On the complexity of Wafer-to-Wafer Integration

M. Bougeret[a,b,*], V. Boudet[a,b], T. Dokka[c], G. Duvillié[b,a,**], R. Giroudeau[a,b]

[a] *LIRMM, 161 rue Ada, 34095 Montpellier Cedex 5*
[b] *Université de Montpellier, 163 rue Auguste Broussonnet, 34090 Montpellier*
[c] *Dept. of Management Science, Lancaster University Management School, Lancaster, LA1 4YX, UK*

Abstract

In this paper we consider the Wafer-to-Wafer Integration problem. A wafer can be seen as a p-dimensional binary vector. The input of this problem is described by m multisets (called "lots"), where each multiset contains n wafers. The output of the problem is a set of n disjoint stacks, where a stack is a set of m wafers (one wafer from each lot). To each stack we associate a p-dimensional binary vector corresponding to the bit-wise AND operation of the wafers of the stack. The objective is to maximize the total number of "1" in the n stacks. We provide $m^{1-\epsilon}$ and $p^{1-\epsilon}$ non-approximability results even for $n = 2$, $f(n)$ non-approximability for any polynomial-time computable function f, as well as a $\frac{p}{r}$-approximation algorithm for any constant r. Finally, we show that the problem is **FPT** when parameterized by p, and we use this **FPT** algorithm to improve the running time of the $\frac{p}{r}$-approximation algorithm.

Keywords: wafer-to-wafer integration problem, approximability, computational complexity, parameterized complexity, multidimensional binary vector assignment

1. Introduction

1.1. Problem definition

In this paper we consider Wafer-to-Wafer Integration problems. In these problems, we are given m multisets $V^1, \ldots, V^m$, where each set V^i contains n binary p-dimensional vectors. For any $j \in [n]$ [1], and any $i \in [m]$, we denote by v^i_j the j^{th} vector of the multiset V^i, and for any $l \in [p]$ we denote by $v^i_j[l] \in \{0, 1\}$ the l^{th} component of v^i_j.

*Corresponding author
**Principal corresponding author
Email addresses: bougeret@lirmm.fr (M. Bougeret), boudet@lirmm.fr (V. Boudet), t.dokka@lancaster.ac.uk (T. Dokka), duvillie@lirmm.fr (G. Duvillié), rgirou@lirmm.fr (R. Giroudeau)

[1] The notation $[n]_j$ stands for $\{j, \ldots, n\}$ and to lighten the notation, we will use the classical notation $[n]$ instead of $[n]_1$.

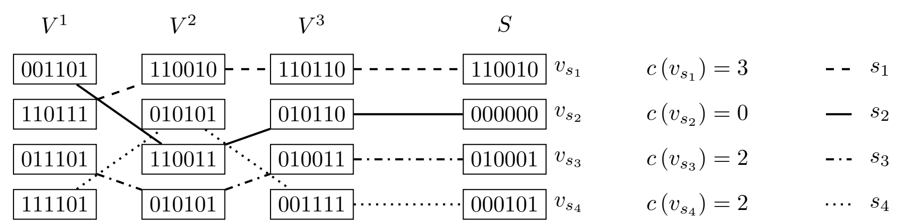

Figure 1: Example of max $\sum 1$ instance with $m = 3, n = 4, p = 6$ and of a feasible solution S of profit $f_{\Sigma 1}(S) = 7$.

Let us now define the output. A stack $s = (v_1^s, \ldots, v_m^s)$ is an $m - tuple$ of vectors such that $v_i^s \in V^i$, for any $i \in [m]$. An output of the problem is a set $S = \{s_1, \ldots, s_n\}$ of n stacks such that for any i and j, the vector v_j^i is contained exactly in one stack. An example of input and output is depicted in Figure 1.

These problems are motivated by an application in IC manufacturing in semiconductor industry, see [**?**] for more details about this application. A wafer can be seen as a string of bad dies (0) and good dies (1). Integrating two wafers corresponds to superimposing the two corresponding strings. In this operation, a position in the merged string is only 'good' when the two corresponding dies are good, otherwise it is 'bad'. The objective of Wafer-to-Wafer Integration is to form n stacks, while maximizing the overall quality of the stacks (depending on the objective function).

Let us now define several objective functions, and the corresponding optimization problems. We consider the operator $\wedge$ which maps two p-dimensional vectors to another one by performing the logical *and* operation on each component of entry vectors. More formally, given two $p-$dimensional vectors u and v, we define $u \wedge v = (u[1] \wedge v[1], u[2] \wedge v[2], \ldots, u[p] \wedge v[p])$. We associate to any stack $s = (v_1^s, \ldots, v_m^s)$ a binary p-dimensional vector $v_s = \bigwedge_{i=1}^m v_i^s$. Then, the profit of a stack s is given by $c(v_s)$, where $c(v) = \sum_{l=1}^p v[l]$. Roughly speaking, the profit of a stack is the number of good bits in the representative vector of this stack, where a good bit (in position l) survives if and only if all the vectors of the stack have a good bit in position l.

We are now ready to define the two following optimization problems:

Set of problems 1 max $\sum 1$ and min $\sum 0$

Input	m multisets of n binary p-dimensional vectors
Output	a set $S = \{s_1, s_2, \ldots, s_n\}$ of n disjoint stacks
Objective functions	max $\sum 1$: maximize $f_{\sum 1}(S) = \sum_{j=1}^n c(v_{s_j})$, the total number of good bits min $\sum 0$: minimize $f_{\sum 0}(S) = np - \sum_{j=1}^n c(v_{s_j})$, the total number of bad bits

Instances of these problems will be denoted by $I[m, n, p]$. The notation $f(S)$ (instead of $f_{\sum 0}(S)$, $f_{\sum 1}(S), \ldots$) will be used when the context is non ambiguous. Note that we use multisets to modelize the sets of wafers since two different wafers can share the same representative vector. In the following, we refer to *multisets* as *sets* and consider two copies of a same vector as two distinct elements.

1.2. Related work

In this paper we consider results in the framework of approximation and fixed parameter tractability theory. We only briefly recall the definitions here and refer the reader to [**? ?**] for more information. For any $\rho > 1$, a ρ-approximation algorithm A (for a maximization problem) is such that for any instance I, $A(I) \geq \frac{Opt(I)}{\rho}$, where $Opt(I)$ denotes the optimal value. The input of a para-meterized (decision) problem Π is a couple (X, κ), where $X \subseteq \Sigma^*$ is a classical decision problem, and $\kappa : \Sigma^* \longrightarrow \mathbb{N}$ is a parameterization. Deciding Π requires to determine for any instance $I \in \Sigma^*$ if $I \in X$. Finally, we say that an algorithm A decides Π in **FPT** time (or that Π is **FPT** parameterized by κ) if and only if there exist a computable function f and a constant c such that for any I, $A(I)$ runs in $\mathcal{O}(f(\kappa(I))|I|^c)$.

The $\max \sum 1$ problem was originally defined in [**?**] as the “yield maximization problem in wafer-to-wafer 3-D integration technology”. Authors of [**?**] point out that "the classical **NP**-hard 3-D matching problem is reducible to the $\max \sum 1$ problem". However, they do not provide the reduction and they only conclude that $\max \sum 1$ is **NP**-hard without stating consequences on the approximability. They also notice that $\max \sum 1$ is polynomial for $m = 2$ (as it reduces to finding a maximum profit perfect matching in a bipartite graph, solved by Hungarian Method), and design the "iterative matching heuristic" (IMH) that computes a solution based on (2D) matchings.

In [**?**] and [**?**] we investigated the $\min \sum 0$ problem by providing a $\frac{4}{3}$-approximation algorithm for $m = 3$ and several $f(m)$-approximation algorithms for arbitrary m (and for a more general profit function c). Furthermore, we also noticed in [**?**] that the natural ILP formulation implied that $\min \sum 0$ and $\max \sum 1$ are polynomial for fixed p. Concerning negative results, the implicit straightforward reduction from k-Dimensional Matching in [**?**] and made explicit in Section 2, shows that $\min \sum 0$ is **NP**-hard, and $\max \sum 1$ is $\mathcal{O}(\frac{m}{\ln m})$ non-approximable. The more complex reduction of [**?**] shows that $\min \sum 0$ is **APX**-hard even for $m = 3$, and thus is very unlikely to admit a **PTAS**.

1.3. Contributions

In this paper we mainly study the $\max \sum 1$ problem, with a particular focus on parameter p. In Subsection 2.2, we provide a strict reduction from Max Clique. Such a reduction proves that, even for $n = 2$, for any ϵ, there is no $\rho(m, p)$-approximation algorithm for $\max \sum 1$ for any function ρ satisfying $\rho(x, x) = x^{1-\epsilon}$ unless **P** = **NP** (this implies in particular no $p^{1-\epsilon}$ and no $m^{1-\epsilon}$ ratios). This reduction implies that, for any polynomial-time computable function f, $\max \sum 1$ is hard to approximate within a factor of $f(n)$. These negative

	$\max \sum 1$	$\min \sum 0$
[?]	**NP**-hard	
[? ?]	$\mathcal{O}(\frac{m}{\ln m})$ inapproximability polynomial for fixed p	for $m = 3$, $\frac{4}{3}$-approximation **APX**-hard $f(m)$ approximation for general m polynomial for fixed p
This paper	for any $\varepsilon > 0$, $p^{1-\varepsilon}$ and $m^{1-\varepsilon}$ non-approximation, even for $n = 2$ $f(n)$ non-approximabtion for any polynomial-time computable function f p/r-approximation in $\mathcal{O}(f(r)poly(m+n+p))$	**FPT**$/p$

Table 1: Overview of results on Wafer-to-Wafer Integration

results show that the simple p-approximation presented in Section 3.1 is somehow the best ratio we can hope for. Nevertheless, looking for better positive results we focus on $\frac{p}{r}$-approximation algorithm for any constant r. It turns out that any exact algorithm with a running time $\mathcal{O}(f(n, m, p))$ for $\max \sum 1$ can be used to derive a $\frac{p}{r}$-approximation running in time $\mathcal{O}(p \times f(n, m, r))$, using a classical shifting technique. This motivates our main result: determining the complexity of the $\max \sum 1$ problem when parameterized by p. The natural ILP in [?] implied that $\max \sum 1$ (and $\min \sum 0$) is polynomial for fixed p. We provide in Section 3.2 another ILP formulation proving that $\max \sum 1$ (and $\min \min 0$) is **FPT** when parameterized by p, using the Frank et al. [?] improvement of the Kannan algorithm [?], and thus improving the complexity of the $\frac{p}{r}$-approximation.

The contributions presented in this paper are summarized in the Table 1.

This paper is a extended version of [?] where we also provide an **EPTAS** for $\max \sum 1$ with fixed n and an exact result for $\max \max 1$ for $n = 2$.

2. Negative Results

In order to obtain negative results for $\max \sum 1$, let us first introduce two related problems defined in the Set of Problems 2.

Roughly speaking, we will see that approximating $\max \sum 1$ is *harder* than approximating these two problems, and that these problems are themselves non-approximable.

Set of problems 2 $\max\max 1$ and $\max_{\neq 0}$

Input	m multisets of n binary p-dimensional vectors
Output	a set S of n disjoint stacks
Objective functions	$\max\max 1$: maximize $f_{\max 1}(S) = \max_{j\in[n]} c\left(v_{s_j}\right)$, the profit of the best stack $\max_{\neq 0}$: maximize $f_{\neq 0}(S) = \lvert\{j \lvert c\left(v_{s_j}\right) \geq 1\}\rvert$, the number of non null stacks

To show that approximability is preserved we will provide strict reductions [**?**]. Indeed, if there is a strict reduction from Π_1 to Π_2, then any polynomial ρ-approximation for Π_2 yields to a ρ-approximation for Π_1. To avoid the technical conditions in the definition of the strict reductions, we will consider a subset of the latter. We will indeed provide reductions that satisfy conditions 1 and 2 of the following property.

Property 1. *Let Π_1 and Π_2 be two maximization problems with their given objective functions m_1 and m_2. Let f be a polynomial function that given any instance x of Π_1 associate an instance $f(x)$ of Π_2. Let g be a polynomial function that given any instance x of Π_1, and feasible solution S_2 of $f(x)$, associates a feasible solution $g(x, S_2)$ of Π_1. If f and g verify the two following conditions:*

1. *$Opt(x) = Opt(f(x))$*
2. *$m_1(g(x, S_2)) \geq m_2(f(x))$*

then (f, g) is a strict reduction.

2.1. Relation between $\max_{\neq 0}$, $\max\max 1$ and $\max\sum 1$

Observation 1. *There exists a strict reduction from $\max\max 1$ to $\max\sum 1$.*

Proof. Let us construct (f, g) as in Property 1. Consider an instance $I'[m', n', p']$ of $\max\max 1$. We construct an instance $I[m, n, p]$ of $\max\sum 1$ as follows: we set $p = p'$, $n = n'$, $m = m' + 1$. The m' sets of $I'[m', n', p']$ remain unchanged in $I[m, n, p]$: $\forall i \in [m']$, $V^i = V'^i$ and the last set $V^{m'+1}$ contains $(n-1)$ "zero vectors" (*i.e.* vectors having only 0) and one "one vector" (*i.e.* vector having only 1).

Informally, the set $V^{m'+1}$ of 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, it is straightforward to see that the following statement is true: $\forall x$, *$\exists$ a solution S' of $\max\max 1$ of profit $f_{\max 1}(S') \geq x \Leftrightarrow \exists$ a solution S of $\max\sum 1$ of profit $f_{\sum 1}(S) = x$.* Thus, we get $Opt_{\max\max 1}(I') = Opt_{\max\sum 1}(I)$. As the previous reduction is polynomial, and a solution of I' can be deduced from a solution of I in polynomial time, we get the desired result. □

Observation 2. *There exists a strict reduction from $\max_{\neq 0}$ to $\max\sum 1$.*

Proof. Consider an instance $I'[m', n', p']$ of $\max_{\neq 0}$. We construct an instance $I[m, n, p]$ of $\max \sum 1$ as follows. The number of components of each vector is left unchanged ($p = p'$), the number of vectors per set is multiplied by p' ($n = n'p'$) and the number of sets is increased by one ($m = m' + 1$). $\forall j = 1, \ldots, m'$, the sets V^j are constructed as follows: $V^i = V'^i \bigcup X$, where X contains $n - n'$ null vectors, and $V^{m'+1}$ contains n' times the following sets of vectors (this is the reason why $n = n'p'$):

$$\{\underbrace{1000\ldots000}_{p=p'}, 0100\ldots000, 0010\ldots000, \ldots, 0000\ldots010, 0000\ldots001\}$$

As an example, the following instance $I'[3, 2, 4]$ of $\max_{\neq 0}$

$$\underbrace{\begin{matrix} V'^1_1 = 1010 \\ V'^1_2 = 1001 \end{matrix}}_{V^1} \qquad \underbrace{\begin{matrix} V'^2_1 = 0001 \\ V'^2_2 = 0100 \end{matrix}}_{V^2} \qquad \underbrace{\begin{matrix} V'^3_1 = 1111 \\ V'^3_2 = 1000 \end{matrix}}_{V^3}$$

is turned into the following one $I[4, 8, 4]$ of $\max \sum 1$:

$$\underbrace{\begin{matrix} v^1_1 = 1010 \\ v^1_2 = 1001 \\ v^1_3 = 0000 \\ v^1_4 = 0000 \\ v^1_5 = 0000 \\ v^1_6 = 0000 \\ v^1_7 = 0000 \\ v^1_8 = 0000 \end{matrix}}_{V^1} \qquad \underbrace{\begin{matrix} v^2_1 = 0001 \\ v^2_2 = 0100 \\ v^2_3 = 0000 \\ v^2_4 = 0000 \\ v^2_5 = 0000 \\ v^2_6 = 0000 \\ v^2_7 = 0000 \\ v^2_8 = 0000 \end{matrix}}_{V^2} \qquad \underbrace{\begin{matrix} v^3_1 = 1111 \\ v^3_2 = 1000 \\ v^3_3 = 0000 \\ v^3_4 = 0000 \\ v^3_5 = 0000 \\ v^3_6 = 0000 \\ v^3_7 = 0000 \\ v^3_8 = 0000 \end{matrix}}_{V^3} \qquad \underbrace{\begin{matrix} v^4_1 = 1000 \\ v^4_2 = 0100 \\ v^4_3 = 0010 \\ v^4_4 = 0001 \\ v^4_5 = 1000 \\ v^4_6 = 0100 \\ v^4_7 = 0010 \\ v^4_8 = 0001 \end{matrix}}_{V^4}$$

Informally, as the set V^m of I turns any non zero stack of I' into a stack of value 1 (by choosing an appropriate vector), maximizing the total number of 1 in I requires to maximize the number of non null stacks in I'.

Let us first check that "$\forall$ solution S' of $\max_{\neq 0}$, $\exists$ a solution S of $\max \sum 1$ of value $f_{\sum 1}(S) = f_{\neq 0}(S')$". Let $\{s'_1, \ldots, s'_x\}$ be the x non null stacks of S', and $\{s'_{x+1}, \ldots, s'_{n'}\}$ be the null stacks of S'. Let us now construct S. For any $i, 1 \leq i \leq x$, let l_i be a non null bit in s'_i. We extend s'_i to a stack s_i by adding a vector v^m_j of set m such that $v^m_j[l_i] = 1$. Notice that such a vector always exists as for any position $l, 1 \leq l \leq p$ there are n' wafers in the set V^m whose bit in position l is equal to 1. Thus, even if the x stacks of S' have the same non null position l, the previous construction is possible. Finally, the $n' - x$ remaining null stacks of S' are extended arbitrarily, and we complete the construction of S by adding $n - n'$ arbitrary stacks (as these stacks use in each of the first $m - 1$ set the set X of null vectors, the value of these stacks is zero). Thus, we get $f_{\sum 1}(S) = x$.

Let us now check that "$\forall$ solution S of $\max \sum 1$, $\exists$ a solution S' of $\max_{\neq 0}$ of value $f_{\neq 0}(S') \geq f_{\sum 1}(S)$. As any vector of set m has only one good bit (*i.e.*, equal to 1), the profit of any stack of S is at most 1, and thus there are exactly x non null stacks $\{s_1, \dots, s_x\}$ in S. By removing the vector in the set V^m in each of these x stacks, we get x non null stacks of I'. Finally, we complete the construction of S' by creating arbitrarily the $n' - x$ remaining stacks, and we get $f_{\neq 0}(S') \geq x$ (notice that the value of S' can be greater than x, as we could have a null stack $s_i \in S$ whose restriction to the first $(m-1)$ set is a non null stack of I').

Thus, we get $Opt_{\max_{\neq 0}}(I') = Opt_{\max \sum 1}(I)$. As the previous reduction is polynomial, and as a solution of S of I can be translated back in polynomial time into a solution S' of I' with $f_{\neq 0}(S') \geq f_{\sum 1}(S)$, we get the desired result. □

According to Observations 1 and 2, any non-approximability result for $\max_{\neq 0}$ or $\max\max 1$ will transfer to $\max\sum 1$. This motivates the next section.

2.2. Hardness of $\max_{\neq 0}$ *and* $\max\max 1$

The reduction from k-Dimensional Matching (k-DM) provided by Dokka et al. in [?] for $\min\sum 0$ can be used as such for $\max_{\neq 0}$. In the following, we present the reduction of [?] and slightly adapt the proof of this article to $\max_{\neq 0}$, showing that, unlike the case of $\min\sum 0$, the reduction preserves approximability.

Theorem 1 (implicit in [?]). *There is a strict reduction from k-DM to $\max_{\neq 0}$.*

Proof. Let us describe the reduction provided in [?]. Let I be an instance of k-DM described by k sets $X_i, 1 \leq i \leq k$ (where X_i are pairwise disjoint) such that $|X_i| = n$, and x k-tuples $t_l \in X_1 \times \cdots \times X_k, 1 \leq l \leq x$. We denote by $a_i^j, 1 \leq j \leq n$ the elements of set X_i.

From this instance, we construct an instance of $\max_{\neq 0}$ composed of k sets, each containing n vectors. The number of bits per vector is equal to x. The j^{th} vector of set i represents the set of k-tuples that use element a_i^j. Thus, we define v_j^i as a string of size x, where the l^{th} bit is set to 1 if and only if a_i^j is used in t_l.

Hence, the l^{th} bit of a stack is 1 if and only if each element of tuple l is selected (by selecting corresponding vector), and then if and only if tuple t_l belongs to solution of k-DM instance. Notice that the value of any stack is at most 1, since a stack represents a tuple.

An example of the reduction is depicted in Figure 2. □

As it is **NP**-hard to approximate k-DM within a factor of $\mathcal{O}(\frac{k}{\ln(k)})$ [?], we get the following corollary:

Corollary 1. *It is **NP**-hard to approximate $\max_{\neq 0}$ within a factor of $\mathcal{O}(\frac{m}{ln(m)})$.*

We can also notice that any $\frac{m}{r}$-approximation ratio (for a constant $r \geq 3$) for $\max_{\neq 0}$ or $\max\sum 1$ would improve the currently best known ratio for k-DM set to $\frac{k+1+\epsilon}{3}$ in [?].

Let us now consider a new reduction which provides inapproximability results according to parameter p even for $n = 2$.

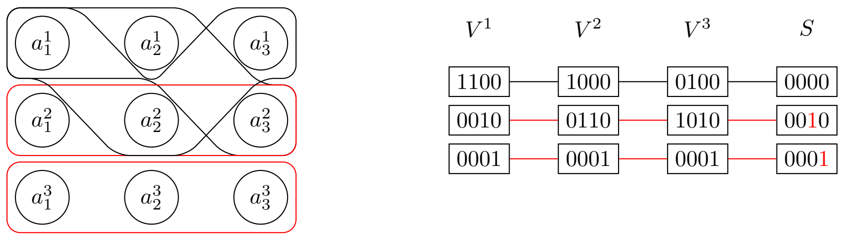

Figure 2: Example of reduction from an instance I of k-DM with $k = 3$, $X_1 = \{a_1^1, a_1^2, a_1^3\}$, $X_2 = \{a_2^1, a_2^2, a_2^3\}$, $X_3 = \{a_3^1, a_3^2, a_3^3\}$, $T = \{(a_1^1, a_2^1, a_3^2), (a_1^1, a_2^2, a_3^1), (a_1^2, a_2^2, a_3^2), (a_1^3, a_2^3, a_3^3)\}$ and a solution of profit 2 to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

Theorem 2. *There is a strict reduction from the* Max Clique *problem to* $\max\max 1$ *for* $n = 2$.

Proof. Let us construct (f, g) as in Property 1. Let us consider an instance $G = (V, E)$ of the Max Clique problem. The corresponding instance of $\max\max 1$ is constructed as follows. We consider $m = |V|$ sets, each having two vectors. All the vectors have $p = |V|$ bits. For each vertex i of V, we create the set $V^i = (v_1^i, v_2^i)$. For any i, we define $v_1^i = (v_1^i[1], v_1^i[2], \ldots, v_1^i[p])$, where $v_1^i[l] = 1$ if and only if $(i, l) \in E$ or $i = l$, and $v_2^i = (v_2^i[1], v_2^i[2], \ldots, v_2^i[p])$, where $v_2^i[l] = 1$ if and only if $i \neq l$. In other words, v_1^i corresponds to the i^{th} row of the adjacency matrix of G, with a 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^{th} component to 0, which corresponds to a penalty for not choosing vertex i.

We first need to state an intermediate lemma. For any stack $s = \{v_1^s, \ldots, v_m^s\}$, 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 1. $\forall i \in [p],\ v_s[i] = 1 \Leftrightarrow ((i \in X_s)$ *and* $(\forall x \in X_s \setminus i,\ (x, i) \in E))$.

$\triangle$ Let us first prove Lemma 1. Suppose i^{th} component of v_s is 1. 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 $v_1^x \in s$. Moreover, $v_s[i] = 1$ implies that $v_1^x[i] = 1$, 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 $\{i', i\} \in E$. $\triangle$

It is now straightforward to prove that $\forall x$, "$\exists$ solution S for $\max\max 1$ of value $f_{\max 1}(S) = x \Leftrightarrow \exists$ a clique X in G of size x." Indeed, suppose first that we have a solution S such that $f_{\max 1}(S) = x$. Let $s = (v_1^s, \ldots, v_m^s)$ be the stack in S of value x, and let $G_s = \{l | v_s[l] = 1\}$ be the set of good bits in the representative vector of s. We immediately get that the vertices corresponding

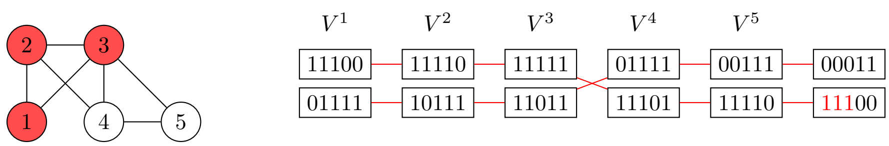

Figure 3: Illustration of the reduction from an instance of the Max Clique problem defined by graph $G = (\{1,2,3,4,5\}, \{(1,2),(1,3),(2,3),(2,4),(3,4),(3,5),(4,5)\})$ admitting a solution of profit 3 to an instance of $\max\max 1$ with $m = p = |V| = 5$, $n = 2$ admitting a solution S of cost $c(S)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

to G_s form 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, $Opt_{\max\max 1}(I)$ is equal to the size of the maximum clique in G. As the previous reduction is polynomial, and as a solution of S of I can be translated back in polynomial time into the corresponding clique in G (of same size), we get the desired result.

An example of this reduction is depicted in Figure 3.

□

Zuckerman shows in [**?**] that, for any ϵ there is no $|V|^{1-\epsilon}$-approximation for the Max Clique problem (with the set of vertices V) unless **P** =**NP**. Since in the previous reduction $m = p = |V|$, we get the following corollary:

Corollary 2. *Even for $n = 2$, for any ϵ, there is no $\rho(m,p)$-approximation for any function ρ satisfying $\rho(x,x) = x^{1-\epsilon}$, $\forall x$, for* $\max\max 1$ *and thus for* $\max\sum 1$. *Notice that in particular, $p^{1-\epsilon}$ and $m^{1-\epsilon}$ are not possible, but for example $(pm)^{\frac{1}{2}}$ is not excluded.*

Since $n = 2$, we can also deduce from this reduction the following negative result:

Corollary 3. *There is no $f(n)$-approximation algorithm for any polynomial-time computable function f for* $\max\max 1$ *and thus for* $\max\sum 1$.

To summarize, the main negative results for $\max\sum 1$ are no $p^{1-\epsilon}$-approximation and no $m^{1-\epsilon}$ approximation for $n = 2$, and no $\mathcal{O}(\frac{m}{\ln m})$-approximation for arbitrary n (using the reduction from k-DM of [**?**]). Notice that it does not seem obvious to adapt the previous reductions to provide the same non-approximability results for $\min\sum 0$. Thus, the question of improving the $f(m)$ ratios provided in [**?**] is still open.

3. Positive Results

In this section, we develop a polynomial-time approximation algorithm for $\max\sum 1$. Then, we show that $\max\sum 1$ and $\min\sum 0$ are **FPT** parameterized by p.

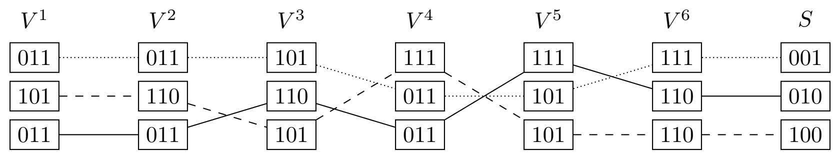

Figure 4: Counter-example showing that Algorithm 1 for $r = 2$ remains a p-approximation. The depicted stacks correspond to an optimal solution of profit 3 whereas the algorithm outputs a solution of profit 1.

3.1. $\frac{p}{r}$-approximation

Given the previous negative results, it seems natural to look for ratio $\frac{p}{r}$, where r is a constant. Let us first see how to achieve a ratio p with Algorithm 1.

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

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

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

Proof. Let $S = S_{\neq 0} \bigcup S_0$ 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 $V'^i = V^i \bigcap S_0$. 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 $w \in V'^{i(l)}$, $w[l] = 0$. In other words, we can say that there is a column of n_2 zeros in the 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 will appear in any solution.

Thus, given S^* an optimal solution, we have $f(S^*) \leq np - n_2 p = n_1 p$. As $f(S) \geq n_1$, we get the desired result. □

Given a fixed integer r (and targeting a ratio $\frac{p}{r}$), a natural way to extend Algorithm 1 is to first look for r-tuples (*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 $(r-1)$-tuples, *etc*. However, even for $r = 2$ this algorithm is not sufficient to get a ratio $\frac{p}{2}$, as shown by the example depicted in Figure 4.

In this example it is not possible to create any stack of value strictly greater than 1 since set V^1 kills positions $\{1, 2\}$ (we say that a set kills positions $\{l_1, l_2\}$ if and only if there is no vector in the set such that $w[l_1] = w[l_2] = 1$), set V^2 kills positions $\{1, 3\}$, and set V^3 kills positions $\{2, 3\}$.

Thus, 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 1. 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 4, the optimal value is 3, and thus the ratio $\frac{p}{2}$ is not verified. In other words, knowing that no stack of profit 2 can be created does not provide better results for Algorithm 1. This motivates the different approach we follow hereafter.

Property 3. *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* $\frac{p}{r}$*-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 run the exact algorithm on these r positions.

Let S^* be an optimal solution for $\max \sum 1$. Let us write $f(S^*) = \sum_{l=1}^{p} a_l$, where $a_l = |\{s \in S^* | v_s[l] = 1\}|$ is the number of stacks in S^* that save position l. $\forall l \in [p-1]_0$, let $X_l = \{l, \dots, (l + r - 1) \ mod \ p\}$, and $\sigma_l = \sum_{t \in X_l} a_t$. Notice that we have $\sum_{l=1}^{p} \sigma_l = r \sum_{l=1}^{p} a_l = r f(S^*)$, as each value a_l appears exactly r times in $\sum_{l=1}^{p} \sigma_l$. This implies $\max_l \sigma_l \geq \frac{r}{p} f(S^*)$.

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 $\frac{p}{r}$-approximation running in $\mathcal{O}(pf(n, m, r))$. □

The previous property motivates the exact resolution of $\max \sum 1$ in polynomial-time for fixed p. It is already proved in [?] that $\min \sum 0$ can be solved in $\mathcal{O}(m(n^{2^p}))$. As this result also applies to $\max \sum 1$, we get a $\frac{p}{r}$-approximation running in $\mathcal{O}(pm(n^{2^r}))$, for any $r \in [p]$. Our objective is now to improve this running time by showing that $\max \sum 1$ (and $\min \sum 0$) are even **FPT** parameterized by p (and not only polynomial for fixed p).

3.2. Faster algorithm for fixed p for $\max \sum 1$

Definition 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* $v = B_t$*.*

First ideas to get an **FPT** *algorithm*

. Let us first recall our previous algorithm in [?] for fixed p. This result is obtained using an integer linear programming formulation of the following form.

The objective function is $\min \sum_{t=0}^{2^p-1} x_t \bar{c}_t$ (recall that in [?] the considered objective function is $\min \sum 0$), where $x_t \in [n]_0$ is an integer variable representing the number of stacks in configuration 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 algorithm of Frank et al. [?] allows us to decide the feasibility of an ILP, under the form $\exists ? x \in \mathbb{Z}^{n_{var}} | Ax \leq b$, in time $\mathcal{O}({n_{var}}^{2.5 n_{var}} L \log L)$, where L is the length of the input. Thus, a classical technique to get an **FPT** algorithm parameterized by p is 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 [?], these constraints are added using z^i_{jt} variables (for $i \in [m], j \in [n], t \in [2^p - 1]_0)$), where $z^i_{jt} = 1$ if and only if v^i_j is assigned to a stack of type t. Nevertheless these new $\mathcal{O}(mn2^p)$ variables prevent us to use [?]. Indeed, the use of these variables leads to a resolution of the ILP formulation in time $\mathcal{O}(((mn+1)2^p)^{2.5(mn+1)2^p} L \log L)$.

Thus, we now come back to the $\max \sum 1$ problem, and our objective is to get rid of these z variables and 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 1 in configuration t.

Definition 2. *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 3. *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 4. *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:

$$\max \sum_{t=0}^{2^p-1} x_t c_t \quad \text{subject to} \quad \begin{array}{l} \sum_{t=0}^{2^p-1} x_t = n \\ \forall 0 \leq t < 2^p,\ x_t \in \mathbb{N} \\ P = \{x_t\} \text{ is feasible} \end{array}$$

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.

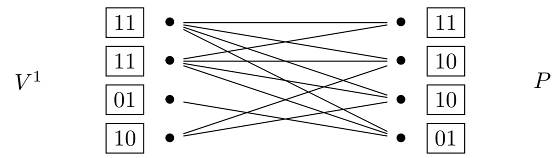

Figure 5: Example showing that satisfying demands of profile P with set 1 requires to find a perfect matching. Edges represent domination between configuration.

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 5. 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 the set V^1, the demand x_2 can be satisfied by using one vector in configuration 2 and one vector in 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 will be introduced in Definition 5. 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 5 (Domination). *A p-dimensional vector v_1 dominates a p-dimensional vector v_2 (denoted by $v_1 \gg v_2$) iff $\forall l \in [p],\ v_2[l] = 1 \Rightarrow v_1[l] = 1$.*

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 $\phi : [n] \to [n]$ 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 6. *For any $i \in [m]$ and any $t \in [2^p - 1]_0$, let b^i_t be the number of vectors of set V^i in configuration t.*

Definition 7 (Graph G^i_P). *Let P be a profile not necessarily feasible. Let $G^i_P = ((\Delta_P, \Lambda^i), E_{\gg})$, where $\Lambda^i = \{\lambda^{i,l}_t, 0 \leq t \leq 2^p - 1, 1 \leq l \leq b^i_t\}$, and $\Delta_P = \{\delta^l_t, 0 \leq t \leq 2^p - 1, 1 \leq l \leq x_t\}$. Let us fix an application $f : \Delta_P \cup \Lambda^i \mapsto [2^p - 1]_0$, that associates to each vertex $\lambda^{i,l}_t$ and to each vertex δ^l_t the vector in configuration t. Λ^i (resp. Δ_P) represents the set of vectors of V^i (resp. the demands of profile P) grouped according to their configurations. Notice that $|\Lambda^i| = |\Delta_P| = n$. Finally, we set $E_{\gg} = \{\{a, b\} | a \in \Delta_P, b \in \Lambda^i, f(a) \ll f(b)\}$.*

We are now ready to show the following proposition.

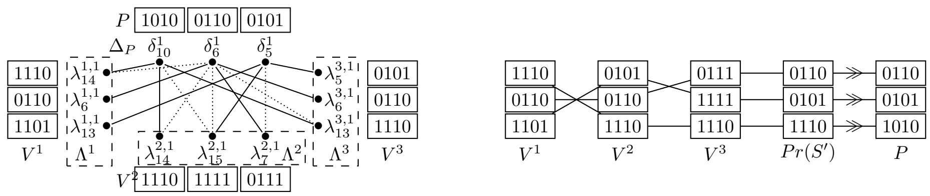

Figure 6: Illustration of Proposition 1 with $m = n = 3$ and $p = 4$. Left: The three G^i_P graphs (edges are depicted by solid and dotted lines), and three matchings (in solid lines) corresponding to S'. Right: Solution S' s.t. $Pr(S') \gg P$.

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

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

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^i_P” 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^i_P, but P is not feasible. This explains the formulation of Proposition 1. An example of the correct formulation is depicted in Figure 6.

Proof. Let P be a profile.

($\Rightarrow$) Let P' be a feasible profile that dominates P. Let $S = \{s_1, \dots, s_n\}$ and $S' = \{s'_1, \dots, s'_n\}$ be two solutions such that S' is feasible, $Pr(S) = P$, $Pr(S') = P'$ (notice that S and P are not necessarily feasible), and $S' \gg S$. Without loss of generality, let us assume that $\forall j$, $v_{s'_j} \gg v_{s_j}$ (*i.e.* the bijection ϕ of Definition 5 is the identity), and let us assume that for any j, $s'_j = (v^1_j, \dots, v^m_j)$. Since $v^i_j \in s'_j$, then for any i, we know that $v^i_j \gg v_{s'_j} \gg v_{s_j}$, $\forall j \in [n]$. This implies a matching of size n in all the graphs G^i_P.

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

W.l.o.g. let us rename $\{\delta_1, \dots, \delta_n\}$ the vertices of Δ_P, and $\{\lambda^i_1, \dots, \lambda^i_n\}$ the vertices of Λ^i such that for any i, $\mathcal{M}^i = \{\{\lambda^i_1, \delta_1\}, \dots, \{\lambda^i_n, \delta_n\}\}$. This implies $f(\lambda^i_1) \gg f(\delta_1)$, $\dots$, $f(\lambda^i_n) \gg f(\delta_n)$. Let us define $S = \{s_1, \dots, s_n\}$, where $\forall j \in [n]$, $s_j = (f(\lambda^1_j), \dots, f(\lambda^m_j))$. Notice that for any j, $s_j \gg f(\delta_j)$, as all the $f(\lambda^i_j) \gg f(\delta_j)$, and combining two vectors $f(\lambda^{i_1}_j) \gg f(\delta_j)$ and $f(\lambda^{i_2}_j) \gg f(\delta_j)$ 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 3 (Hall’s Theorem)**.** *Let $G = ((V^1, V^2), E)$ 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 $\Gamma(\sigma) = \{v_2 \in V^2 | \exists v_1 \in \sigma$ such that $\{v_1, v_2\} \in E\}$.*

Remark 1. *Notice that we cannot use Hall's Theorem directly on graphs G^i_P, 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^i_P.*

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

Proof. ($\Rightarrow$) Let $\sigma_{cfg} = \{t_1, \ldots, t_\alpha\}$. Let $\sigma = \{\delta^l_{t_i}, 1 \leq i \leq \alpha, 1 \leq l \leq x_{t_i}\}$ be the vertices of Δ_P corresponding to the demands in σ_{cfg}. Observe that $\sum_{t\in\sigma_{cfg}} x_t = |\sigma|$. Notice also that $\Gamma(\sigma) = \{\lambda^{i,l}_t, t \in dom(\sigma), 1 \leq l \leq b^i_t\}$ by construction. Thus, $|\sigma| \leq |\Gamma(\sigma)|$ implies $\sum_{t\in\sigma_{cfg}} x_t \leq \sum_{t\in dom(\sigma_{cfg})} b^i_t$.

($\Leftarrow$) Let $\sigma \subseteq \Delta_P$. $\forall t \in [2^p-1]_0$, let $X_t = \{\delta^l_t, 1 \leq l \leq x_t\}$, let $\sigma_t = \sigma \bigcap X_t$. Let $\sigma_{cfg} = \{t_1, \ldots, t_\alpha\} = \{t | \sigma_t \neq \emptyset\}$. 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 $\lambda^{i,l'}_{t'} \in \Gamma(X)$, it means that there is a $t \in \sigma_{cfg}$ such that $\lambda^{i,l'}_{t'} \in \Gamma(X_t)$, and thus there exists l such that $\{\delta^l_t, \lambda^{i,l'}_{t'}\} \in E$ (which implies that $t' \gg t$). As $\sigma_t \neq \emptyset$, there exists l' such that $\delta^{l'}_t \in \sigma_t$, and $\{\delta^{l'}_t, \lambda^{i,l'}_{t'}\} \in E$ as $t' \gg t$.

Finally, the hypothesis with our set σ_{cfg} leads to

$$|\sigma| \leq |X| = \textstyle\sum_{t\in\sigma_{cfg}} x_t \leq \sum_{t\in dom(\sigma_{cfg})} b^i_t = |\Gamma(X)| = |\Gamma(\sigma)|$$

□

Using Propositions 1 and 2, we can now write that for any profile $P = \{x_0, \ldots, x_{2^p-1}\}$:

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

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

$$\begin{aligned} \max \quad & \sum_{t=0}^{2^p-1} x_t c_t \\ \text{subject to} \quad & \forall i \in [m] : \forall \sigma_{cfg} \subseteq [2^p-1]_0, \sum_{t\in\sigma_{cfg}} x_t \leq \sum_{t\in dom(\sigma_{cfg})} b^i_t \\ & \forall t \in [2^p-1]_0, x_t \in \mathbb{N} \end{aligned}$$

This linear program has 2^p variables and $(m2^{2^p} + 2^p)$ constraints. Thus, we can solve it using [**?**] in time $f(p) \times poly(n+m)$ (the $poly(n+m)$ factor comes from the $L \log L$ dependency as stated at the beginning of Section 3.2, with L being the size of the input), we get the following theorem:

Theorem 4. $\max\sum 1$ *and* $\min\sum 0$ *are* **FPT** *when parameterized by* p.

Notice that the objective here is not to optimize the dependence in p, but to highlight a parameter concentrating the hardness of the problem. The dependence of the algorithm in p is huge, the algorithm runs indeed in time $\Omega((2^p)^{2^p})$.

Using Property 3 this ILP leads to the following corollary:

Corollary 4. $\max\sum 1$ *admits a* $\frac{p}{r}$*-approximation algorithm running in time* $f(r)poly(n+m+p)$.

3.3. Additional results

In this section we first provide an **EPTAS** for a variant of $\max\sum 1$ such that the maximum number of zeros per vector is bounded by a constant r. We will denote the latter as $(\max\sum 1)_{\#0\leq r}$. In a second time, we provide an algorithm for $\max_{\neq 0}$ when $n = 2$ based on the resolution of the Maximum Independent Set problem.

Let us first show how the previous **FPT**-algorithm can be used to design an **EPTAS** for $(\max\sum 1)_{\#0\leq r}$.

Theorem 5. *For any fixed* m*,* $(\max\sum 1)_{\#0\leq r}$ *admits an* **EPTAS***.*

Proof. Let us consider a $(\max\sum 1)_{\#0\leq r}$ instance $I[m,n,p]$. Let $k > 1$ be an arbitrary constant. We distinguish between the two following possibilities:

- If $p \leq krm$, the problem is solved optimally using previous ILP based **FPT**-algorithm. Since the latter runs in time $f(p)poly(n+m)$, thus I can be solved in time $f(k)poly(n+m)$.
- If $p > krm$, the profit of every vector $v \in \bigcup_{i=1}^{m} V^i$ satisifies $c\,(v) \geq p - r$. Thus, every possible stacks s satisfies $c\,(s) \geq p - rm > 0$. It follows that any greedy algorithm returns a solution S of profit $c\,(S) \geq n(p - rm)$.

  On the other hand, the profit of every optimal solution S^* is upper bounded by np.

  Since $(\max\sum 1)_{\#0\leq r}$ is a maximization problem, the ratio ρ is defined as:

  $$\rho = \frac{c\,(S^*)}{c\,(S)} \leq \frac{p}{p-rm} = 1 + \frac{rm}{p-rm} < 1 + \frac{rm}{(k-1)rm} = 1 + \frac{1}{k-1}$$

  Therefore every polynomial time algorithm is a $1 + \frac{1}{k-1}$-approximation algorithm.

□

We consider now $\max\max 1$. The latter can be trivially solved in time $\mathcal{O}^*(2^p)$. To achieve this, it is enough to test, for every configuration t, if a feasible stack s in configuration $t_s \gg t$ does exist and, among these feasible stacks, to return the one with the best profit. A natural question arising is to know whether an algorithm with improved running time does exist. We show now that, when considering the special case where $n = 2$, a simple reduction from $\max\max 1$ to INDEPENDENT SET answers positively to this question.

Theorem 6. *Let r be a constant such that there is an algorithm solving* INDEPENDENT SET *in time r^n. Then for $n = 2$, $\max\max 1$ can be solved in time r^p.*

Proof. We show that there is a strict reduction from $\max\max 1$, when $n = 2$, to INDEPENDENT SET.

Let $I[m, n = 2, p]$ be a $\max\max 1$ instance. *W.l.o.g.*, we consider instances I such that two vectors of a same set v_1^i, v_2^i cannot have both the same component l set to zero. Otherwise every solution S would satisfy $v_{s_j}[l] = 0, \forall j = 1, 2$. Thus removing this component from all the vectors of I does not alter the profit of any solution.

A MAX INDEPENDENT SET instance $G = (V, E)$ can be constructed as follows:

- we set $V = [p]$,
- 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 $G = (V, E)$ of size $|IS| = k$.

$\Leftarrow$ Let IS be an independet set in $G = (V, E)$. Thus for each $l \in IS$, we assign to the stack s_1, every vector of any set V^i that satisfies $v_j^i[l] = 0$. If needed, s_1 is completed greedily with vectors of remaining sets.

Note that two vectors of a same set cannot both 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 $l_1, l_2 \in IS$. However, such a pair of components implies an edge in G, thus IS is not an independent set.

The second stack $s_2 \in S$ is such that $\forall l \in IS,\ v_2^s[l] = 1$. Thus $c(S) = \max(c(s_1), c(s_2)) \geq |IS|$.

$\Rightarrow$ Let S be a solution of I of profit $c(S)$. Thus there exists a stack, let us say s_1, such that v_{s_1} contains $c(S)$ component set to one.

If we call $Z = \{l / v_1^{s_1}[l] = 1\}$ the set of component set to one in the representative vector of s_1, we claim that $\forall (l_1, l_2) \in Z^2$, there does not 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 $v_1^s[l_1] = 0, v_1^s[l_2] = 1$ or $v_1^s[l_1] = 1, v_1^s[l_2] = 0$.

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

□

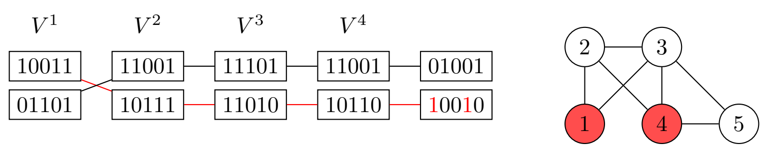

Figure 7: Illustration of the reduction from an instance $I[m = 4, n = 2, p = 5]$ of $\max\max 1$ admitting a solution S of profit $c(S) = 2$ to an instance $G = (\{1,2,3,4,5\}, \{(1,2),(1,3),(2,3),(2,4),(3,4),(3,5),(4,5)\})$ of Independent Set admitting a solution $IS = \{1,4\}$ of profit two.

Theorem 7 ([?])**.** Max-IS *can be solved in* $\mathcal{O}^*(1.2738^{|V|})$*.*

Corollary 5. *For* $n = 2$*,* $\max\max 1$ *can be solved in* $\mathcal{O}^*(1.2738^p)$*.*

4. Conclusion

In this article, we establish that $\max\sum 1$ does not admit any $f(n)$-approximation algorithm and is also $m^{1-\varepsilon}$ and $p^{1-\varepsilon}$ non-approximable for $n = 2$. On the positive side, we provide an **FPT** algorithm for $\max\sum 1$ leading to a $\frac{p}{r}$-approximation algorithm running in $f(r)poly(m+n+p)$, which is the best we can hope for. The existence of an $f(m)$-approximation algorithm for $\max\sum 1$, and even for $\max\max 1$, remains open.