Contents lists available at ScienceDirect

Discrete Optimization

www.elsevier.com/locate/disopt

# 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, France*
[b] *Université de Montpellier, 163 rue Auguste Broussonnet, 34090 Montpellier, France*
[c] *Department of Management Science, Lancaster University Management School, Lancaster, LA1 4YX, UK*

A R T I C L E I N F O

*Article history:*
Received 4 January 2016
Received in revised form 30 June 2016
Accepted 4 July 2016
Available online 29 July 2016

*Keywords:*
Wafer-to-Wafer Integration
Min sum 0
Max sum 1
Approximability
FPT algorithm
Multidimensional binary vector assignment

A B S T R A C T

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.

## 1. Introduction

### 1.1. Problem definition

In this paper we consider Wafer-to-Wafer Integration problems. In these problems, we are given $m$ multisets $V^1, \dots, V^m$, 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 $v^i_j$.

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

* Corresponding author at: LIRMM, 161 rue Ada, 34095 Montpellier Cedex 5, France.
*E-mail 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 $\{j, \dots, n\}$ and to lighten the notation, we will use the classical notation $[n]$ instead of $[n]_1$.

http://dx.doi.org/10.1016/j.disopt.2016.07.001
1572-5286/© 2016 Elsevier B.V. All rights reserved.

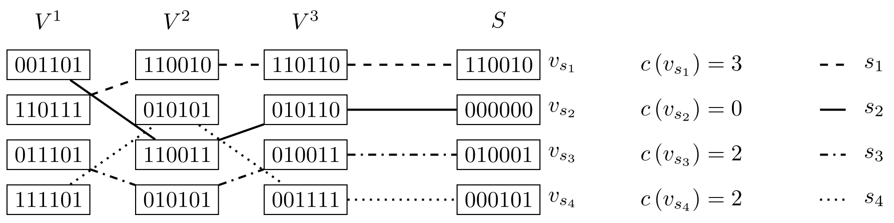

**Fig. 1.** Example of $\max \sum 1$ instance with $m = 3, n = 4, p = 6$ and of a feasible solution $S$ of profit $f_{\Sigma 1}(S) = 7$.

These problems are motivated by an application in IC manufacturing in semiconductor industry, see [1] 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 $u \wedge v = (u[1] \wedge v[1], u[2] \wedge v[2], \dots, u[p] \wedge v[p])$. We associate to any stack $s = (v_1^s, \dots, v_m^s)$ 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 $S = \{s_1, s_2, \dots, s_n\}$ of $n$ disjoint stacks |
| **Objective functions** | $\max \sum 1$: maximize $f_{\sum 1}(S) = \sum_{j=1}^n c\left(v_{s_j}\right)$, the total number of good bits |
| | $\min \sum 0$: minimize $f_{\sum 0}(S) = np - \sum_{j=1}^n c\left(v_{s_j}\right)$, the total number of bad bits |

Instances of these problems will be denoted by $I[m, n, p]$. The notation $f(S)$ (instead of $f_{\sum 0}(S)$, $f_{\sum 1}(S), \dots$) 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 [2,3] for more information. For any $\rho > 1$, a $\rho$-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 parameterized (decision) problem $\Pi$ is a couple $(X, \kappa)$, where $X \subseteq \Sigma^*$ is a classical decision problem, and $\kappa : \Sigma^* \longrightarrow \mathbb{N}$ is a parameterization. Deciding $\Pi$ requires to determine for any instance $I \in \Sigma^*$ if $I \in X$. Finally, we say that an algorithm $A$ decides $\Pi$ in **FPT** time (or that $\Pi$ is **FPT** parameterized by $\kappa$) if and only if there exist a computable function $f$ and a constant $c$ such that for any $I$, $A(I)$ runs in $\mathcal{O}(f(\kappa(I))|I|^c)$.