The 2-machine FJSP with tooling constraints: a theoretical analysis

Luc Libralesso, Vincent Jost, Khadija Hadj Salem, Florian Fontan, Frédéric Maffray

Triangle width: at the intersection of graph theory, scheduling and matrix visualization

Luc Libralesso[1], Vincent Jost[1], Khadija Hadj Salem[2], Florian Fontan[1], and Frédéric Maffray[1]

[1] Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, 38000 Grenoble, France
[2] Université de Tours, LIFAT EA 6300, CNRS, ROOT ERL CNRS 7002, 64 avenue Jean Portalis, 37200 Tours
luc.libralesso@grenoble-inp.fr

Abstract. We introduce a new problem that can be studied from 3 different angles: scheduling, matrix visualization and vertex ordering in hypergraphs. We prove the equivalence of the different formulations of the problem and use them to prove several of its NP-Hard and polynomial subcases. This problem allows to find more elegant (and arguably shorter) proofs for several combinatorial problems. Including "Can a matrix can be made triangular by permuting rows and columns?" and weak k-visit. It also provides an elegant generalization of Johnson's argument for the two-machine flowshop.

Keywords: Combinatorial Optimization · Scheduling · Graph theory · Binary matrix visualization

1 Introduction

In many agendas, one has to first make (costly) investments before being able to carry (rewarding) projects. Each project requires a specific set of investments, while each investment might be needed in several projects. Usually researchers allow to select only a subset of projects to be completed (see [Lus82]). However, we show in this paper that considering the constraint of selecting all projects leads to an elegant problem with real world applications. The question we tackle can be stated as follows: "In which order should we schedule our (predefined sets of) investments and projects, so as to minimize the initial treasury (or loan) required to achieve our agenda?" More conceptually, each project has some required investments that should be paid. Possibly, projects have investments in common. Depending on the investment/project schedule, the initial required treasury varies. We note that it is not allowed to reach a negative amount of treasury. Figure 1 shows an example of possible order of projects. All investments (resp. projects) cost (resp. produce) 1 unit of money except investments 5 and 6 which cost 3 units, and project f produces 2 units.

For more details:

- project a requires investments 1,2

- project b requires 2,3
- project c requires 3,4
- project d requires 4,5
- project e requires 5
- project f requires 2,4,6

In this example, we choose to carry out project a first. Since it requires investments 1 and 2 that both cost 1 units of money, globally, project a costs 2 units of money and produces 1 unit of money. Then, we schedule other projects until every project is completed. Using this specific schedule, we need at least 7 units of treasury to complete this specific project schedule.

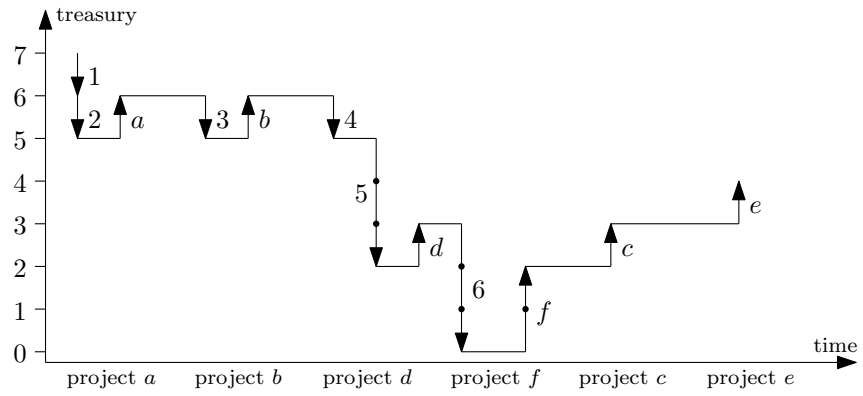


Fig. 1: Example of a treasury diagram where we trace the evolution of the treasury upon projects completion

We now give some examples of applications in which the problem described above arises.

In parallel database join optimization [MR93], one processor loads pages of the secondary memory and executes joins when they are available. We can represent this problem as a hypergraph where vertices are pages and edges are joins to perform. We want to get a page and join load order that minimizes the total execution time.

This problem also occurs for image processing in embedded vision systems (for more details see [HSKM16] and [HSKM18]). These systems consist of calculating a series of image transformations (output tasks). All these transformations require some parts of the original image (input tasks).

Finally, it is also interesting from a theoretical point of view. Although the results can be applied in various fields (scheduling, graph theory and data visualization), the problem is not well studied. It generalizes the following problem, whose computational complexity was asked by Wilf [Wil93]: "Given a square 0/1 matrix, is it possible to make it triangular by permuting both rows and columns?". Wilf's problem was proved $\mathcal{NP}$-Hard [FRV15], but the proof is very

far from straightforward. We show that the triangular matrix problem is a sub-case of triangle width and give a simple $\mathcal{NP}$-Hardness proof.

The structure of the paper is the following. In Section 2, we present three different formulations of triangle width (vertex ordering, scheduling and matrix visualization). We show the equivalence between these formulations. In Section 3, we prove the $\mathcal{NP}$-Hardness of the problem and of some of its sub-cases. In Section 4, we study polynomial time cases of triangle width, in particular, we give an optimal greedy algorithm that solves a generalization of $F2||C_{\max}$ with negative time-lags which is inspired by Johnson's algorithm [GJS76].

2 Definition and Representation

Let V be the set of investments, and E be the set of projects. We express the dependence of a project $e \in E$ on an investment $v \in V$ by an hypergraph $H = (V, E)$ in which the vertex v is incident to the hyper-edge e.

The cost of each investment v is given by the function $p_v : V \to \mathbb{R}^+$. Similarly, the revenue of each project e is given by the function $p_e : E \to \mathbb{R}^+$.

Consider the sequences σ_V (order of investments) and σ_E (order of projects). For a finite set S, we denote by $perm(S)$, the set of permutations on S.

INPUT:

- hypergraph $H = (V, E)$
- $p_v : V \to \mathbb{R}^+$ cost of vertex v (investment)
- $p_e : E \to \mathbb{R}^+$ production of hyper-edge e (project)
- $b \in \mathbb{R}^+$ amount of initial treasury

QUESTION:

Is there a permutation $\sigma \in perm(V \cup E)$ such that:

- for all $e \in E$ and for all $v \in e$, $\sigma(v) < \sigma(e)$
- for all prefix R of σ, $b + \sum_{x \in R} p(x) \geq 0$ where $p(x) = -p_v(x)$ if $x \in V$ and $p(x) = p_e(x)$ if $x \in E$.

As a first result, we observe that given a permutation σ_V of vertices (investments), it is possible to find an optimal permutation σ_E of projects by scheduling them as soon as possible.

Theorem 1 *For any hypergraph $H = (V, E)$, any cost function $p : V \to \mathbb{R}^+$, any profit function $p : E \to \mathbb{R}^+$ and any permutation $\sigma_V \in perm(V)$, an optimal permutation σ_E (depending on σ_V) can be found greedily by scheduling elements of E as soon as possible.*

Proof. We use an exchange argument.

Assume there is a solution given by permutations σ_V, σ_E in which a project e which is not scheduled as soon as possible. Scheduling it just after its last required investment will maintain feasibility and will not increase the amount of initial treasury required.

Considering Theorem 1 results and the solution given in the introduction, we can obtain a new optimal permutation of projects (σ_E) with respect of a given permutation of investments (σ_V). We obtain the schedule described in Figure 2. We obtain the permutation $\sigma_E =< a, b, c, d, e, f >$ and it leads to a required treasury of 5 which is better than the solution given in the introduction which was 7.

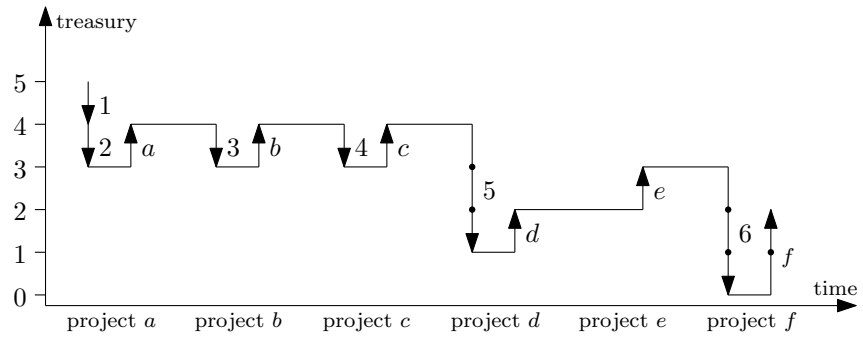


Fig. 2: Another example of a treasury diagram where investments cost money and project produce money

We now present in this section 3 new formulations of this problem using Theorem 1 (*i.e.* vertex ordering, scheduling and binary matrix visualization).

2.1 Formalization and formulation as a vertex ordering problem

Using Theorem 1, we can deduce the following decision model for our problem and express a solution using only a permutation on vertices σ_V (if there is no ambiguity, we call it σ)

INPUT:

- hypergraph $H = (V, E)$
- $p_v : V \to \mathbb{R}^+$ cost of vertex v (investment)
- $p_e : E \to \mathbb{R}^+$ production of hyper-edge e (project)
- $b \in \mathbb{R}^+$ amount of initial treasury

QUESTION:

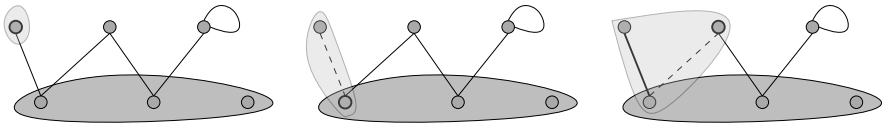
Is there a permutation σ on vertices (investments) such that after each investment, just before starting a project (if one is available), the treasury is positive. More formally, at each step $i \in \{1, 2, \ldots, |V|\}$ (gathering the funds to start the i^{th} investment), the amount of initial treasury (b) together with the amount of money produced by already carried projects (which is $\sum_{e \in E_\sigma(i-1)} p_e(e)$ where

$E_\sigma(i-1)$ denotes the set of hyper-edges induced by the first $i-1$ vertices of the permutation) must be at least the cost of the first i investments ($\sum_{j=1}^{i} p_v(\sigma(j))$)

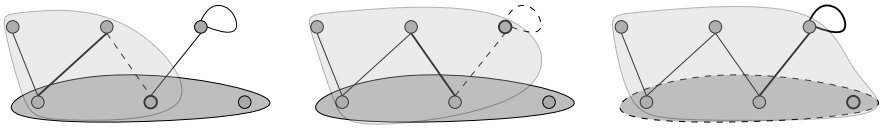
In other words, the permutation σ must satisfy Inequalities (1).

$$\exists\, \sigma \in perm(V), \forall i \in \{1, 2, \ldots, |V|\} \quad , b + \sum_{e \in E_\sigma(i-1)} p_e(e) \geq \sum_{j=1}^{i} p_v(\sigma(j)) \qquad (1)$$

Figure 3 presents an example of a vertex ordering. All vertices cost 1 unit of treasury except the two rightmost ones that cost 3 units of treasury. Each hyper-edge (project) produces 1 unit of treasury except the hyper-edge of degree 3 that produces 2 units of treasury. We start with an initial treasury b of 5.



(a) $i = 1$, funds: $5 - 1 = 4$ (b) $i = 2$, funds: $4 - 1 = 3$ (c) $i = 3$, funds: $3-1+1 = 3$



(d) $i = 4$, funds: $3-1+1 = 3$(e) $i = 5$, funds: $3-3+1 = 1$(f) $i = 6$, funds: $1-3+2 = 0$

Fig. 3: Vertex ordering solution where the treasury initial capacity $b = 5$.

As will be presented in the next section, the vertex ordering formulation is very helpful to obtain complexity results. Indeed, $\mathcal{NP}$-Hardness proofs are tedious to make with other formulations.

2.2 Formulation as a scheduling problem

Let us consider the following two-machine scheduling problem:

INPUT:

- hypergraph $H = (V, E)$
- $p_v : V \to \mathbb{R}^+$ durations of tasks on the input processor.
- $p_e : E \to \mathbb{R}^+$ durations of tasks on the output processor.
- $b \in \mathbb{R}^+$ amount of allowed idle time on the output processor.

The hypergraph H defines the requirements between input and output tasks. An edge e (output task) has all its endpoints in H as requirements.

Each task of V (resp. E) has duration p_v (resp p_e). The goal is to find a schedule defined by permutations σ_V, σ_E such that tasks V (resp. E) are scheduled in these orders on the input processor (resp. output processor) with idle time of b on the output processor. In our context, there is no sense of having an input task which does not have any related output (in this case, we can just remove it). Also, if we have output tasks without any prerequisites, it is dominant to place them at the beginning of the scheduling on the output processor and we do not need to consider them in our problem. Thus, up to preprocessing, we assume that there are neither isolated vertices nor empty edges in the input.

As for the vertex ordering problem, one can note that the order of output tasks (hyper-edges) is strongly related to the order of input tasks (vertices). Indeed, when scheduling an input task v, it is dominant to schedule all output tasks e that have all their precedences scheduled (*i.e.* if there is a feasible solution that do not respect this property, one can exchange output tasks to make this property true without removing feasibility). This argument allows us to have a more compact formulation and only consider the set of permutations σ_V instead of $\sigma_V \times \sigma_E$.

More formally, for each step i of a feasible schedule (scheduling the i^{th} input task), one has to guarantee that the sum of the $(i-1)^{th}$ output task processing times ($\sum_{e \in E_\sigma(i-1)} p_e(e)$, where $E_\sigma(i-1)$ is the set of output tasks induced by the first $i-1$ input tasks of the permutation) and the allowed idle time b is at least the sum of the precedences of the i^{th} output tasks ($\sum_{j=1}^{i} p_v(\sigma(j))$. Otherwise, it implies idle time larger than b.

In other words, one has to satisfy the Inequalities (2).

$$\exists\, \sigma \in perm(V), \forall i \in \{1, 2, \ldots, |V|\} \quad , b + \sum_{e \in E_\sigma(i-1)} p_e(e) \geq \sum_{j=1}^{i} p_v(\sigma(j)) \qquad (2)$$

Figure 4 presents an example of feasible schedule. There are 6 tasks on the input processor and 6 tasks on the output processor. Tasks on the output processor have one or more dependencies represented with arrows.

We now show that satisfying (2) is equivalent to have a feasible schedule.

Theorem 2 *Let σ be a permutation of tasks. σ satisfies (2) if and only if the schedule defined by σ is feasible.*

Proof.

– **If inequalities (2) are verified by a permutation** σ, it is possible to compute a feasible schedule with idle time b on the output processor. Indeed, as we have shown before, one can compute the permutation σ_E. Then we can

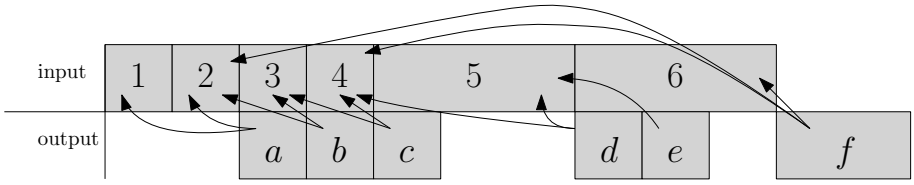


Fig. 4: Example of a feasible schedule

schedule each input task $v \in V$ as soon as possible and schedule each output task $e \in E$ as late as possible. Using this schedule, we have idle time b on the output processor and then all the edges without idle time. It is trivial to see, with those inequalities verified, that such a schedule is feasible.

– **If inequalities (2) are not verified**, we prove by contradiction that it is not possible to compute a feasible schedule with idle time b on the output processor. Consider a feasible schedule such that the output permutation σ_V does not respect inequalities (2) for a position i. We have $b + \sum_{e \in E_\sigma(i-1)} p_e(e) < \sum_{j=1}^{i} p_v(\sigma(j))$. When scheduling the vertex at position i, there is an idle time $b' \leq b$. Since the schedule is feasible, we have $b' + \sum_{e \in E_\sigma(i-1)} p_e(e) \geq \sum_{j=1}^{i} p_v(\sigma(j))$. Thus, it implies that $b + \sum_{e \in E_\sigma(i-1)} p_e(e) \geq \sum_{j=1}^{i} p_v(\sigma(j))$. Contradiction.

2.3 Formulation as a binary matrix visualization problem

Let us consider the following problem:

Consider a binary matrix with rows (resp. columns) of a given height (resp. width). Consider a line parallel to the diagonal by an offset b called the limit (see Figure 5). Is it possible to find a permutation of rows (resp. columns) of the matrix such that each black tile of the matrix is below the limit?

More formally, we can describe the problem as follows:

INPUT:

– hypergraph $H = (V, E)$
– $p_v : V \to \mathbb{R}^+$ width of column v
– $p_e : E \to \mathbb{R}^+$ height of row e
– $b \in \mathbb{R}^+$ offset of the limit

QUESTION:

Is there a permutation of columns (resp. rows) σ_V (resp. σ_E) such that every black tile is below the limit (*i.e.* line parallel to the diagonal).

As in the scheduling version of triangle width, we can notice that the permutation σ_E can be determined once σ_V is given.

We then want to find a permutation of columns σ_V such for every column, all its black tiles lie below the limit. The limit can be defined by the set of points

given in equations (3).

$$\text{limit} = \{(b+i, i) \mid i \in [0; \sum_{e \in E} p_e(e)]\} \tag{3}$$

Since all black tiles in the matrix are composed of rectangles, one can simply check if the upper-right corners are above the limit. Each row at position i must have the sum of the offset (b) and its height ($\sum_{e \in E_\sigma(i-1)} p_e(e)$ where $E_\sigma(i-1)$ is the set of columns with at least one black tile in common with at least one of the first $i-1$ input tasks of the permutation) at least its width ($\sum_{j=1}^{i} p_v(\sigma(j))$).

Doing so, our binary visualization problem can be formulated as in equations (4).

$$\exists\, \sigma \in perm(V), \forall i \in \{1, 2, \ldots, |V|\} \quad , b + \sum_{e \in E_\sigma(i-1)} p_e(e) \geq \sum_{j=1}^{i} p_v(\sigma(j)) \tag{4}$$

Figure 5 presents an example of matrix visualization. The objective is to find a permutation on rows/columns such that all black tiles lie below the limit represented by the dashed red line.

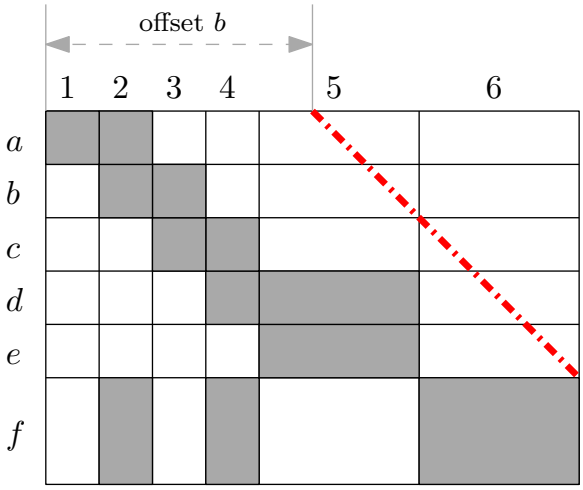


Fig. 5: Example of binary matrix visualization problem. Each row (resp. column) has its own height (resp. width).

2.4 Wrapping up formulations

Theorem 3 (formulation equivalence) *The three triangle width formulations: vertex ordering, scheduling and matrix visualization, are equivalent problems.*

Proof. Since all formulations given by (1), (2) and (4) are strictly equivalent, an optimal solution for one is optimal for the others.

We now discuss the strengths and weaknesses of the 3 proposed formulations:

The vertex ordering formulation is very convenient to obtain complexity results. As we show in the next section, we use this formulation to obtain $\mathcal{NP}$-Hardness proofs.

The scheduling formulation is usually found in applied and industrial problems. However, this formulation makes complexity results difficult to obtain.

During experiments on triangle width, we noticed that instances and solutions were easier to visualize using the matrix form. Indeed, the scheduling version represents hypergraphs as a bipartite graph where the vertices are on one side, the hyper-edges on the other and there is a link between a vertex and hyper-edge if one contains the other. The vertex ordering uses Venn diagrams to represent hypergraphs. Both fail to visualize hypergraphs of reasonable size (i.e. more than 10 vertices and edges). We believe that the matrix visualization form allows to provide valuable information. Along with the results presented in this paper, we provide an interactive web visualization of triangle width instances where one can try to solve instances of triangle width at the following url: `http://librallu.gitlab.io/hypergraph-viz/`.

This paper advocates that each formulation has its strengths and weaknesses. While working on a specific formulation, some results may appear difficult to obtain. Switching to another representation often helps to get elegant and concise arguments. At the beginning of our work on triangle width, we only considered the scheduling version. The $\mathcal{NP}$-Hardness proof was very difficult to obtain. As soon as we changed to the vertex ordering representation, results were much easier to obtain. In order to fully understand the benefit of switching formulations, we invite the reader to think of a $\mathcal{NP}$-Hardness proof for the scheduling formulation without using the vertex ordering formulation.

In the next sections, we consider several sub-problems or related problems, and obtain more elegant results by switching representations.

3 $\mathcal{NP}$-Hard cases

3.1 $p_v = a, p_e = 1$, H is a graph, $1 \leq a \leq |V|$ is $\mathcal{NP}$-Hard

Consider the vertex ordering formulation. We want to find an order σ on the vertices such that:

$$b + |E_\sigma(i-1)| \geq a \cdot i \quad \forall i \in \{1, \dots, |V|\}$$

Where $b \in \{1, \dots, |V|\}$. This problem remains $\mathcal{NP}$-Hard even with $p_v = a, p_e = 1$ and if H is a graph.

Theorem 4 *triangle width is $\mathcal{NP}$-Hard even with $p_v = a, p_e = 1$ and H is a graph.*

Proof. Reduction from Clique, which is defined as follows: Given a graph $G = (V, E)$ and an integer k, is there a set of k vertices pairwise adjacent? Clique is a well known $\mathcal{NP}$-Hard problem [GJ02].

Consider a graph $G = (V, E)$ and an integer $0 \leq k \leq |V|$.

We construct the graph $G' = (V', E')$ as follows :

- $V' = V \cup \{j_1, j_2, \dots j_k\}$
- $E' = E \cup \{(g, j) \mid g \in G, j \in J\}$
- $a = k$
- $b = \frac{k(k+3)}{2}$

An example of such a graph G' is described in Figure 6.

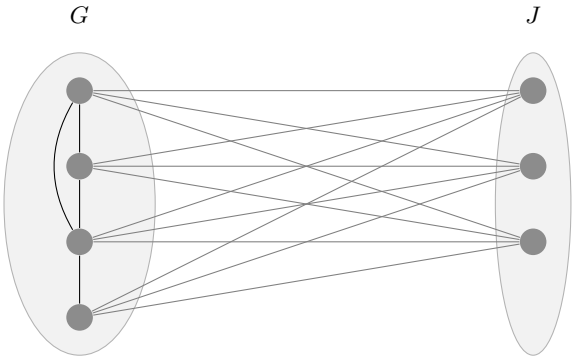


Fig. 6: We consider a new set J of k vertices which are linked to all vertices in G. In this example, we search for a clique of size 3

First, we observe that finding a clique of size k in G is equivalent to find a clique of size $k+1$ in G'. Next, we show that it is also equivalent to the existence of a feasible solution for the vertex ordering in G'.

- there exists a feasible solution for triangle width $\implies$ there exists a clique of size $k+1$ in G': For $i = k+2$, we obtain for all $i \in \{1, 2, \dots, |V|\}$, $b + |E_\sigma(i-1)| \geq a \cdot i$. It implies that $\frac{k(k+3)}{2} + |E_\sigma(i-1)| \geq k \cdot (k+2) \implies |E_\sigma(i-1)| \geq \frac{k(k+1)}{2}$. The only possibility for this inequality to be satisfied is that G' contains a clique of size $k+1$.
- clique of size $k+1$ in G' $\implies$ a feasible solution for triangle width: As the previous case, consider the step $i = k+2$. Since there exists a clique of size $k+1$ in G', we can satisfy steps $i = 1, 2, \dots, k+1$. Then, we can add the remaining vertices in J which contributes each by k. And finally, the remaining vertices of G which will also contribute k. Thus, a clique of size $k+1$ in G' implies a feasible solution for triangle width.

3.2 $p_v = 1, p_e = 1$, H hypergraph is $\mathcal{NP}$-Hard

Theorem 5 *triangle width is $\mathcal{NP}$-Hard, even when $p_v = 1, p_e = 1$ and H is a hypergraph.*

Proof. We perform a reduction from the version with $p_v = a$, $p_e = 1$, H a graph, $1 \leq a \leq |V|$.

We construct the following instance with $p_e = 1, p_v = 1$ as follows:

- $V' = \{v_{u,i} \mid \forall u \in V, \ \forall i \in \{1, 2, \ldots, k\}\}$
- $E' = \{\{v_{u,i} \mid \forall u \in e, \ \forall i \in \{1, 2, \ldots, k\}\} \mid \forall e \in E\}$

The idea is to divide each input task to a separate tasks which belong to the same hyper-edges. It is dominant to schedule the new input tasks by block. Since $a < |V|$ the reduction is polynomial. Thus, solving the sub-case $p_v = 1$ is $\mathcal{NP}$-Hard.

Figure 7 shows an example of column duplication. If we are able to solve the case where columns and rows have the same size, we are also able to solve the case where a column is two times bigger than a row.

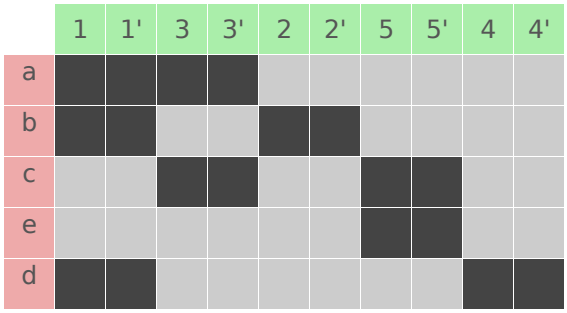


Fig. 7: Matrix where columns are duplicated to simulate a duration of 2 for each column

3.3 Link with weak-k visit

We now study a related vertex ordering problem described in the literature and show that it is a sub-case of triangle width. *Weak-k visit* was originally described in [AFN00] and is known to be $\mathcal{NP}$-Hard. It consists of visiting a graph, starting with k "free" vertices (in the process getting some initial treasury. Again, consecutive vertices are not required to be adjacent). Then, each of the $n - k$ other vertices costs k. The goal is to find a permutation of vertices such that the treasury is always positive. More formally, the problem can be described as follows:

INPUT: A graph $G = (V, E)$, an integer $0 \leq k \leq n$.

OUTPUT: A permutation of vertices $\sigma : V \to \{1, \dots, n\}$ such that:

$$|E_\sigma(i)| \geq k(i-k) \quad k+1 \leq i \leq n \tag{5}$$

Where $E_\sigma(i)$ is the set of edges induced by the first i vertices of the permutation.

We now show that weak k-visit is a sub-problem of triangle width. First, we can notice that, if $|E| < k(n-k)$, the graph is not weak-k visitable since the equation (5) for $i = n$ would not be respected. We can safely assume that $|E| \geq k(n-k)$. Thus, one can only consider the equations for $k+1 \leq i \leq n-1$.

We can reformulate the triangle width as follows and specialize it with edge weights equal to 1 and vertex weights equal to some constant a. We obtain:

$$|E_{i-1}(\sigma)| = a \cdot i - b \quad 1 \leq i \leq n$$

By using the variable $j = i - 1$, we obtain:

$$|E_j(\sigma)| = a(j+1) - b \quad 0 \leq j \leq n-1$$

Since $|E_i(\sigma)| \geq 0$ and $k \geq 0$, we can deduce the definition of *weak-k* visit.

$$|E_\sigma(i)| \geq k(i-k) \quad 0 \leq i \leq n-1$$

Finally, by fixing $j = i, a = k, b = k^2 + k$ one can see that triangle width generalizes weak-k visit problem.

We note that this result also constitutes a new $\mathcal{NP}$-Hardness proof for weak-k visit.

3.4 Can a matrix can be made triangular by permuting rows and columns?

[FRV15] proves that finding if a matrix is triangular if we permute its rows and columns is $\mathcal{NP}$-Hard. We give a new (arguably much simpler) proof by reducing the problem to triangle width, $p_v = 1$, $p_e = 1$, H hypergraph (see section 3.2).

Consider a matrix M and an offset b. We construct a matrix M' where we add b rows made of zeros to M. Then one filled column (C_0) with all ones (See Figure 8. In our matrix representations, zeros are white tiles and ones black tiles). The existence of a permutation making M' triangular is equivalent with finding a permutation making M triangular below b. Indeed, it is dominant to start with column C_0 in our scheduling and the rows with only one 1 in C_0 first. Putting C_0 first, will allow exactly an advance of b for the matrix M. Thus, answering if a matrix can be made triangular by permutations can solve the decision problem of triangle width.

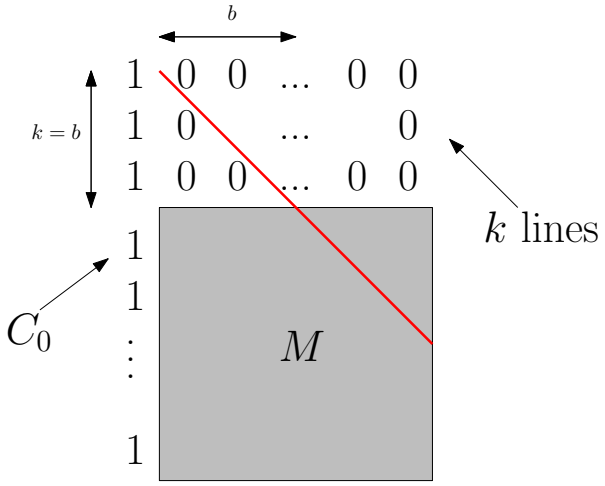


Fig. 8: Example of the matrix used to prove that solving the matrix triangular problem also solves triangle width with $a = 1$

4 Polynomial cases

4.1 Connected orderings

We define a *connected ordering* as follows. Consider a hypergraph $H = (V, E)$. A *connected ordering* orders connected components by block (*i.e.* if two vertices i, j belong to the same connected component, there is no vertex k belonging to another component between i and j in the connected ordering) of decreasing densities ($|E| - |V|$). Within each connected component, for all prefixes $S \subseteq V$, the following vertex belongs to an edge with the maximum number of vertices in S.

One can obtain a connected ordering by executing the steps described by the Algorithm 1.

We now use connected orderings to present some polynomial cases when the processing times of vertices is small. Indeed, with relatively small a, an order exploring the first neighbours of already explored vertices is guaranteed to be optimal (we call this kind of ordering, Prim-like orderings). We present results when H is a hypergraph of fixed edge size (and in particular when H is a graph).

Theorem 6 *If $H = (V, E)$ is a k-uniform hypergraph (*i.e. $|e| = k \ \ \forall e \in E$*) and $a \leq \frac{1}{k-1}$, then there exists a polynomial time algorithm to solve triangle width.*

Proof. The polynomial time algorithm starts by finding a connected ordering as explained above. We claim that the resulting solution is optimal. Indeed, using a connected ordering, except for the first hyper-edge we select, each hyper-edge has a vertex already selected and will cost at most $a \cdot (k-1) \leq \frac{k-1}{k-1} = 1$. This property guarantees that we can add it without decreasing the objective value.

Algorithm 1: Construction of a connected ordering

Input : Hypergraph $H = (V, E)$
Output: permutation of V

```
P ← partition into connected components of H;
sort P by decreasing density (|E| − |V|);
forall c ∈ P do
    start by an arbitrary vertex and add it to the solution;
    while c is not completely visited do
        add an unvisited vertex that is a neighbour of a visited vertex;
    end
end
```

Corollary 1. *If G is a graph and $a \leq 1$, then there exists a polynomial time algorithm to solve triangle width.*

When $a > \frac{1}{k-1}$, the status of the complexity of the problem is open.

4.2 $F2||C_{\max}$ and Johnson's algorithm

We quickly recall $F2||C_{\max}$. Consider two machines (m_1, m_2), n tasks (T) with p_{i1} (resp p_{i2}) the duration of task i on m_1 (resp. on m_2). Each task needs to be scheduled first on m_1, then on m_2. The goal is to find an order σ on the tasks which minimizes the total completion time (makespan or $C_{\max}$). Johnson's rule (see [Pin16]) consists in sorting tasks with the criterion $\min(P_{j1}, P_{i2}) \geq \min(P_{i1}, P_{j2}) \implies i$ is scheduled before j. This leads to an optimal greedy algorithm for $F2||C_{\max}$.

We show that triangle width is a generalization of $F2||C_{\max}$. We use the matrix representation to illustrate and give a visual proof of Johnson's algorithm. Moreover, using this representation, we are able to generalize Johnson's argument and obtain a new rule for $F2||C_{\max}$ with negative time-lags (*i.e.* task i on machine 2 can start before the end of task i on machine 1).

$F2||C_{\max}$ We construct the triangle width as follows:

- $V = T$
- $E = \{\{v\} | v \in T\}$
- $p_e = p_{2e}$
- $p_v = p_{1v}$

We see the triangle width instance as a series of consecutive black blocks. The length of a block is its computation time on machine 1 (and the height of a block its computation time on machine 2). The diagonal line crossing the origin (start of the first task (a)) indicates times where machines 1 and 2 have the same

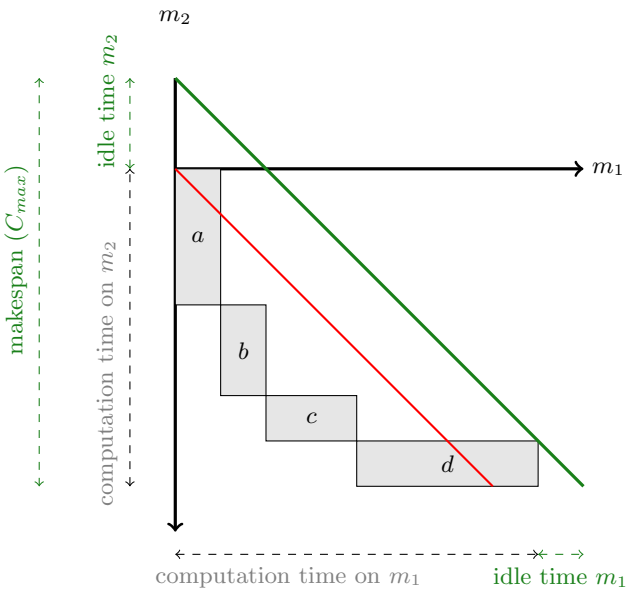


Fig. 9: Visual Argument of equivalence between the formalization using triangle width and $F2||C_{\max}$.

amount of work. Tasks to the right of this line produce more work on machine 1 than on machine 2 (tasks a and d in Figure 9). This implies some idle time on machine 1. The computation time and the idle time on machine 1 give us a lower bound for the minimal C_{max} and the solution of triangle width gives us a feasible solution. Thus, maximizing triangle width on this instance is equivalent to minimizing $F2||C_{\max}$. Figure 9 illustrates this argument.

We provide now Johnson's exchange argument using triangle width visualization (presented on Figure 10). Given two tasks i and j, we can compute the corresponding objective modification by:

- $\min(P_{j1}, P_{i2})$ if i is scheduled before j;
- $\min(P_{i1}, P_{j2})$ if j is scheduled before i

Task i is before task j in the optimal schedule implies that $\min(P_{j1}, P_{i2}) \geq \min(P_{i1}, P_{j2})$ which gives us Johnson's rule ([Pin16]).

4.3 $F2|t_j \in [-\min(p_{1j}, p_{2j}), 0]|C_{\max}$

Using a variant of triangle width, we prove that a $F2||C_{\max}$ generalization is polynomial. To the best of authors knowledge, this variant have not been studied before.

We consider a generalization of $F2||C_{\max}$ where we allow some overlap between task i on machine 1 and task i on machine 2, denoted by t_i. This problem

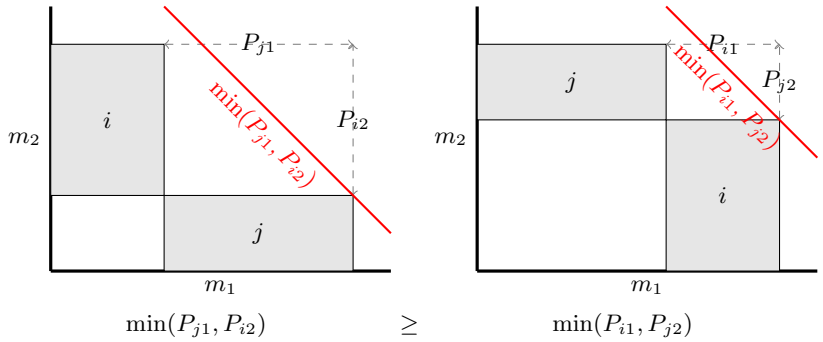


Fig. 10: Exchange argument for Johnson's rule using triangle width visualization

can be easily seen using a variation of triangle width. We now consider rectangles with an empty triangle where the size of this triangle is t_i. Using this representation, we can use the same argument used to prove Johnson's rule.

Theorem 7 *There exists a dominant permutation that schedules tasks i before j if* $\min(P_{j1} + t_i, P_{i2} + t_j) \geq \min(P_{i1} + t_j, P_{j2} + t_i)$.

Proof. First, we prove by contradiction that there exists a dominant permutation schedule, using the same argument for $F2||C_{\max}$ (see [Pin16]). Suppose that no optimal permutation schedule exists. An optimal (non-permutation) schedule has two tasks i, j such that i is scheduled before j on the input machine and j is scheduled before i on the output machine. We can construct another solution by scheduling i just before j on the output machine. The new solution is feasible since every task but i is postponed on the output machine. Also, no task before the end of j on the output machine is postponed, thus, the new solution is also optimal. Hence the contradiction.

Now, let us prove that if i is scheduled before j and $\min(P_{j1} + t_i, P_{i2} + t_j) > \min(P_{i1} + t_j, P_{j2} + t_i)$, the schedule is not optimal. As illustrated in Figure 11, permuting i and j strictly increase the objective function. Thus a schedule where i is before j and $\min(P_{j1} + t_i, P_{i2} + t_j) > \min(P_{i1} + t_j, P_{j2} + t_i)$ is not dominant.

Corollary 2. *There exists a polynomial time greedy algorithm that solves* $F2|t_j \in [-\min(p_{1j}, p_{2j}), 0]|C_{\max}$.

Finally, we note that [Mit59] proposes a related (but different) generalization of Johnson's argument for $F2||C_{\max}$. It considers positive time lags and permutation flowshops.

5 Conclusion

In this paper, we introduced a new problem called triangle width. This problem arises in several real life applications (parallel database optimization, embedded

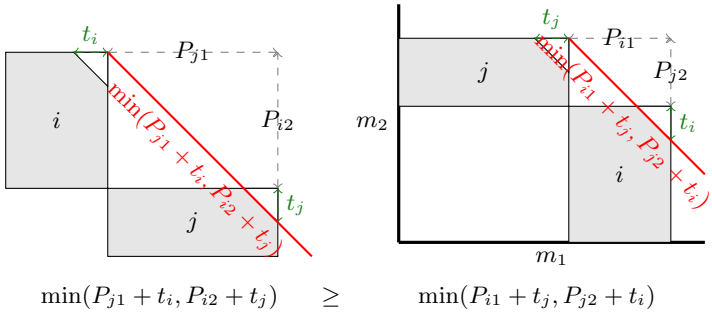


Fig. 11: exchange argument for generalized Johnson's rule using triangle width visualization.

vision system optimization etc.). It is also related to some interesting theoretical questions such as the binary matrix permutation and weak k-visit. We presented it under 3 formulations: a 2 parallel machine scheduling problem which objective is to minimize the makespan, a matrix visualization problem which the objective is to maximize an empty "upper-right triangle" and a vertex ordering problem in a hypergraph where vertices are investments, hyper-edges projects and the objective is to minimize the initial required treasury. The problem turns out to be $\mathcal{NP}$-Hard even if the hypergraph is a graph and vertices cost more than the edges (if the vertices cost less than the edges, the problem is polynomial). If all vertices and edges cost the same, the problem is $\mathcal{NP}$-Hard. Finally, we gave a short proof that the triangular matrix permutation problem is $\mathcal{NP}$-Hard and a visual argument of Johnson's algorithm for $F2||C_{\max}$. This visual argument also helped to find and prove a new generalization of Johnson's rule for the 2 machine flowshop problem with negative time lags.

References

AFN00. Claudio Arbib, Michele Flammini, and Enrico Nardelli. How to survive while visiting a graph. *Discrete applied mathematics*, 99(1-3):279–293, 2000.

FRV15. Guillaume Fertin, Irena Rusu, and Stéphane Vialette. Obtaining a triangular matrix by independent row-column permutations. In *International Symposium on Algorithms and Computation*, pages 165–175. Springer, 2015.

GJ02. Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

GJS76. Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

HSKM16. K. Hadj Salem, Y. Kieffer, and M. Mancini. Formulation and practical solution for the optimization of memory accesses in embedded vision systems. In *Proceedings of the 2016 Federated Conference on Computer Science and Information Systems, FedCSIS 2016, Gdańsk, Poland, September 11-14, 2016*, pages 609–617, 2016.

HSKM18. K. Hadj Salem, Y. Kieffer, and M. Mancini. *Meeting the Challenges of Optimized Memory Management in Embedded Vision Systems Using Operations Research*, pages 177–205. Springer International Publishing, 2018.

Lus82. Hanan Luss. Operations research and capacity expansion problems: A survey. *Operations research*, 30(5):907–947, 1982.

Mit59. L.G. Mitten. Sequencing n jobs on two machines with arbitrary time lags. *Management science*, 5(3):293–298, 1959.

MR93. Marguerite C. Murphy and Doron Rotem. Multiprocessor join scheduling. *IEEE Transactions on Knowledge and Data Engineering*, 5(2):322–338, 1993.

Pin16. Michael L Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2016.

Wil93. Herbert S Wilf. On crossing numbers, and some unsolved problems. *Combinatorics, geometry and probability (Cambridge, 1993)*, pages 557–562, 1993.