Cell Flipping Problems on Permutation Diagrams *

Martin Charles Golumbic
Dept. of Computer Science
University of Haifa
Haifa, Israel
golumbic@cs.haifa.ac.il

Haim Kaplan
School of Computer Science
Tel-Aviv University
Tel-Aviv, Israel
haimk@post.tau.ac.il

Elad Verbin
School of Computer Science
Tel-Aviv University
Tel-Aviv, Israel
eladv@post.tau.ac.il

September 21, 2003

Abstract

Permutation diagrams have been used in circuit design to model a set of single point nets crossing a channel, where the minimum number of layers needed to realize the diagram equals the clique number $\omega(G)$ of its permutation graph, the value of which can be calculated in $O(n \log n)$ time. We consider a generalization of this model motivated by "standard cell" technology in which the numbers on each side of the channel are partitioned into consecutive subsequences, or *cells*, each of which can be left unchanged or flipped (i.e., reversed). We ask, for what choice of flippings will the resulting clique number be minimum or maximum. We show that when one side of the channel is fixed (no flipping), an optimal flipping for the other side can be found in $O(n \log n)$ time for the maximum clique number, and that when both sides are free this can be solved in $O(n^2)$ time. We also prove NP-completeness of finding a flipping that gives a minimum clique number, even when one side of the channel is fixed, and even when the size of the cells is restricted to be less than a small constant. Moreover, since the complement of a permutation graph is also a permutation graph, the same complexity results hold for the stable set (independence) number. In the process of the NP-Completeness proof we also prove

*A preliminary version of this paper has appeared in the Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science (STACS), 1998 [7]. In this version we also settle the main open problem from [7], see Section 4.

NP-Completeness of a restricted variant of a scheduling problem. This new NP-Completeness result may be of independent interest.

Keywords: Permutation graphs, dynamic programming, cell flipping, VLSI layout, clique number, stable set number, independent set number, multiprocessor scheduling.

1 Introduction

1.1 Background

A *permutation diagram* D consists of two horizontal lines L_1 and L_2 each having n distinguished points labeled by a permutation of the numbers $1, 2, \ldots, n$ and n straight line segments connecting i on L_1 with i on L_2. We call each of these points a *position* and its label is called the *contents* of the position. The labels of L_1 are denoted by i, j, and z, etc. The labels of L_2 are denoted by i', j', and z', etc.

The *permutation graph* $G(D) = (V, E)$ of diagram D is an undirected graph with vertices $V = \{1, 2, \ldots, n\}$ and edges $(i, j) \in E$ if and only if segment $i - i'$ intersects segment $j - j'$ in D. See Figure 1.

We may also observe that the number of edges $e = |E|$ equals the number of intersections of segments in D. Further background on permutation graphs can be found in Golumbic [6].

Permutation diagrams have been used in a number of application areas, including in circuit design to model single point nets crossing a channel. The clique number $\omega(G)$ of the permutation graph $G = G(D)$ is the size of the largest complete subgraph. We also define the clique number of a diagram D to be equal to $\omega(G(D))$. The clique number of a permutation diagram equals the minimum number of layers on which the diagram D can be realized (the segments are partitioned into layers where segments on the same layer may not intersect). For a given diagram, $\omega(G)$ can be calculated in $O(n \log n)$ time [4] (see also [6]). We review this algorithm in Section 3.

1.2 Problem definition and results

In this paper, we consider a generalization of the permutation diagram model in which the numbers on each line L_k are partitioned into consecutive subsequences $c_1, \ldots, c_r$, $c'_1, \ldots, c'_{r'}$, called *cells*, each

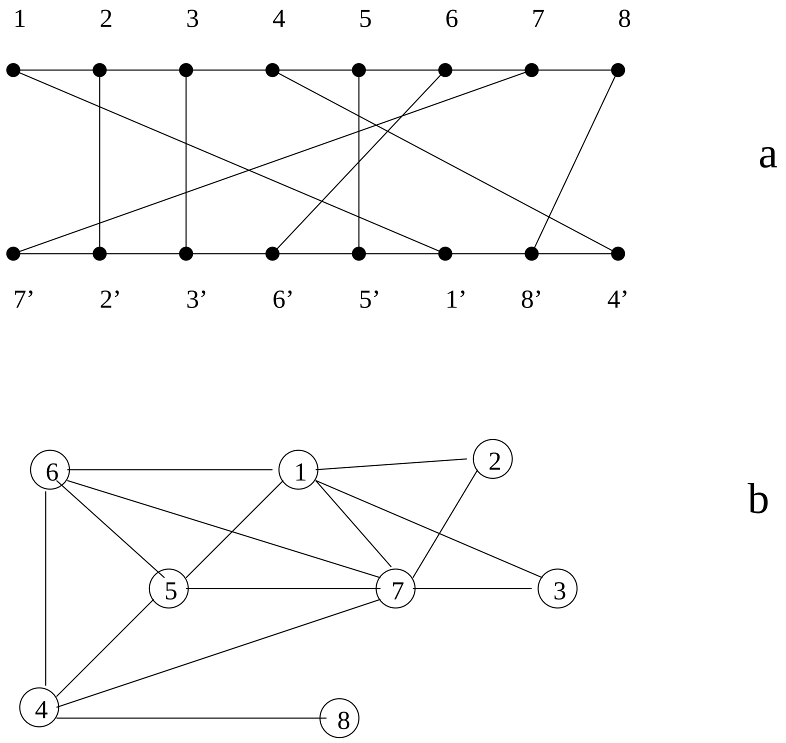

Figure 1: (a) Permutation Diagram D. (b) The Permutation Graph $G(D)$.

of which can be left unchanged or flipped (i.e., reversed). Each choice σ of which cells to flip will yield a different permutation diagram D_σ with its corresponding permutation graph $G_\sigma = G(D_\sigma)$ and clique number $\omega(G_\sigma)$. For a position s, denote by $c(s)$ and $c'(s)$ the cells containing position s on L_1 and L_2, respectively. Note that the contents of position s on L_1 in D_σ may vary between two possible values according to whether σ flips $c(s)$ or not; similarly for the contents of each position s on L_2. If $c(s) = c_l$ and $c(t) = c_k$ we denote $c(s) < c(t)$ if $l < k$, similarly for c'. We raise the question of finding the following two quantities over all possible choices σ of flippings for a given partitioned permutation diagram $\mathcal{D} = < D, \{c_i\}, \{c'_j\} >$:

$$
\begin{aligned}
MAXFLIP(\mathcal{D}) &= \max_{\sigma} \omega(G_\sigma) \\
MINFLIP(\mathcal{D}) &= \min_{\sigma} \omega(G_\sigma)
\end{aligned}
$$

The MAXFLIP and MINFLIP problems are to find $MAXFLIP(\mathcal{D})$ and $MINFLIP(\mathcal{D})$, respectively, given a partitioned permutation diagram $\mathcal{D}$.

Example: Let the labels on L_1 be $[3,4,7],[2,6],[1,5,8]$ where the brackets indicate cells, and let the labels on L_2 be $[6',2',5'],[4',1',7',8',3']$. Position 6 on L_1 has contents 1 if $c(6)$ is not flipped, and contents 8 if $c(6)$ is flipped. The clique number is 4 with no flipping but is reduced to 3 if we flip $[2,6]$, or increased to 5 if we flip $[2,6]$, $[6',2',5']$ and $[4',1',7',8',3']$.

If there are $k = r + r'$ cells, then both MAXFLIP and MINFLIP can be solved in an exhaustive manner by calculating the clique number of all 2^k possible choices. For small values of k (say $k < 5$) this may be reasonable, but for larger values (say $k > 10$) this may be unreasonable. We study the computational complexity of these problems independent of the number k. We give an $O(n^2)$ algorithm for MAXFLIP and prove that MINFLIP is NP-complete. We also show that for the special case of flipping the cells only on one line, MAXFLIP can be solved in $O(n \log n)$, but MINFLIP is still NP-Complete. Our algorithms for MAXFLIP can also produce a flipping σ for which the maximum is achieved within the same time bounds.

Note that the same complexity results immediately hold for the analogous independence (or stability) number problems on flipping a partitioned permutation diagram. This follows since the complement of a permutation graph is also a permutation graph (see [6]).

We mention a very simple case which can be solved in an efficient greedy manner, namely, when each cell is of size at most two. In this case, each cell $[i, j]$ on L_1 is checked once: if segments

$i - i'$ and $j - j'$ do not cross, then leave $[i, j]$ unchanged for MINFLIP and flip it for MAXFLIP. Do the same for each cell $[i', j']$ on L_2. The effect of each such flip decision is local, it either adds the edge (i, j) to the permutation graph or deletes it without changing any other edges. Moreover, this represents *all* the degrees of freedom in the problem. Since adding edges can only increase the clique size and deleting edges can only decrease the clique size, the following result holds.

Proposition 1.1 *If each cell is of size at most two, then MAXFLIP and MINFLIP can be solved in $O(n)$ time.*

We prove, however, that when the cells are limited to be of size ≤ 5, MINFLIP (and even one-sided MINFLIP) is already NP-complete. The status for MINFLIP when restricted to cells of size ≤ 4, or even ≤ 3, is still unresolved.

1.3 Motivation and Applications

In the computer aided design of VLSI circuits using "standard cell" technology, a stage is reached where cell placement on horizontal rows has already been performed, and the only remaining degree of freedom is replacing some of the cells with their "mirror image" with respect to a vertical axis, i.e., cell flipping. Every problem considered for a fixed channel can also be studied in its cell flipping versions. For example, minimum width channel routing in the jog-free Manhattan model [9] is NP-hard even without cell flipping, so its cell flipping versions would also be NP-hard. But the minimum channel density problem is $O(p)$ for a fixed channel with p pins, and its flipping version can be solved in $O(p \log m)$ time, where m is the number of "nets" [1]. (Pins on both sides of the channel are partitioned into disjoints sets called *nets* such that each net is to connected together.)

In combinatorics, the problem of finding the longest decreasing or longest increasing subsequence of a permutation has long been of interest [4, 6]. The one-side cell flipping problems can be restated as follows: Given a permuation of the numbers $1, \ldots, n$ partitioned into cells, find the cell flipping which minimizes/maximizes the longest decreasing/increasing subsequence.

2 An $O(n^2)$ Dynamic Programming Algorithm for MAXFLIP

We start out describing a simple quadratic algorithm that computes the clique number of a (unpartitioned) permutation diagram. Later on we show how to generalize this algorithm to the partitioned case.

Let s be a position on L_1, t a position on L_2, and $tr(s,t)$ the trapezoid defined by positions s and n on L_1 and 1 and t on L_2 (see Figure 2). Define $C(s,t)$ to be the clique number of the permutation diagram induced by line segments with both endpoints in $tr(s,t)$. The algorithm computes $C(s,t)$ for every s and t in increasing order of t. Initially, for every $1 \leq s \leq n$, $C(s,1) = 1$ if $tr(s,1)$ contains a line segment and $C(s,1) = 0$ otherwise. Assume $C(s,l)$ have already been computed for every $l < t$ and every $1 \leq s \leq n$. Let g be the line segment i—i' incident with position t in L_2, (i.e., i' is the label or contents of position t in L_2), and let k be the position labeled i in L_1, (i.e., where g meets L_1). Then $C(s,t) = C(s,t-1)$ if $k < s$; otherwise, $C(s,t)$ is the maximum among $C(s,t-1)$ and $C(k,t-1)+1$.

Now, keeping the algorithm above in mind, we turn back to the partitioned case. Observe that when we fix the directions of the cells $c(s)$ and $c'(t)$, the set of line segments with both endpoints within $tr(s,t)$ is determined. We denote by a pair $\delta = (\delta_1, \delta_2)$, where $\delta_1, \delta_2 \in \{0,1\}$, a direction assignment to $c(s)$ and $c'(t)$. The assignment δ flips $c(s)$ if and only if $\delta_1 = 1$, and flips $c'(t)$ if and only if $\delta_2 = 1$. Given a direction assignment δ we denote by $L(s,t,\delta)$ the set of segments contained in $tr(s,t)$. Let σ be a flipping of the cells $c(s) < c \leq c(n)$ and $c'(1) \leq c' < c'(t)$. For each σ, $L(s,t,\delta)$ induces a permutation diagram obtained by flipping $c(s)$ and $c'(t)$ according to δ and flipping cells $c(s) < c \leq c(n)$, and $c'(1) \leq c' < c'(t)$ according to σ. We denote by $C(s,t,\delta)$ the maximum over all possible σ's of the clique number of the permutation diagram induced by $L(s,t,\delta)$.

Our dynamic programming algorithm computes $C(s,t,\delta)$ for every possible s, t, and δ. The computation proceeds in increasing order of t, the lower right endpoint of the trapezoid. The details are as follows.

For every $1 \leq s \leq n$ we initialize $C(s,1,\delta) = 1$ if $L(s,1,\delta)$ contains one line segment and $C(s,1,\delta) = 0$ otherwise. Fix s, $\delta = (\delta_1, \delta_2)$, and assume that we have already computed $C(s,l,\delta)$ for every $l < t$ and every $1 \leq s \leq n$. To extend the assignment, we calculate two numbers A and

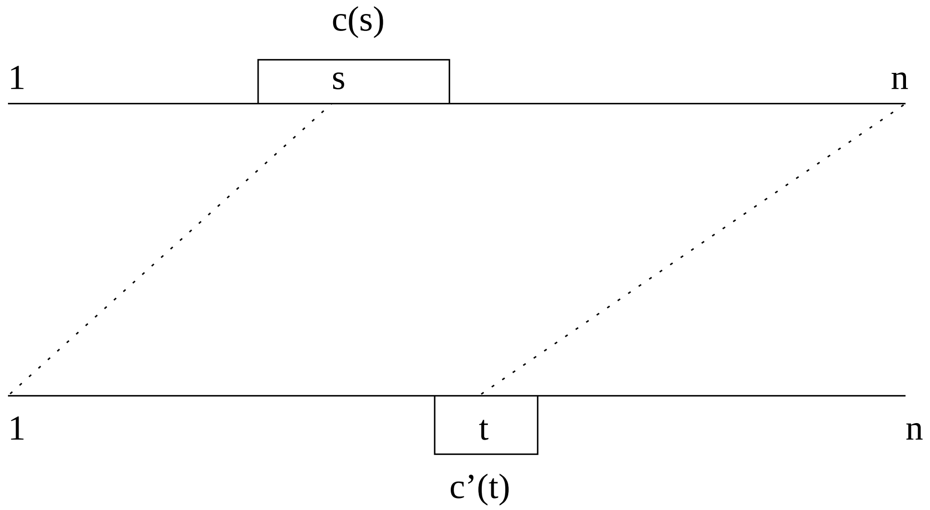

Figure 2: The trapezoid $tr(s,t)$ defined by a position s on L_1 and position t on L_2.

B, eventually setting $C(s,t,\delta)$ to be the larger of A and B.

If $c'(t-1) = c'(t)$ let $A = C(s,t-1,\delta)$. Otherwise, $c'(t-1)$ precedes $c'(t)$, and we let A be the maximum among $C(s,t-1,\delta')$ and $C(s,t-1,\delta'')$ where $\delta' = (\delta_1, 1)$ and $\delta'' = (\delta_1, 0)$; i.e., δ' flips $c'(t-1)$ and δ'' does not flip $c'(t-1)$, and both δ' and δ'' give the same direction to $c(s)$ as δ.

Let g be the line segment incident with position t on L_2 when $c'(t)$ is directed as specified by δ. We denote by $c(g)$ the cell that contains the L_1-endpoint of g.

<u>Case 0</u>: If g is not in $L(s,t,\delta)$ then we can finish setting $C(s,t,\delta) = A$. Otherwise, the line segment g is in $L(s,t,\delta)$. Note that since g is in $L(s,t,\delta)$, $c(s) \leq c(g)$.

<u>Case 1</u>: The cell $c(g)$ is different from $c(s)$, i.e., $c(s) < c(g)$.

Let k_1 be the position of the L_1-endpoint of g when $c(g)$ is flipped, and let k_0 be the position of the L_1-endpoint of g when $c(g)$ is not flipped. If $c'(t-1) = c'(t)$ then assign B to one plus the maximum among $C(k_1, t-1, \delta')$ and $C(k_0, t-1, \delta'')$ where $\delta' = (1, \delta_2)$ and $\delta'' = (0, \delta_2)$. If $c'(t-1) < c'(t)$ then assign B to be one plus the maximum among $C(k_1, t-1, (1,0))$, $C(k_1, t-1, (1,1))$, $C(k_0, t-1, (0,0))$, $C(k_0, t-1, (0,1))$. Now assign $C(s,t,\delta) = \max\{A, B\}$.

<u>Case 2</u>: Cell $c(g)$ is the same as $c(s)$.

We perform a computation similar to the above, the difference stems from the fact that the direction of $c(s)$ is now determined by δ. Let k be the position of the L_1-endpoint of g when the direction of $c(g)$ is fixed as defined by δ. Note that $s < k$ since we already handled Case 0. If $c'(t-1) = c'(t)$

then assign B to be one plus $C(k, t-1, \delta)$, and if $c'(t-1) \neq c'(t)$ then assign B to be one plus the maximum among $C(k, t-1, (\delta_1, 0))$, $C(k, t-1, (\delta_1, 1))$. Finally, set $C(s, t, \delta) = \max\{A, B\}$.

It is straightforward to prove by induction on t that $C(s, t, \delta)$ are computed correctly. Computation of every entry in the table C takes $O(1)$ time and since the size of C is $O(n^2)$ our algorithm runs in $O(n^2)$ time. The space complexity is $O(n)$ since we need to store the values in C only for two consecutive values of t at any one time.

The maximum of the four entries $C(1, n, (0,0))$, $C(1, n, (0,1))$, $C(1, n, (1,0))$, and $C(1, n, (1,1))$ is the answer to MAXFLIP. Therefore, we have proved the following theorem.

Theorem 2.1 *There exists an algorithm for MAXFLIP that runs in $O(n^2)$ time using $O(n)$ space.*

■

By maintaining with each entry $C(s, t, \delta)$ a flipping for which the permutation diagram of $L(s, t, \delta)$ has clique number $C(s, t, \delta)$ our algorithm can be adapted to produce a flipping for which the optimum MAXFLIP value is obtained.

3 $O(n \log n)$ Algorithm for One Side MAXFLIP

When only L_1 is partitioned into cells that can be flipped, one may observe that without loss of generality, we may rename the positions on L_2 to be labeled in increasing order $1', \ldots, n'$. The problem MAXFLIP is then equivalent to finding a flipping σ such that the length of the maximum decreasing subsequence of the integer sequence induced by σ on L_1 is maximized. In this section we show how to solve MAXFLIP by extending the algorithm for finding a maximum decreasing subsequence (or equivalently finding the clique number of a permutation diagram) described by Fredman [4]. The running time of the flipping algorithm that we obtain is of the same order as the ordinary running time of the algorithm for maximum decreasing subsequence, that is $O(n \log n)$. We start by reviewing the algorithm in [4], and subsequently show how to use it for MAXFLIP.

We find the maximum length of a decreasing subsequence of $a_1, \ldots, a_n$ by processing the a_i's in order while maintaining an array T of length n. Initially $T(j) = -\infty$ for every $1 \leq j \leq n$, and after processing a_i, $T(j)$ is the largest number that ends a decreasing subsequence of length j in $a_1, \ldots, a_i$. While processing a_i, $1 \leq i \leq n$, we search for the largest length $0 < l < n$ such that $T(l) > a_i$ (assume $l = 0$ if $T(j) < a_i$ for every $1 \leq j \leq n$). If $T(l+1) < a_i$ then we assign $T(l+1) = a_i$. Since we can search for l using binary search in $O(\log n)$ time, the running time of this algorithm is $O(n \log n)$. It is straightforward to modify this algorithm such that it also produces a decreasing sequence of maximum length.

Our algorithm for MAXFLIP will use the algorithm for maximum decreasing subsequence as a procedure that we denote by MDSS. We run MDSS on the sequences defined by the individual cells, and each time we use it, we supply it with an initialized array to work on. In contrast with the regular MDSS algorithm, the array is not initialized to $-\infty$ but rather with values based on computations performed on previous cells. Let the cells on L_1 be $c_1, \ldots, c_k$ from left to right; assume c_i also denotes the integer sequence defined by cell c_i when it is left unchanged and $\overline{c_i}$ denotes the sequence defined by c_i when it is flipped. An application of MDSS with an array T on integer sequence A is denoted by $MDSS(T, A)$.

Define $F(i, j)$ to be the maximum integer that terminates a decreasing subsequence of length j of an integer sequence defined by some flipping of $c_1, \ldots, c_i$. Let T and T_f be two arrays that we use to communicate input and output with MDSS, initialized with all entries equal to $-\infty$. We start out computing $F(1, j)$ by running $MDSS(T, c_1)$ then $MDSS(T_f, \overline{c_1})$, and setting $F(1, j) = \max\{T(j), T_f(j)\}$ for every $1 \leq j \leq |c_1|$. Assume we have computed $F(i-1, j)$ for some $i < k$ and every j; we compute $F(i, j)$ as follows. We initialize $T(j) = T_f(j) = F(i-1, j)$, and then run $MDSS(T, c_i)$ and $MDSS(T_f, \overline{c_i})$. Finally, we set $F(i, j) = \max\{T(j), T_f(j)\}$ for every j.

Example: Let the input sequence be $[8, 5], [3, 11, 7, 1, 4], [6, 10, 9, 2]$ where the cells are indicated by brackets. While describing our algorithm on this input, we assume that all tables are initialized with $-\infty$ and that each entry not explicitly referred to remains $-\infty$. After executing $MDSS(T, c_1)$ and $MDSS(T_f, \overline{c_1})$, $T(1) = 8$, $T(2) = 5$, and $T_f(1) = 8$. Taking the maximum, we obtain $F(1, 1) = 8$ and $F(1, 2) = 5$. The second iteration starts by setting $T(1) = T_f(1) = F(1, 1)$, and $T(2) = T_f(2) = F(1, 2)$. After executing $MDSS(T, c_2)$ and $MDSS(T_f, \overline{c_2})$, $T(1) = 11$, $T(2) = 7$, $T(3) = 4$, $T(4) = 1$, and $T_f(1) = 11$, $T_f(2) = 7$, $T_f(3) = 4$, $T_f(4) = 3$. Taking maxima, we obtain $F(2, 1) = 11$ and $F(2, 2) = 7$, $F(2, 3) = 4$, and $F(2, 4) = 3$. Similarly, the reader can check that after the last iteration $F(3, 1) = 11$, $F(3, 2) = 10$, $F(3, 3) = 9$, $F(3, 4) = 3$, and $F(3, 5) = 2$. We conclude that the length of a maximum decreasing subsequence of any sequence that can be obtained by flipping some of the cells c_1, c_2, c_3 is 5. Indeed if we do not flip c_1, flip c_2, and either flip or do not flip c_3 we obtain a sequence that contains $8, 5, 4, 3, 2$ as a subsequence. ■

It is straightforward to prove by induction on i that the table $F(i, j)$ is computed correctly. To achieve $O(n \log n)$ running time we implement MDSS such that instead of destructively modifying its input array, it returns a list of the array entries changed together with the final value of each. Note that the length of the output list is bounded by the length of the sequence MDSS is applied to. The maximum computation that follows each pair of calls to MDSS is then carried out only for indices of changed entries. This implementation ensures that for each element in the original sequence, we perform at most two binary searches and two maximum operations, so the $O(n \log n)$ upper bound on the running time follows.

4 One Sided MINFLIP is NP-complete

In this section, we give a reduction to One Sided MINFLIP from a new scheduling-related problem, MIN-MAX TWO-CHOICE VECTOR-SUM. We also show that MIN-MAX TWO-CHOICE VECTOR-SUM is (strongly) NP-Complete by a reduction from SET PACKING, thus proving that One Sided MINFLIP is NP-Complete.

Definition 4.1 MIN-MAX TWO-CHOICE VECTOR-SUM (MIN-MAX-TCVS)

bf Input: A number d, the *dimension*, and n pairs of vectors $(v^1, \hat{v}^1), \ldots, (v^n, \hat{v}^n)$ from Z_+^d (where $Z_+ = \{0, 1, 2, \ldots\}$), and a number Z, the *target*. **Solution:** A set of n vectors, exactly one vector from each of the n pairs, such that the maximum coordinate of the sum of the chosen vectors is at most Z, the target.

In the optimization version of this problem there is no Z and we simply want to minimize the maximum coordinate in the sum of the n vectors.

For $d = 2$, MIN-MAX-TCVS is equivalent to 2-Machine MULTIPROCESSOR SCHEDULING [5, page 238, problem SS8]. MIN-MAX-TCVS can indeed be considered a scheduling problem: Say you want to produce n unique items on a set of d distinguished machines. (All machines may have to participate in the production of an item.) For each item we have two alternatives of how to produce it on the machines, each having a different "load vector", specifying the time each machine has to dedicate in order to complete it. You want to finish all items and ship them out by a deadline Z or as early as possible – this is exactly MIN-MAX-TCVS.

Like many other scheduling problems MIN-MAX-TCVS becomes weakly NP-Complete (that is, it has a pseudo-polynomial algorithm) when d is fixed, instead of being part of the input. In fact, it is similar to the weakly NP-Complete problem *PARTITION* for $d = 2$ (but is not similar to *3-PARTITION* for $d = 3$). For $d = 1$ the problem is trivially solvable. MIN-MAX-TCVS can also be classified as a multiple-choice problem, much like MULTIPLE-CHOICE KNAPSACK [11], MULTIPLE-CHOICE MATCHING [8], or NEIGHBORHOOD TSP.

Next we show that MIN-MAX-TCVS is strongly NP-Complete (for the definition of strong NP-Completeness and for other relevant definitions see [5]):

Theorem 4.2 *MIN-MAX-TCVS is strongly NP-Complete (that is, it is NP-Complete even if all numbers in the input are represented in unary).*

Furthermore, consider the restricted version of MIN-MAX-TCVS in which for every pair of vectors v and $\hat{v}$, v has a 1 in a single coordinate and 0 in all others, and $\hat{v}$ has a single coordinate which is at most 2 and has 0 in all others. This restricted version is also NP-Complete.

Proof. Let us first prove that general MIN-MAX-TCVS is NP-Complete:

We reduce SET PACKING to MIN-MAX-TCVS. *SET PACKING* gets as input a set $S = \{x_1, \ldots, x_s\}$ and a family of subsets of S, $F = \{S_1, S_2, \ldots, S_m\}$, and also a number $K \leq m$. The objective is to determine if there is a collection of at least K disjoint members of F. It is shown to be NP-Complete in [5, page 221, problem *SP3*].

We transform an instance of SET PACKING to an instance of MIN-MAX-TCVS as follows. We pick the dimension to be the number of elements in the collection S plus 1, that is $d = s + 1$. We generate a pair of vectors for each member S_i of F, as follows. For every $1 \leq c \leq s$, the first vector of the pair corresponding to S_i contains $m - K$ in coordinate c if $x_c \in S_i$ and 0 otherwise. Also, the first vector contains 0 in coordinate d. The second vector contains 1 in coordinate d and 0 in all other coordinates. We set Z to be $m - K$.

It is quite easy to verify that there is a solution to the SET PACKING instance if and only if there exists a choice of vectors such that the maximal coordinate of their sum is at most $Z = m - K$, and thus MIN-MAX-TCVS is indeed strongly NP-Complete.

Now we prove that the restricted version of MIN-MAX-TCVS is NP-Complete:

We reduce from a restriction of SET PACKING in which the cardinality of each set in the family $F = \{S_1, S_2, \ldots, S_m\}$ is exactly 3. This restriction of SET PACKING remains NP-Complete, as noted in [5, page 238, problem SS8]. As before, $S = \{x_1, \ldots, x_s\}$ denotes the elements in the universe, and K denotes the number of disjoint sets we look for.

We build an instance of MIN-MAX-TCVS where there is a coordinate for each element, two coordinates for each set, and another *free* coordinate. Thus $d = s + 2m + 1$. The upper bound on the sum in each coordinate is $Z = m - K + 2$.

Let j be either an element-coordinate or a set-coordinate. We have $Z - 2$ pairs of vectors v and $\hat{v}$ where v and $\hat{v}$ are equal and contain 1 in coordinate j and 0 in all other coordinates. For the free coordinate we have two pairs of vectors v and $\hat{v}$ where v and $\hat{v}$ are equal and contain 1 only in the free coordinate and 0 in all other coordinates. These pairs contribute $Z - 2$ to each element-coordinate and to each set-coordinate, and 2 to the free coordinate. So other pairs which we define below are allowed to contribute at most 2 to each element-coordinate and set-coordinate, and at most $Z - 2$ to the free coordinate.

For each set, $S = \{x, y, z\} \in F$, in the SET PACKING instance, we have 5 pairs of vectors $(v_1, \hat{v}_1)$, ..., $(v_5, \hat{v}_5)$, in the MIN-MAX-TCVS instance. These are depicted in Figure 3.

- The vector v_1 contains 1 in the free coordinate, and $\hat{v}_1$ contains 2 in the first coordinate of S.
- The vector v_2 contains 1 in the first coordinate of S, and $\hat{v}_2$ contains 2 in the second coordinate of S.
- The vector v_3 contains 1 in the first coordinate of S, and $\hat{v}_3$ contains 2 in the coordinate of x.
- The vector v_4 contains 1 in the second coordinate of S, and $\hat{v}_4$ contains 2 in the coordinate of y.
- The vector v_5 contains 1 in the second coordinate of S, and $\hat{v}_5$ contains 2 in the coordinate of z.

Given a solution to the SET PACKING instance we can define a solution to MIN-MAX-TCVS by picking $\hat{v}_1$, $\hat{v}_2$, $\hat{v}_3$, $\hat{v}_4$, and $\hat{v}_5$ of the pairs corresponding to S, if S is in the packing, and picking

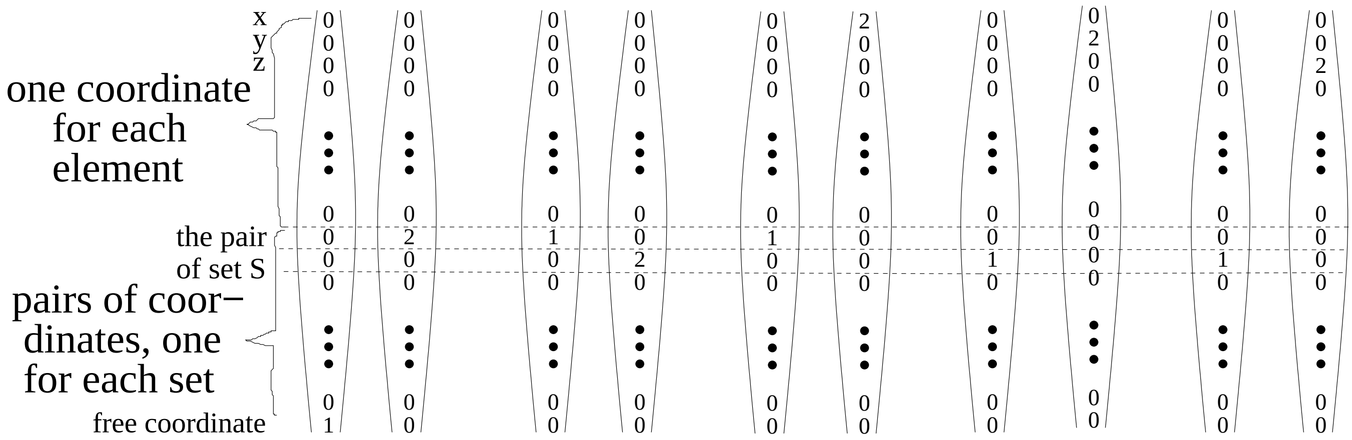

Figure 3: The pairs of vectors that correspond to one set, S, in the SET PACKING instance

v_1, v_2, v_3, v_4, and v_5 otherwise.

To prove the converse, assume that there is a solution to the MIN-MAX-TCVS instance. Consider the 5 pairs of vectors corresponding to one particular set. It is easy to see that if $\hat{v}_1$ is in the solution then this solution must also contain $\hat{v}_2$, $\hat{v}_3$, $\hat{v}_4$, and $\hat{v}_5$. Otherwise, the solution contains v_1 and we may assume that it also contains v_2, v_3, v_4, and v_5. (If it does not, then we can replace the vectors in the corresponding pair, and get another feasible solution). So we can generate a solution to SET PACKING by picking those sets which correspond to pairs for which the MIN-MAX-TCVS solution contains $\hat{v}_1$. From our definition of the MIN-MAX-TCVS instance, it follows that there must be K such sets. $\square$

The special property of MIN-MAX-TCVS is that it is strongly NP-Complete. In contrast, MULTIPROCESSOR SCHEDULING, even if the number of machines is part of the input, is only weakly NP-Complete (i.e., it has a pseudo-polynomial algorithm).

The restricted variant of MIN-MAX-TCVS discussed in Theorem 4.2 is a special case of the problem MULTIPROCESSOR SCHEDULING where you are given d unrelated machines, and all processing times are taken from the set $\{1, 2, \infty\}$. This problem constitutes the one missing case in Lenstra et al.'s [10, section 5] classification of polynomial vs. NP-Complete variants of MULTIPROCESSOR SCHEDULING with restricted processing times. By slightly modifying the proof of Theorem 4.2, we can show that the problem MULTIPROCESSOR SCHEDULING with processing times $\{1, 2, \infty\}$, with the added restriction that for any given job exactly one machine can complete it in one time unit and exactly one machine can complete it in two time units and no other machine can complete it in finite time, is NP-Complete. This result might be of independent interest, so we state it as a theorem:

Theorem 4.3 *MULTIPROCESSOR SCHEDULING with processing times* $p_{ij} \in \{1, 2, \infty\}$ *with the additional constraint that for each* i, $|\{j|p_{ij} = 1\}| = 1$, $|\{j|p_{ij} = 2\}| = 1$ *is NP-Hard.*

As for other restrictions of MIN-MAX-TCVS, if all vectors are drawn from $\{0, 1\}^d$, and furthermore when in each pair of vectors one vector has exactly one entry equal to 1 and all other entries equal to zero, and the other vector has exactly two entries equal to 1 and all other entries equal to zero the problem can also be shown to be NP-Complete. However, if we restrict all vectors to have 1 in a single coordinate and zero in all other coordinates then the problem is equivalent to scheduling unit jobs on d parallel machines where each job has exactly two machines it can run on, and we want to minimize the maximum completion time (the *makespan*). A generalization of this problem where each job has a list of machines it can run on was shown to be polynomial in [2, 3, 10].

From Proposition 1.1, we know that One-Sided MINFLIP is easily solvable in linear time if all cells are of size at most 2. In the next theorem, we show that One-Sided MINFLIP is NP-Complete, even if we only allow the cells to be of size at most a small constant, namely 5. This means that even relatively "natural" instances of MINFLIP are hard to solve.

Theorem 4.4 *One-Sided MINFLIP is NP-Complete.*

Furthermore, it is NP-Complete even when cells are restricted to be of size ≤ 5.

Proof. We will first prove that One-Sided MINFLIP with unbounded cell size is NP-Complete, by a reduction from MIN-MAX-TCVS. Then we show, by slightly modifying the proof and by reducing from the restricted variant of MIN-MAX-TCVS, that we get cells of size ≤ 5. First note that MIN-MAX-TCVS stays strongly NP-Complete even if all elements of the vectors must be at least 1. (We can simply add 1 to all coordinates and add n to Z.) We reduce MIN-MAX-TCVS with elements ≥ 1 to One-Sided MINFLIP as follows.

Given an instance of MIN-MAX-TCVS with all elements at least 1, we generate an instance of One-Sided MINFLIP. Let us denote the n pairs of vectors in the MIN-MAX-TCVS instance as $(v^1, \hat{v}^1), \ldots, (v^n, \hat{v}^n)$. The size of our MINFLIP instance is polynomial in the sum of all elements of all vectors. This proves that MIN-MAX-TCVS is NP-Complete since by Theorem 4.2, MIN-MAX-TCVS is *strongly* NP-Complete. The fixed side of the permutation diagram in the One-Sided MINFLIP instance will be the identity permutation, so we only need to specify the permutation on the other side and its decomposition into cells. This permutation consists of three sections, which we call: a *prefix*, a *middle*, and a *suffix*. The middle consists of n cells each corresponding to a pair of vectors in the MIN-MAX-TCVS instance. The size of the cell which corresponds to the i^{th} pair of vectors, $(v^i, \hat{v}^i)$ is equal to $\sum_{j=1}^{d} v_j^i + \sum_{j=1}^{d} \hat{v}_j^i - d$. Let M denote the size of the middle, i.e., the sum of the sizes of the cells in the middle. Thus, $M = \sum_{i=1}^{n}(\sum_{j=1}^{d} v_j^i + \sum_{j=1}^{d} \hat{v}_j^i) - nd$. The prefix and the suffix consist of cells of size 1 (that is, they cannot be manipulated by flippings). The number of elements in the prefix is $\frac{d(d+1)L}{2}$, and so is the number of elements in the suffix, where $L = M + 1$. In total, our permutation is of the integers $1, \ldots, d(d+1)L + M$.

We now describe how to fill the cells with numbers. We partition the space of integers $1, \ldots, d(d+1)L + M$ into d consecutive intervals: $I_1 = [x_1, y_1]$ $(x_1 = 1)$, $I_2 = [x_2, y_2]$ $(x_2 = y_1 + 1)$, $\ldots$, $I_d = [x_d, y_d]$ $(x_d = y_{d-1} + 1$, $y_d = d(d+1)L + M)$, where the length of I_j is $|I_j| = (d+1) \cdot L + \sum_{i=1}^{n}(v_j^i + \hat{v}_j^i) - n$.

The prefix consists of the following d parts. The first part is a decreasing sequence of the $d \cdot L$ largest elements of the interval I_1 each in a singleton cell. The second part is a decreasing sequence of the $(d-1) \cdot L$ largest elements of interval I_2 each in a singleton cell. In general, the j^{th} part, $1 \leq j \leq d$, is a decreasing sequence of the $(d-j+1) \cdot L$ largest items of I_j each in a singleton cell.

The suffix also consists of d parts and is the mirror image of the prefix. In general, the j^{th} part is a decreasing sequence of the jL smallest elements of interval I_j each in a singleton cell. The construction of the prefix and suffix is demonstrated in Figure 4.

Recall that a clique in D corresponds to a set of pairwise intersecting lines in the drawing. The prefix and suffix are not affected by flippings, so no matter what the flipping is, this part of the construction remains. Thus, our construction of the prefix and suffix will force the maximum decreasing subsequence to be contained in a single interval I_j. (This claim will be shown later in

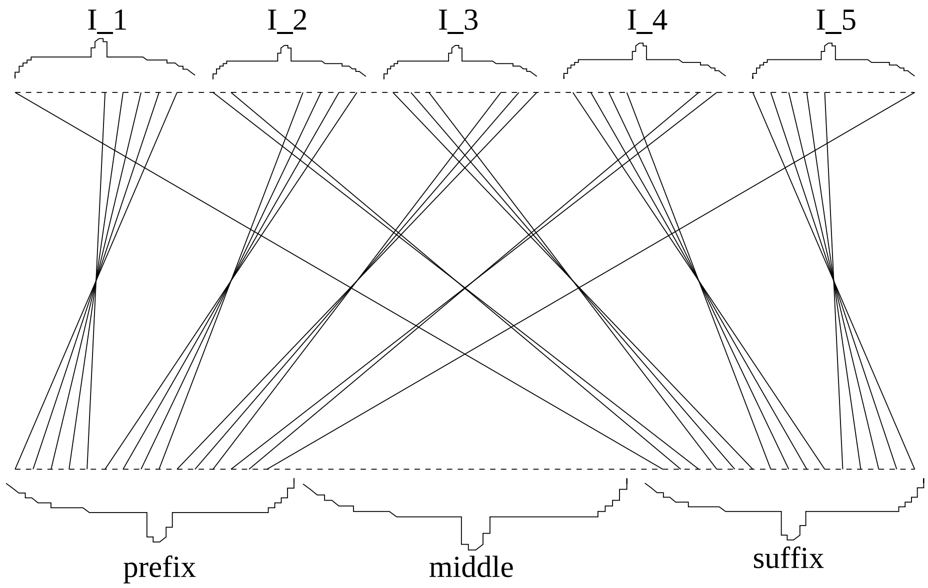

Figure 4: An example for the prefix and suffix for $d = 5$. Each of the lines stands for a "bundle" of L lines that are pairwise intersecting.

the proof.)

Last we define the middle. The middle consists of n cells each corresponding to a pair of vectors in our MIN-MAX-TCVS instance. The cell corresponding to $(v^i, \hat{v}^i)$ consists of d parts, one for each coordinate of v^i and $\hat{v}^i$. The order of these parts in the cell is insignificant. The j^{th} part of the cell consists of $v^i_j + \hat{v}^i_j - 1$ consecutive elements from I_j which we denote by $a_1, \ldots, a_{v^i_j + \hat{v}^i_j - 1}$ ($a_k = a_1 + k - 1$). The element $a_1 \in I_j$ is chosen such that the elements of I_j which we use in the j^{th} part of a particular cell are larger than those used in the j^{th} parts of subsequent cells but smaller than those used in the j^{th} parts of cells preceding it. We order $a_1, \ldots, a_{v^i_j + \hat{v}^i_j - 1}$ such that first we have a decreasing sequence of $a_{v^i_j}$ down to a_1, and subsequently an increasing sequence of $a_{v^i_j + 1}$ to $a_{v^i_j + \hat{v}^i_j - 1}$. (Note that when the cell is reversed we have a decreasing subsequence of length $\hat{v}^i_j$, and when it is not reversed we have a decreasing subsequence of length v^i_j.) Recall that $v^i_j, \hat{v}^i_j \geq 1$ so we have at least one element in each of the d parts of the cell.

Selected parts of a complete construction are demonstrated in Figures 5 and 6.

To complete the proof we show that we can choose one vector of each pair so that their sum in each coordinate is at most Z if and only if in the MINFLIP instance there is a flipping with a

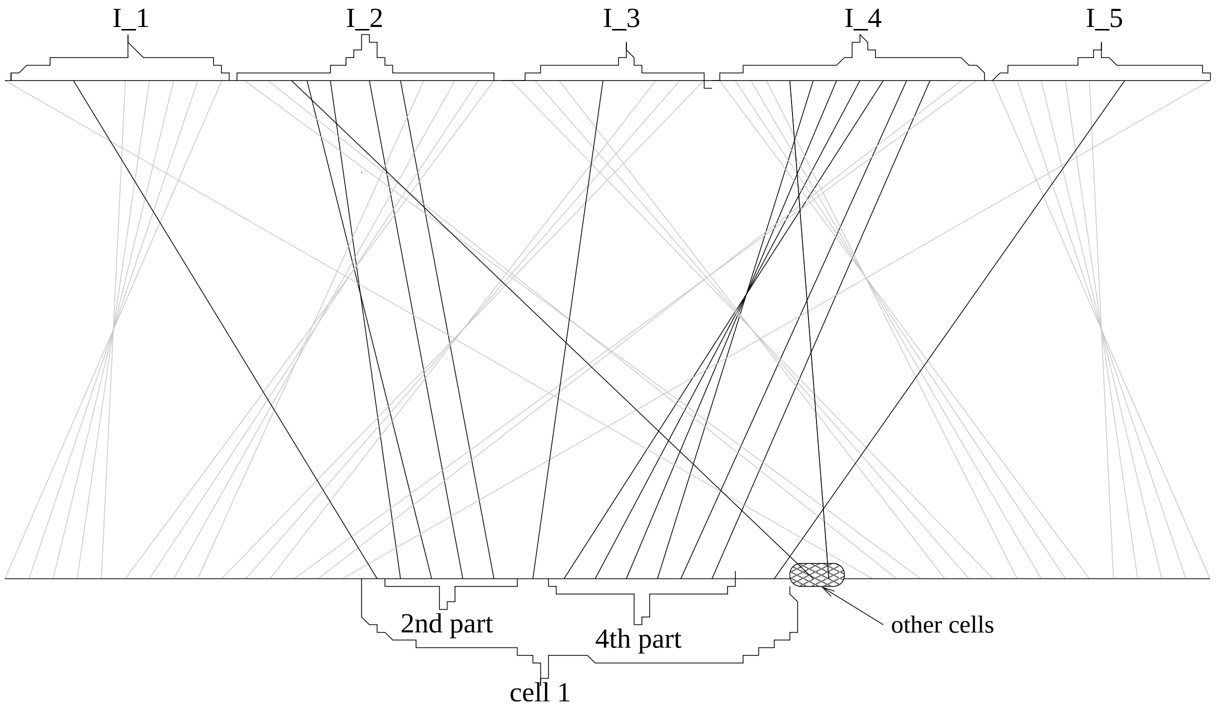

Figure 5: An example containing the prefix, suffix, and first cell of the middle, for $d = 5$ with values $v^1 = (1,2,1,4,1), \hat{v}^1 = (1,3,1,3,1)$. The gray part is the prefix and suffix, already introduced in Figure 4. Note that each of the new black lines stands for one element of the permutation, while each (gray) line in the prefix and suffix stands for a bundle of L pairwise intersecting lines. In order to reduce clutter, other cells are not detailed, except for two representatives from another cell. Note that, according to the reduction, lines that originate at different cells in the bottom line and end at the same interval in the top line always intersect.

maximal decreasing sequence of length $\leq (d+1)L + Z$.

First we show that if there is a flipping F with a maximum decreasing sequence of length $\leq (d+1)L + Z$, then we can choose vectors to solve MIN-MAX-TCVS. We choose vectors as follows. For each pair we choose the second vector if the corresponding cell is flipped in F, and the first vector if it is not flipped in F. We claim that in the sum of the chosen vectors, all coordinates are less than Z. We prove this claim by contradiction.

Say the value of coordinate j in the sum is $Z' > Z$. We show that there is a decreasing subsequence in F of length $(d+1)L + Z'$ which gives a contradiction. This decreasing subsequence

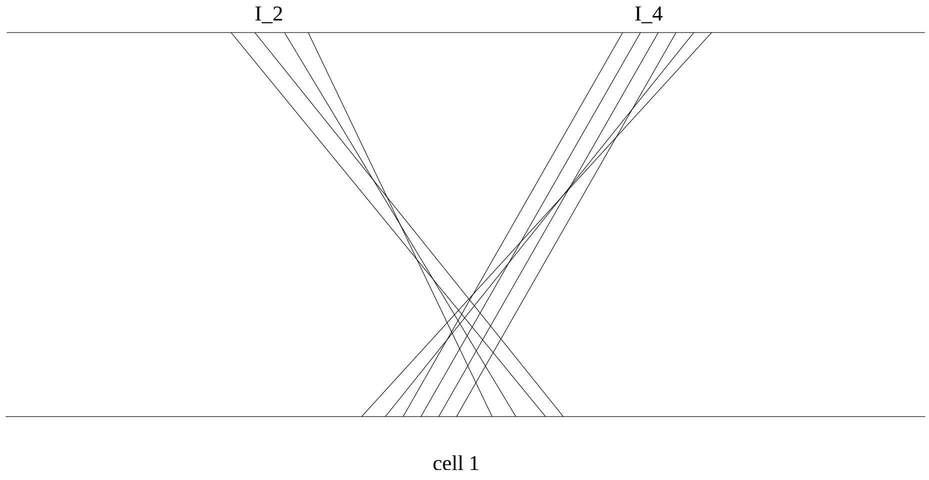

Figure 6: Here we show what happens to the lines that correspond to the non-trivial parts of the first cell in Figure 5 when it is flipped. Note that before the maximum cliques in the second and fourth part were 2 and 4, respectively, and now they are both 3.

is contained in interval I_j. It starts with the $(d+1-j)\cdot L$ elements in the prefix that belong to I_j, and it ends with the $j \cdot L$ elements in the suffix that belong to I_j. From each cell in the middle, the sequence contains the maximum decreasing subsequence which corresponds to coordinate j. It follows from our construction, and the way we chose the vectors, that Z' elements from the middle participate in this decreasing sequence.

To show the converse, we argue that if we can choose vectors so that their sum in each coordinate is at most Z, then there is a flipping F in which each decreasing subsequence is of length $\leq (d+1)L+Z$. We define F by flipping a cell if the solution to MIN-MAX-TCVS contains the second vector of the corresponding pair, and don't flip the cell otherwise. We have to show that in F there is no decreasing subsequence of length greater than $(d+1)L+Z$.

First, we claim that the elements of a maximum decreasing subsequence in the permutation under any flipping, and in particular under F, must contains elements from a single interval I_j (for graphical reference see Figure 4). Since, for any interval I_j, the largest elements of I_j are in the prefix and the smallest elements of I_j are in the suffix, it follows from this claim that a maximum decreasing subsequence starts with the elements of some interval I_j that are in the prefix and ends with elements of the same intervals I_j that are in the suffix.

To prove the claim let us consider a maximal decreasing subsequence A, which contains elements from more than one interval I_ℓ, and derive a contradiction. Let the lowest interval from which A contains elements be I_j and let the highest interval from which A contains elements be I_k, $j < k$. Since A is maximal it contains all the elements from I_k which are in the prefix. (It cannot contain elements from two such intervals I_k and $I_{k'}$ where $k > k'$ since the elements of I_k follow the elements of $I_{k'}$ in the prefix, but are larger.) Also, for a similar reason, A contains all elements from I_j in the suffix. Since $k > j$, A contains $(d + 1 - k) \cdot L + j \cdot L \leq d \cdot L$ elements from the prefix and suffix together. Since the length of the middle is smaller than L, a decreasing subsequence that contains all elements from any interval I_j in the prefix and the suffix is larger than A, so A is not a maximum decreasing sequence.

To complete the proof, it remains to show that when we flip according to F, any decreasing subsequence that is contained in a single interval I_j does not contain more than Z elements from the middle. This follows since otherwise we would get a contradiction to the fact that we defined F based on a solution to MIN-MAX-TCVS. We omit the details which are straightforward.

Finally, for NP-Completeness with cells of size ≤ 5, let us look at what happens when we apply the reduction to an instance of the restricted version of MIN-MAX-TCVS (in which for every pair of vectors v^i and $\hat{v}^i$, v^i has a 1 in a single coordinate and 0 in all others, and $\hat{v}^i$ has a single coordinate which is either 1 or 2 and 0 in all others). First, to get positive vectors we need to add 1 to each coordinate in all vectors and add n to Z. After this modification, for every pair of vectors v^i, $\hat{v}^i$, vector v^i has all coordinates equal to 1 except one coordinate which is equal to 2, and the vector $\hat{v}^i$ has all coordinates equal to 1 except one coordinate which is equal to either 2 or 3.

In the resulting MINFLIP instance according to our construction, the cells in the middle section are of length $d + 2$ or $d + 3$, each of them corresponding to a different pair of vectors. The cell corresponding to the pair v^i and $\hat{v}^i$ has d parts, where the j^{th} part corresponds to v^i_j and $\hat{v}^i_j$. There are two cases:

Case 1: *The vector v^i has a 2 in the same coordinate where the vector $\hat{v}^i$ has a 2 or a 3*: Let j be the index of this coordinate. The j^{th} part of the corresponding cell is of length at most 4 and all other parts are of length 1.

Case 2: *The vector v^i has a 2 in a different coordinate than the one in which vector $\hat{v}^i$ contains a 2* *or a 3*: Let j be the coordinate where v^i has a 2, and let k be the coordinate where $\hat{v}^i$ has 2 or 3. The j^{th} part of the corresponding cell is of length 2, the k^{th} part is of length 2 or 3, and all other parts are of length 1.

Now, our final modification to the reduction is to take each part of size 1 of each cell and put it in a separate new cell following the original cell. It is easy to check that this change does not hurt the correctness of the reductions.

This way we get an equivalent instance of One-Sided MINFLIP, but with cells of size ≤ 5. Now, since the restricted version of MIN-MAX-TCVS is NP-Complete, the theorem follows.□

5 Two-Sided MINFLIP and other Cell Flipping Problems

In [7] we had shown that two-sided MINFLIP is NP-Hard, a result which Theorem 4.4 now generalizes. However, the proof in [7] also shows that MINFLIP does not have a Polynomial Time Approximation Scheme (PTAS), i.e., there is a constant such that no approximation algorithm has an approximation ratio smaller than that constant. We state the appropriate theorem and its corollary here, and refer the interested reader to [7]:

Theorem 5.1 ([7]) *Given a two-sided MINFLIP instance $\mathcal{D}$ and an integer Z, the problem of deciding whether MINFLIP($\mathcal{D}$) $\leq Z$ is NP-complete. This problem remains NP-complete even if Z is fixed, $Z \geq 7$, and the cells in $\mathcal{D}$ are of length at most $2Z + 4$.*

Corollary 5.2 *MINFLIP is inapproximable within $\frac{8}{7}$, unless $P = NP$.*

Currently no inapproximability results are known for one-sided MINFLIP.

Boros, Hammer, Minoux, and Rader [1] investigate the cell flipping problem to minimize channel density for arbitrary nets. Their solution, based on pseudo-Boolean optimization methods, has complexity $O(p \log n)$ for a single channel with p pins and n nets and $O(p(n/c)^c)$ for a stack of c channels. This problem includes the flipping problem on partitioned interval graph representations to minimize the clique number.

6 Acknowledgments

The third author would like to thank Leah Epstein for some helpful discussions.

References

[1] E. Boros, P.L. Hammer, M. Minoux, and D. Rader. Optimal cell flipping to minimize channel density in VLSI design and pseudo-boolean optimization. *Discrete Applied Mathematics*, 90(1-3):69–88, 1999.

[2] R. S. Chang and R. C. T. Lee. On a scheduling problem where a job can be executed only by a limited number of processors. *Computers and Operations Research*, 15(5):471–478, 1988.

[3] Y. L. Chen and Y. H. Chin. Scheduling unit-time jobs on processors with different capabilities. *Computers and Operations Research*, 16(5):409–417, 1989.

[4] M.L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11:29–35, 1975.

[5] M. R. Garey and D. S. Johnson. *Computers and Intractability.* W. H. Freeman, San Francisco, 1979.

[6] M.C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs.* Academic Press, New York, 1980.

[7] M.C. Golumbic and H. Kaplan. Cell flipping in permutation diagrams. In *Proc. 15th annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1373 of *LNCS*, pages 577–586. Springer, 1998.

[8] A. Itai, M. Rodeh, and S.L. Tanimoto. Some matching problems for bipartite graphs. *J. ACM*, 25(4):517–525, 1978.

[9] A.S. LaPaugh. *Algorithms for integrated circuit layout: an analytic approach.* PhD thesis, Dept. of Electrical Engineering and Computer Science, M.I.T., Cambridge, Mass, USA, 1980.

[10] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990. An earlier version of this appeared in the Proceedings of the 28 Annual IEEE Symposium on Foundations of Computer Science.

[11] A. Sinha and A. A. Zoltners. The multiple-choice knapsack problem. *Operations Research*, 27:503–515, 1979.