.e.  $\zeta_p = 0$ ). To each edge  $e_i = \{u, v\} \in E$ ,  $i \in [m_G]$ , we associate a set  $V^i$  with  $|V^i| = \chi$ , where:

- $v_1^i$  represents the vertex  $u$ , that is  $v_1^i[u] = 0$  and  $v_1^i[l] = 1$  for any  $l \in [n_G]$ ,  $l \neq u$ ,
- $v_2^i$  represents the vertex  $v$ , that is  $v_2^i[v] = 0$  and  $v_2^i[l] = 1$  for any  $l \in [n_G]$ ,  $l \neq v$ ,
- for all  $j \in \{3, \dots, \chi\}$ ,  $v_j^i$  is a 1-vector, i.e. it has a 1 at every component.

An example of this construction is depicted in Figure 8.5. Let us now prove that  $G$  admits a  $\chi$ -coloring if and only if  $I$  has a solution of cost  $p = n_G$ .

$\Rightarrow$  Let  $S_j \subseteq U$ ,  $j \in [\chi]$  be the  $\chi$  color classes (notice that the  $S_j$  are pairwise disjoint, some of them may be empty, and  $\bigcup_{j \in [\chi]} S_j = U$ ). To each  $S_j$  we associate a stack  $s_j$  such that  $v_{s_j}[l] = 0$  if and only if  $r \in S_j$ . It remains to prove that the solution  $S = \{s_1, \dots, s_\chi\}$  is feasible, as its cost is exactly  $p$  by construction. Let us consider a set  $V^i$  where  $v_1^i$  (resp.  $v_2^i$ ) represents a vertex  $u$  (resp.  $v$ ). As  $\{u, v\}$  is an edge of  $G$ , we know that  $u$  and  $v$  have two different colors, i.e. that  $u \in S_j$  and  $v \in S_{j'}$ , for some  $j, j' \in [\chi]$  with  $j \neq j'$ . Thus, we can add  $v_1^i$  to stack  $s_j$ ,  $v_2^i$  to stack  $s_{j'}$ , and the  $\chi - 2$  other  $v_j^i$  ( $j \geq 3$ ) in an arbitrary way. Since the only 0 in  $v_1^i$  (resp.  $v_2^i$ ) is at the  $u^{\text{th}}$  (resp.  $v^{\text{th}}$ ) component, we have  $v_1^i \wedge v_{s_j} = v_{s_j}$  (resp.  $v_2^i \wedge v_{s_{j'}} = v_{s_{j'}}$ ), which proves that  $S$  is feasible.

$\Leftarrow$ . Let  $S = \{s_1, \dots, s_\chi\}$  be the stacks of an optimal solution. For  $j \in [\chi]$ , let  $S_j = \{l \in [p] \mid v_{s_j}[l] = 0\}$ . Notice that  $\bigcup_{j=1}^\chi S_j = U$ , and as  $I$  is of cost  $p$ , all the  $S_j$  are pairwise disjoint and form a partition of  $U$ . Moreover, as for any  $i \in [m]$ ,  $v_1^i$  and  $v_2^i$  have been assigned to different stacks, the corresponding vertices have been assigned to different colors, and thus each  $S_j$  induces an independent set, which completes the reduction.

It is known, thanks to [IPZ01] that there is no  $2^{o(|U|)}$  algorithm for deciding whether a graph  $G = (U, E)$  admits a  $\chi\text{-COLORING}$ , for any  $\chi \geq 3$  (under ETH). As we can see, the value of the optimal solution for  $n\text{-min } \sum 0$  in the previous reduction equals the number of vertices in the instance of  $\chi\text{-COLORING}$ , which proves that  $n\text{-min } \sum 0$  cannot be solved in  $2^{o(k)}$  for any  $n \geq 3$ . □

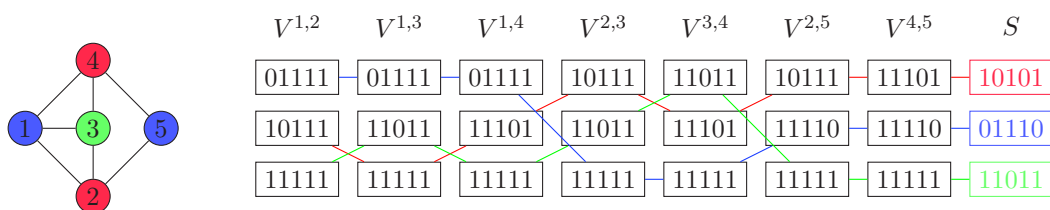


Figure 8.5: Example of reduction from a positive instance of  $\chi$ -COLORING, with  $\chi = 3$ ,  $U = [5]$ ,  $E = \{\{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{3, 4\}, \{2, 5\}, \{4, 5\}\}$ , to an instance of  $\min \sum 0$  with  $m = |E| = 6$ ,  $n = \chi = 3$ ,  $p = |U| = 5$  admitting a solution of cost  $p$ .

Finally, remark that as for the parameterization by  $p$ , one could ask if  $\min \sum 0$  is in **FPT** when parameterized by the first lower bound  $\beta$ . However, we can see in the previous reduction that we obtain a graph with  $\beta = 2$ , and thus the problem is even not in **XP** unless **P = NP**.

### 8.3 Conclusion and open questions

Contrary to  $\max \sum 1$ ,  $\min \sum 0$  appears to be more tractable from the approximation point of view but also from the parameterized complexity point of view. Indeed the problem admits several trivial approximation algorithms such as the  $n$  or the  $m$  approximation algorithm<sup>2</sup> but also admit a kernel of size  $\mathcal{O}(k^2m)$  while  $\max \sum 1$  is **W[1]**-hard when parameterized by the standard parameter  $k$ .

However this tractability has a drawback in terms of negative results. Contrary to  $\max \sum 1$ , it remains several open questions on the approximability of  $\min \sum 0$ . The problem is known to be **APX**-hard even when  $m = 3$  and  $\#0 \leq 2$  [DCS14] or when  $n = 2$  and  $\#0 \leq 1$  while the best known ratio is given by the *heaviest first* heuristic of [DCS14] approximating  $\min \sum 0$  within a factor:

$$\rho = \frac{1}{2}(m+1) - \frac{1}{4}\log(m-1)$$

This leads to a huge gap between upper and lower bound. We provide a Gap-reduction that fills somehow this gap, but the  $(n - \varepsilon)$ -inapproximability is based on the disputed Unique Game Conjecture. Thus the question on the existence of a better inapproximability bound assuming only **P**  $\neq$  **NP** is left open.

This closes our theoretical study of the *wafer-to-wafer integration* problems. Next part is devoted to real instance tackling and we will show among other things that, matching based heuristics are good candidates to tackle real life instances. Indeed, despite its approximation ratio being equal to  $4/3$ , the performance of the algorithm in average are very interesting in practice.

<sup>2</sup>The cost function being additive and submodular, we can apply the result of Dokka et al. [DCS14] stating that any greedy algorithm is an  $m$ -approximation algorithm.





— PART III —  
**PRACTICAL  
ASPECTS**



# 9

## Preliminaries

---

### Contents

---

<b>9.1</b>	<b>Related work</b>	<b>132</b>
<b>9.2</b>	<b>Instance generation</b>	<b>133</b>
9.2.1	Considered model	133
9.2.2	Model restrictions	134
9.2.3	Alternative model	134
<b>9.3</b>	<b>Experiment characteristics</b>	<b>134</b>

---

As for physics, statistics or chemistry, computer science is a prism introducing mathematics in the perception of our surrounding world. As such it aims at developing links between the power of abstraction of fundamental sciences and the extreme complexity of real world applications. This naturally motivates a double approach of the problems being at once theoretical and practical, especially when the latter has been introduced as a tool to tackle real worlds issues.

This naturally motivates a practical analysis of the internal mechanisms of the problems. This part aims at presenting the work we perform to those ends. In a first time we introduce the state of the art in terms of practical tackling of the problem. In a second time we discuss on the simulation of the problems. We present empirical constraints and hypotheses that apply on real world instances and present instances generator taking the latter into consideration.

We finish the preliminaries but discussing about the characteristics of the experiment environment.

## 9.1 Related work

Contrary to the theoretical aspects, the practical aspects have been quite intensely studied. It follows that the literature on this topic is relatively dense. However, the important applications of the problem in IC manufacturing led to the study of some very specific variants. That is why we only focus on reference works on this topic.

The consideration of the yield in wafer-to-wafer integration has been first introduced by Smith et al. in [Smi+07]. However the formalization of the problem and the introduction of the framework we use to tackle it have been introduced by Reda et al. in [RSS09]. In Chapter 5 we mainly focused on the proposed heuristics for  $\max \sum 1$ . But authors of the article provide several experiments designed to analyze the impact of the natural parameters  $m, n, p$  and  $Y_W$  on the solution quality, where  $Y_W$  is defined as the die yield. In other word, it can be seen as the ratio of the overall number of viable dies in all the wafer compared to the total number of dies. More formally, the authors define it as:

$$Y_W = \frac{1}{mnp} \sum_{i \in [m]} \sum_{j \in [n]} \sum_{l \in p} v_j^i[l]$$

. They show that the matching heuristics can lead to significant improvement especially when  $m$  takes high values, when the individual wafer yield  $Y_W$  is low or when  $p$  takes low values. In a last time, they try to evaluate the profit we can expect by increasing the number of wafers per set when the die yield is low.

In line with the previous work of Reda et al. [RSS09], Verbree et al. [Ver+10] propose a mathematical equation that approximates the expected stack yield of the matching heuristic. They confront their mathematical model with experiment results on more than 10000 instances. Based on these very large number of instances, they propose refinement of the experiment of Reda et al. They indeed investigate the influence of the number of wafers per set for different values of  $Y_W, p$  and  $m$ . They show for example that the improvement of the final die yield we can get from considering a set of 50 wafers instead of a set of 25 wafers is more important if the number of layers in the stack (and thus the number of sets) is equal to 6 instead of 3. They finally address an interesting problem arising when the failures map of the wafer is not known. In this problem the objective is to determine in function of  $Y_W, p$  and  $m$  whether the profit that could be made thanks with the die yield increase thanks to matching heuristic counterbalance or even overcome the expenses needed to compute the failure map of the wafers.

These two papers are the reference works on the study of yield improvement based on matching heuristic approach. However one can cite other interesting works either on slightly different problems or different approach of the same problem.

In [Tao+10], the authors focused on the yield improvement of wafer-to-wafer integration with *running repositories*. In the problems we consider, the sets of wafers are implicitly considered as *static repositories*. In such a configuration, we are given  $n$  wafers, each of these wafers has to be integrated into a stack. When considering *running repositories*, the number of wafers per set is constant and equal to  $n$ . In other

words, as soon as a wafer of a set is used to design a stack, a new wafer is inserted in each set. The process stops after the creation of  $n$  stacks. Such a process can be seen as an online version of the *static repositories* configuration. The authors investigate the yield improvement that some adaptations of the matching algorithm can provide. They indeed consider different decision criteria. At each iteration a feasible solution maximizing the overall value of the criterion is computed. Among the  $n$  possible stacks of the feasible solution, the one that contribute the most to the value of the criterion is selected. The set are then replenished and a new feasible solution is computed. They prove that for every considered criterion, one can expect a more or less important die yield improvement when compared to greedy algorithm that selects at each iteration the feasible stack of maximum profit according to the considered criterion.

One can also cite different methods for improving the expected die yield. In [TH11], the authors consider a technique inspired from reparation of faulty sectors in memory chips manufacturing called the redundancy technique. In this technique, additional memory sectors are added to the chip. As soon as a failure of a sector is detected, a modification of the addressing of the chip is performed to point on the additional viable memory. Based on this principle, Taouil et al. [TH11] investigate the yield improvement we can expect by adding an extra wafer on the created stack that aims at *repairing* locations that are almost viable, *i.e.* locations in which  $m - 1$  dies are viable. They show that this technique can lead to significant improvement of the die yield especially on instances where matching heuristics are quite bad.

To finish, we mention work of Giroudeau et al. [Gir+12] that partially motivated this PhD thesis, which is the only reference, to our knowledge, that aims at optimally tackling the wafer-to-wafer integration. They propose a branch-and-price algorithm for the latter. However, such an algorithm appears to be inefficient in practice because of somehow inefficient domination rules.

## 9.2 Instance generation

### 9.2.1 Considered model

A natural question arising when considering experiments is the way to generate the instances on which the experiments will be performed. In this section we present the model we used to generate our instances. We also present the limits we identified for such a modelization and finally briefly present a more complex but more accurate model that have been used to generate instances in [RSS09; Ver+10; Tao+10].

The model we used is a simplified model that aims at taking into consideration the distribution of the failures on a wafer in the industrial processes. Indeed, one can observe that the failures are not uniformly distributed on a wafer. The dies located on the rim being more likely faulty than the one located in the center of the wafer.

Moreover, one can also see that the expected yield of two consecutive engraved wafers is very similar. It follows that wafers of a same set share almost the same expected yield.

Based on these observations, Di Natale et al. [Nat+13] define the probability for a die to be faulty as:

$$P(v_j^i[l] = 0) = 47.5 \left(1 - \frac{l}{p}\right) + 50(Y - Y_W) \quad (9.1)$$

In the previous equation,  $Y$  corresponds to the current wafer yield, *i.e.* it represents the yield computed on the position 1 to  $l-1$ . We can see that the probability decreases with the index of the considered die and decreases with the number of nonfunctional dies in the considered set ensuring thus a behaviour similar to previously presented constraints.

### 9.2.2 Model restrictions

As for any simplified model, the latter present some restrictions.

First the linear decreasing of the probability of failures seems to be inappropriate. Works of Stapper et al. [MP89; Sta89a; Sta89b; Sta86; SAS83; TB91; SR95] among others, show that a negative binomial distribution law seems to be more appropriate. They also show that the failures occurring near from the center of the wafer, due to the presence of particles, are not uniformly distributed, but shows a slight tendency to be localized in the same area for wafers of a same set.

Given a position  $l$ , the failure probability of the die located at position  $l$  strongly depends on the number of failures occurring in the position 1 to  $l-1$ . This dependence leads to side effects on the overall distribution of failures on the wafer. Indeed, two wafers having similar distribution of failures in their center likely have similar failures distribution on their whole area. We will see in Section 10.3, that such a behaviour may explain unexpected results.

### 9.2.3 Alternative model

The generation of random wafers has been extensively studied in the late 80's. These studies led to a quite accurate model based on the negative binomial distribution. In this model the yield of an individual wafer is given by:

$$\left(1 + \left(\frac{AD_0}{\alpha}\right)\right)^{-\alpha}$$

where  $A$  is the area of a die,  $D_0$  is the defect density and  $\alpha$  is the clustering ratio.

## 9.3 Experiment characteristics

Let us now define the characteristics of the following experiments. Every experiment is performed on a workstation equipped with Intel Xeon processors providing twenty cores running at 2.8 GHz each. The latter is equipped with 68Gb of dynamic memory. the use of such a powerful machine is justified by the applications of the considered

---

problem. Indeed, it seems reasonable to consider such a computing power to tackle these problems during industrial processes.

We use the CPLEX 12.6 interactive solver as a black box to solve ILP formulations, with default configuration unless explicitly otherwise stated.

Furthermore we only consider CPU time. As a reminder the CPU time tries to measure the amount of time needed by a single processor machine to solve the problem. Roughly speaking, it can be seen as the sum of the computation durations overall all the cores being part of the computation.





# 10

## ILP formulations

---

---

### Contents

---

<b>10.1 Model description</b> . . . . .	<b>138</b>
10.1.1 MDA inspired formulation . . . . .	138
10.1.2 Hub set formulation . . . . .	142
<b>10.2 ILP comparison</b> . . . . .	<b>143</b>
10.2.1 LP relaxations comparison . . . . .	143
10.2.2 Computational Results . . . . .	148
<b>10.3 ILP based heuristics</b> . . . . .	<b>159</b>
10.3.1 Principle . . . . .	159
10.3.2 Computational Results . . . . .	160
<b>10.4 Sets aggregation technique</b> . . . . .	<b>166</b>
10.4.1 Modus Operandi . . . . .	166
10.4.2 Computational results . . . . .	166
<b>10.5 Conclusion</b> . . . . .	<b>167</b>

---

This chapter is devoted to present and compare some ILP based resolution methods. In a first time we introduce two formulations. The first one introduced by Reda et al. [RSS09] uses the fact that the Wafer-to-Wafer integration problems are particular cases of MDA. They propose an adaptation of the classical ILP formulation used to describe the problem. However we provide in this manuscript, the improved formulation of Dokka et al. [Dok+12] in which unnecessary constraints are removed. We provide a study on the limitations of such a formulation. We will indeed see that this formulation has memory complexity being exponential in the parameter  $m$ , we thus

try to determine what are the values of parameters  $n$  and  $m$  for which the formulation returns solution without exceeding memory capacity.

The second ILP has been introduced by Dokka et al. [Dok+12]. This formulation is an ad hoc formulation motivated by the memory issues of the previous formulation. In a first time, we propose a slight improvement of the formulation by reducing the number of constraints and then show that the LP relaxation of this formulation is weaker than the LP relaxation of the previous one.

Computational results are then provided. The aim is to numerically compare the formulations but also to study the behaviour of the second formulation in function of the input parameters  $(m, n, p, Y_W)$ . Based on the conclusion of this work, we propose ILP based heuristics in Chapter 10 and provide computational results to evaluate their performances compared to exact resolution methods.

To finish, we present in conclusion a list of experimentations designed to answer the question arising throughout the chapter.

## 10.1 Model description

In this section we present two ILP formulations modelizing  $\min \sum 0$ . But first let us make few remarks on conventions we use in this part. While the notions of vectors, coordinates and  $m$ -tuples are appropriate to handle theoretical aspects of the considered problem, we consider that the equivalent notions of wafers, dies (or positions) and stacks give a better intuition of the mechanisms involved in the practical resolution. That is why we will use this terminology in the sections devoted to computational results presentation.

### 10.1.1 MDA inspired formulation

#### Model overview

As pointed out in Section 5.1, the considered problems are some particular cases of the MDA problem. The ILP formulation used to modelize the latter can thus be used to modelize the  $\max \sum 1$  and  $\min \sum 0$  problems. Reda et al. [RSS09] first introduce this ILP formulation for  $\max \sum 1$ , while Dokka et al. [Dok+12] introduce a slightly improved version of the latter for  $\min \sum 0$ . In both cases, a preprocessing operation is used to compute the costs of the  $n^m$  possible  $m$ -tuples.

Once the costs have been computed, the formulation can be described as follows. Given an  $m$ -tuple  $a = (v^1, v^2, \dots, v^m) \in V^1 \times V^2 \times \dots \times V^m$ , we introduce the variable  $x_a$  such that:

$$\begin{cases} x_a = 1 & \text{if and only if } a \text{ represents a stack in the solution } S \\ x_a = 0 & \text{otherwise} \end{cases}$$

We get then the following formulation.

---

**Integer Linear Program VI** MDA inspired formulation
 

---

$$\text{Minimize } \sum_{a \in [n]^m} (p - c(a))x_a \quad (\text{VI.1})$$

$$\text{Subject To } \sum_{a: v \in a} x_a = 1 \quad \forall v \in \bigcup_{i=1}^m V^i \quad (\text{VI.2})$$

$$x_a \in \{0, 1\} \quad \forall a \in [n]^m \quad (\text{VI.3})$$


---

The  $nm$  Equations (VI.2) ensure that, given a vector  $v^i$  in any set  $V^i$ ,  $v^i$  belongs to exactly one  $m$ -tuple and thus ensure that all vectors belong to the solution and that the selected  $m$ -tuples are disjoint.

With its  $n^m$  integrality constraints, this formulation thus has  $n^m$  variables and  $nm + n^m$  constraints. While the computation runtime of ILP solvers can be important, we expect the memory to be the limiting factor. This motivates the next subsection.

**Limits of the formulation**

This subsection aims at determining the memory limits of the MDA inspired formulation in function of the four input parameters:

1. the number of sets  $m$ ,
2. the number of wafers per set  $n$ ,
3. the number of dies per wafer  $p$ ,
4. the average wafer yield  $Y_W$ .

To do so, ten instances have been generated for each combination of the following parameters values:

1.  $m \in \{3, 4, 5, 6\}$ ,
2.  $n \in \{25, 50, 75, 100, 125, 150, 175, 200, 225\}$ ,
3.  $p \in \{10, 100, 1000\}$ ,
4.  $Y_W \in \{50, 70, 90\}$ .

We then try to solve optimally every instance using CPLEX, the objective being to determine for each instance type, the number of instances exceeding memory capacity of the machine running the ILP solver. Results are summarized in Figure 10.1.

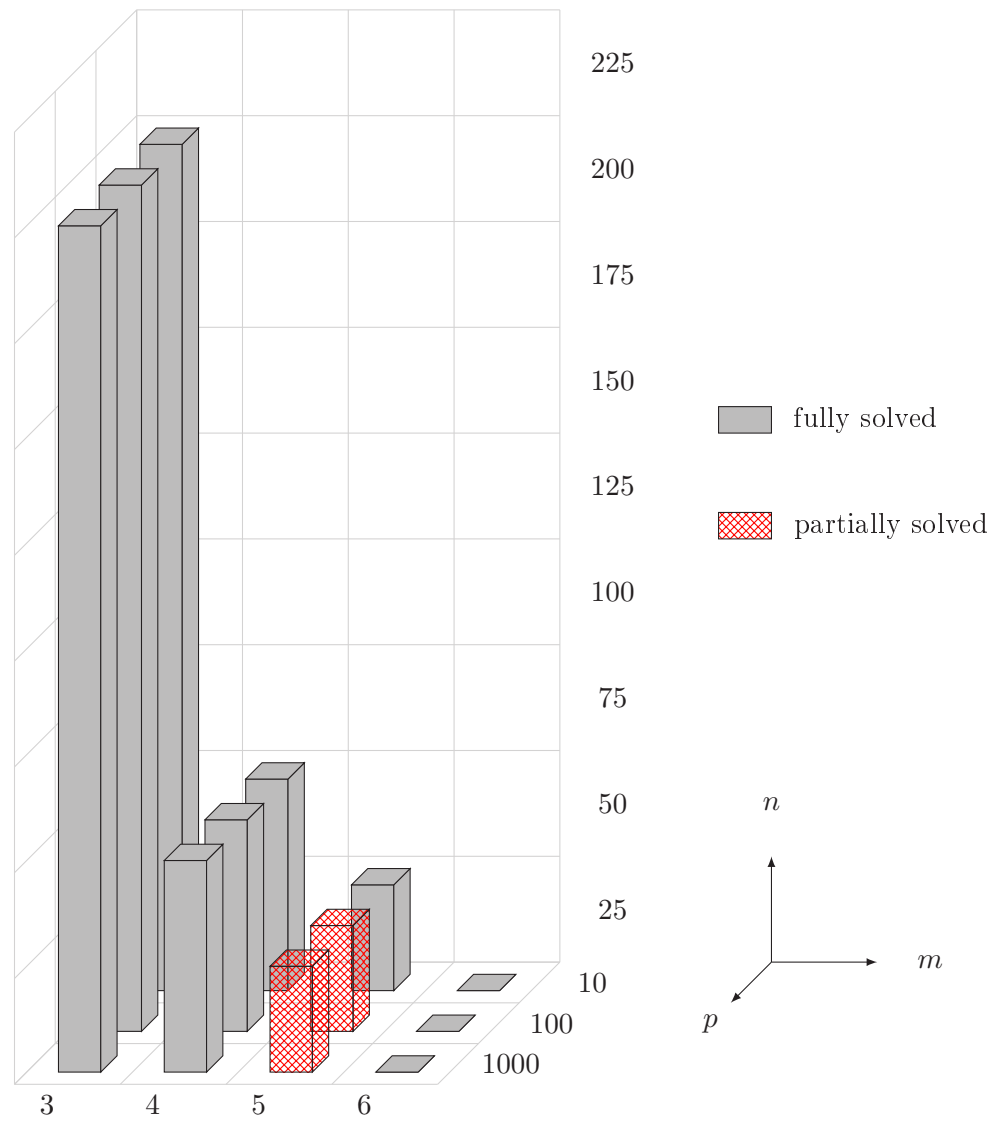


Figure 10.1: Representation of the instances that can be solved with Formulation VI without reaching the memory break point.

$m$	$n$	$p$	$Y_W$	# solved
5	25	10	50	10
			70	10
			90	10
		100	50	0
			70	1
			90	5
		1000	50	4
			70	2
			90	4

Table 10.1: Number of solved instances for  $m = 5$  and  $n = 25$  in function of  $p$  and  $Y_W$ .

The exponential dependence in  $m$  of the number of variables gives strong intuition on the memory consumption of the formulation when  $m$  increases. Indeed, with  $n^m$  variables, the number of wafers per set that the solver can handle is supposed to drastically decrease when  $m$  increases.

Such a behaviour is noticeable in Figure 10.1. With  $m = 3$  sets, the solver handles up to  $n = 225$  wafers per set, while it only handles up to  $n = 50$  wafers per set for  $m = 4$  sets. Furthermore, when the number of sets is equal to  $m = 5$ , the memory break point is reached from time to time even for sets containing only  $n = 25$  wafers. Details of solved instances are given in Table 10.1.

In light of these results, one can see that for  $m = 5$  and  $n = 25$ , the number of dies has strong impact on the number of solved instances. However, the number of variables has no dependence in  $p$ , neither does the number of constraints. This phenomenon could be the results of the following factors:

- the size of the ILP has logarithmic dependence in the product  $pY_W$ . Indeed, the expected cost of each  $m$ -tuple is equal to  $pY_W$ . The cost of each possible  $m$ -tuple appearing in the objective function of the formulation, the size of the ILP is expected to logarithmically increase when  $pY_W$  increases.
- while the 5\_25\_100 instances fit the expected behaviour when considering previous remark, the behaviour of the 5\_25\_1000 instances suggests that the memory overflow is not only the consequence of the increase of the factor  $pY_W$ . The low proportion of 5\_25\_1000\_70 instances solved could be explained by a combination of various mechanisms such as CPLEX internal branching strategies, inherent complexity of particular instances<sup>1</sup> or even external factors<sup>2</sup> However,

<sup>1</sup>Instances with wafers showing very high or very low similarity on failures distribution are easier to solve than instances composed with uniformly spread failures.

<sup>2</sup>The machine used to perform the experiments being shared, this behaviour could be explained by an increase of memory load due to other processes running on the latter.

more advanced studies on a more significant number of instances have to be performed in order to clearly explain which mechanisms are involved in this asymmetry between 5\_25\_10, 5\_25\_100 and 5\_25\_1000 instances.

Let us insist on the fact that considering these previous factors as impact factors on the memory consumption is only an hypothesis requiring to be confirmed with properly designed experiments.

### 10.1.2 Hub set formulation

The introduction of new ILP formulations is motivated by the limitations of the previously defined formulation (VI).

This alternative formulation, provided by Dokka et al. in [Dok+12] relies on the following observation: for every solution the  $m$ -tuples can be renumbered such that  $\forall i \in [m], j \in [n], v_j^i \in s_j$ <sup>3</sup>. Hence, instead of assigning vectors of  $V^i, i = 1, \dots, m$  to  $m$ -tuples  $s_j, j = 1, \dots, n$ , vectors of  $V^j, j = 2, \dots, m$  can be assigned to vectors  $v_j^1, j = 1, \dots, n$ .

In this case, the set  $V^1$  acts like a hub on which vectors of other sets will be *plugged*.

In a formal way, we define variable  $z(u, v), \forall u \in V^1, v \in \bigcup_{i=2, \dots, m} V^i$  such that:

$$\begin{cases} z(u, v) = 1 & \text{if and only if vectors } u \text{ and } v \text{ are contained in the same } m\text{-tuple} \\ z(u, v) = 0 & \text{otherwise} \end{cases}$$

In order to define the objective function, we define as well  $y(u, l), \forall u \in V^1, l \in [p]$ , such that  $y(u, l)$  is equal to the value of the  $l^{\text{th}}$  component of the  $m$ -tuple containing the vector  $u$ .

We can now present the ILP formulation (VII).

However this formulation can be slightly improved by taking into consideration the constraints imposed by Inequalities VII.3 to aggregate some of the Inequalities VII.4. Indeed, the  $z(u, v)$  are not independent over the  $v$  of a same set  $V^i$ . This leads us to introduce the Formulation (VIII).

On one hand, Equations (VIII.2) ensure that given a vector  $v \in \bigcup_{i=2, \dots, m} V^i, v$  is assigned to exactly one vector of  $V^1$ . On the other hand, Equations (VIII.3) ensure that each vector  $u \in V^1$  is assigned to exactly one vector in each  $V^i, \forall i = 2, \dots, m$ . Thus the combination of these two sets of Equations ensure that for any  $u \in V^1$  and any  $i \in [m]_2$ , exactly one pair  $(u, v^i)$  is selected in the solution. It follows that, given a set  $V^i, i \in [m]_2$ , the  $z(u, v)$  variables, for  $v \in V^i$ , define a complete assignment in  $V^1 \times V^i$ .

The Inequalities (VIII.4) force  $y(u, l)$  to be equal to 0 if at least one of the vectors selected in the  $m$ -tuple has a zero in position  $l$ . Otherwise, the maximization of the objective function forces  $y(u, l)$  to be equal to 1.

In this formulation, we count  $n^2(m-1) + np$  variables ( $n^2(m-1)$   $z$  variables and  $np$   $y$  variables) and  $n(m-1)(p+2)$  constraints. This formulation is significantly

<sup>3</sup>Remember that  $[j]_i$ , with  $i \leq j$  stands for  $\{i, i+1, \dots, j\}$

---

**Integer Linear Program VII** Hub set formulation
 

---

$$\text{Minimize } np - \sum_{u \in V^1} \sum_{l=1}^p y(u, l) \quad (\text{VII.1})$$

$$\text{Subject To } \sum_{u \in V^1} z(u, v) = 1 \quad \forall v \in \bigcup_{i=2}^m V^i \quad (\text{VII.2})$$

$$\sum_{v \in V^i} z(u, v) = 1 \quad \forall u \in V^1, i = 2, \dots, m \quad (\text{VII.3})$$

$$y(u, l) \leq \min(u[l], v[l])z(u, v) \quad \forall u \in V^1, i = 2, \dots, m, v \in V^i, l = 1, \dots, p \quad (\text{VII.4})$$


---

smaller than the MDA based formulation previously introduced. It follows that this formulation is able to handle instances with bigger  $m$  values without high memory requirement. However, we see in the next section that this huge lessening of memory usage leads to a lessening of the ILP strength.

## 10.2 ILP comparison

### 10.2.1 LP relaxations comparison

In this section we compare the LP relaxation of formulations (VI) and (VIII). We consider an alternative writing of formulation (VI), depicted by formulation (IX). In this writing, a variable  $x_a$ , associated to an  $m$ -tuple  $a = (v^1, v^2, \dots, v^m)$ , is turned into  $x(v^1, v^2, \dots, v^m)$ .

We also consider an alternative writing of formulation (VIII), depicted in (X) in which each set is considered independently. In other words, a variable  $z(u, v)$  for a vector  $u \in V^1$  and a vector  $v \in \bigcup_{i=2}^m V^i$  is turned into a variable  $z(u, i, j)$ ,  $i = 2, \dots, m$ ,  $j = 1, \dots, n$ .

We now show that the memory requirement reduction performed to design the Formulation (VIII) leads to a decrease of the LP relaxation strength.

**Theorem 10.1.** *The linear relaxation of Formulation (VI) is stronger than the linear relaxation of Formulation (VIII).*

*Proof.* To prove such a Theorem we show that every solution satisfying the constraints of Formulation (IX) also satisfies the constraints of Formulation (X).

We consider  $\tilde{S} = \{\tilde{x}(v^1, \dots, v^m)\}$  a general solution of the linear relaxation of Formulation (IX). We consider as well  $\hat{S} = \{\hat{y}(u, l), \hat{z}(u, i, j)\}$  a general solution of



---

**Integer Linear Program VIII** Improved hub set formulation
 

---

$$\text{Minimize } np - \sum_{u \in V^1} \sum_{l=1}^p y(u, l) \quad (\text{VIII.1})$$

$$\text{Subject To } \sum_{u \in V^1} z(u, v) = 1 \quad \forall v \in \bigcup_{i=2}^m V^i \quad (\text{VIII.2})$$

$$\sum_{v \in V^i} z(u, v) = 1 \quad \forall u \in V^1, i = 2, \dots, m \quad (\text{VIII.3})$$

$$y(u, l) \leq \sum_{v \in V^i} \min(u[l], v[l]) z(u, v) \quad \forall u \in V^1, i = 2, \dots, m, l = 1, \dots, p \quad (\text{VIII.4})$$


---

linear relaxation of Formulation (X).

We define the linear relations (10.1) that will allow us to compute the  $\hat{z}$  values from the  $\tilde{x}$  one.

$$\hat{z}(u, i, v^i) = \sum_{v^2 \in V^2} \cdots \sum_{v^{i-1} \in V^{i-1}} \sum_{v^{i+1} \in V^{i+1}} \cdots \sum_{v^m \in V^m} \tilde{x}(u, \dots, v^{i-1}, v^i, v^{i+1}, \dots, v^m) \quad (10.1)$$

Intuitively, the vector  $v^i$  is assigned to vector  $u$  if and only if there exists an  $m$ -tuple selected by Formulation (IX) containing both of them.

In a first time, we show that given  $\tilde{S}$ , we can construct  $\hat{S}$  that satisfies all constraints of Formulation (X). The  $\hat{z}$  variables are computed using linear relations (10.1).

Once  $\hat{z}(u, i, j)$  are defined,  $\hat{y}(u, l)$  are easily computable using Inequations (X.4) since  $u[l]$  and  $v^i[l]$  are part of the input.

Let us now show that  $\hat{S}$  satisfies all the constraints of (X). By definition of  $\hat{S}$ , the Inequations (X.4) are satisfied. We now consider Equations (IX.2)  $\forall i = 2, \dots, m, v^i \in V^i$ :

$$\sum_{v^1 \in V^1} \cdots \sum_{v^{i-1} \in V^{i-1}} \sum_{v^{i+1} \in V^{i+1}} \cdots \sum_{v^m \in V^m} \tilde{x}(v^1, \dots, v^i, \dots, v^m) = 1$$

---

**Integer Linear Program IX** Alternative MDA inspired formulation
 

---

$$\text{Maximize } \sum_{v^1 \in V^1} \cdots \sum_{v^m \in V^m} c((v^1, \dots, v^m)) x(v^1, v^2, \dots, v^m) \quad (\text{IX.1})$$

$$\text{Subject To } \sum_{v^1 \in V^1} \cdots \sum_{v^{i-1} \in V^{i-1}} \sum_{v^{i+1} \in V^{i+1}} \cdots \sum_{v^m \in V^m} x(v^1, \dots, v^i, \dots, v^m) = 1 \quad \forall v^i \in V^i, i = 1, \dots, m \quad (\text{IX.2})$$

$$x(v^1, \dots, v^m) \in \{0, 1\} \quad \forall v^1 \in V^1, \dots, v^m \in V^m \quad (\text{IX.3})$$


---

We highlight with brackets the right part of Equation (10.1):

$$\Leftrightarrow \sum_{v^1 \in V^1} \left( \sum_{v^2 \in V^2} \cdots \sum_{v^{i-1} \in V^{i-1}} \sum_{v^{i+1} \in V^{i+1}} \cdots \sum_{v^m \in V^m} \tilde{x}(v^1, \dots, v^i, \dots, v^m) \right) = 1$$

We can then replace this term by the left term of Equation(10.1):

$$\Leftrightarrow \sum_{v^1 \in V^1} (\bar{z}(v^1, i, v^i)) = 1$$

A solution satisfying Equation (IX.2) in Formulation (IX) for every  $i \in [m]_2$  leads to a solution that satisfies Equations (X.3) in Formulation (X).

Let us now consider in the same set of equations, the Equations (IX.2) for  $i = 1$  and  $\forall v^1 \in V^1$ . We can write, for every  $i' = 2, \dots, m$ :

$$\sum_{v^2 \in V^1} \cdots \sum_{v^{i'} \in V^{i'}} \cdots \sum_{v^m \in V^m} \tilde{x}(v^1, \dots, v^{i'}, \dots, v^m) = 1$$

As previously, we highlight with brackets the right part of Equation (10.1):

$$\Leftrightarrow \sum_{v^{i'} \in V^{i'}} \left( \sum_{v^2 \in V^2} \cdots \sum_{v^{i'-1} \in V^{i'-1}} \sum_{v^{i'+1} \in V^{i'+1}} \cdots \sum_{v^m \in V^m} \tilde{x}(v^1, \dots, v^{i'}, \dots, v^m) \right) = 1$$

---

**Integer Linear Program X** Alternative hub set formulation
 

---

$$\text{Maximize } \sum_{u \in V^1} \sum_l^p y(u, l) \quad (\text{X.1})$$

$$\text{Subject To } \sum_{u \in V^1} z(u, i, v^i) = 1 \quad \forall v^i \in V^i, i = 2, \dots, m \quad (\text{X.2})$$

$$\sum_{v^i \in V^i} z(u, i, v^i) = 1 \quad \forall u \in V^1, i = 2, \dots, m \quad (\text{X.3})$$

$$y(u, l) \leq \sum_{v^i \in V^i} \min(u[l], v^i[l]) z(u, i, v^i) \quad \forall u \in V^1, i = 2, \dots, m, l = 1, \dots, p \quad (\text{X.4})$$

$$z(u, i, v^i) \in \{0, 1\} \quad \forall u \in V^1, i = 2, \dots, m, v^i \in V^i \quad (\text{X.5})$$

$$y(u, l) \in \{0, 1\} \quad \forall u \in V^1, l = 1, \dots, p \quad (\text{X.6})$$


---

We replace:

$$\Leftrightarrow \sum_{v^{i'} \in V^{i'}} \left( \bar{z}(v^1, i, v^{i'}) \right) = 1$$

Equations (X.2) and (X.3) are thus satisfied by  $\hat{S}$ . Note that the existence of a solution  $\hat{S}$  that satisfies constraints of Formulation (X) is not sufficient to prove Theorem 10.1. The objective function of (X) is indeed not linear in regard to  $\hat{z}$  variables. Thus we have to prove that given a solution  $\tilde{S}$  for (IX) we can construct a solution  $\hat{S}$  such that:

$$c(\tilde{S}) \leq c(\hat{S}) \quad (\text{10.2})$$

Remark that, if we call  $v(v^1, v^2, \dots, v^m)$  the representative vector of the  $m$ -tuple  $(v^1, v^2, \dots, v^m)$ , thus we can write:

$$\begin{aligned} \forall i \in [m], v^i \in V^i, l \in [p] : \\ v(v^1, v^2, \dots, v^m)[l] &= \min \left( v^1[l], \min_{i \in [m]_2} v^i[l] \right) \end{aligned}$$

Since  $v[l] = \min_{i \in [m]_2} (v^i[l])$  obviously implies that  $\forall i = 2, \dots, m, v[l] \leq v^i[l]$ , we can write:

$$\Rightarrow \forall v^1 \in V^1, i \in [m]_2, v^i \in V^i, l \in [p] : \\ v(v^1, v^2, \dots, v^m)[l] \leq \min (v^1[l], v^i[l])$$

We can multiply the both sides by  $\hat{z}(v^1, i, v^i)$  without changing the orientation of the inequality since  $\hat{z}(v^1, i, v^i) \in \{0, 1\}$ :

$$\Rightarrow \forall v^1 \in V^1, i \in [m]_2, v^i \in V^i, l \in [p] : \\ v(v^1, \dots, v^m)[l] \cdot \hat{z}(v^1, i, v^i) \leq \min (v^1[l], v^i[l]) \cdot \hat{z}(v^1, i, v^i)$$

The inequality being satisfied for any individual  $v^i \in V^i$ , the inequality on the overall sum is also satisfied:

$$\Rightarrow \forall v^1 \in V^1, i \in [m]_2, l \in [p] : \\ \sum_{v^i \in V^i} v(v^1, \dots, v^m)[l] \cdot \hat{z}(v^1, i, v^i) \leq \sum_{v^i \in V^i} \min (v^1[l], v^i[l]) \cdot \hat{z}(v^1, i, v^i)$$

We use the Equation(10.1) to replace the  $\hat{z}$  variable in the left part of the inequality:

$$\Rightarrow \forall v^1 \in V^1, i \in [m]_2, l \in [p] : \\ \sum_{v^i \in V^i} v(v^1, \dots, v^m)[l] \cdot \sum_{v^1 \in V^1} \cdots \sum_{v^{i-1} \in V^{i-1}} \\ \sum_{v^{i+1} \in V^{i+1}} \cdots \sum_{v^m \in V^m} \tilde{x}(v^1, \dots, v^m) \leq \sum_{v^i \in V^i} \min (v^1[l], v^i[l]) \cdot \hat{z}(v^1, i, v^i)$$

Similarly to what has been done previously, the inequality being verified to every individual  $i$ , it is specifically verified for the minimum value over all the  $i$ :

$$\Rightarrow \forall v^1 \in V^1, l \in [p] : \\ \sum_{v^2 \in V^2} \cdots \sum_{v^{i-1} \in V^{i-1}} \sum_{v^i \in V^i} \sum_{v^{i+1} \in V^{i+1}} \\ \cdots \sum_{v^m \in V^m} v(v^1, \dots, v^m)[l] \cdot \tilde{x}(v^1, \dots, v^m) \leq \min_{i \in [m]_2} \left( \sum_{v^i \in V^i} \min (v^1[l], v^i[l]) \cdot \hat{z}(v^1, i, v^i) \right)$$

By using the Inequalities (X.4)

$$\Rightarrow \forall v^1 \in V^1, l \in [p] : \\ \sum_{v^2 \in V^2} \cdots \sum_{v^m \in V^m} v(v^1, \dots, v^m)[l] \cdot \tilde{x}(v^1, \dots, v^m) \leq \hat{y}(v^1, l)$$

We can sum up over the dies indices:

$\Rightarrow \forall v^1 \in V^1 :$

$$\sum_{v^2 \in V^2} \cdots \sum_{v^m \in V^m} c((v^1, \dots, v^m)) \cdot \tilde{x}(v^1, \dots, v^m) \leq \sum_{l \in [p]} \hat{y}(v^1, l)$$

We end the proof by summing up over the vectors of set  $V^1$ :

$$\Rightarrow \sum_{v^1 \in V^1} \cdots \sum_{v^m \in V^m} c((v^1, \dots, v^m)) \cdot \tilde{x}(v^1, \dots, v^m) \leq \sum_{v^1 \in V^1} \sum_{l \in [p]} \hat{y}(v^1, l)$$

Thus the profit of the computed solution  $\hat{S}$  is greater than the profit of  $\tilde{S}$ , this ends the proof of Theorem 10.1.  $\square$

We will see in the next section that this result is experimentally validated as the Formulation (IX) optimally solves  $\min \sum 0$  in less time than the Formulation (X).

## 10.2.2 Computational Results

In this section, we present a comparison of the Formulations (VI) and (VIII) based on computational results. In this purpose, ten instances have been generated for each combination of the following parameters values:

- $m \in \{3, 6, 9\}$ ,
- $n \in \{25, 50\}$ ,
- $p \in \{10, 100, 1000\}$ ,
- $Y_W \in \{50, 70, 90\}$ .

The following settings apply to all the experiments on ILP based resolution methods:

- the instances have been generated using the model of Di Natale et al. [Nat+13] described in Section 9.2.1.
- In order to define reference data, we implemented an oblivious algorithm randomly integrating wafers of the different sets. This approach is similar to the one presented by Reda et al. [RSS09]. These reference data offer two main advantages. First, the performance of this algorithm is dependent on the overall similarity over all the wafers of a same instance. Thus the performances of this algorithm may differ from the expected yield based on wafer yield. For example, given an instance with three sets of wafers, if the yield of the individual wafers is 90%, then the expected yield of every created stack is given by  $1 - 0.9^3 = 27\%$ .

Secondly, it allows us to determine whether our results can be (carefully) compared to the computational results of the literature even though the method used to generate wafers is different. Indeed, both of the generation methods are focused on failures clustering over the wafers of a same instances. It follows that similar performances between our random assignment algorithm and the one of Reda et al. [RSS09] may indicate that the correspondence between the failures maps of the wafer are similar. Obviously, such a correlation between random algorithm only gives information of the analogies between the wafers of different set but not on the inherent structure of the wafers failures maps. That is why the comparison between our results and other results of the literature based on different generation methods will be done very carefully and have to be considered as hypotheses need to be validated on instances generated in the same way.

- Two types of experiments are realized. The first one, that we will denote as the *long term* experiment, aims at determining whether we can hope to optimally solve the considered instances. In this experiment, the solver is interrupted after 54000 seconds<sup>4</sup> of computation. Furthermore, every CPLEX parameters is set to default values especially the *Feasibility Pump Heuristic* used to find feasible solutions. In this configuration, CPLEX tries to determine the most appropriate configuration to find the optimal solution. On the other hand, the *short term* experiment aims at giving informations on the behaviour of solving methods when the amount of time needed to get a solution is short. It thus aims at determining the efficiency of such a method when integrated into industrial processes. In this experiment, the solver is interrupted after 1800 seconds<sup>5</sup> of computation. For the latter, the *Feasibility Pump Heuristic* is configured to find good feasible solutions. In such a setting, the objective is no more to make easier the optimality proof of solutions but to quickly converge to the optimal value. This setting will be denoted in the following as the *Aggressive* setting.
- For ease of reading purposes, for each of the studied parameter, we only present a sample of the results. The complete results are given in Appendix.
- In the following tables, the average gap given is computed only on instances that led to an interruption of the solver. In other words, given a set of ten instances sharing the same parameters values and such that four instances have been solved optimally, the average gap is computed on the six instances for which resolution has been interrupted before reaching the optimal value. In a similar way, the average time is computed only on optimally solved instances.

---

<sup>4</sup>Recall that the time we consider is the CPU time.

<sup>5</sup>This represents one roughly one minute of real time computation on a dedicated machine with 32 cores.

Table 10.2: Impact of the Feasibility Pump Heuristic on the Formulation (VI)

	Default FPH (Long Term)	Agressive FPH (Short Term)	Speed Up Factor
3 25 10 90	0.78	0.61	1.28
3 25 10 70	2.13	0.51	4.18
3 25 10 50	3.21	0.51	6.30
3 25 100 50	9.59	0.82	11.70
3 25 100 70	2.36	0.47	5.02
3 25 100 90	3.12	0.6	5.20
3 25 1000 90	2.36	0.44	5.36
3 25 1000 70	20.17	0.83	24.30
3 25 1000 50	8.61	0.73	11.79
3 50 10 50	20.87	6.75	3.09
3 50 10 70	16.22	6.54	2.48
3 50 10 90	11.79	7.23	1.63
3 50 100 50	66.38	10.68	6.22
3 50 100 70	50.54	8.68	5.82
3 50 100 90	40.01	7.79	5.13
3 50 1000 50	73.37	15.57	4.71
3 50 1000 70	90.94	11.97	7.60
3 50 1000 90	60.54	7.93	7.63

### Impact of the *Feasibility Pump Heuristic*

We first investigate the impact of the *Feasibility Pump Heuristic* on the performances of Formulations (VI) and (VIII). First and foremost we consider the Formulation (VI). We obviously restrict ourselves to instances for which results are returned, *i.e.* instances in which  $m = 3$ . Furthermore since these instances are all optimally solved in both of the experiments, we restrict our consideration to the computation time.

We can see in Table 10.2 that no matter the set of instances we consider, using *Aggressive Feasibility Pump Heuristic* significantly decreases the computation time.

Considering the Formulation (VIII) the comparison is a little bit more complicated because of the amount and the heterogeneity of the data and the difference of computation time before interruption. While an overview of the results on Formulation (VIII) is given by Table A.3 in Annex, the former remark motivates a decomposition of the analysis. In a first time we consider the sets of instances that are entirely solved to the optimality in both experiments. The computation times are given on Table 10.3. As for Formulation (VI), the results show that the *Aggressive Feasibility Pump Heuristic* significantly decreases the computation time except for the 9\_25\_10\_90 instances.

In a second time, we consider sets of instances such that none of the instances have been solved to optimality. Results are depicted in Table 10.4. Except for the sets of instances 6\_50\_1000\_50, 9\_25\_1000\_70 and 9\_25\_1000\_90 the gap between the best known lower bound and the best integer solution is larger in experiment with the *Aggressive Feasibility Pump Heuristic*. The average gap increase is equal to 5.27% while the average increase of failures is equal to 1.43%. Furthermore, the results show that for big values of  $p$  the results computed by the experiment with aggressive settings in 1800 seconds are better than those of experiment with default settings in 54000 seconds.

Table 10.3: Impact of the Feasibility Pump Heuristic on the Formulation (VIII) restricted to sets of instances solved to optimality in both experiment.

	Default FPH (Long Term)	Aggressive FPH (Short Term)	Speed up factor
3 25 10 50	12.26	1.353	9.06
3 25 10 70	24.26	1.475	16.45
3 25 10 90	2.26	0.259	8.73
3 50 10 50	69.13	19.299	3.58
3 50 10 70	106.50	22.292	4.78
3 50 10 90	20.22	9.233	2.19
6 25 10 90	40.55	18.313	2.21
9 25 10 90	195.73	243.776	0.80

We also can remark that the set of instances 6\_50\_1000\_70, 6\_50\_1000\_90, 9\_50\_1000\_90, 9\_50\_1000\_70 and 9\_50\_1000\_90 with the largest gap increase (more than 20%) are also sets of instances for which the *aggressive Feasibility Pump Heuristic* improves the objective function with less computation times.

It follows that comparing gap variations does not seem to be an appropriate way to evaluate efficiency of the aggressive setting with respect to the default configuration. This motivates an analysis based on the failure rate per stack. However the conclusion we can make on such a restrictive analysis have to be seen as research trails. The latter have thus to be confirmed by properly designed experiments. We propose possible experiments in Section 10.5.

To finish we consider the rest of the sets of instances, *i.e.* sets of instances for which at least one of the instance has been optimally solved in at least one of the experiment. Results are depicted in Table 10.5. The comparison between experiments on these instances is the most sensitive since the previous approaches do not seem to be appropriate. A time-based comparison is obviously not relevant due to the large difference between the maximum computation duration. Furthermore, as presented in previous paragraph, gap based comparisons are not relevant. We thus focus on the failure rate per stack.

Set of instances 6\_25\_10\_50 highlight the difficulties of such a comparison. The 6\_25\_10\_50 instances highlight the artificial increase of the Gap value when the number of optimally solved instances increases. The results show that despite the large difference of maximum computation duration, the increase of the number of failures is limited. Indeed, for the set of instances 6\_25\_10\_50, an increase of failures of 0.12% corresponds to an average increase equal to 0.3 failure per instance. For the set of instances 6\_25\_10\_70 the average increase of failures is equal to 1 extra failure per wafer.

To conclude, despite a limited computation duration, the solution quality of the *short term* experiment is only slightly reduced thanks to the *Aggressive Feasibility Pump Heuristic*. This configuration appears to be particularly appropriate to handle our problems. Indeed, according to CPLEX manual, such a settings can lead to difficulty in finding feasible solution. However, feasible solutions are easy to find due to



Table 10.4: Impact of the Feasibility Pump Heuristic on the Formulation (VIII) restricted to set of instances such that no instance have been solved to optimality in both experiments.

	Default FPH (Long Term)		Aggressive FPH (Short Term)		Differences	
	Gap	Fails	Gap	Fails	Gap	Fails
3 25 100 50	5.06%	79.09%	5.79%	79.69%	0.73%	0.60%
3 25 100 70	7.05%	56.56%	8.16%	57.16%	1.11%	0.60%
3 25 100 90	2.78%	20.90%	3.93%	21.11%	1.15%	0.21%
3 25 1000 50	9.17%	83.67%	9.69%	84.14%	0.52%	0.47%
3 25 1000 70	13.59%	61.39%	14.23%	61.84%	0.64%	0.45%
3 25 1000 90	12.23%	23.78%	13.14%	24.04%	0.91%	0.26%
3 50 100 50	7.01%	78.33%	7.33%	78.49%	0.32%	0.16%
3 50 100 70	9.83%	55.79%	11.12%	56.35%	1.29%	0.56%
3 50 100 90	7.13%	20.56%	8.55%	20.76%	1.42%	0.20%
3 50 1000 50	10.83%	83.69%	12.02%	84.81%	1.19%	1.12%
3 50 1000 70	16.01%	61.28%	17.42%	62.32%	1.41%	1.04%
3 50 1000 90	17%	23.93%	19.32%	24.61%	2.32%	0.68%
6 25 100 50	16.13%	91.68%	17.03%	92.44%	0.90%	0.76%
6 25 100 70	27.17%	75.84%	30.83%	79.78%	3.66%	3.94%
6 25 100 90	28.87%	32.50%	33.22%	34.61%	4.35%	2.11%
6 25 1000 50	18.60%	96.96%	18.95%	97.31%	0.35%	0.35%
6 25 1000 70	32.15%	83.70%	32.65%	84.08%	0.50%	0.38%
6 25 1000 90	37.67%	38.00%	37.92%	38.08%	0.25%	0.08%
6 50 10 50	3.02%	65.17%	3.50%	65.6 %	0.48%	0.43%
6 50 10 70	6.10%	43.72%	7.43%	44.32%	1.33%	0.60%
6 50 100 50	19.18%	91.81%	22.00%	94.54%	2.82%	2.73%
6 50 100 70	31.22%	76.23%	37.86%	82.71%	6.64%	6.48%
6 50 100 90	38.61%	33.29%	45.05%	36.9 %	6.44%	3.61%
6 50 1000 50	21.05%	97.07%	21.00%	96.96%	-0.05%	-0.11%
6 50 1000 70	35.14%	83.40%	64.05%	82.98%	28.91%	-0.42%
6 50 1000 90	45%	37.48%	72.58%	37.48%	27.58%	0.00%
9 25 10 50	1.69 %	69.88%	2.92%	70.64%	1.23%	0.76%
9 25 10 70	5.15 %	49.6%	6.90%	50.56%	1.75%	0.96%
9 25 100 50	19.04%	95.56%	20.09%	96.81%	1.05%	1.25%
9 25 100 70	35.11%	86.66%	37.12%	89.12%	2.01%	2.46%
9 25 100 90	43.85%	43.06%	46.42%	44.87%	2.57%	1.81%
9 25 1000 50	19.37%	99.39%	19.65%	99.45%	0.28%	0.06%
9 25 1000 70	37.17%	93.23%	37.08%	92.62%	-0.09%	-0.61%
9 25 1000 90	48.03%	48.26%	46.52%	46.83%	-1.51%	-1.43%
9 50 10 50	5.62%	67.78%	17.00%	77.1 %	11.38%	9.32%
9 50 10 70	9.90%	46.28%	26.25%	56.62%	16.35%	10.34%
9 50 100 50	23.22%	97.1%	25.33%	98.95%	2.11%	1.85%
9 50 100 70	41.11%	88.61%	43.09%	91.17%	1.98%	2.56%
9 50 100 90	48.18%	40.48%	54.13%	45.32%	5.95%	4.84%
9 50 1000 50	22.13%	99.39%	50.20%	99.31%	28.07%	-0.08%
9 50 1000 70	40.13%	91.98%	67.38%	91.48%	27.25%	-0.50%
9 50 1000 90	53.77%	46.30%	77.35%	45.40%	23.58%	-0.90%
					5.27%	1.43 %

Table 10.5: Impact of the Feasibility Pump Heuristic on the Formulation (VIII) on sets of instances containing at least one instance solved to the optimality by at least one of the two formulations.

	Default FPH (Long Term)				Aggressive FPH (Short Term)				Fails
	Time	#	Gap	Fails	Time	#	Gap	Fails	Fails
6 25 10 50	14104.79	8	5.10%	68.2%	1078.943	3	1.76%	68.32%	0.12%
6 25 10 70	24912.32	4	2.65%	46.04%	-	0	2.67%	46.44%	0.4%
6 50 10 90	16464.56	8	6.25%	15.4%	321.987	6	6.37%	15.4 %	0%
9 50 10 90	6499.96	5	3.23%	16.28%	198.587	4	3.25%	16.28%	0%
									0.13%

the particular structure of the considered problems.

In the following, we only consider the results of the *short term* experiment. This choice is motivated by the former study. Furthermore, this experimentation gives informations on the performance of this formulation in real world applications.

### Impact of the number of sets

We then investigate the impact of the number of sets on the performances of the different ILP formulations. However the study of this parameter on the memory complexity of the Formulation (VI) performed in Section 10.1.1 led us to investigate the performance on the latter in function of  $m$ . This investigation gives strong clues on the fact that this formulation is unadapted for the resolution of large scale instances.

We thus focus on the performances of the Formulation (VIII). The Table 10.6 depicts the evolution of the number of exactly solved instances and the amount of time needed to solve instances optimally in function of  $m$ .

As expected, the results obviously show that the number of failures increases with the number of wafers integrated in the stacks. The other expected result is that the increase of failures from 3 to 6 sets is more important than the one between 6 and 9 layers in the stacks. This can be explained by the failure clustering in the wafer map. Intuitively, since the failures are localized at roughly the same location, the probability to turn a viable position into a faulty one decreases with the number of failures and thus with the number of sets.

Contrary to Formulation (VI), the Formulation (VIII) is able to handle and to return feasible solutions for large scale instances. However, the solution quality, especially for instances with  $Y_W \leq 70$  is very bad. Furthermore, when compared to random assignment algorithm, the improvement is low. For example, we can see an improvement of the quality of 0,06% on the 9\_25\_1000\_50 instances. This corresponds to a reduction of only 15 faulty dies on an instances that contains 25000 of them.

Table 10.6: Impact of the number of sets on the Formulations (VI) and (VIII)

	random	Formulation (VI)				Formulation (VIII)				
	Fails	Time	#	Fails	Ratio	Time	#	Gap	Fails	Ratio
3 25 10 90	23.76%	0.616	10	14.48%	0.609	0.259	10	-	14.48%	0.609
6 25 10 90	35.16%	-	-	-	-	18.313	10	-	17.28%	0.491
9 25 10 90	44.8%	-	-	-	-	243.776	10	-	18.4 %	0.411
3 25 10 70	62.32%	0.517	10	41.12%	0.660	1.475	10	-	41.12%	0.660
6 25 10 70	84.24%	-	-	-	-	-	0	2.67%	46.44%	0.551
9 25 10 70	91.88%	-	-	-	-	-	0	6.90%	50.56%	0.550
3 25 10 50	85.68%	0.516	10	63.67%	0.743	1.353	10	-	63.67%	0.743
6 25 10 50	97%	-	-	-	-	1078.943	3	1.76%	68.32%	0.704
9 25 10 50	99.52%	-	-	-	-	-	0	2.92%	70.64%	0.710
3 25 1000 90	24.91%	0.44	10	23.54%	0.945	-	0	13.14%	24.04%	0.965
6 25 1000 90	39.37%	-	-	-	-	-	0	37.92%	38.08%	0.967
9 25 1000 90	48.72%	-	-	-	-	-	0	46.52%	46.83%	0.961
3 25 1000 70	63.07%	0.837	10	60.91%	0.966	-	0	14.23%	61.84%	0.981
6 25 1000 70	84.66%	-	-	-	-	-	0	32.65%	84.08%	0.993
9 25 1000 70	93.18%	-	-	-	-	-	0	37.08%	92.62%	0.994
3 25 1000 50	85.52%	0.737	10	83.32%	0.974	-	0	9.69%	84.14%	0.984
6 25 1000 50	97.43%	-	-	-	-	-	0	18.95%	97.31%	0.999
9 25 1000 50	99.51%	-	-	-	-	-	0	19.65%	99.45%	0.999

Table 10.7: Impact of the number of wafers per set on the performances of ILP formulations.

	random	Formulation (VI)				Formulation (VIII)				
	Fails	Time	#	Fails	Ratio	Time	#	Gap	Fails	Ratio
3 25 10 90	23.76%	0.616	10	14.48%	0.609	0.259	10	-	14.48%	0.609
3 50 10 90	24.36%	7.231	10	13.56%	0.557	9.233	10	-	13.56%	0.557
3 25 100 70	63.00%	0.472	10	56.04%	0.890	-	0	8.16%	57.16%	0.907
3 50 100 70	62.58%	8.685	10	54.63%	0.873	-	0	11.12%	56.35%	0.900
3 25 1000 50	85.52%	0.737	10	83.32%	0.974	-	0	9.69%	84.14%	0.984
3 50 1000 50	84.99%	15.572	10	82.56%	0.971	-	0	12.02%	84.81%	0.998
6 25 10 50	97%	-	-	-	-	1078.943	3	1.76%	68.32%	0.704
6 50 10 50	96.76%	-	-	-	-	-	0	3.50%	65.6 %	0.678
6 25 10 90	35.16%	-	-	-	-	18.313	10	-	17.28%	0.491
6 50 10 90	37.52%	-	-	-	-	321.987	6	6.37%	15.4 %	0.410
6 25 1000 70	84.66%	-	-	-	-	-	0	32.65%	84.08%	0.993
6 50 1000 70	83.52%	-	-	-	-	-	0	64.05%	82.98%	0.994
9 25 10 50	99.52%	-	-	-	-	-	0	2.92%	70.64%	0.710
9 50 10 50	99.32%	-	-	-	-	-	0	17.00%	77.1 %	0.776
9 25 10 90	44.8%	-	-	-	-	243.776	10	-	18.4 %	0.411
9 50 10 90	45.14%	-	-	-	-	198.587	4	3.25%	16.28%	0.361
9 25 1000 70	93.18%	-	-	-	-	-	0	37.08%	92.62%	0.994
9 50 1000 70	91.99%	-	-	-	-	-	0	67.38%	91.48%	0.994

### Influence of the number of wafers per set

In this section we focus on the influence of parameter  $n$  on the performances of the ILP formulations. Some interesting results are depicted in Table 10.7 while the whole results are depicted in Table A.4. Note that we will not discuss about the repercussions on the objective value of the set aggregation. This technique, consisting in the construction of instances with 50, 75, 100, . . . wafers per lot by assembling instances with 25 wafers per lot, will be subject to a dedicated study in Section 10.4.

The results first show that the computational complexity of these problems increases with the number of wafers. This behaviour is particularly noticeable on instances 6\_25\_10\_90 and 9\_25\_10\_90. Moreover, we expect an increase of the solution quality due to the increase of  $n$ , since, given a wafer of the instance, the probability of finding a wafer with similar failures map is higher.

The results are generally consistent with the expected behaviour. However the high computational complexity of some instances (especially instances with a high value for  $p$ ) leads to a decrease of the quality solution when solved with Formulation (VIII).

### Influence of the average wafer yield

We now focus on the impact of the average wafer yield  $Y_W$  on the Formulation (VIII). A selection of results is given in Table 10.8.

In a first time, we consider the impact of the wafer yield on the solution quality. A first obvious result is the increase of overall solution quality when  $Y_W$  increases. Furthermore, when compared to random assignments, a high individual wafer yield implies a smaller value of the ratio of the number of failures in the ILP formulation over the number of failures in the random assignment, no matter the considered formulation.

In a second time we consider the Formulation (VI). For the latter, the results show that the more the wafer yield is low, the more the computation time is important. This can be explained by the logarithmic dependence of the formulation in the value of  $Y_W$ .

The behaviour of the Formulation (VIII) is different from the former behaviour. Indeed, the hardest instances, when considering the computation time, are instances such that  $Y_W = 70$ . Such a comportment could be explained by the model used in the instance generator. This model designed to cluster the failures on the rim of the wafers may lead to particular failures maps when  $Y_W \approx 70$ . In these instances, one can see that the failures map can be cut into three more or less distinct areas.

1. The outer rim, on which probability of failure is very high due to a combination of quite low  $Y_W$  and high distance to the center of wafers. This area is relatively thin, since the high density of failures leads to a rapid decrease of failure probability.
2. The inner rim, which is the most important part in term of area. In this area, the probability of failure is close to 50%, leading to well spread failures on this area.

Table 10.8: Impact of  $Y_W$  on the performances of ILP formulations.

	random	Formulation (VI)				Formulation (VIII)				
	Fails	Time	#	Fails	Ratio	Time	#	Gap	Fails	Ratio
3 25 10 50	85.68%	0.516	10	63.67%	0.743	1.353	10	-	63.67%	0.743
3 25 10 70	62.32%	0.517	10	41.12%	0.660	1.475	10	-	41.12%	0.660
3 25 10 90	23.76%	0.616	10	14.48%	0.609	0.259	10	-	14.48%	0.609
3 25 100 50	85.69%	0.821	10	78.62%	0.917	-	0	5.79%	79.69%	0.930
3 25 100 70	63.00%	0.472	10	56.04%	0.890	-	0	8.16%	57.16%	0.907
3 25 100 90	24.99%	0.6	10	20.81%	0.833	-	0	3.93%	21.11%	0.845
3 25 1000 50	85.52%	0.737	10	83.32%	0.974	-	0	9.69%	84.14%	0.984
3 25 1000 70	63.07%	0.837	10	60.91%	0.966	-	0	14.23%	61.84%	0.981
3 25 1000 90	24.91%	0.44	10	23.54%	0.945	-	0	13.14%	24.04%	0.965
3 50 10 50	84.2%	6.759	10	60.67%	0.721	19.299	10	-	60.67%	0.721
3 50 10 70	62.66%	6.544	10	38.96%	0.622	22.292	10	-	38.96%	0.622
3 50 10 90	24.36%	7.231	10	13.56%	0.557	9.233	10	-	13.56%	0.557
3 50 100 50	84.97%	10.684	10	77.22%	0.909	-	0	7.33%	78.49%	0.924
3 50 100 70	62.58%	8.685	10	54.63%	0.873	-	0	11.12%	56.35%	0.900
3 50 100 90	24.85%	7.798	10	20.04%	0.806	-	0	8.55	20.76%	0.836
3 50 1000 50	84.99%	15.572	10	82.56%	0.971	-	0	12.02%	84.81%	0.998
3 50 1000 70	62.51%	11.977	10	60.08%	0.961	-	0	17.42%	62.32%	0.997
3 50 1000 90	24.81%	7.931	10	23.26%	0.937	-	0	19.32%	24.61%	0.992
6 25 10 50	97%	-	-	-	-	1078.943	3	1.76%	68.32%	0.704
6 25 10 70	84.24%	-	-	-	-	-	0	2.67%	46.44%	0.551
6 25 10 90	35.16%	-	-	-	-	18.313	10	-	17.28%	0.491
6 50 100 50	97.03%	-	-	-	-	-	0	22.00%	94.54%	0.974
6 50 100 70	83.57%	-	-	-	-	-	0	37.86%	82.71%	0.990
6 50 100 90	38.46%	-	-	-	-	-	0	45.05%	36.9 %	0.959
9 25 10 50	99.52%	-	-	-	-	-	0	2.92%	70.64%	0.710
9 25 10 70	91.88%	-	-	-	-	-	0	6.90%	50.56%	0.550
9 25 10 90	44.8%	-	-	-	-	243.776	10	-	18.4 %	0.411
9 50 1000 50	99.38%	-	-	-	-	-	0	50.20%	99.31%	0.999
9 50 1000 70	91.99%	-	-	-	-	-	0	67.38%	91.48%	0.994
9 50 1000 90	46.65%	-	-	-	-	-	0	77.35%	45.40%	0.973

This area is, to our opinion, the one that explain the computational complexity of these instances due to its area size.

3. The wafer center, in which the failure density is very low.

Note that this partition into areas is accurate no matter the average wafer yield. However, the values of  $Y_W$  determine the size of each of these areas. The more the areas 1 and 3 are important, the more the instances are easy to solve with Formulation (VIII) since the combinatorics in this surfaces is quite limited. When considering instances with  $Y_W = 70$  likely have a important inner rim area, explaining at the same time, the increase of computation time when compared to instances where  $Y_W = 50$  or  $Y_W = 90$ .

More generally, based on these observations, we conjecture that the Formulation (VIII) has strong dependency in the homogeneity of the failures distribution. We propose in Section 10.5 some experiments to validate the former assumption.

### Impact of the number of dies

To finish, we focus on the parameter  $p$ . Let us first consider the influence of such a parameter on the solution quality. As expected, the number of dies per wafer is not an impact factor on the quality of solution returned by random assignment. Indeed, the die yield being fixed between the compared sets of instances, the number of failures is proportional to the number of dies. Therefore, when considering the failure clustering, the performance of random assignment should not be impacted by an increase of the number dies per wafers. Note that this observation is not at variance as the observation given by Reda et al. in [RSS09]. Indeed, in their experimentation on the impact of the number of dies, the overall failure is not constant. In fact, they consider the impact of the die area on solution quality, the size of a wafer and the defect density being constant. It follows that the number of failures per wafer is a constant implying that the failures rate per wafer increases as the number of dies decreases.

More surprisingly, the quality of the solutions computed by ILP formulations degrades as the number of dies increase. We explain this behaviour by the impact of the number of dies on the failures distribution. Indeed, with a high number of dies, the failures clustering is less pronounced as the sensitivity of failure probability to the distance to the center is more fine-grained. It follows that even though the general distribution of failures over the wafers are similar, the exact location of these failures differs from a wafer to another. This hypothesis is strengthen by the asymptotic increase of the failures rate in function of  $p$ . Indeed, the increase of the failures rate between instances with  $p = 10$  and  $p = 100$  is more important than the one between instances with  $p = 100$  and  $p = 1000$ . This confirms that the general distribution of failures are similar between instances. However, this asymptotic behaviour needs to be validated with experiments on larger number of dies.

When considering the efficiency of the ILP formulations, the analysis of Section 10.1 gives us clues on the sensibility of the latter in function of  $p$ . The only dependence in  $p$  of the Formulation (VI) is that the latter has logarithmic size in function of  $p$ . This formulation is thus expected to show limited variations as the parameter increases.

Table 10.9: Impact of  $p$  on the performances of ILP formulations.

	random	Formulation (VI)				Formulation (VIII)				
	Fails	Time	#	Fails	Ratio	Time	#	Gap	Fails	Ratio
3 25 10 50	85.68%	0.516	10	63.67%	0.743	1.353	10	-	63.67%	0.743
3 25 100 50	85.69%	0.821	10	78.62%	0.917	-	0	5.79%	79.69%	0.930
3 25 1000 50	85.52%	0.737	10	83.32%	0.974	-	0	9.69%	84.14%	0.984
3 25 10 70	62.32%	0.517	10	41.12%	0.660	1.475	10	-	41.12%	0.660
3 25 100 70	63.00%	0.472	10	56.04%	0.890	-	0	8.16%	57.16%	0.907
3 25 1000 70	63.07%	0.837	10	60.91%	0.966	-	0	14.23%	61.84%	0.981
3 25 10 90	23.76%	0.616	10	14.48%	0.609	0.259	10	-	14.48%	0.609
3 25 100 90	24.99%	0.6	10	20.81%	0.833	-	0	3.93%	21.11%	0.845
3 25 1000 90	24.91%	0.44	10	23.54%	0.945	-	0	13.14%	24.04%	0.965
3 50 10 50	84.2%	6.759	10	60.67%	0.721	19.299	10	-	60.67%	0.721
3 50 100 50	84.97%	10.684	10	77.22%	0.909	-	0	7.33%	78.49%	0.924
3 50 1000 50	84.99%	15.572	10	82.56%	0.971	-	0	12.02%	84.81%	0.998
3 50 10 70	62.66%	6.544	10	38.96%	0.622	22.292	10	-	38.96%	0.622
3 50 100 70	62.58%	8.685	10	54.63%	0.873	-	0	11.12%	56.35%	0.900
3 50 1000 70	62.51%	11.977	10	60.08%	0.961	-	0	17.42%	62.32%	0.997
3 50 10 90	24.36%	7.231	10	13.56%	0.557	9.233	10	-	13.56%	0.557
3 50 100 90	24.85%	7.798	10	20.04%	0.806	-	0	8.55	20.76%	0.836
3 50 1000 90	24.81%	7.931	10	23.26%	0.937	-	0	19.32%	24.61%	0.992
6 25 10 50	97%	-	-	-	-	1078.943	3	1.76%	68.32%	0.704
6 25 100 50	97.32%	-	-	-	-	-	0	17.03%	92.44%	0.950
6 25 1000 50	97.43%	-	-	-	-	-	0	18.95%	97.31%	0.999
6 25 10 70	84.24%	-	-	-	-	-	0	2.67%	46.44%	0.551
6 25 100 70	84.64%	-	-	-	-	-	0	30.83%	79.78%	0.942
6 25 1000 70	84.66%	-	-	-	-	-	0	32.65%	84.08%	0.993
6 25 10 90	35.16%	-	-	-	-	18.313	10	-	17.28%	0.491
6 25 100 90	39.28%	-	-	-	-	-	0	33.22%	34.61%	0.881
6 25 1000 90	39.37%	-	-	-	-	-	0	37.92%	38.08%	0.967
6 50 10 50	96.76%	-	-	-	-	-	0	3.50%	65.6 %	0.678
6 50 100 50	97.03%	-	-	-	-	-	0	22.00%	94.54%	0.974
6 50 1000 50	97.06%	-	-	-	-	-	0	21.00%	96.96%	0.999
6 50 10 70	82.98%	-	-	-	-	-	0	7.43%	44.32%	0.534
6 50 100 70	83.57%	-	-	-	-	-	0	37.86%	82.71%	0.990
6 50 1000 70	83.52%	-	-	-	-	-	0	64.05%	82.98%	0.994
6 50 10 90	37.52%	-	-	-	-	321.987	6	6.37%	15.4 %	0.410
6 50 100 90	38.46%	-	-	-	-	-	0	45.05%	36.9 %	0.959
6 50 1000 90	38.05%	-	-	-	-	-	0	72.58%	37.48%	0.985
9 25 10 50	99.52%	-	-	-	-	-	0	2.92%	70.64%	0.710
9 25 100 50	99.49%	-	-	-	-	-	0	20.09%	96.81%	0.973
9 25 1000 50	99.51%	-	-	-	-	-	0	19.65%	99.45%	0.999
9 25 10 70	91.88%	-	-	-	-	-	0	6.90%	50.56%	0.550
9 25 100 70	93.34%	-	-	-	-	-	0	37.12%	89.12%	0.955
9 25 1000 70	93.18%	-	-	-	-	-	0	37.08%	92.62%	0.994
9 25 10 90	44.8%	-	-	-	-	243.776	10	-	18.4 %	0.411
9 25 100 90	49.18%	-	-	-	-	-	0	46.42%	44.87%	0.912
9 25 1000 90	48.72%	-	-	-	-	-	0	46.52%	46.83%	0.961
9 50 10 50	99.32%	-	-	-	-	-	0	17.00%	77.1 %	0.776
9 50 100 50	99.40%	-	-	-	-	-	0	25.33%	98.95%	0.995
9 50 1000 50	99.38%	-	-	-	-	-	0	50.20%	99.31%	0.999
9 50 10 70	91.36%	-	-	-	-	-	0	26.25%	56.62%	0.620
9 50 100 70	92.08%	-	-	-	-	-	0	43.09%	91.17%	0.990
9 50 1000 70	91.99%	-	-	-	-	-	0	67.38%	91.48%	0.994
9 50 10 90	45.14%	-	-	-	-	198.587	4	3.25%	16.28%	0.361
9 50 100 90	46.98%	-	-	-	-	-	0	54.13%	45.32%	0.965
9 50 1000 90	46.65%	-	-	-	-	-	0	77.35%	45.40%	0.973

On the other hand, the Formulation (VIII) has  $n^2(m - 1) + np$  variables and  $n(m - 1)(p + 2)$  constraints. We thus expect  $p$  to be an impact factor on the efficiency of this formulation.

Both of these expectations are validated by our experiment. Indeed, even for high values of  $m$  and  $n$ , we can expect optimal values to be computed by Formulation (VIII) if the number of dies is low. This motivates the design of a new heuristic based on this ILP formulation.

## 10.3 ILP based heuristics

### 10.3.1 Principle

The previous section shows that we can expect the Formulation (VIII) to be efficient when the number of dies of the instance is low. It seems thus natural to use this property to design heuristics. One possible way to reach this goal is to use the following scheme:

---

**Algorithm 3:** Reduced dies heuristic principle

---

**Data:** An integer  $\phi$   
 Select  $\phi$  positions;  
 Solve optimally the instance reduced to the  $\phi$  selected positions;  
 Apply previously computed assignment on the initial instance;

---

It seems natural that the number  $\phi$  of dies in the newly created instance as well as the way to select these  $\phi$  positions are impact factors on the efficiency of such a heuristic. In the following we propose two ways to select these dies and perform tests for  $\phi \in \{10, 50\}$  on the instances used in the previous ones.

Let us now describe the position selection procedures. The first one is led by the following principle: “Given an instance of wafer-to-wafer integration problem, we may want to ensure that the positions having the best yield are lead to viable position in the created stacks.” The objective is then to identify the positions maximizing the number of good dies in the reduced instance. Based on this principle, we define the heuristic 4.

---

**Algorithm 4:** Viable dies preserving heuristic (VDP)

---

**Data:** An integer  $\phi$   
 Select the  $\phi$  positions that maximize the overall number of good dies;  
 Solve optimally the instance reduced to the  $\phi$  selected positions;  
 Apply previously computed assignment on the initial instance;

---

The second one is led by the following principle: “Given an instance of wafer-to-wafer integration problem, we may try to maximize the number of saved positions in the created stacks, as other positions have higher probability to lead to viable positions.”



The objective now is therefore to identify the positions maximizing the overall number of bad dies in the reduced instance. We thus define the heuristic 5.

---

**Algorithm 5:** Bad dies stacking heuristic (BDS)

---

**Data:** An integer  $\phi$

Select the  $\phi$  positions that maximize the overall number of bad dies;

Solve optimally the instance reduced to the  $\phi$  selected positions;

Apply previously computed assignment on the initial instance;

---

### 10.3.2 Computational Results

This section is devoted to the results presentation of the experiments performed on previously introduced heuristics. We focus on the efficiency of such heuristics in function of the numbers of selected dies ( $\phi \in \{10, 50\}$ ) and the way these dies are selected (BDS or VDP).

Before introducing the results let us make few remarks.

- First of all, in the following we denote as BDS- $\phi$  (resp. VDP- $\phi$ ) the heuristic BDS (resp. VDP) that select  $\phi$  positions.
- We only consider instances satisfying  $p > \phi$  and  $m \in \{6, 9\}$  as instances with  $m = 3$  can be solved to the optimality with the Formulation (VI).
- In the following tables, we respect the following conventions. The given computation time is computed on instances that have been solved to optimality. It follows that when no time is given, every reduced instance of the considered set reached the computation duration limit of 1800 seconds measured in CPU time. Similarly, the value of the Gap stands for the average gap value returned by CPLEX for instances that are not solved to optimality. This value only gives information on the quality of the solution returned by CPLEX when solving reduced instances. The latter thus does not give informations on the quality of the solution returned by the heuristics. Such informations are given by the Ratio value which is equal to the cost of the heuristic solution over the cost of the solution returned by a random assignment.

#### Influence of the dies selection heuristic

The selection of the positions composing the reduced instance is obviously a factor having strong impact on the performances of considered heuristics. This section aims at determining how the latter influences both computation time and solution quality. The results, given in Tables 10.10 and 10.11, show that how positions are selected drives the heuristics very sensitive to the wafer yield and the number of dies in the initial instances.

Table 10.10: Influence of the wafer yield on the performances of BDS-10 and VDP-10 heuristics.

	BDS-10		VDP-10		Form (VIII)	
	Time	Ratio	Time	Ratio	Time	Ratio
6 25 100 50	3.254	0.988	1800	0.968	1800	0.950
6 50 100 50	74.249	0.986	1800	0.962	1800	0.974
9 25 100 50	72.792	0.990	1800	0.966	1800	0.973
9 50 100 50	771.284	0.986	1800	0.971	1800	0.995
6 25 100 70	852.477	0.971	1631.654	0.961	1800	0.942
6 50 100 70	1800	0.964	1800	0.956	1800	0.990
9 25 100 70	1800	0.972	1800	0.953	1800	0.955
9 50 100 70	1800	0.978	1800	0.952	1800	0.990
6 25 100 90	1800	0.918	0.265	0.994	1800	0.881
6 50 100 90	1800	0.895	0.325	0.995	1800	0.959
9 25 100 90	1800	0.916	0.874	0.982	1800	0.912
9 50 100 90	1800	0.928	0.922	0.989	1800	0.965
6 25 1000 50	0.214	1.000	1800	0.996	1800	0.999
6 50 1000 50	13.664	1.000	1800	0.997	1800	0.999
9 25 1000 50	0.533	1.000	1800	0.997	1800	0.999
9 50 1000 50	43.672	0.999	1800	0.997	1800	0.999
6 25 1000 70	42.786	0.998	1156.615	0.995	1800	0.993
6 50 1000 70	1800	0.996	1800	0.996	1800	0.994
9 25 1000 70	406.526	0.999	1800	0.996	1800	0.994
9 50 1000 70	1800	0.998	1800	0.995	1800	0.994
6 25 1000 90	1800	0.985	0.034	0.967	1800	0.967
6 50 1000 90	1800	0.995	0.132	0.985	1800	0.985
9 25 1000 90	1800	0.990	0.131	0.973	1800	0.961
9 50 1000 90	1800	0.992	0.438	0.975	1800	0.973

Table 10.11: Influence of the wafer yield on the performances of BDS-50 and VDP-50 heuristics.

	BDS-50		VDP-50		Form (VIII)	
	Time	Ratio	Time	Ratio	Time	Ratio
6 25 100 50	1800	0.949	1800	0.95	1800	0.950
6 50 100 50	1800	0.977	1800	0.967	1800	0.974
9 25 100 50	1800	0.965	1800	0.969	1800	0.973
9 50 100 50	1800	0.993	1800	0.982	1800	0.995
6 25 100 70	1800	0.915	1800	0.935	1800	0.942
6 50 100 70	1800	0.955	1800	0.939	1800	0.990
9 25 100 70	1800	0.936	1800	0.944	1800	0.955
9 50 100 70	1800	0.978	1800	0.968	1800	0.990
6 25 100 90	1800	0.834	1800	0.937	1800	0.881
6 50 100 90	1800	0.895	1800	0.936	1800	0.959
9 25 100 90	1800	0.877	1800	0.897	1800	0.912
9 50 100 90	1800	0.921	1800	0.921	1800	0.965
6 25 1000 50	1800	0.997	1800	0.995	1800	0.999
6 50 1000 50	1800	0.999	1800	0.996	1800	0.999
9 25 1000 50	1800	0.999	1800	0.996	1800	0.999
9 50 1000 50	1800	1.0	1800	0.997	1800	0.999
6 25 1000 70	1800	0.993	1800	0.993	1800	0.993
6 50 1000 70	1800	0.998	1800	0.993	1800	0.994
9 25 1000 70	1800	0.995	1800	0.994	1800	0.994
9 50 1000 70	1800	0.999	1800	0.994	1800	0.994
6 25 1000 90	1800	0.978	0.26	0.983	1800	0.967
6 50 1000 90	1800	0.985	0.924	0.997	1800	0.985
9 25 1000 90	1800	0.978	162.699	0.991	1800	0.961
9 50 1000 90	1800	0.994	659.668	0.994	1800	0.973

Indeed, we can see in Table 10.10, that VDP-10 has overall better solution quality except for instances with 100 initial dies per wafer and an individual wafer yield being equal to 90%. In such a configuration, the outer rim of the wafer is almost nonexistent due to the combination of a high value of  $Y_W$  and of a medium value of  $p$ . It follows that most of the failures are quite well spread over the inner rim while the wafer center, composed with almost only viable dies, covers a large area on the wafer.

This leads the BDS heuristic to select positions composed with a mix of viable dies and failures while the VDP heuristic selects positions coming from the wafer center area, and thus almost exclusively composed with viable dies. In the reduced instances of VDP-10 every assignment is almost optimal. This explains why the computation is very fast (less than 1 second measured in CPU time) and why the computed solution are quite bad. The more balanced BDS-10 instances lead, in turn, to more computation time (none of the reduced instance has been solved to optimality) but produce better solutions.

Note that the previous phenomenon does not necessary apply on instances with  $p = 1000$  since the increase of the number of dies leads to instances admitting a more consequent outer rim in which dies are selected by BDS-10, leading to positions with a larger number of failures. This explains why performances of BDS-10 degrade when  $p = 1000$ . However, the good performances of VDP on these instances are not explained by this kind of arguments. Such a phenomenon motivates a more advanced study of the structure of reduced instances.

Note also that a similar arguments explains:

- the bad solution quality of BDS-10, since reduced instances mainly consists in position with a high rate of faulty dies. Since these positions are very likely to be faulty positions in solutions of the initial instances, the BDS-10 algorithm appears to be inadequate.
- the good solutions quality of BDS-50 in instances with  $Y_W = 90$ ,
- the overall better quality of VDP-10 and VDP-50 of the other instances.

It follows that the efficiency of proposed heuristic is strongly related to the individual wafer yield in the reduced instances. Investigation on the impact of the latter on the performances appears to be appropriate. Furthermore, we conjecture that, using a positions selection algorithm ensuring a heterogeneity between viable and faulty dies in the reduced instance, can drastically improve the final yield.

### Impact of the number of dies in reduced instances

This section aims at determining the influence of the number of dies in reduced instances on the quality of the considered heuristics. Naturally, we expect that an increased size of the reduced instances leads to better solution as the heuristic have more informations on the initial instance. On a first hand adding informations into the reduced instances leads to a diminution of the impact of the mechanisms presented in previous section, improving thus efficiency of BDS heuristic. On the other hand we

Table 10.12: Influence of the number of dies in reduced instances on the performances of BDS-10 and BDS-50 heuristics.

	BDS-10		BDS-50		Form (VIII)	
	Time	Ratio	Time	Ratio	Time	Ratio
6 25 100 50	3.254	0.988	1800	0.949	1800	0.950
6 25 100 70	852.477	0.971	1800	0.915	1800	0.942
6 25 100 90	1800	0.918	1800	0.834	1800	0.881
6 25 1000 50	0.214	1.000	1800	0.997	1800	0.999
6 25 1000 70	42.786	0.998	1800	0.993	1800	0.993
6 25 1000 90	1800	0.985	1800	0.978	1800	0.967
6 50 100 50	74.249	0.986	1800	0.977	1800	0.974
6 50 100 70	1800	0.964	1800	0.955	1800	0.990
6 50 100 90	1800	0.895	1800	0.895	1800	0.959
6 50 1000 50	13.664	1.000	1800	0.999	1800	0.999
6 50 1000 70	1800	0.996	1800	0.998	1800	0.994
6 50 1000 90	1800	0.995	1800	0.985	1800	0.985
9 25 100 50	72.792	0.990	1800	0.965	1800	0.973
9 25 100 70	1800	0.972	1800	0.936	1800	0.955
9 25 100 90	1800	0.916	1800	0.877	1800	0.912
9 25 1000 50	0.533	1.000	1800	0.999	1800	0.999
9 25 1000 70	406.526	0.999	1800	0.995	1800	0.994
9 25 1000 90	1800	0.990	1800	0.978	1800	0.961
9 50 100 50	771.284	0.986	1800	0.993	1800	0.995
9 50 100 70	1800	0.978	1800	0.978	1800	0.990
9 50 100 90	1800	0.928	1800	0.921	1800	0.965
9 50 1000 50	43.672	0.999	1800	1.0	1800	0.999
9 50 1000 70	1800	0.998	1800	0.999	1800	0.994
9 50 1000 90	1800	0.992	1800	0.994	1800	0.973

Table 10.13: Influence of the number of dies in reduced instances on the performances of VDP-10 and VDP-50 heuristics.

	VDP-10		VDP-50		Form (VIII)	
	Time	Ratio	Time	Ratio	Time	Ratio
6 25 100 50	1800	0.968	1800	0.95	1800	0.950
6 25 100 70	1631.654	0.961	1800	0.935	1800	0.942
6 25 100 90	0.265	0.994	1800	0.937	1800	0.881
6 25 1000 50	1800	0.996	1800	0.995	1800	0.999
6 25 1000 70	1156.615	0.995	1800	0.993	1800	0.993
6 25 1000 90	0.034	0.967	0.26	0.983	1800	0.967
6 50 100 50	1800	0.962	1800	0.967	1800	0.974
6 50 100 70	1800	0.956	1800	0.939	1800	0.990
6 50 100 90	0.325	0.995	1800	0.936	1800	0.959
6 50 1000 50	1800	0.997	1800	0.996	1800	0.999
6 50 1000 70	1800	0.996	1800	0.993	1800	0.994
6 50 1000 90	0.132	0.985	0.924	0.997	1800	0.985
9 25 100 50	1800	0.966	1800	0.969	1800	0.973
9 25 100 70	1800	0.953	1800	0.944	1800	0.955
9 25 100 90	0.874	0.982	1800	0.897	1800	0.912
9 25 1000 50	1800	0.997	1800	0.996	1800	0.999
9 25 1000 70	1800	0.996	1800	0.994	1800	0.994
9 25 1000 90	0.131	0.973	162.699	0.991	1800	0.961
9 50 100 50	1800	0.971	1800	0.982	1800	0.995
9 50 100 70	1800	0.952	1800	0.968	1800	0.990
9 50 100 90	0.922	0.989	1800	0.921	1800	0.965
9 50 1000 50	1800	0.997	1800	0.997	1800	0.999
9 50 1000 70	1800	0.995	1800	0.994	1800	0.994
9 50 1000 90	0.438	0.975	659.668	0.994	1800	0.973

will see that this extra information may also causes loss of efficiency on certain sets of instances.

The results in Table 10.12 show that the BDS heuristic has overall better results with a larger number of dies in the reduced instances. These results strengthen the explanations we gave in the previous section regarding the overall poor performances of this heuristic. Indeed, a larger number of positions in the reduced instances leads to a selection of positions composed with more viable dies than the positions selected in BDS-10.

Results also show that, except few sets of instances, BDS-50 returns better solution than the Formulation (VIII).

The results in Table 10.13 show that, while increasing the number of dies in the reduced instance improve the quality of the solutions returned by BDS heuristic, such an increase can lead to loss of efficiency for VDP. Such a phenomenon occurs on instances with large number of dies ( $p = 1000$ ) and a high individual wafer yield ( $Y_W = 90\%$ ). It is again explained by the combination of favourable parameters and positions selection routine.

Remember that VDP heuristics selects among all the positions the ones that are composed with the minimum number of faulty dies. It follows that, for such instances, most of the selected positions are nearly perfect positions. Thus the few selected

positions, composed with a more substantial number of faulty dies, have a strong impact on the optimal solution of the reduced instances. Optimal assignments in VDP-50 are indeed very different from those in VDP-10. Well, recall that in the wafer generation model of Di Natale et al. [Nat+13], wafers that have similar failures maps in the wafer center area have similar overall failures maps. Thus optimal assignment in reduced instances of VDP-10 likely led to assignment of similar wafers in the initial instances. Optimal assignments in VDP-50 reduced instances are driven by more faulty positions and thus positions placed further from the wafer center. The probability that these assignments lead to assignment of similar wafers is thus reduced.

On the other hand, the increase of dies in reduced instances leads to a noticeable improvement of the solution quality on instances with  $p = 100$  dies and having a wafer yield equal to 90%.

For both of the heuristics, with an increased number of dies in reduced instances, none of the instance has been solved to optimality. However let us remark that, solving the latter to optimality does not necessarily implies an improvement of the solution computed by the heuristics.

## 10.4 Sets aggregation technique

In this section we investigate another aspect of the problem. We try to determine the potential profit we can earn by considering instances with a larger number of sets. This question is motivated by optimizing industrial processes.

### 10.4.1 Modus Operandi

In order to measure the maximum impact of the sets aggregation technique, we want to ensure an optimal resolution of considered instances. This motivates the use of Formulation (VI) restricted to instances with  $m = 3$ . In the following we will focus on the aggregation of four instances. In other words, we construct a instance with  $n = 100$  from four instances with  $n = 25$ .

For each combination of parameter  $p \in \{10, 100, 1000\}$  and  $Y_W \in \{50, 70, 90\}$ , we generate 80 instances with  $m = 3$  and  $n = 25$ . We can then compute 20 instances with  $n = 100$ . Each of these instances are optimally solved with Formulation (VI) and we compare the evolution of the objective values and the computation time between the four instances with  $n = 25$  and the corresponding instance with  $n = 100$ .

The tests are performed on the same machine as all other performed tests, *i.e.* a workstation equipped with Intel Xeon processors with twenty cores running at 2.86Ghz with 68Gb of dynamic memory. The ILP solver used to perform the computation is CPLEX 12.6.

### 10.4.2 Computational results

The results of the experiment are given in Table 10.14. They show a substantial increase of the objective value with up to 578.9 additional viable dies for instances

Table 10.14: Impact of the set aggregation technique on the time computation and objective function

$p$	$Y_W$	$n = 100$ instances		$n = 25$ instances		Difference		
		Time	Obj. Value	Time	Obj. Value	$\Delta$ Time	$\Delta$ Obj.	$\Delta$ Fails
1000	50	35029.62	82006.9	10.26	82585.8	-35019.36	578.9	0.58%
	70	21420.80	59410.3	10.39	59988.3	-21410.41	578.1	0.58%
	90	14106.18	23081.1	14.25	23456.4	-14091.93	375.3	0.38%
100	50	25606.76	7599.2	15.54	7800.5	-25591.21	201.3	2.01%
	70	11620.44	5345.1	11.44	5547.5	-11608.00	202.4	2.02%
	90	8433.96	1935.3	7.20	2066.0	-8426.76	130.7	1.31%
10	50	2864.43	584.4	6.19	641.6	-2858.24	57.2	5.72%
	70	1091.83	366.9	5.77	419	-1086.06	52.1	5.21%
	90	295.29	128.6	5.85	148.8	-289.43	20.2	2.02%

with  $p = 1000$  dies and  $Y_W = 50$ . Even though this improvement only represents 0.58% of the overall number of dies, it corresponds to an increase of 3.21% of the number of viable 3D-chips.

Furthermore and as expected, we remark a larger improvement for instances with low and average wafer yield ( $Y_W \in \{50, 70\}$ ). Indeed, for the instances having a good wafer average, the few failures are quite well spread over the later. It follows that given a wafer, finding very similar in the concatenated instances is quite unlikely, contrary to instances with bad wafer yield.

We can also see that, the more the number of dies increases, the more the percentage of “saved” dies decreases. Actually, the phenomenon is due to a diminution of clustering as the number of dies increases. Indeed, as explained in Section 10.2.2, the general distribution on the wafer area is quite similar no matter the number of dies, however an increase number of dies can be seen as an increase of details on the wafer, leading to failures located at close but distinct locations.

To finish, the improvement of the objective value has a cost in terms of computation time. For example, the time required to solve optimally the 3\_100\_1000\_50 instances is roughly 3400 times the time required to solve the four corresponding 3\_25\_1000\_50. Such an increase of computation prevents the use of sets aggregation technique as it is. However, it can still be used in the conception of prototypes for which individual wafer yield is very low. Furthermore, the advantages of this technique motivates, to our opinion, efforts to develop resolution techniques with limited dependence in  $n$ .

## 10.5 Conclusion

In this chapter, we presented two ILP formulations coming from the literature on Wafer-to-Wafer integration problem. After having slightly improved one of those formulations, we proved that none of them are suitable for large scale general instances. However, the experiments we provided help us to better understand the complexity of the problem, and determine the limits of each of the considered ILP formulations.



Furthermore, the results we obtained confirm that the number of dies has a strong impact on the difficulty of the problem. We conjecture that reducing the number of dies in the considered instance can lead to interesting and efficient heuristics. This work begs a lot of questions that can be answered thanks to appropriately designed experiments.

In a first time, further experiments to confirm the impact of the *Feasibility Pump Heuristic* on the performance of Formulation (VIII) should be performed. Two experiments could be performed, the first one using default CPLEX parameters but limiting the computation time to 1800 seconds and the second using the aggressive setting with a computation time limit set to 54000 seconds.

We conjecture the impact of the failures clustering on the Formulation (VIII) performances. Running the latter on instances with uniformly distributed failures could validate this conjecture.

In order to be able to compare the performances of our ILP based heuristics to the literature results, it could be interesting to perform the experiment we presented on pseudo random generated wafer using the negative binomial distribution model.

Based on behaviour of presented heuristics, considering other dies selection strategies to design reduced instances can be an interesting research trail. We show that good performances of the heuristics are explained by selected positions presenting a mix of faulty and viable dies. Thus selecting dies that respect some yield conditions could lead to an improvement of the solutions quality of the ILP based heuristics.

A last research trail lies in the use of the Formulation (IV) to solve optimally reduced instances. This formulation has been used in Section 7.2.2 to prove the problem membership in **FPT**. Therefore, we can hope the latter to be more efficient than Formulation (VIII) to solve instances with  $p = 10$ .

# 11

## Matching based heuristics

---

---

### Contents

---

<b>11.1 Set selection strategies</b> . . . . .	<b>170</b>
11.1.1 Iterative Matchings . . . . .	170
11.1.2 Lot Partitioning . . . . .	170
11.1.3 Balanced Binary Tree Strategy . . . . .	172
<b>11.2 Computational results</b> . . . . .	<b>173</b>
<b>11.3 Conclusion</b> . . . . .	<b>175</b>

---

The previous chapter focused on the resolution of the problem using ILP formulations. However a recurrent technique in the literature is the use of matchings to perform the assignment [Tao+10; Ver+10; RSS09]. In each of these references, the authors consider the *sequential* heuristic that we presented in Chapter 5. Recall that this algorithm, whose pseudo code is given by Algorithm 6, has been proved to be a  $3/2$ -approximation algorithm for the  $\min \sum 0$  problem for fixed  $m = 3$ . Furthermore a simple sort of the set according to their weight improves the ratio guarantee to  $4/3$ , for fixed  $m = 3$ .

Nevertheless, this theoretical improvement has never, to our knowledge, been taken into consideration in the experimentation. We provide in the following a comparison between performances of the *sequential* heuristic and its sorted version also introduced in Chapter 5 as the *heaviest first* heuristic.

We also propose and test different set selection strategies. Some of these strategies lead to approximation algorithm.

## 11.1 Set selection strategies

In this section we quickly present the considered set selection strategies as well as approximation algorithms based on these strategies. In a first time we recall the iterative matching strategy on which both *sequential* and *heaviest first* heuristics are based. In a second time we present the lot partitioning strategy that, given an integer  $k$ , constructs a  $k$ -ary tree in which each leaf is a set and each internal node represents a wafer-to-wafer integration problems with  $k$  sets. In a last section we present the balanced binary tree strategy. The latter is quite similar to the lot partitioning approach but the constructed tree is binary and balanced.

### 11.1.1 Iterative Matchings

Let us first quickly recall the principle of this strategy. We consider an instance of  $\min \sum 0^1$  with  $m$  sets. In a first time the optimal assignment between vectors of  $V^1$  and vectors of  $V^2$  is computed. At each iteration the algorithm solves optimally the assignment between the resulting matching of the previous iteration and the next unprocessed set.

The pseudo-code of this strategy is given by Algorithm 6. Dokka et al. prove in [DCS14] that this algorithm is a  $m/2$ -approximation algorithm for  $\min \sum 0$ . They also prove that the Algorithm 7 is a  $\frac{m+1}{2} - \frac{1}{4} \ln(m-1)$ -approximation algorithm.

---

#### Algorithm 6: Sequential heuristic

---

```

 $M \leftarrow$  resulting optimal assignment between  $V^1$  and  $V^2$ ;
for  $i = 3, \dots, m$  do
     $M \leftarrow$  resulting optimal assignment between  $M$  and  $V^i$ ;
return  $M$ ;

```

---



---

#### Algorithm 7: Heaviest first heuristic

---

```

Sort the sets according to decreasing costs;
Apply the Sequential algorithm on the sorted sets;

```

---

### 11.1.2 Lot Partitioning

In this section we present the lot partitioning algorithm. Given a  $\min \sum 0$  instance, with  $m$  sets, an integer  $k$ , and a routine that solves  $\min \sum 0$  instance with a number of sets less or equal to  $k$ , the algorithm proceeds as follows. The input sets are partitioned into  $\lfloor m/k \rfloor$  lots containing  $k$  sets complemented by a set that contains less than  $k$  sets, if necessary. These lots define smaller  $\min \sum 0$  instances having a reduced number of sets. It then solved each reduced instance thanks to the provided routine. For each

---

<sup>1</sup>This strategy is also suitable for  $\max \sum 1$  instances by modifying the way the cost are computed.

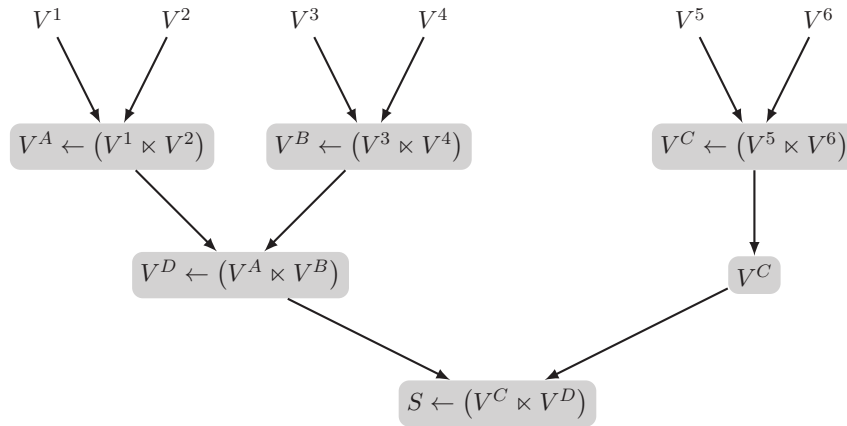


Figure 11.1: Illustration of the running scheme of Algorithm 8 on an instance with  $m = 6$ .

lot, a representative set is computed based on solution returned by the routine. The algorithm iterates these steps until a solution to the initial instance is computed.

The Figure 11.1 depicts the resolution scheme of a  $\min \sum 0$  instance with  $m = 6$  and  $k = 2$ . The initial sets of wafers are first partitioned into three reduced  $\min \sum 0$  of two sets each. A representative set  $V^A$  is computed based on resolution of the initial instance restricted to  $(V^1, V^2)$ . In the same way, the representative set  $V^B$  (resp.  $V^C$ ) is computed based on resolution of the initial instance restricted to  $(V^3, V^4)$  (resp.  $V^5, V^6$ ). These three representative sets define a new  $\min \sum 0$  instances that will be solved using the same steps. These operations are applied until a solution  $S$  is computed.

In the following we consider the Algorithm 8 being the particular lot partitioning algorithm when  $k = 2$ .

---

**Algorithm 8:** Lot partitioning heuristic

---

```

while the number of remaining sets is greater than 1 do
  if  $m$  is even then
    Partition the sets into  $m/2$  lots of size 2;
  else
    Partition the sets into  $\lfloor m/2 \rfloor - 1$  lots of size 2 completed with a lot of size 3;
    Compute a representative of the lot of size 3 using sequential heuristic;
  for each lot of size 2 do
    Compute the optimal matching between the both sets;
    Compute the representative sets based on optimal matching;
  Replace lots of sets by their representative set;
return the last representative set;
  
```

---

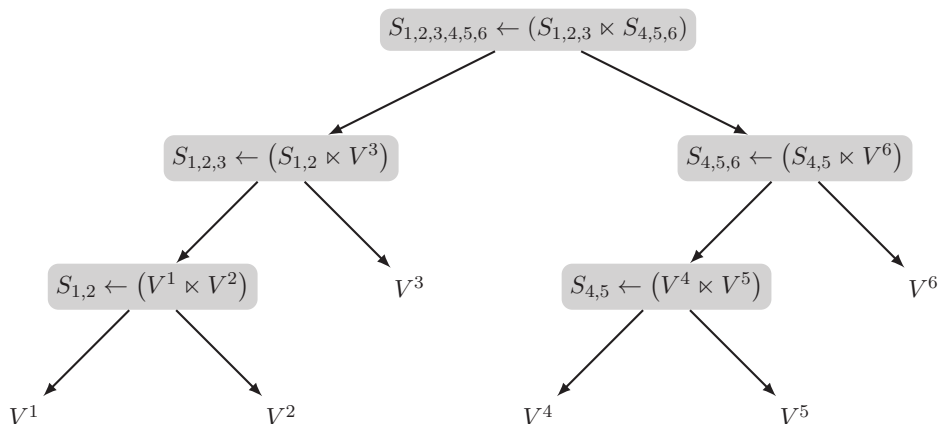


Figure 11.2: Illustration of the running scheme of Algorithm 9 on an instance with  $m = 6$ .

**Theorem 11.1.** *The Algorithm 8 is an  $m/2$  approximation algorithm.*

*Proof.* Let us consider two cases.

**$m$  is even.** Remark that the first iteration of the algorithm generates  $m/2$  representative sets  $V^{a_1}, V^{a_2}, \dots, V^{a_{m/2}}$  whose cost satisfies  $c(V^{a_i}) \leq c(Opt)$ . In the worst case, no faults are mapped when superimposing wafers of these sets. It follows that the cost  $c(S)$  of the computed solution  $S$  is given by  $c(S) = \sum_{i=1}^{m/2} c(V^{a_i}) \leq \frac{m}{2} c(Opt)$ .

**$m$  is odd.** In this case the first iteration of the algorithm generate  $\lfloor m/2 \rfloor - 1$  representative sets whose cost is less than the cost of the optimal solution completed by a representative set whose cost is bounded by  $3/2c(Opt)$ . It follows that the computed solution  $S$  verifies  $c(S) \leq m/2c(Opt)$ .

□

### 11.1.3 Balanced Binary Tree Strategy

---

**Algorithm 9:** Balanced binary tree heuristic

---

RecursiveSplit(1,  $m$ );

---

Given a min  $\sum 0$  instance  $I$  with  $m$  sets, this recursive algorithm solves  $I$  by solving two smaller instances  $I_1$  and  $I_2$  with respectively  $\lceil m/2 \rceil$  and  $\lfloor m/2 \rfloor$  sets and by computing the optimal matching between the two partial solutions.

An example of execution scheme of Algorithm 9 is given in Figure 11.2.

---

**Function** RecursiveSplit( $s, e$ )
 

---

```

if  $s = e$  then
  return  $V^s$ ;
if  $e - s = 1$  then
  return resulting optimal assignment between  $V^s$  and  $V^e$ ;
 $V^{left} \leftarrow$  RecursiveSplit( $s, \lceil (e + s)/2 \rceil$ );
 $V^{right} \leftarrow$  RecursiveSplit( $\lceil (e + s)/2 \rceil + 1, e$ );
return ( $V^{left} \times V^{right}$ );

```

---

## 11.2 Computational results

The previous section led us to introduce several matching based heuristics. This section is devoted to their comparison from a computational point of view. The experiments are performed in the same environment as the experiments performed on ILP formulations. Let us recall the main points.

- The experiments are performed on a workstation equipped with Intel Xeon processors providing twenty cores running at 2.8GHz and with 68Gb of dynamic memory. Note that the number of cores is not relevant for this experiment as none of the algorithms implementations take advantage of the parallelization.
- We use the Hungarian method to compute the optimal matching. All the algorithms are implemented in C and compiled with gcc 4.8.4 with the -O2 flag.
- Ten instances have been generated using the Di Natale et al. [Nat+13] wafer generation model, for each combination of the following parameters.
  1.  $m \in \{3, 6, 9\}$ ,
  2.  $n \in \{25, 50\}$ ,
  3.  $p \in \{10, 100, 1000\}$ ,
  4.  $Y_W \in \{50, 70, 90\}$ .
- Every instance has been solved using Algorithms 6, 7, 8 and 9 completed slightly modified Algorithms 8 and 9 in which set have been first sorted by decreasing costs.
- For each instance, we kept the value of the best solution among the matching based algorithm. These results are included in the result Table as Best.
- Results are compared to ILP based heuristics 5 and 4 both with a remaining number a dies equal to 50.

A complete overview of the results is given in Table 11.1.

Table 11.1: Results

	6	7	8	Sort + 8	9	Sort + 9	Best	4	5
3 25 10 50	0.760	0.760	0.760	0.760	0.760	0.760	0.759	-	-
3 25 10 70	0.675	0.673	0.675	0.673	0.675	0.673	0.671	-	-
3 25 10 90	0.613	0.620	0.613	0.620	0.613	0.620	0.613	-	-
3 25 100 50	0.926	0.926	0.926	0.926	0.926	0.926	0.925	0.941	0.953
3 25 100 70	0.899	0.898	0.899	0.898	0.899	0.898	0.897	0.933	0.924
3 25 100 90	0.845	0.844	0.845	0.844	0.845	0.844	0.844	0.956	0.865
3 25 1000 50	0.977	0.977	0.977	0.977	0.977	0.977	0.976	0.994	0.998
3 25 1000 70	0.969	0.969	0.969	0.969	0.969	0.969	0.969	0.996	0.994
3 25 1000 90	0.949	0.949	0.949	0.949	0.949	0.949	0.949	0.985	0.977
3 50 10 50	0.737	0.736	0.737	0.736	0.737	0.736	0.734	-	-
3 50 10 70	0.641	0.643	0.641	0.643	0.641	0.643	0.641	-	-
3 50 10 90	0.566	0.565	0.566	0.565	0.566	0.565	0.564	-	-
3 50 100 50	0.918	0.918	0.918	0.918	0.918	0.918	0.917	0.939	0.949
3 50 100 70	0.883	0.883	0.883	0.883	0.883	0.883	0.882	0.928	0.915
3 50 100 90	0.819	0.820	0.819	0.820	0.819	0.820	0.819	0.960	0.834
3 50 1000 50	0.974	0.974	0.974	0.974	0.974	0.974	0.974	0.994	0.997
3 50 1000 70	0.964	0.965	0.964	0.965	0.964	0.965	0.964	0.994	0.993
3 50 1000 90	0.942	0.942	0.942	0.942	0.942	0.942	0.941	0.992	0.978
6 25 10 50	0.739	0.739	0.739	0.739	0.756	0.758	0.736	-	-
6 25 10 70	0.587	0.586	0.587	0.586	0.604	0.603	0.581	-	-
6 25 10 90	0.509	0.512	0.509	0.512	0.527	0.528	0.505	-	-
6 25 100 50	0.936	0.935	0.936	0.935	0.946	0.945	0.934	0.950	0.965
6 25 100 70	0.879	0.880	0.879	0.880	0.893	0.890	0.877	0.935	0.936
6 25 100 90	0.804	0.805	0.804	0.805	0.820	0.817	0.801	0.937	0.877
6 25 1000 50	0.982	0.983	0.982	0.983	0.985	0.985	0.982	0.995	0.999
6 25 1000 70	0.963	0.962	0.963	0.962	0.967	0.967	0.962	0.993	0.995
6 25 1000 90	0.926	0.926	0.926	0.926	0.930	0.931	0.925	0.983	0.978
6 50 10 50	0.697	0.698	0.697	0.698	0.714	0.713	0.695	-	-
6 50 10 70	0.549	0.551	0.549	0.551	0.565	0.568	0.548	-	-
6 50 10 90	0.436	0.433	0.436	0.433	0.445	0.439	0.430	-	-
6 50 100 50	0.921	0.922	0.921	0.922	0.932	0.932	0.920	0.967	0.969
6 50 100 70	0.862	0.860	0.862	0.860	0.876	0.874	0.860	0.939	0.956
6 50 100 90	0.766	0.767	0.766	0.767	0.783	0.783	0.765	0.936	0.880
6 50 1000 50	0.979	0.979	0.979	0.979	0.982	0.982	0.978	0.996	0.999
6 50 1000 70	0.957	0.957	0.957	0.957	0.962	0.962	0.957	0.993	0.995
6 50 1000 90	0.927	0.926	0.927	0.926	0.933	0.933	0.926	0.997	0.992
9 25 10 50	0.740	0.746	0.740	0.746	0.779	0.777	0.738	-	-
9 25 10 70	0.583	0.587	0.583	0.587	0.627	0.627	0.582	-	-
9 25 10 90	0.445	0.449	0.445	0.449	0.474	0.475	0.444	-	-
9 25 100 50	0.950	0.950	0.950	0.950	0.964	0.964	0.949	0.969	0.977
9 25 100 70	0.881	0.882	0.881	0.882	0.906	0.905	0.880	0.944	0.955
9 25 100 90	0.782	0.780	0.782	0.780	0.815	0.812	0.779	0.897	0.895
9 25 1000 50	0.989	0.989	0.989	0.989	0.992	0.992	0.988	0.996	0.999
9 25 1000 70	0.967	0.967	0.967	0.967	0.974	0.974	0.967	0.994	0.998
9 25 1000 90	0.921	0.920	0.921	0.920	0.932	0.933	0.920	0.991	0.985
9 50 10 50	0.702	0.702	0.702	0.702	0.734	0.735	0.700	-	-
9 50 10 70	0.525	0.525	0.525	0.525	0.560	0.560	0.522	-	-
9 50 10 90	0.383	0.382	0.383	0.382	0.402	0.404	0.379	-	-
9 50 100 50	0.934	0.935	0.934	0.935	0.953	0.953	0.934	0.982	0.993
9 50 100 70	0.864	0.865	0.864	0.865	0.891	0.890	0.864	0.968	0.978
9 50 100 90	0.747	0.749	0.747	0.749	0.785	0.784	0.746	0.921	0.921
9 50 1000 50	0.985	0.985	0.985	0.985	0.990	0.990	0.985	0.997	1.000
9 50 1000 70	0.960	0.960	0.960	0.960	0.969	0.969	0.960	0.994	0.999
9 50 1000 90	0.915	0.915	0.915	0.915	0.930	0.929	0.915	0.994	0.994

To lighten the table results, the computation time of considered heuristics is not provided. Except for the Best heuristic that runs in few milliseconds, no matter the considered heuristic, the computation time is less than one millisecond.

The results show that, even though sorting the sets improves the guarantee on the ratio, the Algorithm 7 does not provide better computational results compared to 6. The performances of these two algorithms are comparable no matter the considered instances. This can be explained by the fact that all the sets of a same instance have the same yield. The sort operation has thus almost non influence of the algorithm.

We also observe the same tendencies as Reda et al. [RSS09] and Verbree et al. [Ver+10], namely high performances of these heuristics when  $m$  takes high value and/or  $p$  and  $Y_W$  takes low values.

The two versions of Algorithm 8 provide comparable results, following the same tendencies and providing comparable solutions quality. However the both versions of Algorithm 9 may lead to significantly worst solutions when  $m$  increases. This can be explained by the order the matching are performed by the algorithm. Indeed, in the balanced binary tree strategy the failures map of the computed solution for the left subtree can significantly differs from the one of the computed solution in the right subtree. This leads to a loss of performances in terms of solutions quality.

An interesting results concerns the Best heuristic that applies successively all of the six algorithms and selects the best solutions. The latter provides improvement for instances with a high number of sets and a low or average number of dies per wafer.

### 11.3 Conclusion

These experiments on matching based strategies close this practical part. We proposed in this part, several heuristics to tackle the problem. However none of them significantly improves the existing techniques.

In this part, we laid the foundations for different way of handling the problem: reducing the number of dies. In Section 4.2 we presented some reduction rules allowing us to remove unnecessary dies. We expect this approach to significantly improve the performance of the solving strategies on instances with either a very high  $Y_W$  or a very low one.

We also provided experimentation results highlighting interesting internal mechanisms of the problem. Combined with the results we can expect from the proposed experiments, we should be able to provide new interesting solving strategies.





# 12

## Perspective and conclusion

---

---

This chapter marks the end of this manuscript, but also the end of three exciting and stimulating years of research. Even though lot of works is still ongoing, we tried to present interesting insights into the problems. On a theoretical aspect, despite the lack of mind blowing techniques or brilliant results, we provided a complete overview of the problem complexity both from an approximability and fixed parameter complexity point of view. Indeed, the few questions that we left open confirm emphasize the latter.

We also tried to be as exhaustive as can be even on the practical aspects of the problem. On a positive side, we provided a new and interesting way to handle the problem complexity. The die reduction principle also tends to validate the tendencies expected from theoretical analysis. However, on a negative side our practical approach provided no performance improvement compared to already existing methods. Furthermore, we kept on going on already explored paths: ILP formulations and Matching Based Heuristics. However, we now reconsider it try to tackle the problem by using different techniques such as Constraints Programming, Branch and Bound algorithms or parallel algorithms.

In light of this work, we can imagine a great number of research trails. From a theoretical point of view, we can consider new problems strongly related to wafer-to-wafer integration problems. One can for instance cite  $\max \alpha$ -stack or  $\max \alpha$ -column. In both of these problems, we are given an integer  $\alpha$  and a BMVA input. The objective of the first one is to maximize the number  $m$ -tuple of profit greater than  $\alpha$ . Intuitively we try to maximize the number of wafer stacks of good quality. In the second wafer, the objective is, as for  $\max \sum 1$ , to maximize the overall number of good dies in the created stacks, however, a die at position  $l$  in the stack is considered to be viable if at least  $\alpha$  wafers composing the stack have viable die at position  $l$ .

We can remark in our work that most of the inapproximability results from  $\max \sum 1$

are derived from inapproximability results for  $\max \max 1$ . The expression power provided by  $\max \sum 1$  has not been properly exploited in the presented results. We think that this expressivity could be exploited to provide better negative results for the latter. Similarly, except for very constrained instances, the presented work do not take advantage of the more simple structure of  $\max \max 1$ . An interesting question to answer concerns the existence of an  $\mathcal{O}(m)$ -approximation algorithm for  $\max \max 1$ .

To conclude, this work raises lot of interesting questions that only ask to be answered.

## Bibliography

---

- [AK97] Paola Alimonti and Viggo Kann. “Hardness of Approximating Problems on Cubic Graphs”. In: *Algorithms and Complexity, Third Italian Conference, CIAC '97, Rome, Italy, March 12-14, 1997, Proceedings*. 1997, pp. 288–298.
- [Aus+99] Giorgio Ausiello et al. *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer Science & Business Media, 1999.
- [BÇ99] Rainer E. Burkard and Eranda Çela. *Linear assignment problems and extensions*. Springer, 1999.
- [BCS94] Hans-Jürgen Bandelt, Yves Crama, and Frits C. R. Spieksma. “Approximation Algorithms for Multi-Dimensional Assignment Problems with Decomposable Costs”. In: *Discrete Applied Mathematics* 49.1-3 (1994), pp. 25–50.
- [BDG16] Marin Bougeret, Guillaume Duval, and Rodolphe Giroudeau. “Approximability and Exact Resolution of the Multidimensional Binary vector Assignment Problem”. In: *Proceedings of the 4th International Symposium on Combinatorial Optimization, ISCO 2016, Vietri-Sul-Mare, Italy, May 17-19, 2016*. 2016, to be published.
- [BDM09] Rainer E. Burkard, Mauro Dell’Amico, and Silvano Martello. *Assignment Problems*. SIAM, 2009.
- [BJK14] Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. “Kernelization Lower Bounds by Cross-Composition”. In: *SIAM J. Discrete Math.* 28.1 (2014), pp. 277–305.
- [BK10] Nikhil Bansal and Subhash Khot. “Inapproximability of Hypergraph Vertex Cover and Applications to Scheduling Problems”. In: *Proceedings of the 36th International Colloquium on Automata, Languages and Programming, ICALP 2010, Bordeaux, France, July 6-10, 2010, Part I*. 2010, pp. 250–261.
- [BM11] Adrian Bondy and Uppaluri S.R. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer London, 2011.
- [Bou+15] Marin Bougeret et al. “Multidimensional Binary Vector Assignment Problem: Standard, Structural and Above Guarantee Parameterizations”. In: *Proceedings of the 20th International Symposium on Fundamentals of Computation Theory, FCT 2015, Gdańsk, Poland, August 17-19, 2015*. 2015, pp. 189–201.

- [Bou+16] Marin Bougeret et al. “On the Complexity of Wafer-to-Wafer Integration”. In: *Discrete Optimization* (2016), to be published.
- [BRW96] Rainer E. Burkard, Rüdiger Rudolf, and Gerhard J. Woeginger. “Three-dimensional Axial Assignment Problems with Decomposable Cost Coefficients”. In: *Discrete Applied Mathematics* 65.1-3 (1996), pp. 123–139.
- [CKX10] Jianer Chen, Iyad A. Kanj, and Ge Xia. “Improved upper bounds for vertex cover”. In: *Theor. Comput. Sci.* 411.40-42 (2010), pp. 3736–3756.
- [Coo71] Stephen A. Cook. “The Complexity of Theorem-Proving Procedures”. In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*. 1971, pp. 151–158.
- [Cre+99] Pierluigi Crescenzi et al. “Structure in Approximation Classes”. In: *SIAM J. Comput.* 28.5 (1999), pp. 1759–1782.
- [Cre97] Pierluigi Crescenzi. “A Short Guide to Approximation Preserving Reductions”. In: *Proceedings of the Twelfth Annual IEEE Conference on Computational Complexity, Ulm, Germany, June 24-27, 1997*. IEEE Computer Society, 1997, pp. 262–273.
- [CS92] Yves Crama and Frits C.R. Spieksma. “Approximation algorithms for three-dimensional assignment problems with triangle inequalities”. In: *European Journal of Operational Research* 60.3 (1992), pp. 273–279.
- [CT97] Marco Cesati and Luca Trevisan. “On the Efficiency of Polynomial Time Approximation Schemes”. In: *Inf. Process. Lett.* 64.4 (1997), pp. 165–171.
- [Cyg+13] Marek Cygan et al. “On multiway cut parameterized above lower bounds”. In: *TOCT* 5.1 (2013), p. 3.
- [Cyg+15] Marek Cygan et al. *Parameterized Algorithms*. Springer, 2015.
- [DCS14] Trivikram Dokka, Yves Crama, and Frits C. R. Spieksma. “Multi-dimensional vector assignment problems”. In: *Discrete Optimization* 14 (2014), pp. 111–125.
- [DF13] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
- [DF95] Rodney G. Downey and Michael R. Fellows. “Fixed-Parameter Tractability and Completeness II: On Completeness for  $W[1]$ ”. In: *Theor. Comput. Sci.* 141.1&2 (1995), pp. 109–131.
- [DF99] Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999.
- [Die12] Reinhard Diestel. *Graph Theory, 4th Edition*. Vol. 173. Graduate texts in mathematics. Springer, 2012.
- [Din+05] Irit Dinur et al. “A New Multilayered PCP and the Hardness of Hypergraph Vertex Cover”. In: *SIAM J. Comput.* 34.5 (2005), pp. 1129–1146.

- [Dok+12] Trivikram Dokka et al. “Approximation Algorithms for the Wafer to Wafer Integration Problem”. In: *Approximation and Online Algorithms - 10th International Workshop, WAOA 2012, Ljubljana, Slovenia, September 13-14, 2012, Revised Selected Papers*. 2012, pp. 286–297.
- [Dru15] Andrew Drucker. “New Limits to Classical and Quantum Instance Compression”. In: *SIAM J. Comput.* 44.5 (2015), pp. 1443–1479.
- [Duv+15] Guillaume Duvillié et al. “On the Complexity of Wafer-to-Wafer Integration”. In: *Algorithms and Complexity - 9th International Conference, CIAC 2015, Paris, France, May 20-22, 2015. Proceedings*. 2015, pp. 208–220.
- [Esc05] Bruno Escoffier. “Approximation polynomiale de problèmes d’optimisation: Aspects structurels et opérationnels”. PhD thesis. Ph. D. Thesis, Laboratoire LAM-SADE, Paris Dauphine, 2005.
- [FG06] Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006.
- [Gir+12] Rodolphe Giroudeau et al. “Circuits intégrés en 3D”. In: *Roadef*. 2012, p. 17.
- [GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.
- [Gut+13] Gregory Gutin et al. “Corrigendum. The Linear Arrangement Problem Parameterized Above Guaranteed Value”. In: *Theory Comput. Syst.* 53.4 (2013), pp. 690–691.
- [GY12] Gregory Gutin and Anders Yeo. “Constraint Satisfaction Problems Parameterized above or below Tight Bounds: A Survey”. In: *The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday*. 2012, pp. 257–286.
- [HSS03] Elad Hazan, Shmuel Safra, and Oded Schwartz. “On the Complexity of Approximating k-Dimensional Matching”. In: *Approximation, Randomization, and Combinatorial Optimization: Algorithms and Techniques, 6th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2003 and 7th International Workshop on Randomization and Approximation Techniques in Computer Science, RANDOM 2003, Princeton, NJ, USA, August 24-26, 2003, Proceedings*. 2003, pp. 83–97.
- [IP01] Russell Impagliazzo and Ramamohan Paturi. “On the Complexity of k-SAT”. In: *J. Comput. Syst. Sci.* 62.2 (2001), pp. 367–375.
- [IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. “Which Problems Have Strongly Exponential Complexity?” In: *J. Comput. Syst. Sci.* 63.4 (2001), pp. 512–530.

- [Kan83] Ravi Kannan. “Improved Algorithms for Integer Programming and Related Lattice Problems”. In: *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*. Ed. by David S. Johnson et al. ACM, 1983, pp. 193–206.
- [Kan91] Viggo Kann. “Maximum Bounded 3-Dimensional Matching is MAX SNP-Complete”. In: *Inf. Process. Lett.* 37.1 (1991), pp. 27–35.
- [Kar72] Richard M. Karp. “Reducibility Among Combinatorial Problems”. In: *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York*. 1972, pp. 85–103.
- [Kho02] Subhash Khot. “On the Power of Unique 2-Prover 1-Round Games”. In: *Proceedings of the 17th Annual IEEE Conference on Computational Complexity, Montréal, Québec, Canada, May 21-24, 2002*. IEEE Computer Society, 2002, p. 25.
- [Kum+05] Rakesh Kumar et al. “Heterogeneous Chip Multiprocessors”. In: *IEEE Computer* 38.11 (2005), pp. 32–38.
- [Lee90] Jan van Leeuwen, ed. *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*. Elsevier and MIT Press, 1990.
- [Lev73] Leonid A. Levin. “Universal sequential search problems”. In: *Problemy Peredachi Informatsii* 9.3 (1973), pp. 115–116.
- [LMS11] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. “Slightly Superexponential Parameterized Problems”. In: *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*. 2011, pp. 760–776.
- [Lok+14] Daniel Lokshtanov et al. “Faster Parameterized Algorithms Using Linear Programming”. In: *ACM Trans. Algorithms* 11.2 (2014), 15:1–15:31.
- [MP89] Fred J. Meyer and Dhiraj K. Pradhan. “Modeling Defect Spatial Distribution”. In: *IEEE Trans. Computers* 38.4 (1989), pp. 538–546.
- [MR99] Meena Mahajan and Venkatesh Raman. “Parameterizing above Guaranteed Values: MaxSat and MaxCut”. In: *J. Algorithms* 31.2 (1999), pp. 335–354.
- [MRS09] Meena Mahajan, Venkatesh Raman, and Somnath Sikdar. “Parameterizing above or below guaranteed values”. In: *J. Comput. Syst. Sci.* 75.2 (2009), pp. 137–153.
- [Nat+13] Giorgio Di Natale et al. “Exact wafer matching process wafer to wafer integration”. In: *Whorshop 3D integration Applications*. 2013, N–A.
- [Pap94] Christos H. Papadimitriou. *Computational complexity*. Addison-Wesley, 1994.
- [Pas04] Vangelis Th. Paschos. *Complexité et approximation polynomiale*. Hermès Science, 2004.

- [Pie68] William P. Pierskalla. “Letter to the editor—the multidimensional assignment problem”. In: *Operations Research* 16.2 (1968), pp. 422–431.
- [PY91] Christos H. Papadimitriou and Mihalis Yannakakis. “Optimization, Approximation, and Complexity Classes”. In: *Journal of Computer and System Science* 43.3 (1991), pp. 425–440.
- [RSS09] Sherief Reda, Gregory Smith, and Larry Smith. “Maximizing the Functional Yield of Wafer-to-Wafer 3-D Integration”. In: *IEEE Trans. VLSI Syst.* 17.9 (2009), pp. 1357–1362.
- [SAS83] Charles H. Stapper, Frederick M. Armstrong, and Kiyotaka Saji. “Integrated circuit yield statistics”. In: *Proceedings of the IEEE* 71.4 (1983), pp. 453–470.
- [Smi+07] Greg Smith et al. “Yield considerations in the choice of 3D technology”. In: *Semiconductor Manufacturing, 2007. ISSM 2007. International Symposium on*. IEEE. 2007, pp. 1–3.
- [Spi00] Frits C.R. Spijksma. “Multi index assignment problems: complexity, approximation, applications”. In: *Nonlinear Assignment Problems*. Springer, 2000, pp. 1–12.
- [SR95] Charles H. Stapper and Raymond J. Rosner. “Integrated circuit yield management and yield analysis: Development and implementation”. In: *Semiconductor Manufacturing, IEEE Transactions on* 8.2 (1995), pp. 95–102.
- [Sta86] Charles H. Stapper. “On yield, fault distributions, and clustering of particles”. In: *IBM Journal of Research and Development* 30.3 (1986), pp. 326–338.
- [Sta89a] Charles H. Stapper. “Fact and fiction in yield modeling”. In: *Microelectronics Journal* 20.1 (1989), pp. 129–151.
- [Sta89b] Charles H. Stapper. “Simulation of spatial fault distributions for integrated circuit yield estimations”. In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 8.12 (1989), pp. 1314–1318.
- [SV97] Amit Sahai and Salil P. Vadhan. “A Complete Promise Problem for Statistical Zero-Knowledge”. In: *38th Annual Symposium on Foundations of Computer Science, FOCS '97, Miami Beach, Florida, USA, October 19-22, 1997*. 1997, pp. 448–457.
- [Tao+10] Mottaqiallah Taouil et al. “On maximizing the compound yield for 3D Wafer-to-Wafer stacked ICs”. In: *2011 IEEE International Test Conference, ITC 2010, Austin, TX, USA, November 2-4, 2010*. 2010, pp. 183–192.



- 
- [TB91] Aakash Tyagi and Magedy A. Bayoumi. “A generalized Poisson based model for defect spatial distribution in WSI”. In: *Wafer Scale Integration, 1991. Proceedings.,[3rd] International Conference on*. IEEE. 1991, pp. 149–155.
- [TH11] Mottaqiallah Taouil and Said Hamdioui. “Layer Redundancy Based Yield Improvement for 3D Wafer-to-Wafer Stacked Memories”. In: *16th European Test Symposium, ETS 2011, Trondheim, Norway, May 23-27, 2011*. 2011, pp. 45–50.
- [Tre01] Luca Trevisan. “Non-approximability results for optimization problems on bounded degree instances”. In: *Proceedings on 33rd Annual ACM Symposium on Theory of Computing, July 6-8, 2001, Heraklion, Crete, Greece*. 2001, pp. 453–461.
- [Vaz01] Vijay V. Vazirani. *Approximation algorithms*. Springer, 2001.
- [Ver+10] Jouke Verbree et al. “On the cost-effectiveness of matching repositories of pre-tested wafers for wafer-to-wafer 3D chip stacking”. In: *15th European Test Symposium, ETS 2010, Prague, Czech Republic, May 24-28, 2010*. 2010, pp. 36–41.
- [Wil02] Herbert S. Wilf. *Algorithms and complexity*. A K Peters, 2002.
- [WLC10] Jianxin Wang, Wenjun Li, and Jianer Chen. “A parameterized algorithm for the hyperplane-cover problem”. In: *Theor. Comput. Sci.* 411.44-46 (2010), pp. 4005–4009.
- [Zuc07] David Zuckerman. “Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number”. In: *Theory of Computing* 3.1 (2007), pp. 103–128.

# A

---

## Computational results

---

---

### Contents

---

A.1 ILP comparison . . . . .	185
------------------------------	-----

---

### A.1 ILP comparison

Table A.1: Comparison of the performances of Formulation (VI) and Formulation (VIII) (long term experiment)

	random	Formulation (VI)				Formulation (VIII)				
	Fails	Time	#	Fails	Ratio	Time	#	Gap	Fails	Ratio
3 25 10 50	85.68%	3.21	10	63.67%	0.743	12.26	10	-	63.67%	0.743
3 25 10 70	62.32%	2.13	10	41.12%	0.66	24.26	10	-	41.12%	0.66
3 25 10 90	23.76%	0.78	10	14.48%	0.609	2.26	10	-	14.48%	0.609
3 25 100 50	85.69%	9.59	10	78.62%	0.917	-	0	5.06%	79.09%	0.923
3 25 100 70	63.00%	2.36	10	56.04%	0.89	-	0	7.05%	56.56%	0.898
3 25 100 90	24.99%	3.12	10	20.81%	0.833	-	0	2.78%	20.90%	0.836
3 25 1000 50	85.52%	8.61	10	83.32%	0.974	-	0	9.17%	83.67%	0.978
3 25 1000 70	63.07%	20.17	10	60.91%	0.966	-	0	13.59%	61.39%	0.973
3 25 1000 90	24.91%	2.36	10	23.54%	0.945	-	0	12.23%	23.78%	0.955
3 50 10 50	84.2 %	20.87	10	60.67%	0.721	69.13	10	-	60.67%	0.721
3 50 10 70	62.66%	16.22	10	38.96%	0.622	106.50	10	-	38.96%	0.622
3 50 10 90	24.36%	11.79	10	13.56%	0.557	20.22	10	-	13.56%	0.557
3 50 100 50	84.97%	66.38	10	77.22%	0.909	-	0	7.01%	78.33%	0.922
3 50 100 70	62.58%	50.54	10	54.63%	0.873	-	0	9.83%	55.79%	0.892
3 50 100 90	24.85%	40.01	10	20.04%	0.806	-	0	7.13%	20.56%	0.828
3 50 1000 50	84.99%	73.37	10	82.56%	0.971	-	0	10.83%	83.69%	0.985
3 50 1000 70	62.51%	90.94	10	60.08%	0.961	-	0	16.01%	61.28%	0.98
3 50 1000 90	24.81%	60.54	10	23.26%	0.937	-	0	17%	23.93%	0.964
6 25 10 50	97.0%	-	-	-	-	14104.79	8	5.10%	68.2%	0.703
6 25 10 70	84.24%	-	-	-	-	24912.32	4	2.65%	46.04%	0.547
6 25 10 90	35.16%	-	-	-	-	40.55	10	-	17.28%	0.491
6 25 100 50	97.32%	-	-	-	-	-	0	16.13%	91.68%	0.942
6 25 100 70	84.64%	-	-	-	-	-	0	27.17%	75.84%	0.896
6 25 100 90	39.28%	-	-	-	-	-	0	28.87%	32.50%	0.827
6 25 1000 50	97.43%	-	-	-	-	-	0	18.60%	96.96%	0.995
6 25 1000 70	84.66%	-	-	-	-	-	0	32.15%	83.70%	0.989
6 25 1000 90	39.37%	-	-	-	-	-	0	37.67%	38.00%	0.965
6 50 10 50	96.76%	-	-	-	-	-	0	3.02%	65.17%	0.674
6 50 10 70	82.98%	-	-	-	-	-	0	6.10%	43.72%	0.527
6 50 10 90	37.52%	-	-	-	-	16464.56	8	6.25%	15.4%	0.41
6 50 100 50	97.03%	-	-	-	-	-	0	19.18%	91.81%	0.946
6 50 100 70	83.57%	-	-	-	-	-	0	31.22%	76.23%	0.912
6 50 100 90	38.46%	-	-	-	-	-	0	38.61%	33.29%	0.865
6 50 1000 50	97.06%	-	-	-	-	-	0	21.05%	97.07%	1.0
6 50 1000 70	83.52%	-	-	-	-	-	0	35.14%	83.40%	0.999
6 50 1000 90	38.05%	-	-	-	-	-	0	45%	37.48%	0.985
9 25 10 50	99.52%	-	-	-	-	-	0	1.69 %	69.88%	0.702
9 25 10 70	91.88%	-	-	-	-	-	0	5.15 %	49.6%	0.54
9 25 10 90	44.8%	-	-	-	-	195.73	10	-	18.4%	0.411
9 25 100 50	99.49%	-	-	-	-	-	0	19.04%	95.56%	0.961
9 25 100 70	93.34%	-	-	-	-	-	0	35.11%	86.66%	0.928
9 25 100 90	49.18%	-	-	-	-	-	0	43.85%	43.06%	0.876
9 25 1000 50	99.51%	-	-	-	-	-	0	19.37%	99.39%	0.999
9 25 1000 70	93.18%	-	-	-	-	-	0	37.17%	93.23%	1.001
9 25 1000 90	48.72%	-	-	-	-	-	0	48.03%	48.26%	0.991
9 50 10 50	99.32%	-	-	-	-	-	0	5.62%	67.78%	0.682
9 50 10 70	91.36%	-	-	-	-	-	0	9.90%	46.28%	0.507
9 50 10 90	45.14%	-	-	-	-	6499.96	5	3.23%	16.28%	0.361
9 50 100 50	99.40%	-	-	-	-	-	0	23.22%	97.1%	0.977
9 50 100 70	92.08%	-	-	-	-	-	0	41.11%	88.61%	0.962
9 50 100 90	46.98%	-	-	-	-	-	0	48.18%	40.48%	0.862
9 50 1000 50	99.38%	-	-	-	-	-	0	22.13%	99.39%	1.0
9 50 1000 70	91.99%	-	-	-	-	-	0	40.13%	91.98%	1.0
9 50 1000 90	46.65%	-	-	-	-	-	0	53.77%	46.30%	0.993

Table A.2: Comparison of the performances of Formulation (VI) and Formulation (VIII) (short term experiment)

	random	Formulation (VI)				Formulation (VIII)				
	Fails	Time	#	Fails	Ratio	Time	#	Gap	Fails	Ratio
3 25 10 50	85.68%	0.516	10	63.67%	0.743	1.353	10	-	63.67%	0.743
3 25 10 70	62.32%	0.517	10	41.12%	0.660	1.475	10	-	41.12%	0.660
3 25 10 90	23.76%	0.616	10	14.48%	0.609	0.259	10	-	14.48%	0.609
3 25 100 50	85.69%	0.821	10	78.62%	0.917	-	0	5.79%	79.69%	0.930
3 25 100 70	63.00%	0.472	10	56.04%	0.890	-	0	8.16%	57.16%	0.907
3 25 100 90	24.99%	0.6	10	20.81%	0.833	-	0	3.93%	21.11%	0.845
3 25 1000 50	85.52%	0.737	10	83.32%	0.974	-	0	9.69%	84.14%	0.984
3 25 1000 70	63.07%	0.837	10	60.91%	0.966	-	0	14.23%	61.84%	0.981
3 25 1000 90	24.91%	0.44	10	23.54%	0.945	-	0	13.14%	24.04%	0.965
3 50 10 50	84.2%	6.759	10	60.67%	0.721	19.299	10	-	60.67%	0.721
3 50 10 70	62.66%	6.544	10	38.96%	0.622	22.292	10	-	38.96%	0.622
3 50 10 90	24.36%	7.231	10	13.56%	0.557	9.233	10	-	13.56%	0.557
3 50 100 50	84.97%	10.684	10	77.22%	0.909	-	0	7.33%	78.49%	0.924
3 50 100 70	62.58%	8.685	10	54.63%	0.873	-	0	11.12%	56.35%	0.900
3 50 100 90	24.85%	7.798	10	20.04%	0.806	-	0	8.55	20.76%	0.836
3 50 1000 50	84.99%	15.572	10	82.56%	0.971	-	0	12.02%	84.81%	0.998
3 50 1000 70	62.51%	11.977	10	60.08%	0.961	-	0	17.42%	62.32%	0.997
3 50 1000 90	24.81%	7.931	10	23.26%	0.937	-	0	19.32%	24.61%	0.992
6 25 10 50	97%	-	-	-	-	1078.943	3	1.76%	68.32%	0.704
6 25 10 70	84.24%	-	-	-	-	-	0	2.67%	46.44%	0.551
6 25 10 90	35.16%	-	-	-	-	18.313	10	-	17.28%	0.491
6 25 100 50	97.32%	-	-	-	-	-	0	17.03%	92.44%	0.950
6 25 100 70	84.64%	-	-	-	-	-	0	30.83%	79.78%	0.942
6 25 100 90	39.28%	-	-	-	-	-	0	33.22%	34.61%	0.881
6 25 1000 50	97.43%	-	-	-	-	-	0	18.95%	97.31%	0.999
6 25 1000 70	84.66%	-	-	-	-	-	0	32.65%	84.08%	0.993
6 25 1000 90	39.37%	-	-	-	-	-	0	37.92%	38.08%	0.967
6 50 10 50	96.76%	-	-	-	-	-	0	3.50%	65.6 %	0.678
6 50 10 70	82.98%	-	-	-	-	-	0	7.43%	44.32%	0.534
6 50 10 90	37.52%	-	-	-	-	321.987	6	6.37%	15.4 %	0.410
6 50 100 50	97.03%	-	-	-	-	-	0	22.00%	94.54%	0.974
6 50 100 70	83.57%	-	-	-	-	-	0	37.86%	82.71%	0.990
6 50 100 90	38.46%	-	-	-	-	-	0	45.05%	36.9 %	0.959
6 50 1000 50	97.06%	-	-	-	-	-	0	21.00%	96.96%	0.999
6 50 1000 70	83.52%	-	-	-	-	-	0	64.05%	82.98%	0.994
6 50 1000 90	38.05%	-	-	-	-	-	0	72.58%	37.48%	0.985
9 25 10 50	99.52%	-	-	-	-	-	0	2.92%	70.64%	0.710
9 25 10 70	91.88%	-	-	-	-	-	0	6.90%	50.56%	0.550
9 25 10 90	44.8%	-	-	-	-	243.776	10	-	18.4 %	0.411
9 25 100 50	99.49%	-	-	-	-	-	0	20.09%	96.81%	0.973
9 25 100 70	93.34%	-	-	-	-	-	0	37.12%	89.12%	0.955
9 25 100 90	49.18%	-	-	-	-	-	0	46.42%	44.87%	0.912
9 25 1000 50	99.51%	-	-	-	-	-	0	19.65%	99.45%	0.999
9 25 1000 70	93.18%	-	-	-	-	-	0	37.08%	92.62%	0.994
9 25 1000 90	48.72%	-	-	-	-	-	0	46.52%	46.83%	0.961
9 50 10 50	99.32%	-	-	-	-	-	0	17.00%	77.1 %	0.776
9 50 10 70	91.36%	-	-	-	-	-	0	26.25%	56.62%	0.620
9 50 10 90	45.14%	-	-	-	-	198.587	4	3.25%	16.28%	0.361
9 50 100 50	99.40%	-	-	-	-	-	0	25.33%	98.95%	0.995
9 50 100 70	92.08%	-	-	-	-	-	0	43.09%	91.17%	0.990
9 50 100 90	46.98%	-	-	-	-	-	0	54.13%	45.32%	0.965
9 50 1000 50	99.38%	-	-	-	-	-	0	50.20%	99.31%	0.999
9 50 1000 70	91.99%	-	-	-	-	-	0	67.38%	91.48%	0.994
9 50 1000 90	46.65%	-	-	-	-	-	0	77.35%	45.40%	0.973

Table A.3: Impact of the Feasibility Pump Heuristic on the Formulation (VIII)

	Default FPH (Long Term)				Aggressive FPH (Short Term)			
	Time	#	Gap	Fails	Time	#	Gap	Fails
3 25 10 50	12.26	10	-	63.67%	1.353	10	-	63.67%
3 25 10 70	24.26	10	-	41.12%	1.475	10	-	41.12%
3 25 10 90	2.26	10	-	14.48%	0.259	10	-	14.48%
3 25 100 50	-	0	5.06%	79.09%	-	0	5.79%	79.69%
3 25 100 70	-	0	7.05%	56.56%	-	0	8.16%	57.16%
3 25 100 90	-	0	2.78%	20.90%	-	0	3.93%	21.11%
3 25 1000 50	-	0	9.17%	83.67%	-	0	9.69%	84.14%
3 25 1000 70	-	0	13.59%	61.39%	-	0	14.23%	61.84%
3 25 1000 90	-	0	12.23%	23.78%	-	0	13.14%	24.04%
3 50 10 50	69.13	10	-	60.67%	19.299	10	-	60.67%
3 50 10 70	106.50	10	-	38.96%	22.292	10	-	38.96%
3 50 10 90	20.22	10	-	13.56%	9.233	10	-	13.56%
3 50 100 50	-	0	7.01%	78.33%	-	0	7.33%	78.49%
3 50 100 70	-	0	9.83%	55.79%	-	0	11.12%	56.35%
3 50 100 90	-	0	7.13%	20.56%	-	0	8.55%	20.76%
3 50 1000 50	-	0	10.83%	83.69%	-	0	12.02%	84.81%
3 50 1000 70	-	0	16.01%	61.28%	-	0	17.42%	62.32%
3 50 1000 90	-	0	17%	23.93%	-	0	19.32%	24.61%
6 25 10 50	14104.79	8	5.10%	68.2%	1078.943	3	1.76%	68.32%
6 25 10 70	24912.32	4	2.65%	46.04%	-	0	2.67%	46.44%
6 25 10 90	40.55	10	-	17.28%	18.313	10	-	17.28%
6 25 100 50	-	0	16.13%	91.68%	-	0	17.03%	92.44%
6 25 100 70	-	0	27.17%	75.84%	-	0	30.83%	79.78%
6 25 100 90	-	0	28.87%	32.50%	-	0	33.22%	34.61%
6 25 1000 50	-	0	18.60%	96.96%	-	0	18.95%	97.31%
6 25 1000 70	-	0	32.15%	83.70%	-	0	32.65%	84.08%
6 25 1000 90	-	0	37.67%	38.00%	-	0	37.92%	38.08%
6 50 10 50	-	0	3.02%	65.17%	-	0	3.50%	65.6 %
6 50 10 70	-	0	6.10%	43.72%	-	0	7.43%	44.32%
6 50 10 90	16464.56	8	6.25%	15.4%	321.987	6	6.37%	15.4 %
6 50 100 50	-	0	19.18%	91.81%	-	0	22.00%	94.54%
6 50 100 70	-	0	31.22%	76.23%	-	0	37.86%	82.71%
6 50 100 90	-	0	38.61%	33.29%	-	0	45.05%	36.9 %
6 50 1000 50	-	0	21.05%	97.07%	-	0	21.00%	96.96%
6 50 1000 70	-	0	35.14%	83.40%	-	0	64.05%	82.98%
6 50 1000 90	-	0	45%	37.48%	-	0	72.58%	37.48%
9 25 10 50	-	0	1.69 %	69.88%	-	0	2.92%	70.64%
9 25 10 70	-	0	5.15 %	49.6%	-	0	6.90%	50.56%
9 25 10 90	195.73	10	-	18.4%	243.776	10	-	18.4 %
9 25 100 50	-	0	19.04%	95.56%	-	0	20.09%	96.81%
9 25 100 70	-	0	35.11%	86.66%	-	0	37.12%	89.12%
9 25 100 90	-	0	43.85%	43.06%	-	0	46.42%	44.87%
9 25 1000 50	-	0	19.37%	99.39%	-	0	19.65%	99.45%
9 25 1000 70	-	0	37.17%	93.23%	-	0	37.08%	92.62%
9 25 1000 90	-	0	48.03%	48.26%	-	0	46.52%	46.83%
9 50 10 50	-	0	5.62%	67.78%	-	0	17.00%	77.1 %
9 50 10 70	-	0	9.90%	46.28%	-	0	26.25%	56.62%
9 50 10 90	6499.96	5	3.23%	16.28%	198.587	4	3.25%	16.28%
9 50 100 50	-	0	23.22%	97.1%	-	0	25.33%	98.95%
9 50 100 70	-	0	41.11%	88.61%	-	0	43.09%	91.17%
9 50 100 90	-	0	48.18%	40.48%	-	0	54.13%	45.32%
9 50 1000 50	-	0	22.13%	99.39%	-	0	50.20%	99.31%
9 50 1000 70	-	0	40.13%	91.98%	-	0	67.38%	91.48%
9 50 1000 90	-	0	53.77%	46.30%	-	0	77.35%	45.40%

Table A.4: Impact of the number of wafers per set on the performances of ILP formulations (complete results).

	random	Formulation (VI)				Formulation (VIII)				
	Fails	Time	#	Fails	Ratio	Time	#	Gap	Fails	Ratio
3 25 10 50	85.68%	0.516	10	63.67%	0.743	1.353	10	-	63.67%	0.743
3 50 10 50	84.2%	6.759	10	60.67%	0.721	19.299	10	-	60.67%	0.721
3 25 10 70	62.32%	0.517	10	41.12%	0.660	1.475	10	-	41.12%	0.660
3 50 10 70	62.66%	6.544	10	38.96%	0.622	22.292	10	-	38.96%	0.622
3 25 10 90	23.76%	0.616	10	14.48%	0.609	0.259	10	-	14.48%	0.609
3 50 10 90	24.36%	7.231	10	13.56%	0.557	9.233	10	-	13.56%	0.557
3 25 100 50	85.69%	0.821	10	78.62%	0.917	-	0	5.79%	79.69%	0.930
3 50 100 50	84.97%	10.684	10	77.22%	0.909	-	0	7.33%	78.49%	0.924
3 25 100 70	63.00%	0.472	10	56.04%	0.890	-	0	8.16%	57.16%	0.907
3 50 100 70	62.58%	8.685	10	54.63%	0.873	-	0	11.12%	56.35%	0.900
3 25 100 90	24.99%	0.6	10	20.81%	0.833	-	0	3.93%	21.11%	0.845
3 50 100 90	24.85%	7.798	10	20.04%	0.806	-	0	8.55	20.76%	0.836
3 25 1000 50	85.52%	0.737	10	83.32%	0.974	-	0	9.69%	84.14%	0.984
3 50 1000 50	84.99%	15.572	10	82.56%	0.971	-	0	12.02%	84.81%	0.998
3 25 1000 70	63.07%	0.837	10	60.91%	0.966	-	0	14.23%	61.84%	0.981
3 50 1000 70	62.51%	11.977	10	60.08%	0.961	-	0	17.42%	62.32%	0.997
3 25 1000 90	24.91%	0.44	10	23.54%	0.945	-	0	13.14%	24.04%	0.965
3 50 1000 90	24.81%	7.931	10	23.26%	0.937	-	0	19.32%	24.61%	0.992
6 25 10 50	97%	-	-	-	-	1078.943	3	1.76%	68.32%	0.704
6 50 10 50	96.76%	-	-	-	-	-	0	3.50%	65.6 %	0.678
6 25 10 70	84.24%	-	-	-	-	-	0	2.67%	46.44%	0.551
6 50 10 70	82.98%	-	-	-	-	-	0	7.43%	44.32%	0.534
6 25 10 90	35.16%	-	-	-	-	18.313	10	-	17.28%	0.491
6 50 10 90	37.52%	-	-	-	-	321.987	6	6.37%	15.4 %	0.410
6 25 100 50	97.32%	-	-	-	-	-	0	17.03%	92.44%	0.950
6 50 100 50	97.03%	-	-	-	-	-	0	22.00%	94.54%	0.974
6 25 100 70	84.64%	-	-	-	-	-	0	30.83%	79.78%	0.942
6 50 100 70	83.57%	-	-	-	-	-	0	37.86%	82.71%	0.990
6 25 100 90	39.28%	-	-	-	-	-	0	33.22%	34.61%	0.881
6 50 100 90	38.46%	-	-	-	-	-	0	45.05%	36.9 %	0.959
6 25 1000 50	97.43%	-	-	-	-	-	0	18.95%	97.31%	0.999
6 50 1000 50	97.06%	-	-	-	-	-	0	21.00%	96.96%	0.999
6 25 1000 70	84.66%	-	-	-	-	-	0	32.65%	84.08%	0.993
6 50 1000 70	83.52%	-	-	-	-	-	0	64.05%	82.98%	0.994
6 25 1000 90	39.37%	-	-	-	-	-	0	37.92%	38.08%	0.967
6 50 1000 90	38.05%	-	-	-	-	-	0	72.58%	37.48%	0.985
9 25 10 50	99.52%	-	-	-	-	-	0	2.92%	70.64%	0.710
9 50 10 50	99.32%	-	-	-	-	-	0	17.00%	77.1 %	0.776
9 25 10 70	91.88%	-	-	-	-	-	0	6.90%	50.56%	0.550
9 50 10 70	91.36%	-	-	-	-	-	0	26.25%	56.62%	0.620
9 25 10 90	44.8%	-	-	-	-	243.776	10	-	18.4 %	0.411
9 50 10 90	45.14%	-	-	-	-	198.587	4	3.25%	16.28%	0.361
9 25 100 50	99.49%	-	-	-	-	-	0	20.09%	96.81%	0.973
9 50 100 50	99.40%	-	-	-	-	-	0	25.33%	98.95%	0.995
9 25 100 70	93.34%	-	-	-	-	-	0	37.12%	89.12%	0.955
9 50 100 70	92.08%	-	-	-	-	-	0	43.09%	91.17%	0.990
9 25 100 90	49.18%	-	-	-	-	-	0	46.42%	44.87%	0.912
9 50 100 90	46.98%	-	-	-	-	-	0	54.13%	45.32%	0.965
9 25 1000 50	99.51%	-	-	-	-	-	0	19.65%	99.45%	0.999
9 50 1000 50	99.38%	-	-	-	-	-	0	50.20%	99.31%	0.999
9 25 1000 70	93.18%	-	-	-	-	-	0	37.08%	92.62%	0.994
9 50 1000 70	91.99%	-	-	-	-	-	0	67.38%	91.48%	0.994
9 25 1000 90	48.72%	-	-	-	-	-	0	46.52%	46.83%	0.961
9 50 1000 90	46.65%	-	-	-	-	-	0	77.35%	45.40%	0.973



## List of Figures

---

1	Influence of the assignment on the final die yield. . . . .	12
2.1	Positive and negative instances of HAMILTONIAN CYCLE. . . . .	23
2.2	Incomplete hierarchy of approximation classes. . . . .	31
2.3	General reduction scheme. . . . .	33
2.4	Gap-reduction principle. . . . .	37
2.5	Perfect instance of UNIQUE LABEL COVER. . . . .	39
2.6	Instance of UNIQUE LABEL COVER partially satisfied. . . . .	40
4.1	Example of input of wafer-to-wafer integration problem. . . . .	56
4.2	Walk on the dies. . . . .	57
4.3	Modelization of an input of wafer-to-wafer integration problem. . . . .	57
4.4	Example of solution $S$ for the instance of wafer-to-wafer integration problem depicted on Figure 4.3. . . . .	58
5.1	Cartography of polynomial reductions for $\max \sum 1$ . . . . .	68
5.2	Cartography of polynomial reductions for $\min \sum 0$ . . . . .	70
6.1	Overview of the results for $\max \max 1$ . . . . .	72
6.2	Overview of the results for $\min \min 0$ . . . . .	73
6.3	Example of reduction from $k$ -DIMENSIONAL MATCHING to $\max_{\neq 0}$ . . . . .	74
6.4	Example of reduction from MAX CLIQUE to $\max \max 1$ . . . . .	76
6.5	Example of reduction from $\max \max 1$ to $(\max \max 1)_{\#0 \leq 1}$ . . . . .	78
6.6	Example of construction of a $(\text{BMVA})_{\#0 \leq 1}$ instance from a $k$ -uniform hypergraph. . . . .	79
6.7	Example of reduction from MIN-HS to $\min \min 0$ . . . . .	82
6.8	Example of reduction from $\max \max 1$ to INDEPENDENT SET. . . . .	85
7.1	Overview of the results for $\max \sum 1$ . . . . .	89
7.2	Counter-example of the extension of the $p$ -approximation algorithm. . . . .	93
7.3	Use of matchings to satisfy profile demands. . . . .	95
7.4	Illustration of Proposition 7.1. . . . .	97
7.5	AND-cross composition example. . . . .	102
7.6	Illustration of block partitioning and of shape of partial stacks. . . . .	104
7.7	Illustration of reachable shape. . . . .	105
8.1	Overview of the results for $\min \sum 0$ . . . . .	111
8.2	Example of $\min \sum 0$ instance such that no optimal solution creates a 1-stack. . . . .	118
8.3	Example of reduction from 2- $\min \sum 0$ to BIP-OCT. . . . .	123



8.4	Example of reduction from BIP-OCT to OCT. . . . .	125
8.5	Example of reduction from $\chi$ -COLORING to $\min \sum 0$ . . . . .	127
10.1	Representation of the instances that can be solved with Formulation VI without reaching the memory break point. . . . .	140
11.1	Illustration of the running scheme of Algorithm 8 on an instance with $m = 6$ . . . . .	171
11.2	Illustration of the running scheme of Algorithm 9 on an instance with $m = 6$ . . . . .	172

## List of Tables

---

10.1	Number of solved instances for $m = 5$ and $n = 25$ in function of $p$ and $Y_W$ . . . . .	141
10.2	Impact of the Feasibility Pump Heuristic on the Formulation (VI) . . .	150
10.3	Impact of the Feasibility Pump Heuristic on the Formulation (VIII) restricted to sets of instances solved to optimality in both experiment. . . . .	151
10.4	Impact of the Feasibility Pump Heuristic on the Formulation (VIII) restricted to set of instances such that no instance have been solved to optimality in both experiments. . . . .	152
10.5	Impact of the Feasibility Pump Heuristic on the Formulation (VIII) on sets of instances containing at least one instance solved to the optimality by at least one of the two formulations. . . . .	153
10.6	Impact of the number of sets on the Formulations (VI) and (VIII) . . .	154
10.7	Impact of the number of wafers per set on the performances of ILP formulations. . . . .	154
10.8	Impact of $Y_W$ on the performances of ILP formulations. . . . .	156
10.9	Impact of $p$ on the performances of ILP formulations. . . . .	158
10.10	Influence of the wafer yield on the performances of BDS-10 and VDP-10 heuristics. . . . .	161
10.11	Influence of the wafer yield on the performances of BDS-50 and VDP-50 heuristics. . . . .	162
10.12	Influence of the number of dies in reduced instances on the performances of BDS-10 and BDS-50 heuristics. . . . .	164
10.13	Influence of the number of dies in reduced instances on the performances of VDP-10 and VDP-50 heuristics. . . . .	165
10.14	Impact of the set aggregation technique on the time computation and objective function . . . . .	167
11.1	Results . . . . .	174
A.1	Comparison of the performances of Formulation (VI) and Formulation (VIII) (long term experiment) . . . . .	186
A.2	Comparison of the performances of Formulation (VI) and Formulation (VIII) (short term experiment) . . . . .	187
A.3	Impact of the Feasibility Pump Heuristic on the Formulation (VIII) . . .	188
A.4	Impact of the number of wafers per set on the performances of ILP formulations (complete results). . . . .	189



# Integer Linear Programs

---

I	Generic formulation for 3DA . . . . .	64
II	Alternative ILP formulation for $\max \sum 1$ . . . . .	67
III	XP ILP formulation for $\max \sum 1$ . . . . .	68
IV	FPT-formulation sketch . . . . .	95
V	FPT-formulation with Hall constraints . . . . .	98
VI	MDA inspired formulation . . . . .	139
VII	Hub set formulation . . . . .	143
VIII	Improved hub set formulation . . . . .	144
IX	Alternative MDA inspired formulation . . . . .	145
X	Alternative hub set formulation . . . . .	146



## List of Algorithms

---

1	Kernelization from FPT-algorithm . . . . .	49
2	$p$ -approximation for $\max \sum 1$ . . . . .	92
-	Function $\text{MinSumZeroDP}(l, Sh)$ . . . . .	106
3	Reduced dies heuristic principle . . . . .	159
4	Viable dies preserving heuristic (VDP) . . . . .	159
5	Bad dies stacking heuristic (BDS) . . . . .	160
6	Sequential heuristic . . . . .	170
7	Heaviest first heuristic . . . . .	170
8	Lot partitioning heuristic . . . . .	171
9	Balanced binary tree heuristic . . . . .	172
-	Function $\text{RecursiveSplit}(s, e)$ . . . . .	173