Algorithms and Hardness Results for the Jump Number Problem, the Joint Replenishment Problem, and the Optimal Clustering of Frequency-Constrained Maintenance Jobs

by

Claudio Telha Cornejo

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2012

Author . . . . . .
Sloan School of Management
Jan 15, 2011

Certified by . . . . . .
Andreas S. Schulz
Patrick J. McGovern Professor of Management
Professor of Mathematics of Operations Research
Thesis Supervisor

Accepted by . . . . . .
Dimitris Bertsimas
Boeing Leaders for Global Operations Professor
Co-Director, Operations Research Center

Algorithms and Hardness Results for the Jump Number Problem, the Joint Replenishment Problem, and the Optimal Clustering of Frequency-Constrained Maintenance Jobs

by

Claudio Telha Cornejo

Submitted to the Sloan School of Management
on Jan 15, 2011, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

In the first part of this thesis we present a new, geometric interpretation of the jump number problem on 2-dimensional 2-colorable (2D2C) partial order. We show that the jump number of a 2D2C poset is equivalent to the maximum cardinality of an independent set in a properly defined collection of rectangles in the plane. We then model the geometric problem as a linear program. Even though the underlying polytope may not be integral, we show that one can always find an integral optimal solution. Inspired by this result and by previous work of A. Frank, T. Jordan and L. Vegh [13, 14, 15] on set-pairs, we derive an efficient combinatorial algorithm to find the maximum independent set and its dual, the minimum hitting set, in polynomial time. The combinatorial algorithm solves the jump number problem on convex posets (a subclass of 2D2C posets) significantly faster than current methods. If n is the number of nodes in the partial order, our algorithm runs in $O((n\log n)^{2.5})$ time, while previous algorithms ran in at least $O(n^9)$ time.

In the second part, we present a novel connection between certain sequencing problems that involve the coordination of activities and the problem of factorizing integer numbers. We use this connection to derive hardness results for three different problems:

- The Joint Replenishment Problem with General Integer Policies.
- The Joint Replenishment Problem with Correction Factor.
- The Problem of Optimal Clustering of Frequency-Constrained Maintenance Jobs.

Our hardness results do not follow from a standard type of reduction (e.g., we do not prove **NP**-hardness), and imply that no polynomial-time algorithm exists for the problems above, unless Integer Factorization is solvable in polynomial time.

Thesis Supervisor: Andreas S. Schulz
Title: Patrick J. McGovern Professor of Management
Professor of Mathematics of Operations Research

Acknowledgments

I would like to thank all those who have supported me during these 4 years.

First of all, I want to thank my family for their support. They have always been there, in all my happy and not so happy times.

I want to thank my advisor, Andreas Schulz, for all his support during my PhD. studies. It still amazes me how he always managed to find problems that I like, even though I am extremely picky when it comes to research problems. The good relationship we had during these four years was determinant in my enjoyable experience as a doctoral student.

Jose Soto was already a friend of mine before I came to MIT, and I do not think there is enough space here to express how much I value his friendship now. A complete chapter of this thesis is the result of nearly two years of work together. I also want to thank Giannina Miranda, his wife, for organizing so many dinners and for being such a reliable friend.

I always enjoyed our regular outings with my classmates: Allison Chang, Sun Wei, Phillip Keller, Matthew Fontana, Shubham Gupta and Mallory Soldner. I am particularly grateful to Allison Chang for being such a great friend and for her help in the late stages of my thesis.

I always liked research, but working straight for more than two hours was never my thing. Going for coffee with David Goldberg to have an interesting discussion was always the most fun way to have a break. He is not only the most enthusiastic researcher I have ever met, he is also a very good friend, and helped me through the hard times at MIT.

Finally, I want to thank Juliane Dunkel for her support in the last couple of years. She was always there to help me when I needed it. But more importantly, she became one of my closest friends during my time in Cambridge.

Cambridge, December 2011 *Claudio Telha*

Contents

List of Figures

Chapter 1

Introduction

This thesis considers three problems, one from Partial Order Theory and the other two from Operations Management. The main focus is on the question of whether there are theoretically efficient algorithms to solve them.

The most widely accepted notion of a theoretically efficient algorithm is that of a **polynomial time algorithm**. Given a function $f : \mathbb{Z}_+ \to \mathbb{Z}_+$, an algorithm $\mathscr{A}$ runs in time $O(f(n))$ if there is a constant c such that the running time of $\mathscr{A}$ on instances of encoding size n is at most $cf(n)$, for all n greater than a constant n_0. The class of decision problems[1] that can be solved with an $O(f(n))$-time algorithm, where $f(n)$ is a polynomial function of n, is denoted by **P**. The fact that **P** only considers decision problems is not a major limitation, as most problems can be written as decision problems for this purpose.

Identifying **P** with the class of problems admitting efficient algorithm is not a statement exempt of criticism, as an algorithm with running time n^{30} is hardly efficient in practice. However, most interesting problems solvable in polynomial time are eventually solved with a fast polynomial algorithm[2]. For example, the first polynomial algorithm to asymptotically approximate the volume of a polyhedron in $\mathbb{R}^m$ runs in time $O(m^{23})$ [11]. The ideas introduced in this algorithm were subsequently improved in a sequence of papers ending with an $O(m^4)$ algorithm for the same problem, ignoring logarithmic factors [28].

The theory of **NP**-completeness revealed that many interesting problems in Combinato-

[1] Problems where the output is either yes or no

[2] Say, $O(n^4)$ with a relatively low constant c hidden in the $O(\cdot)$ notation

rial Optimization and Operations Research are **NP**-hard, a statement that implies that they are unlikely to be in **P**. The **NP**-hardness of a problem not only shows its inherent difficulty, but also gives evidence that approximate algorithms are necessary. But for many problems, it is still open whether they admit polynomial time algorithms or if they are **NP**-hard.

1.1 Summary

In Chapter 2 of this thesis we provide efficient algorithms to compute the **jump number** of **2-dimensional 2-colorable** partial orders. The jump number is a constant associated to each partial order. We show that for the class of 2-dimensional 2-colorable partial orders, this constant can be computed in polynomial time. More precisely, we provide an $O(n^8)$ time algorithm based on linear programming followed by an $O(n^{2.5}\log n)$ algorithm that exploits the combinatorial structure of the problem.

These algorithms allow us to extend the classes of partial orders for which the jump number can be computed in polynomial time, as well as to provide a significantly faster algorithm for the class of convex partial orders, a subclass of 2-dimensional 2-colorable partial orders where prior to our work, the fastest algorithm for the jump number ran in $O(n^9)$ time [10].

In Chapter 4 of this thesis we show that three optimization problems in Operations Management, the **Joint Replenishment Problem with General Integer Policies**, the **Joint Replenishment Problem with Correction Factor** and the **Clustering of Frequency Constrained Maintenance Jobs** are unlikely to have polynomial time algorithms. For these results, we introduce a notion of hardness weaker than **NP**-hardness, but still enough to imply that polynomial time algorithms are unlikely to exist. The hardness results also reveal an interesting connection between those problems and the problem of factorizing integer numbers. Finally, these results support the use of heuristics to solve these problems. In Chapter 3 we provide a **fully polynomial time approximation scheme** for the Joint Replenishment Problem with General Integer Policies. This is a family of approximation algorithms that allow to approximate the optimal objective value to any desired, but fixed, accuracy in polynomial time.

1.2 Some prerequisites

This thesis assumes that the reader has basic familiarity with Linear Programming, basic Graph Theory and the notions of **NP**-complete and co**NP**-complete problems. A basic exposition on these subjects can be found in [24] Here, we briefly describe some particular results of Linear Programming that will be useful for us.

Linear programming is the problem of optimizing a linear function in $\mathbb{R}^n$ subject to m linear inequalities. It can be written as $\max\{cx : Ax \leq b\}$, where $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$. Linear programming is a useful tool to efficiently solve, in theory and practice, many problems in Combinatorial Optimization and Operations Research. However, linear programs do not explicitly exploit the structure of the problems they model, leaving open the possibility of faster polynomial time algorithms.

The **dual** of the **primal** linear program $\max\{cx : Ax \leq b\}$ is the program $\min\{by : yA = c, y \geq 0\}$. The Theorem of Strong Duality guarantees that the optimal solution for both programs (when they exist) have exactly the same value. In particular, any feasible solution x for the primal and any feasible solution y for the dual satisfying $cx = by$ are optimal for the primal and the dual, respectively. In other words, duality is a certificate of optimality.

Chapter 2

The Jump Number Problem

In this chapter, we study an optimization problem coming from partial order theory, the **Jump Number Problem**. Our most important contribution is a collection of efficient algorithms for some well studied instances of this problem. These algorithms are described geometrically, and are supported by several techniques from combinatorial optimization.

Most of the results introduced here are joint work with José A. Soto [44, 43].

An example

The Jump Number Problem can be stated as a scheduling problem. Suppose that we have to process six tasks labeled a, b, c, d, e, f sequencially, where some tasks have **precedence constraints**:

- Task d cannot be scheduled before tasks a and b have been completed.
- Task e cannot be scheduled before tasks a, b and c have been completed.
- Task f cannot be scheduled before task c has been completed.

Several schedules comply with these precedences. For example, we could sort them (from first to last) in the order $a \to b \to c \to d \to e \to f$ or in the order $a \to b \to d \to c \to f \to e$.

In the Jump Number Problem we aim to find the schedule minimizing the total delay between tasks. This delay is the total time "wasted" between the completion of a task and the beginning of the next one. We assume that the only consecutive pairs of tasks that

produce a delay are those not linked by a precedence constraint. In those cases the delay is equal to 1. For example, there is no delay if we schedule a and d consecutively, but there is a delay of 1 unit of time if we schedule a and b consecutively.

If we decide to schedule the tasks in the order $a \to b \to d \to c \to f \to e$, the pairs of consecutive tasks linked by a precedence constraint ($b \to d$ and $c \to f$) are called **bumps**. In contrast, the pairs not linked by a precedence constraint ($a \to b$, $d \to c$ and $f \to e$) are called **jumps**. So this schedule has 3 jumps and 2 bumps. It is not hard to see that this schedule minimizes the number of jumps (which is exactly the total delay) and therefore this schedule is the optimal solution to this jump number instance. A non-optimal solution, for example, is the schedule $a \to b \to c \to d \to e \to f$, with 5 jumps and 0 bumps.

In our example, the number of bumps plus the number of jumps is equal to 5 for every schedule, so it follows that minimizing the number of jumps is equivalent to maximize the number of bumps. This simple property is a key idea of this chapter, since we consider instances of the Jump Number Problem for which maximizing the number of bumps can be interpreted as a geometrical problem. Let us see some of the ingredients of this geometrical interpretation. We initially use a directed graph to visualize the precedences, with nodes representing tasks and arcs representing precedences:

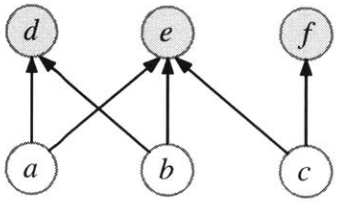


Figure 2-1: A bipartite set of precedences.

Note that every arc starts with a task from $\{a,b,c\}$ and ends with a task from $\{d,e,f\}$. We color white the nodes $\{a,b,c\}$ and we color gray the nodes $\{d,e,f\}$ to emphasize this distinction between the nodes. This is a very strong property: the fact that the nodes can be bi-colored is equivalent to say that the set of precedences is bipartite. Note that with bipartite instances we cannot infer implicit precedences using transitivity. For non-bipartite instances, implicit precedences must be taken into account when computing the jump number.

But so far there is no geometry involved. If we now position the nodes as indicated in Figure 2-2, we see that arcs pointing towards the north-east, starting at a white point

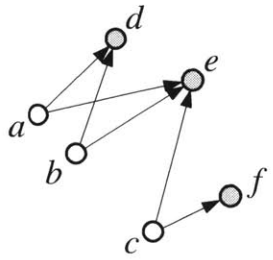


Figure 2-2: A geometric representation of the precedences in Figure 2-1.

and ending at a gray point are exactly the precedence arcs. One of the main results of this chapter shows that finding the schedule maximizing the number of bumps (i.e. minimizing the number of jumps) is equivalent to find the maximum number of disjoint rectangles having the precedence arcs as diagonals.

Several well known families of bipartite precedences can be represented using this geometric method. For these families, we design new efficient algorithms for the Jump Number Problem.

2.1 Partial orders

The role of tasks and precedences in the example we just showed corresponds to what in combinatorics is called a partially ordered set. And in the same context, schedules correspond to linear extensions. In this section we give a precise definition of the Jump Number Problem from the perspective of partial order theory. We then study in detail a particular family of partial orders that admits a geometric representation from where we can compute the jump number efficiently.

Definitions

A **partial order** $\preceq$ on a finite set X is a binary relation that is reflexive, antisymmetric and transitive on X. It becomes a **total order** if, additionally, every pair of elements $u, v \in X$

is **comparable**, that is, either $u \preceq v$ or $v \preceq u$ holds. We use $u \sim v$ to denote comparable elements.

A pair $P = (X, \preceq)$, where $\preceq$ is a partial order on X is called a **partially ordered set**, or just **poset**. A **linear extension** of P is a poset $(X, \preceq_L)$ where $\preceq_L$ is a total order on X which is **compatible** with $\preceq$, that is, $u \preceq v$ implies $u \preceq_L v$ for every $u, v \in X$. The set of linear extensions of P will be denoted by $\mathscr{L}_P$.

Colloquially, we translate $u \preceq v$ as "u is smaller than v under $\preceq$". Similarly, we say that u is strictly smaller than v under $\preceq$ if u is smaller than v but u and v are not equal. We denote this by $u \prec v$. A **chain** of a poset $P = (X, \preceq)$ is a subset $X' \subset X$ such that $(X', \preceq_{|X'})$, the restriction of the poset P to X', is a total order. A **convex chain** is a chain X' that has no other elements in between, that is, if $u \preceq v \preceq w$, and $u, w \in X'$, then $v \in X'$.

Finally, a poset $P = (X, \preceq)$ is **bipartite** if X can be written as the union of two disjoint sets $X = A \cup B$ so that $u \prec v$ implies $u \in A$ and $v \in B$.

Basic representations of a partial order.

Every poset $P = (X, \preceq)$ can be represented using a directed graph where each element of X is represented by a node and each comparison $u \prec v$ is represented by a directed arc from u to v. We call this the **natural representation** of P.

The **adjacency matrix representation of a partial order** $P = (X, \preceq)$ enumerates the elements of X, say $X = \{x_1, x_2, \ldots, x_{|X|}\}$ and represents P using a binary matrix M^P of size $|X| \times |X|$, where $M^P_{ij} = 1$ if and only if $x_i \preceq x_j$. When the poset is bipartite, say $X = A \cup B$, it is sufficient to provide the submatrix of M^P associated to $A \times B$ only. We call this a **biadjacency matrix representation**.

But so far none of these representations use the properties of the partial orders. In the **Hasse diagram**, we represent the poset using a directed graph where nodes become elements, and arcs become comparisons, but the Hasse diagram does not include comparisons that can be inferred by using transitivity. More precisely, $u \prec v$ is drawn in the Hasse diagram if and only if there is no element w so that $u \prec w \prec v$. We say that v **covers** u in this case. The advantage of this representation is that it is easier to visualize and is more

compact than the standard graph representation. Figure 2-3 shows two Hasse diagrams.

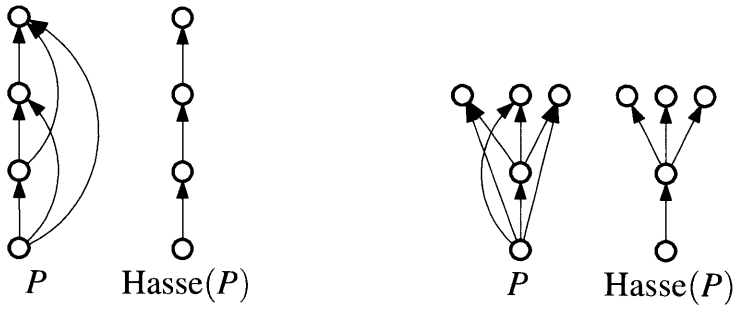


Figure 2-3: Some Hasse diagrams, where a directed arrow from u to v indicates that v covers u. In the graphical version, the arrows always point upwards, so they can be deleted.

Another characterization of a poset is based on total orders. We can see a linear extension L of a poset P as a way to decide the comparisons not established under P. It is possible to prove [47] that if u and v are incomparable under P, then there is a linear extension of P where u is smaller than v, and another one where v is smaller than u. Therefore we can implicitly define P as the set of comparisons that are simultaneously compatible with all the linear extensions of P, that is

$$P = \bigcap_{L \in \mathscr{L}_P} L,$$

where the intersection of posets with the same ground set preserves the ground set and intersects the partial orders.

Given the last property, it is natural to ask what is the shortest representation of P as the intersection of linear extensions. The **dimension** of a poset P is the minimum number of linear extensions of P whose intersection is P. For example, a poset where every pair of elements is comparable (a total order, or **chain**) has trivially dimension 1, while a poset where no pair of elements is comparable (an **antichain**) has dimension 2, because it is the intersection of any total order of the elements and its reverse[1].

The structure of a poset is determined by the comparable pairs and by the relative order of each comparable pair. But even if the relative order is missing, the comparable pairs are still enough to study several poset properties, the jump number among them [19]. The

[1] By reverse, we mean that "smaller than" becomes "larger than".

comparability graph $G(P)$ of a poset $P = (X, \preceq)$ is the undirected graph with vertex set X, and where there is an edge between u and v if and only if u and v are comparable. Note that a comparability graph does not contain enough information to recover the partial order, and therefore is not a representation of the partial order.

2.2 Geometrical representation of some partial orders

In this section we introduce two classes of a partial orders. What they have in common is that elements can be identified with integral points in $\mathbb{Z}^m$ and that convex chains can be seen as hyper-rectangles. We start by formalizing a particular notion of representation. An **embedding** of a poset $(X, \preceq_X)$ on a poset $(Y, \preceq_Y)$ is a function $f : X \to Y$, where $u \preceq_X v$ if and only if $u, v \in X$ and $f(u) \preceq_Y f(v)$. We can identify $(X, \preceq_X)$ with $(f(X), \preceq_Y |_{f(X)})$ as long as we only study poset properties.

We will define each of these posets geometrically, and then prove the equivalence to their classical definitions. We do this just for clarity, as by default we will see posets geometrically.

In this section, a point $u \in \mathbb{R}^m$ will be denoted as $u = (u_1, u_2, \ldots, u_m)$.

The m-dimensional partial order

An **m-dimensional** partial order P is a collection of points in $\mathbb{Z}^m$ with the partial order $\leq_{\mathbb{Z}^m}$ defined as follows:

$$u \leq_{\mathbb{Z}^m} v \quad \text{if and only if} \quad u_i \leq v_i \text{ for } i = 1, \ldots, m.$$

The following lemma "justifies" the term **poset dimension**.

Lemma 2.2.1 ([47]). *Every poset $P = (X, \preceq)$ can be embedded in $(\mathbb{Z}^m, \leq_{\mathbb{Z}^m})$, for some m. The smallest value of m is equal to the dimension of the poset.*

Proof. Let $L_i = (X, \preceq_{L_i}), i = 1, \ldots, m$ be linear extensions of P with $\cap_{i=1}^{m} L_i = P$ (in particular m can be the dimension of P). For each $u \in X$, construct a vector $u' \in \mathbb{Z}^m$ where u'_i is

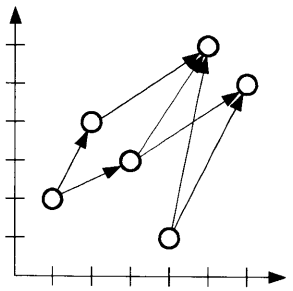


Figure 2-4: A 2-dimensional poset and its Hasse diagram.

equal to the position of the element u in the linear extension L_i. Clearly, $u \preceq_{L_i} v$ if and only $u'_i \leq v'_i$, and therefore, $u \preceq v$ if and only if $u' \leq_{\mathbb{Z}^m} v'$. Therefore, $f(u) = u'$ for $u \in X$ defines an embedding of P in $(\mathbb{Z}^m, \leq_{\mathbb{Z}^m})$. The inverse of this construction shows that if P can be embedded in $(\mathbb{Z}^m, \leq_{\mathbb{Z}^m})$, then the dimension of P is at most m. □

In this work, it is convenient to add an additional assumption to the geometrical representation that is implicit in the proof of Lemma 2.2.1.

Lemma 2.2.2. *Every m-dimensional partial order with k elements can be represented in* $(\{1,2,\ldots,k\}^m, \leq_{\mathbb{Z}^m})$ *so that no two points share a common coordinate value. We call this a* ***rook representation***[2] *of an m-dimensional poset.*

As an example, the 2-dimensional poset in Figure 2-4 is in rook representation.

The m-dimensional 2-colorable partial order

We now define a bipartite version of the m-dimensional poset. To the best of our knowledge, this class has not been defined before.

An ***m*-dimensional 2-colorable partial order** is a collection of points $S \subseteq \mathbb{Z}^m$ with a coloring function $f : S \to \{\text{white, gray}\}$ and the following partial order: $u \preceq v$ if and only if $f(u) = \text{white}$, $f(v) = \text{gray}$ and $u \leq_{\mathbb{Z}^m} v$.

The following lemma is a simple extension of a result in Soto [43].

Lemma 2.2.3. *An* ***m-dimensional 2-colorable partial order*** *has dimension at most* $m+1$.

[2] In chess, rooks placed in those positions would not block each other.

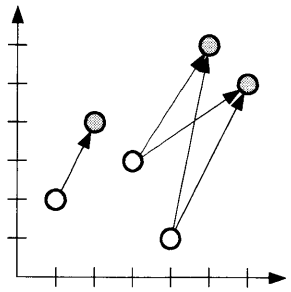


Figure 2-5: A 2-dimensional 2-colorable poset and its Hasse diagram.

Proof. If white is mapped to -1 and gray is mapped to 1, then the constraint $f(u) =$ white, and $f(v) =$ gray can be written as $f(u) < f(v)$. This allows us to replace the coloring condition by a partial order constraint in one additional dimension. □

Again, we can assume that no two points share the a common coordinate value. This is shown in the same way as in Lemma 2.2.2 for m-dimensional posets.

Lemma 2.2.4. *Every m-dimensional 2-colorable partial order with k elements can be represented in* $(\{1,2,\ldots,k\}^m, \leq_{\mathbb{Z}^m})$ *and the same coloring function, so that no two points share a common coordinate value. We call this a* ***rook representation*** *of an m-dimensional 2-colorable poset.*

The 2-dimensional 2-colorable poset in Figure 2-5 is in rook representation.

2.3 The jump number of posets representable in the plane

The jump number has a simple geometrical interpretation when the partial order is 2-dimensional 2-colorable. Since they are the main subject of this chapter, we will call them 2D2C posets.

Since we will work in the plane, we will use some ad-hoc notation. We denote the coordinates of $\mathbb{Z}^2$ as x and y, so that a point p in the plane can be written as (p_x, p_y). For any set $S \in \mathbb{Z}^2$, the **projection** $\{s_x : s \in S\}$ of S onto the x axis will be denoted by S_x. We define S_y, the projection onto the y axis, in a similar way.

Given two sets S and S' of $\mathbb{Z}^2$, we write $S_x < S'_x$ if the projection S_x is to the left of the projection S'_x, that is, if $p_x < p'_x$ for all $p \in S$, $p' \in S'$. We extend this convention to

$S_x > S'_x, S_x \leq S'_x$ and $S_x \geq S'_x$, as well as to the projections onto the y-axis. Given two sets S and S' of $\mathbb{Z}^2$, we write $S_x < S'_x$ if the projection S_x is to the left of the projection S'_x, that is, if $p_x < p'_x$ for all $p \in S$, $p' \in S'$. We extend this convention to $S_x > S'_x, S_x \leq S'_x$ and $S_x \geq S'_x$, as well as to the projections onto the y-axis.

Finally, we denote by $\Gamma(a,b)$ the rectangle with bottom-left corner $a \in A$ and upper-right corner $b \in B$. That is, $\Gamma(a,b) = \{p \in \mathbb{R}^2 : a_x \leq p_x \leq b_x, a_y \leq p_y \leq b_y\}$. You can assume that this definition only holds for a and b such that $a_x < b_x$ and $a_y < b_y$.

For notational convenience, we will describe a 2D2C poset as a set $S = A \cup B \subseteq \mathbb{Z}^2$, where A are the white points and B are the gray points. **We will always assume by default that the points of the poset are in rook representation**. Although $A \cup B$ is enough to describe the partial order, we explicitly state the comparabilities using a set $\mathscr{R}$ of geometric rectangles that identifies each comparability $a \leq_{\mathbb{Z}^n} b$, with $a \in A$ and $b \in B$ using the rectangle $\Gamma(a,b)$ having the segment between a and b as diagonal. In other words,

$$\mathscr{R} = \{\Gamma(a,b) : a \in A, b \in B \text{ and } a \leq_{\mathbb{Z}^2} b\}.$$

We call a and b the **defining vertices** of $\Gamma(a,b)$. We are now ready to interpret the jump number of 2D2C posets geometrically, using the set $\mathscr{R}$. We say that $R = \Gamma(a,b)$ and $R' = \Gamma(a',b')$ **intersect** if the rectangles R and R' have a non-empty geometric intersection. A collection of pairwise not-intersecting rectangles is called an **independent set of rectangles**.

Lemma 2.3.1. *For any two rectangles $R = \Gamma(a,b)$ and $R' = \Gamma(a',b')$, with a,b,a',b' all distinct, the following three statements are equivalent:*

1. *The rectangles R and R' intersect.*
2. *The following four comparabilities hold: $a \leq_{\mathbb{Z}^2} b, a \leq_{\mathbb{Z}^2} b', a' \leq_{\mathbb{Z}^2} b$ and $a' \leq_{\mathbb{Z}^2} b'$.*
3. *There is no linear extension of $(\{a,b,a',b'\}, \leq_{\mathbb{Z}^2})$ with bumps between a and b, and also between a' and b'.*

Proof. Let us prove this in three steps:

- (1) $\Rightarrow$ (2): Note that if there is a point $p \in R \cap R'$, then $\max(a_x, a'_x) \leq p_x \leq \min(b_x, b'_x)$ and $\max(a_y, a'_y) \leq p_y \leq \min(b_y, b'_y)$, so the four comparabilities hold.

- (2) $\Rightarrow$ (3): The only two possible linear extensions satisfying the requirement of (3) are $a \to b \to a' \to b'$ and $a' \to b' \to a \to b$. Both violate one of the four comparabilities in (2).

- (3) $\Rightarrow$ (1): By contradiction, suppose that $R \cap R' = \emptyset$. Then one of the rectangles must be either to the left or above the other one. Suppose R is located to the left of R'. Then, it is easy to check that $a \to b \to a' \to b'$ is a linear extension of $(\{a, b, a', b'\}, \leq_{\mathbb{Z}^2})$. If R' is located to the left of R, then $a' \to b' \to a \to b$ is a linear extension of $(\{a, b, a', b'\}, \leq_{\mathbb{Z}^2})$. Both cases are therefore incompatible with (3). A similar argument can be used when one rectangle is above the other one.

□

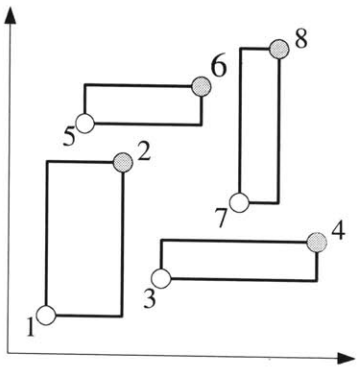


Figure 2-6: A linear extension (numerically ordered) of a 2D2C poset and the corresponding bumps.

Now suppose $P = (A \cup B, \mathscr{R})$ is a 2D2C poset. Using Lemma 2.3.1, it is easy to show that the bumps of any linear extension of P corresponds to a set of disjoint rectangles in $\mathscr{R}$. Surprisingly, any collection of disjoint rectangles in $\mathscr{R}$ corresponds to a subset of bumps of some linear extension of P. Although this result follows from a characterization of the jump number of chordal bipartite posets [34], here we present a self-contained geometric proof.

Lemma 2.3.2. *Let $R_i = \Gamma(a_i, b_i), i = 1, \ldots, m$ be a collection of disjoint rectangles. Let $A = \{a_1, \ldots, a_m\}$, $B = \{b_1, \ldots, b_m\}$ and $\mathscr{S} = \{R_i\}_{i=1,\ldots,m}$. Then there is a linear extension of $P = (A \cup B, \mathscr{R})$ where each R_i is a bump.*

Proof. Consider the directed graph H with node set $\mathscr{S}$, and directed arcs from R_i to R_j if and only if $a_i \leq_{\mathbb{Z}^2} b_j$ and $i \neq j$.

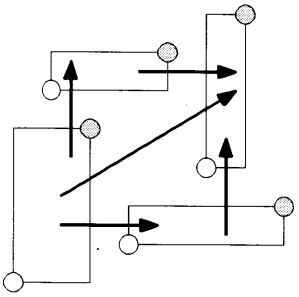


Figure 2-7: The directed graph H in the proof of Lemma 2.3.2.

If H is acyclic, then the transitive closure of H is a partial order, so we can pick any linear extension of this transitive closure, say $R_{j_1} \to R_{j_2} \to \ldots \to R_{j_m}$, and the order $a_{j_1} \to b_{j_1} \to a_{j_2} \to b_{j_2} \to \ldots \to a_{j_m} \to b_{j_m}$ will be a linear extension of P where each R_i is a bump. This follows from the fact that every comparability in this order is compatible with P, by construction.

The remainder of the proof is a geometric argument that shows that H is acyclic. Suppose, by contradiction, that there is a cycle in H. Relabeling if necessary, let $R_1 \to R_2 \to \ldots \to R_{k-1} \to R_k$ be a cycle with minimum number of nodes. We divide the analysis in two cases:

- Suppose first that $k = 2$. Then Lemma 2.3.1 implies that either $a_1 \not\leq_{\mathbb{Z}^2} b_2$ or $a_2 \not\leq_{\mathbb{Z}^2} b_1$, which contradicts the definition of H.

- Suppose, on the contrary, that $k \geq 3$. For every i from 1 to k, it is easy to see that $a_i \not\leq_{\mathbb{Z}^2} a_{i+1}$, where addition must be read cyclically in $\{1, 2, \ldots, k\}$. Otherwise, we would have that $a_i \leq_{\mathbb{Z}^2} b_i \leq_{\mathbb{Z}^2} a_{i+1} \leq_{\mathbb{Z}^2} b_{i+2}$, and therefore the cycle is not the shortest. For the same reason, we also have that $a_i \not\leq_{\mathbb{Z}^2} b_{i+2}$ for every i (cyclically).

  Suppose now that $(a_1)_x < (a_2)_x$. We can then prove that $(a_2)_x < (a_3)_x$. To see this, note that $a_1 \not\leq_{\mathbb{Z}^2} b_3$ and $a_2 \leq_{\mathbb{Z}^2} b_3$ implies that b_3 must live in the region marked

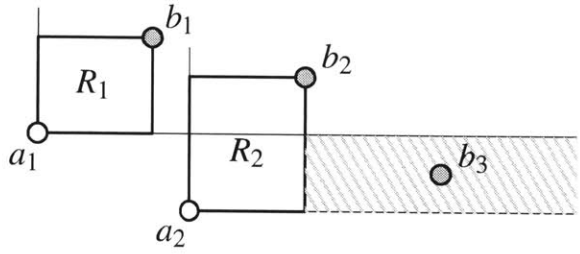


Figure 2-8: Regions in proof of Lemma 2.3.2

in Figure 2-8, and therefore $(a_2)_x < (a_3)_x$. Iterating this argument, it follows that $(a_i)_x < (a_{i+1})_x$ implies $(a_{i+1})_x < (a_{i+2})_x$. But then $a_k \not\leq_{\mathbb{Z}^2} a_1$, which contradicts the fact that $a_k \leq_{\mathbb{Z}^2} b_1$.

□

The following corollary is immediate:

Theorem 2.3.1. *The jump number problem of a 2D2C poset $P = (A \cup B, \mathscr{R})$ is equivalent (under polynomial time reductions) to the maximum independent set of rectangles problem in $\mathscr{R}$.*

Proof. We can assume that P is in rook representation. Let $\mathscr{R}'$ be a maximum independent set of rectangles in $\mathscr{R}$. Using Lemma 2.3.1, there is a linear extension L of a subset of $A \cup B$ where each rectangle in $\mathscr{R}'$ is a bump. Extend this to a linear extension of P, by appending every point in A not already in L to the beginning of L, and every point in B not already in L to the end of L. This proves that there is a linear extension with at least $|\mathscr{R}'|$ bumps. On the other hand, again by Lemma 2.3.1, the bumps of any linear extension correspond to an independent set of rectangles, and therefore any linear extension must have at most $|\mathscr{R}'|$ bumps.

□

Using topological sort, finding a total order of an acyclic graph can be done in linear time in the number of arcs, which is $O(n^2)$ for a 2D2C poset with n points. Therefore, the reduction of the jump number problem to an independent set of rectangles problem is unlikely to be the bottleneck of any algorithm solving the jump number via this equivalence.

The case of 2-dimensional posets

For 2-dimensional partial orders, a geometric characterization using independent sets of rectangles was given by Ceroi [6]. Let $P = (S, \mathbb{Z}^2)$ be a 2-dimensional poset, and let $\mathscr{C}$ be the set of convex chains of P. Ceroi associates to each convex chain $C \in \mathscr{C}$ with smallest element u and largest element v the rectangle $R = \{p \in \mathbb{R}^2 : u \leq_{\mathbb{R}^2} p \leq_{\mathbb{R}^2} v\}$.

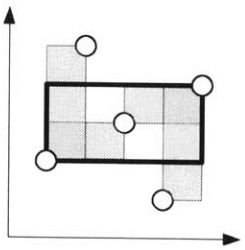


Figure 2-9: Rectangles corresponding to convex chains in a 2-dimensional poset. For Theorem 2.3.2, the weights of every rectangle is equal to 1, except for the thick rectangle that has weight 2.

Using Lemma 2.3.1, it is easy to prove that the set of (maximal) convex chains of any linear extension correspond to sets of disjoint rectangles. This, together with a statement similar to Lemma 2.3.2 translate into the following result:

Theorem 2.3.2 (Ceroi [6]). *Let $P = (S, \mathbb{Z}^2)$ be a 2-dimensional partial order. Then the jump number of P is equivalent (under polynomial time reductions) to a maximum weighted independent set of the rectangles associated to convex chains. The weight assigned to a convex chain is the number of points in the chain minus one.*

The proof is quite similar to the proof of Theorem 2.3.1 for 2D2C posets. The weights chosen by Ceroi are quite natural, since a convex chain $C \in \mathscr{C}$ is a collection of $|C| - 1$ bumps. In the case of 2D2C posets, convex chains have length 1 or 2 and therefore we obtain an unweighted[3] problem.

What happens after dimension 2?

For both, m-dimensional and m-dimensional 2-colorable posets with $m \geq 3$, the reduction of the jump number problem to an independent set of convex chains problem is not applicable.

[3] weights are 0 or 1 only.

While it is true that linear extensions induce independent sets of hyper-rectangles having weight identical to the number of bumps, not every independent set of hyper-rectangles can be transformed into a linear extension where those rectangles are convex chains. An example is the following set of three parallelepipeds in dimension 3:

1. The first parallelepiped determined by $(1,3,1)$ and $(4,4,2)$.

2. The second parallelepiped determined by $(3,1,1)$ and $(4,2,4)$.

3. The third parallelepiped determined by $(1,1,3)$ and $(2,4,4)$.

It is easy to see that they are disjoint. But since $(1,3,1) \leq_{\mathbb{Z}^3} (2,4,4)$, $(3,1,1) \leq_{\mathbb{Z}^3} (4,4,2)$ and $(1,1,3) \leq_{\mathbb{Z}^3} (4,2,4)$, there is no way to sort the extremes of those parallelepipeds into a linear extension.

Why rook representation?

In most of this thesis, the use of a rook representation helps the intuition, and so it will be assumed by default. There are cases, however, where using a more flexible representation is more natural. One example is the following. Let us say that a 2-dimensional 2-colorable poset $P = (A \cup B, \mathscr{R})$ has two elements in B that have exactly the same comparabilities with the elements of A. It would be convenient to match the two elements in B to exactly the same point in the plane.

We defined the geometric representation so that each element in the poset is a different point in the plane. However, it causes no contradiction to assume that A and B can be non-disjoint multisets. Points assigned to multiple elements can be split using the operation indicated in Figure 2-10.

We will use this more flexible geometric definition in Section 2.4, when we introduce biconvex, convex and bi-interval posets. It is indeed possible to prove most of the results of this chapter under this definition, but this adds many technical details that makes the exposition lengthier. For example, rectangles associated to bumps can now be degenerated (points or segments).

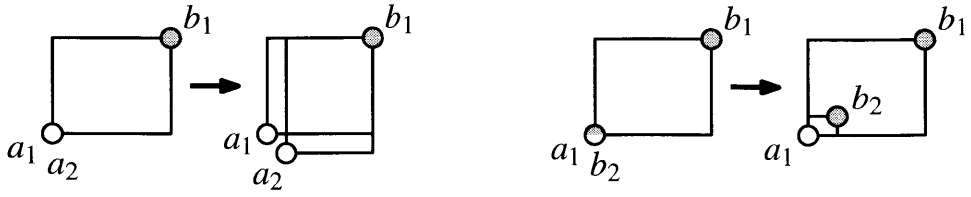


Figure 2-10: Splitting operations. The left side shows how to split points from the same side of the bipartition (in the example, points from A). The right side shows how to split points from different sides of the bipartition.

2.4 The 2D2C posets and its subclasses

Many important classes of partial orders are subclasses of 2D2C posets. In this section, we give priority to the geometric definition of these classes, even though none of them were initially conceived this way. We do this to keep the geometric intuition as the fundamental viewpoint. Eventually, we will go outside the class of 2D2C posets in order to briefly describe the state of the art of this problem. In this section, and all the subsequent ones, we reserve the variable n to denote the number of elements in the poset we are working on.

Given any particular class of posets, the **recognition problem** is the one of deciding whether a particular poset belong to this class. The class recognition problem is usually applied to posets given in natural representation, not the geometric one.

The smallest subclass: bipartite 2-dimensional posets

Since 2D2C posets are a bipartization of 2-dimensional posets, it is reasonable to consider the class of 2D2C posets for which this bipartization is unnecessary. A **bipartite 2-dimensional poset** $P = (A \cup B, \mathscr{R})$ is a 2D2C poset in which A and B are antichains. Equivalently, a bipartite 2-dimensional poset is a bipartite poset that is simultaneously 2-dimensional.

One of the most traditional characterizations of bipartite 2-dimensional posets is the following:

Lemma 2.4.1 ([5]). *A bipartite poset $P = (A \cup B, \mathscr{R})$ is 2-dimensional if and only if we can enumerate $A = \{a_1, a_2, \ldots, a_{|A|}\}$ and $B = \{b_1, b_2, \ldots, b_{|B|}\}$ so that if $a_i \preceq_P b_j$ and $a_k \preceq_P b_l$ for some $i < k$ and $l < j$, then $a_i \preceq_P b_l$ and $a_k \preceq_P b_j$ must also hold. This property is called*

strong ordering.

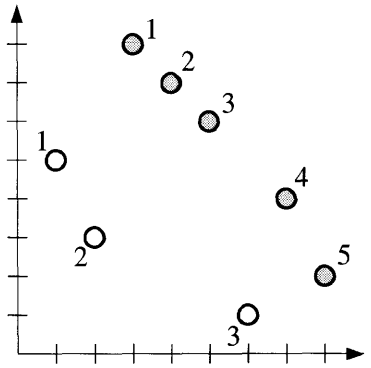


Figure 2-11: A bipartite 2-dimensional poset and a possible strong ordering.

The intuition behind this lemma is easy to see geometrically (see Figure 2-11) : if A and B are antichains in rook representation, then sorting the points according to the coordinate x immediately gives a strong ordering.

This class has strong structure, and the jump number can be solved extremely efficiently. Steiner and Stewart gave an $O(|\mathscr{R}|)$ algorithm [45], while later on Fauck [12] and Brandstädt [4] gave an $O(n)$ algorithm. In Section 2.8 we describe an algorithm that solves a generalization of the jump number in time $O(n)$, using an approach similar to the one of Brandstädt.

As we will discuss later, recognizing bipartite 2-dimensional posets in natural representation can be done in polynomial time, as it is also easy to construct the geometric representation from the natural representation or even from the strong ordering.

Convex and biconvex posets

In a bipartite 2-dimensional poset $P = (A \cup B, \mathscr{R})$, if we use the strong ordering of A and B to index the rows and columns of the biadjacency matrix M^P, we obtain a matrix where in every row and column the set of ones form a consecutive block (i.e. with no zeros in between). A matrix with this property is called **biconvex**. When only the set of columns satisfy this property, the matrix is called **convex**.

Not every biconvex biadjacency matrix comes from a bipartite 2-dimensional poset. An example is the matrix

$$M^P = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix},$$

which is clearly biconvex, but it does not admit a strong ordering[4].

It is natural to define (**convex**) **biconvex posets** as the class of bipartite posets with a (convex) biconvex biadjacency matrix. These two classes are not equivalent. For example, the biadjacency matrix

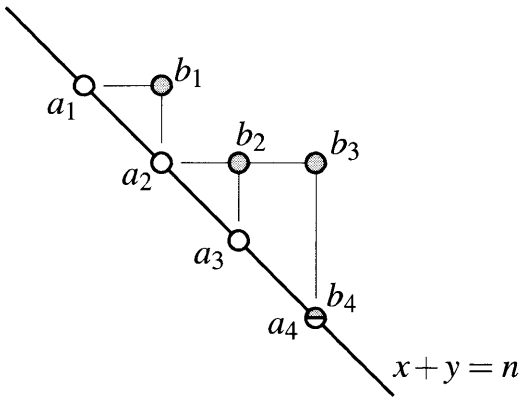
$$M^P = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix},$$

corresponds to a convex poset. But there is no way to sort the columns of M^P so that the rows have consecutive ones.

Convex posets are 2D2C posets with a very particular geometric representation. The following lemma describe this representation, although it is not of the rook type.

Lemma 2.4.2. *A poset P is convex if and only if there exists a representation $P = (A \cup B, \mathscr{R})$, where the points of A are placed in the line $x+y=n$ and the points of B are in the halfspace $x+y \geq n$. (Note: the representation allows A and B to be non-disjoint multisets, see page 28).*

Proof. Let M^P be a convex matrix, and let $A = \{a_1, a_2, \ldots, a_{|A|}\}$ and $B = \{b_1, b_2, \ldots, b_{|B|}\}$ be the set of points in $\mathbb{Z}^2$ that we will associate to the set of rows and columns of M^P, enumerated according to the rows and columns of M^P, respectively. We will assume that each point $b \in B$ is comparable with some point in A, as those b with no comparability can be easily positioned afterwards. First, place all the points of A in the line $x+y=n$, by setting $a_i = (i, n-i)$. For the points $b_j \in B$, if the j-th column of M_P has ones between the i_1-th and i_2-th row (for $i_1 \leq i_2$) set $b_j = (i_2, n-i_1)$. The latter corresponds to put b_j at the intersection of the horizontal line passing through a_{i_1} and the vertical line passing through a_{i_2} (see Figure 2-12 for a concrete example). It is easy to check that b_j is only comparable



$$M^P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix},$$

Figure 2-12: A convex poset in non-rook representation. Note that two elements are mapped to the same location.

with a_i satisfying $i_1 \leq i \leq i_2$ (see Figure 2-12), and therefore $M^P_{ij} = 1$ if and only if $a_i \preceq_P b_j$.

The converse follows from a similar argument. Given $P = (A \cup B, \mathscr{R})$, where the points of A are on the line $x+y=n$ and the points of B satisfy $x+y \geq n$, it is easy to see that each point in B is comparable with a set of consecutive points of A in the line $x+y=n$. Therefore, ordering the rows of M^P according to the order induced by this line gives a convex biadjacency matrix.

□

The line $x+y=n$ could have been replaced by other lines with negative slope. But the construction given in the proof places all the points in $\{1,2,\ldots,n\} \times \{1,2,\ldots,n\}$.

The state of the jump number problem prior to the work in [44] changed dramatically from biconvex to convex posets. While in both classes, the jump number problem was known to be polynomial time solvable, the fastest algorithm for biconvex posets ran in $O(n^2)$ time [4], while for convex posets the fastest algorithm ran in $O(n^9)$ time [10].

Bi-interval posets

If convex posets can be represented by placing points of A in the line $x+y=n$ and the points of B in the halfspace $x+y \geq n$, **bi-interval posets** extend this by allowing the points of A to lie in the halfspace $x+y \leq n$. As with convex posets, it is convenient to assume that

[4]To see this easily, note that in any strong order the second row and the third column of M^P cannot be first or last, while the first column of M^P must be either first or last.

A and B can be non-disjoint multisets.

This definition can be easily shown to be equivalent to the traditional definition of bi-interval posets.

Lemma 2.4.3. *A poset $P = (A \cup B, \mathscr{R})$ is bi-interval if and only if we can associate to each element $v \in A \cup B$ an interval $I_v \subseteq [1,n]$ with integral extremes so that $a \preceq_P b$ for $a \in A$ and $b \in B$ if and only if $I_a \cap I_b \neq \emptyset$. (Note: the representation allows A and B to be non-disjoint multisets, see page 28).*

Proof. Let $P = (A \cup B, \mathscr{R})$ a bi-interval poset, where the points of A lie on the halfspace $x+y \leq n$ and the points of B lie on the halfspace $x+y \geq n$. It is easy to see that we can assume that all the points lie in $\{1,2,\ldots,n\} \times \{1,2,\ldots,n\}$. We assign, to each point $a \in A$, the interval $I_a = [a_x, n-a_y]$, and to each point $b \in B$ the interval $I_b = [n-b_y, b_x]$. Clearly, each interval is non-empty and has integral extremes. Suppose $a \preceq_P b$ for some $a \in A$, $b \in B$. Then we additionally have $a_x \leq b_x$ and $a_y \leq b_y$, and therefore $\max\{n-b_y, a_x\} \leq \min\{n-a_y, b_x\}$. The latter implies that I_a and I_b cannot be disjoint.

For the converse, note that if we map the interval $I_v = [i_1, i_2]$ to the point $v = (i_1, n-i_2)$ for the points $v \in A$ and to the point $v = (i_2, n-i_1)$ for the points $v \in B$, we obtain a geometric representation where the points $a \in A$ satisfy $x+y \leq n$, the points $b \in B$ satisfy $x+y \geq n$, and the condition $I_a \cap I_b \neq \emptyset$ is equivalent to $a \preceq_P b$.

□

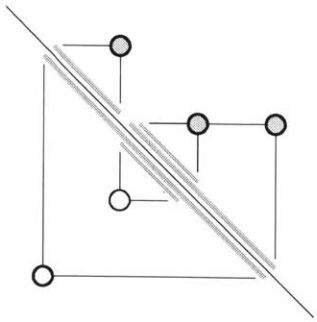


Figure 2-13: A bi-interval poset represented in the plane. The gray segments correspond to the interval representation.

Müller [33] showed that bi-interval posets is a strict subclass of 2D2C posets that can be recognized in polynomial time. In contrast to convex posets, prior to [44] it was not

known whether there were polynomial time algorithms for the jump number of bi-interval posets.

2-directional orthogonal-ray posets.

In 2009, Shrestsa et al. [42] introduced the **2-directional orthogonal-ray posets**[5]. The elements of a 2-directional orthogonal-ray poset P are rays in $\mathbb{R}^2$, which can be either horizontal rays going rightwards or vertical rays going downwards. Given two rays r_1 and r_2 in P, we say that $r_1 \preceq_P r_2$ if and only if r_1 is horizontal, r_2 is vertical, and the two rays intersect.

More precisely, an horizontal ray in the poset has the form $[a_x, \infty) \times \{a_y\}$ while a vertical ray in the poset has the form $\{b_x\} \times (-\infty, b_y]$, where a and b are points in $\mathbb{R}^2$. It is easy to show that we can assume that a and b have integral coordinates, and therefore these two types of rays intersect if and only if $a \leq_{\mathbb{Z}^2} b$ (see Figure 2-14). Hence, we have the following

Lemma 2.4.4. *The class of 2D2C posets is equivalent to the class of 2-directional orthogonal-ray posets.*

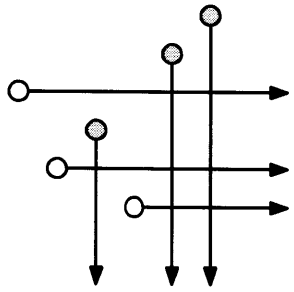


Figure 2-14: A 2D2C poset in orthogonal-ray form. Comparable elements are intersecting rays.

Shreshta et al. [42] give several properties of (what we now call) 2D2C posets. One of them is the following characterization:

Lemma 2.4.5 ([42]). *A bipartite poset P is a 2D2C poset if and only it can be represented using a biadjacency matrix M^P having no submatrices of the following form:*

[5] however, they work with the comparability graph, so they call them graphs and not posets.

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}.$$

We remark that Lemma 2.4.5 is valid for one particular biadjacency matrix representation. Another result shown in the same paper is the following:

Lemma 2.4.6 ([42]). *A bipartite poset P is a 2D2C poset if and only the complement of its comparability graph is a* ***circular arc graph****, that is, the intersection graph of arcs in a circle (see Figure 2-15).*

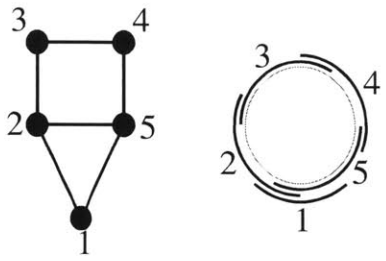


Figure 2-15: A graph in circular arc representation.

Given a 2D2C poset P, Shreshta et al. [42] show how to obtain, in $O(n^2)$ time, a biadjacency matrix M^P satisfying the conditions of Lemma 2.4.5. They also prove that from such a M^P the geometric representation of P using rays can be found in $O(n^2)$ time. But all this holds assuming that they have access to the the circular ray graph representation of the complement of P. This missing part follows from a result of McConnell [30]. Altogether, this gives:

Lemma 2.4.7 (Implicit in [42]). *We can find a geometrical representation of a 2D2C poset in $O(n^2)$ time.*

Beyond 2D2C posets: chordal bipartite posets

There are many possible generalizations of 2D2C posets. Here we only focus on one particular class that has a direct relation with our work.

A graph is said to be **chordal** if every cycle of the graph has a **chord**, that is, an arc connecting some pair of non-consecutive nodes in the cycle. Müller [34] shows that the jump number problem is NP-hard for posets with chordal bipartite comparability graph

(we call them **chordal bipartite posets**). It is not hard to see that 2D2C posets are chordal bipartite. Indeed, this is implicitly shown in the proof of Lemma 2.3.2[6].

A matching in a graph $G = (V, E)$ is **cross-free** if the matching has no **crossing** edges, that is, a pair (u, v) and (u', v') where (u, v') and (u', v) are both in E. For chordal bipartite posets, the jump number is equivalent to find a **maximum cross-free matching** in the comparability graph [34]. An alternative way to prove Theorem 2.3.1 uses this result, together with the observation that crossing edges correspond to intersecting rectangles not sharing any defining vertices. This is depicted in Figure 2-16.

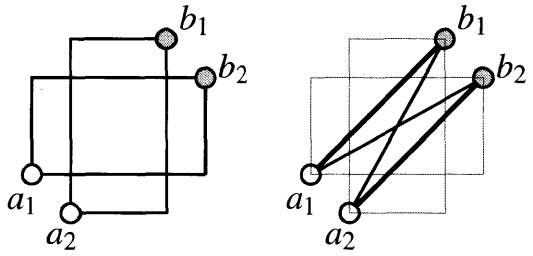


Figure 2-16: Intersecting rectangles correspond to crossing edges.

2.5 A linear programming algorithm for 2D2C posets

We now present a linear programming based algorithm that solves the jump number of a 2D2C poset $P = (A \cup B, \mathscr{R})$ in rook-representation in polynomial time. As in the previous section, we denote $|A \cup B|$ as n. We will also denote $\{1, \ldots, n\}$ as $[n]$.

Let us start from one of the classic integer program formulations for the maximum independent set of a collection of rectangles $\mathscr{R}$:

$$\mathbf{mis}_{\mathrm{IP}}(\mathscr{R}) \equiv \max \left\{ \sum_{R \in \mathscr{R}} x_R : \sum_{R:\ q \in R} x_R \leq 1, q \in [n]^2; x \in \{0,1\}^{\mathscr{R}} \right\} .$$

Informally, the constraints of this integer program state that at most one rectangle of the solution can touch any "interesting" point in the plane. The removal of the integrality

[6] In the proof of this lemma, the fact that H is acyclic implies that all chordless cycles in the comparability graph of a 2D2C poset have length at most 4.

constraint leads to the linear relaxation:

$$\mathbf{mis}_{\mathrm{LP}}(\mathscr{R}) \equiv \max\left\{ \sum_{R\in\mathscr{R}} x_R : \sum_{R:\ q\in R} x_R \leq 1, q \in [n]^2; x \geq 0 \right\} ,$$

which is widely used in other contexts to obtain exact or approximate solutions to the corresponding integer program. In what follows, we discuss some of the properties of this linear program.

Integrality of the linear relaxation

Usually, the first step when using a linear relaxation of an integer program is to check whether the underlying **independent set polytope**

$$\mathrm{P}(\mathscr{R}) \equiv \{x \in \mathbb{R}^{\mathscr{R}} : \sum_{R:\ q\in R} x_R \leq 1, q \in [n]^2; x \geq 0\}$$

has only integral vertices (which is equivalent to say that $\mathrm{P}(\mathscr{R})$ is an integral polytope). It is not hard to see that this claim is false in general. A simple counter-example is the poset shown in Figure 2-17. If $\mathrm{P}(\mathscr{R})$ was integral for this poset, then any linear maximization

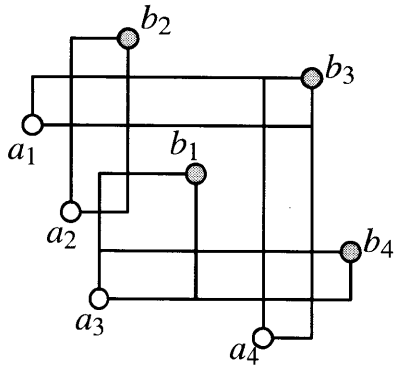


Figure 2-17: A 2D2C poset with non-integral independent set polytope.

problem over $\mathrm{P}(\mathscr{R})$ should have an integral optimal solution. But this is not true when the objective c satisfies $c_R = 1$ for the five rectangles shown in Figure 2-17, and 0 otherwise: any optimal solution must satisfy $x_R = 1/2$ for each of the five rectangles.

There are several important results regarding the integrality of $\mathrm{P}(\mathscr{R})$. The **intersection graph** of a collection of rectangles $\mathscr{R}$, denoted by $\mathscr{I}(\mathscr{R})$ is the graph having the rectangles

in $\mathscr{R}$ as vertices, and with arcs connecting every pair of intersecting rectangles:

$$\mathscr{I}(\mathscr{R}) \equiv (\mathscr{R}, E), \quad \text{where } E = \{(R, R') \in \mathscr{R}^2 : R \cap R' \neq \emptyset\}.$$

A graph is **perfect** if the chromatic number of every induced subgraph equals the size of a maximum clique in that subgraph. A classical result on perfect graphs (see, e.g. [40]) establishes a connection between perfect graphs and the independent set polytope. In our case, this result can be phrased as follows:

Theorem 2.5.1 ([40]). *Given a family of rectangles $\mathscr{R}$, the polytope* $\mathrm{P}(\mathscr{R})$ *is integral if and only if the intersection graph $\mathscr{I}(\mathscr{R})$ is perfect.*

The more recent Strong Perfect Graph Theorem [7] gives a complete characterization of perfect graphs in terms of forbidden subgraphs. The five rectangles shown in Figure 2-17 form an **odd-hole** in $\mathscr{I}(\mathscr{R})$, which is one of these forbidden structures. In this work we will not use this theorem to imply perfectness, but these ones:

Theorem 2.5.2 ([40]). *Every comparability graph is perfect.*

Theorem 2.5.3 (Weak Perfect Graph Theorem). *The complement of a perfect graph is perfect.*

Since Figure 2-17 is a biconvex graph, the only subfamily of 2D2C posets for which $\mathrm{P}(\mathscr{R})$ could be always integral is bipartite 2-dimensional. This turns out to be true. Although this result is known, we present a proof that will be used later.

Lemma 2.5.1. *For bipartite 2-dimensional posets $(A \cup B, \mathscr{R})$, $\mathscr{I}(\mathscr{R})$ is perfect.*

Proof. Let us introduce a partial order $\searrow$ of the rectangles in $\mathscr{R}$. We will say that $R \searrow S$ if R and S are disjoint and either $R_x < S_x$ or $R_y > S_y$ holds. It is not hard to verify that $D = (\mathscr{R}, \searrow)$ is a partial order whose comparability graph is the complement of $\mathscr{I}(\mathscr{R})$. Therefore, by Theorems 2.5.2 and 2.5.3, the graph $\mathscr{I}(\mathscr{R})$ is perfect. □

Finding an integral solution using uncrossing

While not every vertex of $P(\mathscr{R})$ is integral, we can still find an optimal solution to

$$\mathbf{mis}_{\mathrm{LP}}(\mathscr{R}) = \max\left\{ \sum_{R\in\mathscr{R}} x_R \colon \sum_{R\colon q\in R} x_R \leq 1, q \in [n]^2; x \geq 0 \right\}$$

that is integral. The key idea is to find the optimal fractional solution to $\mathbf{mis}_{\mathrm{LP}}(\mathscr{R})$ that minimizes the geometric area. More precisely, if z^* denotes the optimal cost of $\mathbf{mis}_{\mathrm{LP}}(\mathscr{R})$, we consider the following linear program, that uses area(R) to denote the geometric area[7] of R:

$$\overline{\mathbf{mis}}_{\mathrm{LP}}(\mathscr{R}) = \min\left\{ \sum_{R\in\mathscr{R}} \mathrm{area}(R) x_R : \sum_{R\in\mathscr{R}} x_R = z^*; \sum_{R\colon q\in R} x_R \leq 1, q \in [n]^2; x \geq 0 \right\} .$$

Let x^* be an arbitrary but fixed optimal extreme point of $\overline{\mathbf{mis}}_{\mathrm{LP}}(\mathscr{R})$, and let $\mathscr{R}_0$ be the set of rectangles in $\mathscr{R}$ having non-zero value in x^*:

$$\mathscr{R}_0 = \{R \in \mathscr{R} : x^*_R > 0\}.$$

The set $\mathscr{R}_0$ is usually called the **support** of x^*. The properties of this set are important for the linear programming algorithm we present here and for a combinatorial algorithm we will present later on. To start, note that rectangles in $\mathscr{R}_0$ must be inclusion-wise minimal, and therefore:

Lemma 2.5.2. *Every rectangle in $\mathscr{R}_0$ does not contain any point in $A \cup B$ other than the two defining vertices.*

It follows from this observation that the intersection described in Figure 2-18 (left) cannot arise in $\mathscr{R}_0$. What is non-trivial to see is that the opposite type of intersection shown in Figure 2-18 (right) cannot arise. We will call the two type of intersections just mentioned **corner-intersections**.

Proposition 2.5.1. *There are no corner-intersections in $\mathscr{R}_0$.*

[7] i.e. width times height.

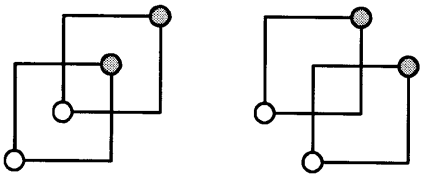


Figure 2-18: Corner intersections.

Proof. Suppose that R and R' have a corner-intersection as shown in Figure 2-19 (left). Let $\lambda = \min\{x^*_R, x^*_{R'}\}$. Figure 2-19 (left) shows two other rectangles, R'' and R''', that also

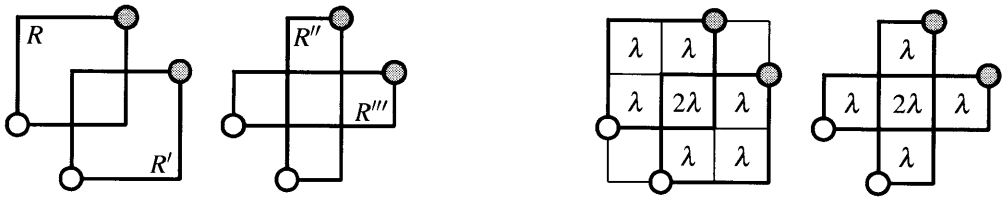


Figure 2-19

belong to $\mathscr{R}$. We modify x^* as follows:

- We reduce the values of x^*_R and $x^*_{R'}$ by λ.
- We increase the values of $x^*_{R''}$ and $x^*_{R'''}$ by λ.

Let $\bar{x}^*$ be this modified solution. We can show that $\bar{x}^*$ is feasible for $\mathbf{mis}_{\mathrm{LP}}(\mathscr{R})$, as the total weight of the rectangles covering any point in the plane can only decrease with the modification:

$$\sum_{S \ni q} x^*_S \geq \sum_{S \ni q} \bar{x}^*_S, \quad \forall S \in \mathscr{R}.$$

This is graphically shown in Figure 2-19 (right), and not only implies that the constraints $\sum_{S \ni q} \bar{x}^*_S \leq 1$ hold, but also implies that the values $\bar{x}^*_S$ are between 0 and 1.

We can also easily check that $\sum_{S \in \mathscr{R}} \bar{x}^*_S = \sum_{S \in \mathscr{R}} x^*_S = z^*$, and therefore $\bar{x}^*$ is also feasible for $\overline{\mathbf{mis}}_{\mathrm{LP}}(\mathscr{R})$.

Finally, the modified solution $\bar{x}^*$ has

$$\lambda \left((\mathrm{area}(R) + \mathrm{area}(R')) - (\mathrm{area}(R'') + \mathrm{area}(R''')) \right) > 0$$

less area than the original solution x^*, contradicting the optimality of x^*. □

A **corner-free intersection (c.f.i.)** family is a collection of rectangles whose intersections have one of the forms shown in Figure 2-20. More precisely, a corner-free intersection between two rectangles R and S arises if and only if $R_x \subseteq S_x$ and $S_y \subseteq R_y$ holds or if $S_x \subseteq R_x$ and $R_y \subseteq S_y$ holds.

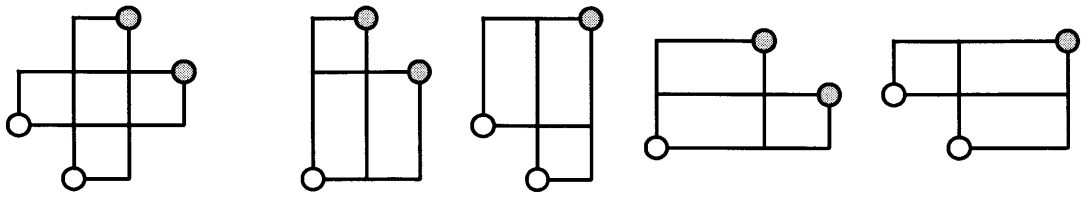


Figure 2-20: Corner-free intersections

Corner-free intersections have a special structure as the following proposition shows.

Proposition 2.5.2. *Let $\mathscr{K}$ be any c.f.i. family. Then $\mathscr{I}(\mathscr{K})$ is a comparability graph.*

Proof. Consider the following relation $\hookrightarrow$ of the rectangles in $\mathscr{K}$. We say that $R \hookrightarrow S$ if and only if

1. R intersects S.

2. R is shorter than S (i.e., the length of R_y is smaller than or equal to the lenght of S_y).

The collection $\mathscr{K}$ is a c.f.i. family, therefore, when (1) holds, the statement in (2) becomes equivalent to $S_y \subseteq R_y$ and equivalent to $R_x \subseteq S_x$. It is easy to check that $\hookrightarrow$ is reflexive and antisymmetric. We can also prove that $\hookrightarrow$ is transitive: suppose $R \hookrightarrow S$ and $S \hookrightarrow T$. It follows that $T_y \subseteq S_y \subseteq R_y$ and $R_x \subseteq S_x \subseteq T_x$. Hence, $R \cap T = R_x \times T_y \neq \emptyset$ and R is shorter than T, so $R \hookrightarrow T$.

Therefore $\hookrightarrow$ is a partial order. To conclude, note that two rectangles in $\mathscr{K}$ intersect if and only if they are comparable, and therefore $\mathscr{I}(\mathscr{K})$ is the comparability graph of $(\mathscr{K}, \hookrightarrow)$. □

Note that Proposition 2.5.1, together with the inclusion-wise minimality of the rectangles in $\mathscr{R}_0$, is equivalent to say that $\mathscr{R}_0$ is a c.f.i. family. Together with the result above, we have all we need to prove the integrality of x^*.

Proposition 2.5.3. *The point x^* is integral. In particular, $\mathscr{R}_0$ is an independent set.*

Proof. Since $\mathscr{R}_0$ is the support of x^*, the linear program $\mathbf{mis}_{\mathrm{LP}}(\mathscr{R}_0)$ also has x^* as optimal solution. But $\mathscr{I}(\mathscr{R}_0)$ is a comparability graph by Proposition 2.5.2, and therefore it is perfect. Using Theorem 2.5.1, it follows that $P(\mathscr{R}_0)$ is integral. To conclude, we only need to prove that x^* is an extreme point of $P(\mathscr{R}_0)$. By contradiction, if x^* is not an extreme point of $P(\mathscr{R}_0)$ then it can be written as convex combination of two points in $P(\mathscr{R}_0) \subseteq P(\mathscr{R})$, contradicting the fact that x^* is extreme point of $P(\mathscr{R})$ □

Since linear programming admits polynomial time algorithms, it follows that:

Theorem 2.5.4. *The jump number of a 2D2C poset can be computed in polynomial time.*

Using the fastest algorithm for linear programming, the running time would be $O(n^8)$, which is slightly better than the $O(n^9)$ dynamic programming algorithm of Dahlhaus for convex posets [10]. We will not discuss algorithmic improvements to our linear programming method, since the combinatorial algorithm presented in the next section is far faster.

A link to the next section

We close this section with a small result that links what we have covered so far to the combinatorial algorithm in the next section.

The dual of $\mathbf{mis}_{\mathrm{LP}}(\mathscr{R})$ is:

$$\mathbf{mhs}_{\mathrm{LP}}(\mathscr{R}) = \min\left\{ \sum_{p\in[n]^2} y_p : \sum_{p\in R} y_p \geq 1, R \in \mathscr{R}; y \geq 0 \right\}.$$

The dual is called the **minimum hitting set** linear program, because it tries to minimize the number of points that hit all the rectangles in $\mathscr{R}$ (a point p **hits** a rectangle R when $p \in R$). Since we were able to find an optimal integral vertex of $\mathbf{mis}_{\mathrm{LP}}(\mathscr{R})$, it is reasonable to ask whether there is also an optimal integral vertex of $\mathbf{mhs}_{\mathrm{LP}}(\mathscr{R})$. More precisely, if we call $\mathbf{mis}_{\mathrm{IP}}(\mathscr{R})$ and $\mathbf{mhs}_{\mathrm{IP}}(\mathscr{R})$ the integer program versions of $\mathbf{mis}_{\mathrm{LP}}(\mathscr{R})$ and $\mathbf{mhs}_{\mathrm{LP}}(\mathscr{R})$, the question is whether the last inequality in the following relations can be turned into a equality:

$$\mathbf{mis}_{\mathrm{IP}}(\mathscr{R}) = \mathbf{mis}_{\mathrm{LP}}(\mathscr{R}) = \mathbf{mhs}_{\mathrm{LP}}(\mathscr{R}) \leq \mathbf{mhs}_{\mathrm{IP}}(\mathscr{R}).$$

Note that the first equality follows from Proposition 2.5.3, while the second equality is an application of strong duality.

What we prove here is that we can obtain the full chain of equalities if, instead of $\mathscr{R}$, we use a c.f.i. family $\mathscr{K}$ (in particular $\mathscr{K}$ satisfies the main properties we shown for $\mathscr{R}_0$). This result can be derived from the perfectness of $\mathscr{I}(\mathscr{K})$, but a more elementary way to prove it uses the fact $\mathscr{I}(\mathscr{K})$ is a comparability graph. The advantage of this approach is that it provides some combinatorial intuition that cannot be obtained directly with linear programming techniques. If $\hookrightarrow$ is a partial order on $\mathscr{K}$ with intersecting rectangles as comparable elements, then independent sets in $\mathscr{K}$ are just antichains in $(\mathscr{K}, \hookrightarrow)$ while collections of rectangles hit by a point are just sets of pairwise intersecting rectangles, and therefore they are chains in $(\mathscr{K}, \hookrightarrow)$. It follows that the following two pairs of problems are equivalent:

- The maximum independent set of a c.f.i. family $\mathscr{K}$ and the maximum cardinality of an antichain in $(\mathscr{K}, \hookrightarrow)$. The latter is the **maximum antichain problem**.
- The minimum hitting set of a c.f.i. family $\mathscr{K}$ and the minimum number of chains in $(\mathscr{K}, \hookrightarrow)$ containing all the elements of $\mathscr{K}$. The latter is the description of the **minimum chain covering problem**.

A classic result from poset theory states that

Theorem 2.5.5 (Dilworth's Theorem). *For any partial order, the maximum cardinality of an antichain equals the size of a minimum chain covering.*

In our context, this result directly translates into:

Theorem 2.5.6. *For any c.f.i. family $\mathscr{K}$,*

$$\mathbf{mis}_{\mathrm{IP}}(\mathscr{K}) = \mathbf{mis}_{\mathrm{LP}}(\mathscr{K}) = \mathbf{mhs}_{\mathrm{LP}}(\mathscr{K}) = \mathbf{mhs}_{\mathrm{IP}}(\mathscr{K})$$

But these equivalences have algorithmic implications, since maximum antichains and minimum chain coverings can be found efficiently using combinatorial algorithms. What we describe here is one of the fastest algorithms for these two problems. They will be used later to solve the maximum independent set and the minimum hitting set of 2D2C posets.

Theorem 2.5.7. *There is a $O(m^{2.5})$ deterministic algorithm for solving the maximum antichain and the minimum chain covering of a poset $P = (X, \preceq)$ with m elements.*

Proof. Note that the minimum chain covering can be transformed into **minimum chain partition** (where the chains are restricted to be disjoint) just by deleting repetitions of elements appearing in multiple chains.

The algorithm for minimum chain partition operates as follows. Given the poset $P = (X, \preceq)$, it constructs a bipartite graph $G = (X \cup X, E)$ (i.e., two copies of the set X form the bipartition) where $(u, v) \in E$ if and only if $u \prec v$. Note that each partition of P into k chains induces a matching in G where pairs of consecutive elements in the chains are edges of the matching. Similarly, any matching can be transformed into a collection of disjoint chains, by putting u and v in the same chain if and only if (u, v) is an edge of the matching.

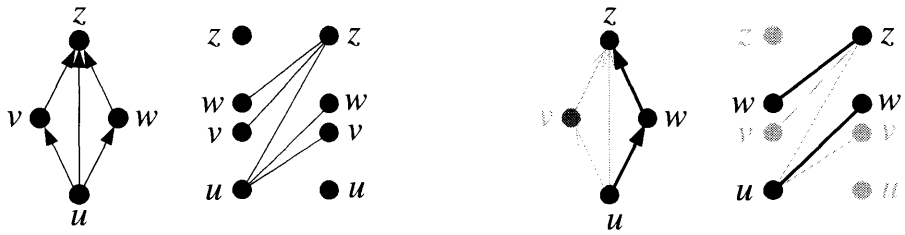


Figure 2-21: Solving minimum chain partitions using matchings
Left side: a poset P in natural representation and its graph G defined in Lemma 2.5.7.
Right side: chains in P are matchings in G.

Since a matching with k elements induces a partition into $n - k$ chains, it follows that the minimum chain partition of P is equivalent to the maximum matching in G. The latter can be found in $O(m^{2.5})$ using the algorithm of Mucha and Sankowsky [31].

The algorithm for minimum chain partition directly uses the equivalence above. In contrast, the algorithm for maximum antichain first finds a **minimum vertex cover** C of G in $O(m^{2.5})$ time, again using the maximum matching algorithm in [31]. The maximum antichain is the set V of elements in X with none of their two copies (in G) in the vertex cover. It is easy to see that V is actually an antichain, because if $u, v \in V$ were comparable, then at one of the extremes of the edge (u, v) should be part of the vertex cover. It is also easy to see that $|V| = |X| - |C|$, and therefore the maximality of the antichain directly follows from König's Theorem [40]. □

Corollary 2.5.1. *The maximum independent set and the minimum hitting set of a c.f.i. family $\mathscr{K}$ can be found in $O(|\mathscr{K}|^{2.5})$ time.*

What we just described is one of the two key elements of the algorithm described in the next section. The other key element is the combinatorial construction of a particular c.f.i. family that plays the role of $\mathscr{R}_0$ in the linear programming algorithm.

2.6 A combinatorial algorithm for 2D2C posets

We now propose a combinatorial algorithm for solving the maximum independent set of a 2D2C poset $P = (A \cup B, \mathscr{R})$. In a nutshell, what we do in this section is to efficiently construct a family $\mathscr{K} \subseteq \mathscr{R}$ with the following properties:

1. It only contains inclusion-wise minimum rectangles.
2. It is a c.f.i. family.
3. Its maximum independent set is a maximum independent set in $\mathscr{R}$.

Note that those are the main properties we have shown for $\mathscr{R}_0$, the c.f.i. family used in the linear programming algorithm (even though for $\mathscr{R}_0$ we ended up proving that it was already an independent set). It follows from the discussion in page 43 that a maximum independent set in $\mathscr{R}$ can be efficiently found by solving the maximum antichain problem in $(\mathscr{K}, \hookrightarrow)$. The main difference with respect to the linear programming algorithm is that $\mathscr{K}$ is determined with a faster combinatorial procedure.

Before going into the details, we need some additional notation. Given a rectangle R, let $\mathbf{A}(R)$ be the bottom-left corner of the rectangle R and $\mathbf{B}(R)$ be the top-right corner of the rectangle R . The notation is chosen so that when R is a rectangle of a 2D2C poset $(A \cup B, \mathscr{R})$, then $\mathbf{A}(R) \in A$ and $\mathbf{B}(R) \in B$ are the two defining vertices of R. However, we will also use this notation for rectangles not in $\mathscr{R}$.

Let $\mathscr{R}_{\downarrow}$ be the set of inclusion-wise minimal rectangles. The construction of the c.f.i. family $\mathscr{K}$ uses an specific order for the rectangles in $\mathscr{R}_{\downarrow}$. We say that R appears before S in the **right-top** order if either

- $\mathbf{A}(R)_x < \mathbf{A}(S)_x$, or
- $\mathbf{A}(R)_x = \mathbf{A}(S)_x$ and $\mathbf{A}(R)_y < \mathbf{B}(S)_y$.

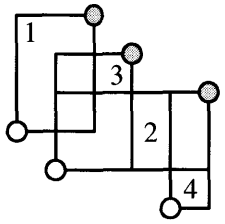


Figure 2-22: Some rectangles labeled in right-top order.

To construct $\mathscr{K}$, we just process the rectangles in $\mathscr{R}_{\downarrow}$ one by one according to the right-top order, adding them to $\mathscr{K}$ if and only if $\mathscr{K}$ remains corner-free. Since this clearly gives a c.f.i. family with inclusion-wise minimal rectangles, all that remains to do is to argue that the maximum independent set in $\mathscr{K}$ is also maximum in $\mathscr{R}$. The proof relies on the dual problem, the minimum hitting set, as we show next.

The flipping subroutine

If we could prove that $\mathbf{mhs}_{\mathrm{IP}}(\mathscr{K}) = \mathbf{mhs}_{\mathrm{IP}}(\mathscr{R})$, we would immediately conclude that a maximum independent set in $\mathscr{K}$ is also maximum in $\mathscr{R}$. This easily follows from the fact that $\mathbf{mhs}_{\mathrm{IP}}(\mathscr{K}) = \mathbf{mis}_{\mathrm{IP}}(\mathscr{K}) \leq \mathbf{mis}_{\mathrm{IP}}(\mathscr{R}) \leq \mathbf{mhs}_{\mathrm{IP}}(\mathscr{R})$.

To prove $\mathbf{mhs}_{\mathrm{IP}}(\mathscr{K}) = \mathbf{mhs}_{\mathrm{IP}}(\mathscr{R})$ we use a constructive procedure that essentially moves the points of a minimum hitting set for $\mathscr{K}$ so that in their new position they not only hit $\mathscr{K}$, but the entire $\mathscr{R}$. As we will discuss in the next section, many of the ideas for this algorithm are already present in the algorithmic proof of Frank for a min-max result of Györi on intervals [13]. Nevertheless, we will give a full description here.

Given two points $p, q \in \mathbb{Z}^2$ with $p_x <_{\mathbb{Z}^2} q_x$, the **flipping** of p and q "moves" these two points to the new coordinates $r = (p_x, q_y)$ and $s = (q_x, p_y)$. Technically, this correponds to delete p and q from some generic set and replace them by r and s.

The flipping algorithm will start from a minimum hitting set $H = H_0$ of $\mathscr{K}$, and will continue to flip pairs of points in H in any arbitrary order, as long as H hits all the rectangles in $\mathscr{K}$. We call a flip **admissible** when this property holds.

We claim that at some point no more flips are admissible. To see this, note that the potential $\phi(H) = \sum_{p \in H} p_x p_y \geq 0$ decreases after the flipping of p and q by

$$p_x p_y + q_x q_y - p_x q_y - q_x p_y = (p_x - q_x)(p_y - q_y),$$

which is greater than or equal to one. Therefore, no more than $\phi(H_0)$ flips can be done. It is easy to see that $\phi(H_0) \leq n^3$, which gives a polynomial bound on the number of flips made by the algorithm.

If we denote by H^* the final set of points obtained, we know that $|H^*| = |H_0|$, and H^* hits all the rectangles in $\mathscr{K}$. The main theorem of this section establishes that H^* also hits $\mathscr{R}_\downarrow$ and, in consequence, $\mathscr{R}$.

Theorem 2.6.1. *[44] H^* is a hitting set for $\mathscr{R}_\downarrow$. In particular,* $\mathbf{mhs}_{\mathrm{IP}}(\mathscr{K}) = \mathbf{mhs}_{\mathrm{IP}}(\mathscr{R})$.

Proof. Suppose that H^* does not hit $\mathscr{R}_\downarrow$. From all the rectangles not hit by H^*, let $R = \Gamma(a,b) \in \mathscr{R}_\downarrow \setminus \mathscr{K}$ be the one that appears last in the right-top order. Also, let $R' = \Gamma(a',b')$ be the first rectangle added to $\mathscr{K}$ that has corner-intersection with R. Finally, let $T = \Gamma(a,b')$ and $S = \Gamma(a',b)$ be the two other rectangles we can form using the same defining vertices. This is depicted in Figure 2-23.

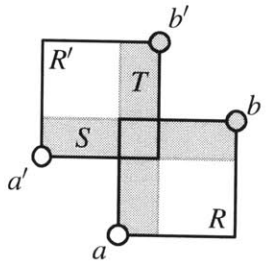


Figure 2-23: Rectangles in the proof of Theorem 2.6.1.

Since R and R' are inclusion-wise minimal rectangles having corner-intersection, it is easy to check that S and T are also inclusion-wise minimal. Our proof has two main claims:

Claim 1: The rectangle S is in $\mathscr{K}$.

Suppose, by contradiction, that S is not in $\mathscr{K}$. Then there must exist a rectangle $U = (c,d)$ appearing before S and having corner intersection with S. It follows that c lies in the zone $Z_1 = [0, a'_x) \times (a'_y, b_y)$, and d lies in the zone $Z_2 = (a'_x, b_x) \times (b_y, n]$ as shown in

Fig. 2-24 (left). Note that (a'_x, b_y) is in both R' and U, and therefore they intersect. But since they are in $\mathscr{K}$, their intersection must be corner-free.. Using this last fact, and that $c \in Z_1$ we conclude that d is either equal to b' or it lies in the zone $Z_3 = (b'_x, n] \times (a'_y, b_y)$. See Fig. 2-24 (right).

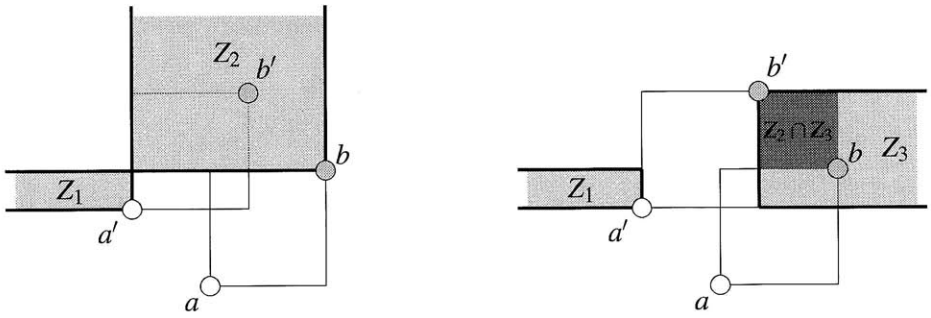


Figure 2-24: Zones in proof of Theorem 2.6.1.

From all the above, we conclude that R and U have corner-intersection. This contradicts the choice of R' since U appears before R' in right-top order. Therefore, Claim 1 is true.

For the next claim, given that T appears after the last rectangle not hit by H^*, it follows that T is hit by a point $q \in H^*$. By Claim 1, there is also a point $p \in H^*$ hitting S. The following claim concludes the proof of the theorem, as it contradicts the properties of H^*.

Claim 2: It is admissible to flip p and q in H^*.

Since $p \in S \setminus R$ and $q \in T \setminus R$ we have that $p_x < a_x \leq q_x$ and $p_y \leq b_y < q_y$. Let $r = (p_x, q_y)$, $s = (q_x, p_y)$ be the potential flipped positions of p and q, and suppose that there is a rectangle $U \in \mathscr{K}$ hit by H^* but not by $(H^* \setminus \{p,q\}) \cup \{r,s\}$. If the rectangle U is hit by p (but not by r or s), then its top-right corner $\mathbf{B}(U)$ must be in the region $[p_x, q_x) \times [p_y, q_y)$. In particular, $\mathbf{B}(U) \in R' \setminus \{a', b'\}$, contradicting the inclusion-wise minimality of R'. On the other hand, if U is hit by q (but not by r or s), then its bottom-left corner $\mathbf{A}(U)$ must be in $(p_x, q_x] \times (p_y, q_y]$. As before, this means that $\mathbf{A}(U) \in R' \setminus \{a', b'\}$, contradicting the inclusion-wise minimality of R'. Therefore, it is admissible to flip p and q. □

From this theorem, and the discussion presented in this section, the main result of this section follows:

Theorem 2.6.2. *For the c.f.i. family $\mathscr{K}$ constructed in this section:*

$$\mathbf{mhs}_{\mathrm{IP}}(\mathscr{K}) = \mathbf{mis}_{\mathrm{IP}}(\mathscr{K}) = \mathbf{mis}_{\mathrm{IP}}(\mathscr{R}) = \mathbf{mhs}_{\mathrm{IP}}(\mathscr{R}).$$

.

With this result, we have shown that finding a maximum independent set in $\mathscr{K}$ is all we need to obtain a maximum independent set in $\mathscr{R}$. It is very easy to see that a naive implementation of this algorithm will run in polynomial time. In the next section, we discuss a very efficient implementation that runs in $O\left((n\log n)^{2.5}\right)$ time.

2.7 Implementation of the combinatorial algorithm

Let us show how to implement the algorithm described in Section 2.6 in $O\left((n\log n)^{2.5}\right)$ time. Recall that for a 2D2C poset $P = (A \cup B, \mathscr{R})$ in rook representation, this algorithm solves $\mathbf{mis}(\mathscr{R})$ by first constructing an specific c.f.i. family $\mathscr{K} \subseteq \mathscr{R}$ and then finding a maximum independent set there.

Recall that by Corollary 2.5.1, the maximum independent set in $\mathscr{K}$ can be found in $O\left(|\mathscr{K}|^{2.5}\right)$ time.

In this thesis, we will not analyze the running time of the minimum hitting set algorithm for $\mathscr{R}$ that is implicit in the flipping subroutine. Naively, this algorithm can be implemented in $O(n^6 \log n)$, but using the improved implementation and analysis by Soto [43], the running time can be reduced to $O\left(n^{2.5}\sqrt{\log n}\right)$.

Constructing the family $\mathscr{K}$

Recall that in the construction of $\mathscr{K}$ we process the rectangles of $\mathscr{R}_{\downarrow}$ in right-top order, adding them to $\mathscr{K}$ as long as this set remains corner-free. In this subsection, $\mathscr{K}'$ denotes the set that is dynamically updated to generate $\mathscr{K}$.

With a direct implementation, this construction takes at least $O(|\mathscr{K}|\,|\mathscr{R}_{\downarrow}|)$ time, as we need to check whether the rectangles to be processed have corner-intersection with the

rectangles in $\mathscr{K}'$. Here we show how to reduce the construction time to $O(n^2 \log n)$ by reducing the time invested in this checking. The critical observation is the following:

Lemma 2.7.1. *Let $R \in \mathscr{R}_{\downarrow}$ be a rectangle about to be processed. Then R has a corner-intersection with a rectangle of $\mathscr{K}'$ if and only if one of the bottom-right corners of the rectangles in $\mathscr{K}'$ is in the interior of R.*

Proof. Since the rectangles are processed in the right-top order, if R has a corner-intersection with a rectangle $R' \in \mathscr{K}'$, then R must contain the bottom-right vertex of R'. For the converse, note that the only type of intersection where the bottom-right vertex of one rectangle is included in the interior of the other one is corner-intersection. See figures 2-18 and 2-20. □

Motivated by the lemma above, as we process the rectangles in $\mathscr{R}_{\downarrow}$ we will dynamically update a collection of lists $\{\mathrm{L}[y]\}_{y=1,\ldots,n}$, where $\mathrm{L}[y]$ contains the x-coordinates of all the bottom-right corners of rectangles $R \in \mathscr{K}'$ satisfying $\mathbf{A}(R)_y = y$. We will always keep these lists sorted from smallest to largest. Since each time we add a rectangle to $\mathscr{K}'$ we need to insert one number into a single list $\mathrm{L}[y]$, the total cost of updating is $O(|\mathscr{K}| \log n)$ using binary search.

Note that all the rectangles $R \in \mathscr{R}_{\downarrow}$ sharing the same defining corner $\mathbf{A}(R)$ are consecutive in right-top order. The improved routine for checking corner-intersections processes all these rectangles in a single batch. From now on suppose that we are currently processing the rectangles R having $\mathbf{A}(R) = a$. For $y = 1, \ldots, n$, let **leftmost**(y) be the minimum number greater than a_x in $\cup_{a_y < y' \leq y} \mathrm{L}[y']$. Using Lemma 2.7.1, it is easy to show that R has corner-intersection with some rectangle in $\mathscr{K}'$ if and only **leftmost**$(\mathbf{B}(R)_y - 1) < b_x$.

Lemma 2.7.2. *For fixed $a \in A$, the function* **leftmost**$()$ *can be computed in $O(n \log n)$ time.*

Proof. For every $y \in \{1, 2, \ldots, n\}$ we can find the smallest number in $\mathrm{L}[y]$ strictly greater than a_x. Call this number LeftMost(y). With binary search, this can be done in $O(n \log n)$ time. Then, using the recursion **leftmost**$(y) = \min\{$**leftmost**$(y-1), \mathrm{LeftMost}(y)\}$, we can compute the function **leftmost**$()$ in additional $O(n)$ time. □

With the **leftmost**() function computed, we process all the rectangles $R \in \mathscr{R}_{\downarrow}$ with $\mathbf{A}(R) = a$, checking whether $\mathbf{leftmost}(\mathbf{B}(R)_y) < \mathbf{B}(R)_x$. If so, R has a corner-intersection with a rectangle $\mathscr{K}$ and is discarded. Otherwise, R is added to $\mathscr{K}$ and $\mathrm{L}(a_y)$ is updated. Note that **leftmost**() does not change during all the batch process. When all rectangles satisfying $\mathbf{A}(R) = a$ has been processed, **leftmost**() is recomputed.

For each batch, we can compute **leftmost**() in $O(n \log n)$ time. Since at most $O(n)$ rectangles having a common defining corner can be inclusion-wise minimal, this function has to be queried $O(n)$ times during a batch. Therefore, the total time spent during a batch is $O(n \log n)$. There are $O(n)$ batches, so the total time is $O(n^2 \log n)$.

Bounding the size of $\mathscr{K}$ and efficiency of the algorithm

From the discussion above, the family $\mathscr{K}$ can be constructed in $O(n^2 \log n)$ time. Finding an independent set in $\mathscr{K}$ requires to build the intersection graph (that can be easily constructed in $O(|\mathscr{K}|)^2$ time), and then solving the associated maximum antichain problem in $O\left(|\mathscr{K}|^{2.5}\right)$ time. We need a good bound for $\mathscr{K}$ in order to obtain an improved analysis. The following result can be found in Soto's thesis:

Lemma 2.7.3 ([43]). *The cardinality of any c.f.i. family in a 2D2C poset with n elements is* $O(n \log n)$.

Using this, we obtain the following running time:

Theorem 2.7.1. *The jump number of 2D2C posets can be solved in* $O\left((n \log n)^{2.5}\right)$ *time.*

We want to remark that the running time can be reduced to $O((n \log n)^{2.38})$ in expectation if we allow randomization. This follows from a faster randomized algorithm for maximum matching.

2.8 The maximum weighted bump problem

Our algorithms for the jump number of 2D2C posets are based on the fact that the number of jumps plus the number of bumps of every linear extension of a poset P is constant. But

there is no weighted version of this equation, so we cannot extend our results to the case where the jumps are weighted. According to what we have done so far, it seems more natural to to extend our results to the case where weights are assigned to the bumps of a poset and the problem is to find the linear extension with maximum weighted bump. We call this the **maximum weighted bump problem**. A similar variant was already considered by Ceroi [6] for 2-dimensional posets[8].

For 2D2C posets $P = (A \cup B, \mathscr{R})$ with weights $\{w_R\}_{R \in \mathscr{R}}$ on their possible bumps, the maximum weighted bump problem is equivalent to find the maximum weighted independent set of the rectangles in $\mathscr{R}$, using w_R as the weight function. Note that in previous sections we have considered the case where w_R is constant for all $R \in \mathscr{R}$.

Unfortunately, the algorithms presented in this work for the unweighted case do no carry over to the weighted case. For example, the uncrossing argument in the linear programming algorithm does not extend. In fact, there is a significant difference in the complexity of the maximum weighted bump number for 2D2C posets.

Lemma 2.8.1 ([44], [43]). *The maximum weighted bump number is NP-hard for 2D2C posets.*

Despite this negative result, we can still provide efficient algorithms for some subclasses of 2D2C posets, as we see in the following subsections.

Bipartite 2-dimensional posets

Recall that for bipartite 2-dimensional posets $P = (A \cup B, \mathscr{R})$ the intersection graph $\mathscr{I}(\mathscr{R})$ is perfect (see Lemma 2.5.1), and therefore we can still solve the maximum weighted independent in $\mathscr{R}$ by finding an optimal vertex of

$$\mathbf{mis}_{\mathrm{LP}}(\mathscr{R}) \equiv \max \left\{ \sum_{R \in \mathscr{R}} w_R x_R \colon \sum_{R \colon q \in R} x_R \leq 1, q \in [n]^2; x \geq 0 \right\} .$$

But we can provide a faster combinatorial algorithm. Recall the following partial order introduced in the proof of Lemma 2.5.1: we say that $R \searrow S$ if and only if R and S are

[8] although they call it "maximum weighted jump problem".

disjoint and either $R_x < S_x$ or $R_y > S_y$ holds. We claimed that the comparability graph of $(\mathscr{R}, \searrow)$ is the complement of $\mathscr{I}(\mathscr{R})$. It follows from here that maximum independent sets in $\mathscr{R}$ correspond to maximum chains in $(\mathscr{R}, \searrow)$, which are just longest paths in $(\mathscr{R}, \searrow)$ (seen as a directed acyclic graph).

Since the longest path problem in directed acyclic graphs can be solved in linear time in the number of nodes and arcs [8], this observation immediately leads to an $O(|\mathscr{R}|^2)$ algorithm. In what follows we describe an improved version of this algorithm that runs in $O(n^2)$ time.

Theorem 2.8.1 ([44]). *There is an $O(n^2)$ algorithm for the maximum weighted bump of bipartite 2-dimensional posets $P = (A \cup B, \mathscr{R})$.*

Proof. For simplicity, let us assume that all the weights are different. Our algorithm exploits the geometric structure of the independent sets in $\mathscr{R}$. Since A and B are antichains in $\mathbb{Z}^2$, the condition $R \searrow S$ implies that $\mathbf{A}(R)_x < \mathbf{A}(S)_x$ and $\mathbf{B}(R)_y > \mathbf{B}(S)_y$. Geometrically, this condition orders the rectangles of any independent set from top-left to bottom-right.

Let $\mathscr{R}^* \subset \mathscr{R}$ be any maximum weighted independent set, and let $R \searrow S \searrow T$ be three consecutive rectangles in $\mathscr{R}^*$. We can then extract the following information about S (see Figure 2-25):

- **Down-right scenario:**

  If $R_y > S_y$ and $S_x < T_x$, then (i) S is the heaviest rectangle with corner $\mathbf{B}(S)$. In particular, S is determined by $\mathbf{B}(S)$.

- **Down-down scenario:**

  If $R_y > S_y$ and $S_y > T_y$, then (ii) S is the heaviest rectangle below R with corner $\mathbf{A}(S)$. In particular, S is determined by $\mathbf{A}(R)_y$ and $\mathbf{A}(S)$.

- **Right-down scenario:**

  If $R_x < S_x$ and $S_y > T_y$, then (iii) S is the heaviest rectangle with corner $\mathbf{A}(S)$. In particular, S is determined by $\mathbf{A}(S)$.

- **Right-right scenario:**

If $R_x < S_x$ and $S_x < T_x$, then (iv) S is the heaviest rectangle to the right of R with corner $\mathbf{B}(S)$. In particular, S is determined by $\mathbf{B}(S)$ and $\mathbf{B}(R)_x$.

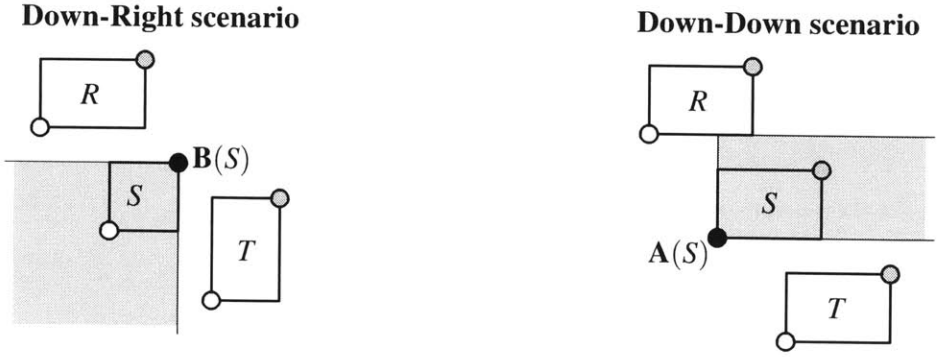


Figure 2-25: Proof of Theorem 2.8.1

For $R \in \mathscr{R}$, let $V(R)$ be the maximum weight of a path in $(\mathscr{R}, \searrow)$ that starts with R. For $a \in A$, let $V_{\downarrow}(a)$ be the maximum weight of a path using only rectangles below a. Similarly, for $b \in B$, let $V_{\rightarrow}(b)$ be the maximum weight of a path using only rectangles to the right of b.

Clearly, $V_{\downarrow}(a) = \max\{V(S)\colon S \text{ rectangle below } a\}$, but from the decomposition into scenarios we can restrict S to be a rectangle below a satisfying properties (i) or (ii). Using this idea, we define:

$$\mathscr{S}_a^{(\mathrm{i})} \equiv \{S : S \text{ is the heaviest rectangle with } \mathbf{B}(S) = b \text{ for some } b \text{ below } a\},$$
$$\mathscr{S}_a^{(\mathrm{ii})} \equiv \{S : \text{ for some } a' \in A,\ S \text{ is the heaviest rectangle below } a \text{ with } \mathbf{A}(S) = a'\},$$
$$\mathscr{S}_b^{(\mathrm{iii})} \equiv \{S : S \text{ is the heaviest rectangle with } \mathbf{A}(S) = a \text{ for some } a \text{ to the right of } b\},$$
$$\mathscr{S}_b^{(\mathrm{iv})} \equiv \{S : \text{ for some } b' \in B,\ S \text{ is the heaviest rectangle to the right of } b \text{ with } \mathbf{B}(S) = b'\}$$

and compute the recursion as follows:

$$V(R) = \max\left\{V_{\downarrow}(\mathbf{A}(R)), V_{\rightarrow}(\mathbf{B}(R))\right\} + w_R,$$
$$V_{\downarrow}(a) = \max\left\{V(S)\colon S \in \mathscr{S}_a^{(\mathrm{i})} \cup \mathscr{S}_a^{(\mathrm{ii})}\right\},$$
$$V_{\rightarrow}(b) = \max\left\{V(S)\colon S \in \mathscr{S}_b^{(\mathrm{iii})} \cup \mathscr{S}_b^{(\mathrm{iv})}\right\}.$$

With some trivial preprocessing we can precompute $\mathscr{S}_a^{(\mathrm{i})}$ for all $a \in A$ in $O(n^2)$ time. We can also precompute $\mathscr{S}_a^{(\mathrm{ii})}$ for all $a \in A$ in $O(n^2)$ time: we fix $a' \in A$, and then traverse the points $a \in A$ from bottom to top. As we do this traversal, we can find the heaviest rectangle S below a satisfying $\mathbf{A}(S) = a'$, for all $a \in A$. This finds one rectangle S belonging to each $\mathscr{S}_a^{(\mathrm{ii})}$, in $O(n)$ time. Iterating now on $a' \in A$, we determine the entire sets $\mathscr{S}_a^{(\mathrm{ii})}$ in $O(n^2)$ time.

After this preprocessing, the rectangles in $\mathscr{S}_a^{(\mathrm{i})}$ and $\mathscr{S}_a^{(\mathrm{ii})}$ can be accessed in $O(1)$ time. The same holds for $\mathscr{S}_b^{(\mathrm{iii})}$ and $\mathscr{S}_b^{(\mathrm{iv})}$, via a similar argument. Since the cardinality of each of these sets is $O(n)$, and there are $O(n)$ values $V_{\downarrow}(a)$ and $V_{\rightarrow}(b)$ to compute, the complete recursion for these terms can be evaluated in $O(n^2)$. Finally, the maximum weight of an independent set is just $\max_R\{V(R)\}$, which can trivially be found with $O(\mathscr{R}) = O(n^2)$ additional time. The independent set itself can be found by backtracking the recursion, giving an overall running time of $O(n^2)$.

If there are repeated weights, we break ties in Properties (i) and (ii) by choosing the rectangle S of smallest height and we break ties in Properties (iii) and (iv) by choosing the rectangle S of smallest width. This does not affect the asymptotic running time. □

We can further improve the running time of the algorithm when the weights are 0 or 1 only. Since just specifying the weights of the rectangles requires $O(\mathscr{R})$ space, this requires certain assumptions on the the way the partial order is given.

Theorem 2.8.2. *The maximum weighted bump problem of bipartite 2-dimensional posets can be solved in $O(n)$ time when the weights are either 0 or 1 and the input is given as a biadjacency matrix M^P in strong ordering form (see Theorem 2.4.1), where we can access the first and last 1 of every row and column in $O(1)$ time.*

Proof. Our algorithm uses a simplified version of the the algorithm for arbitrary weights introduced in Theorem 2.8.1. We completely ignore the rectangles of weight 0 in the following discussion.

Recall that for each $b \in B$, $\mathscr{S}_a^{(\mathrm{i})}$ contains at most a single rectangle S satisfying $\mathbf{B}(S) = b$, the one with minimum height. But $\mathscr{S}_a^{(\mathrm{i})}$ corresponds to the Down-Right scenario, that assumes that the rectangle immediately next to S in the independent set is located to the right

of S. Therefore, we can just delete from $\mathscr{S}_a^{(\mathrm{i})}$ all the rectangles except the one minimizing $\mathbf{B}(S)_x$ (or, equivalently, maximizing $\mathbf{B}(S)_y$). This rectangle can be easily identified in the biadjacency matrix, see Figure 2-26: first, we find the last element b comparable with a. Then we look at the column $\bar{b}$ immediately to the right of b, and find the first row $\bar{a}$ that has a 1. The rectangle we look for is $\Gamma(\bar{a},\bar{b})$, that we denote by S_a.

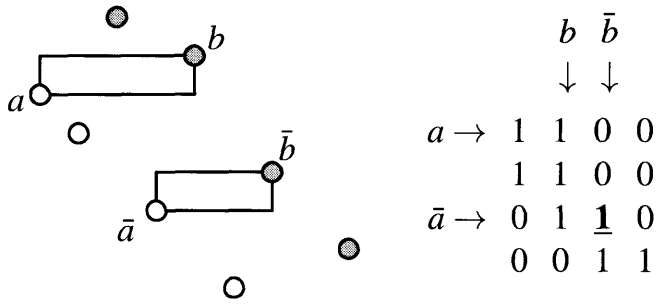


Figure 2-26: Proof of Theorem 2.8.2

Similarly, for each $a' \in A$, $\mathscr{S}_a^{(\mathrm{ii})}$ contains at most a single rectangle S satisfying $\mathbf{A}(S) = a'$, the one with minimum height. But since $\mathscr{S}_a^{(\mathrm{ii})}$ corresponds to the Down-Down scenario, we can just delete from $\mathscr{S}_a^{(\mathrm{ii})}$ all the rectangles except the one maximizing $\mathbf{A}(S)_y$. This rectangle is actually the same as S_a, the remaining rectangle in $\mathscr{S}_a^{(\mathrm{i})}$.

With a similar argument, it is easy to show that $V_{\rightarrow}(b) = V(S_b)$, where S_b is the rectangle in $\mathscr{S}_b^{(\mathrm{iii})}$ that minimizes $\mathbf{B}(S_b)_x$. This gives us a simplified recursion:

$$\begin{aligned} V(R) &= \max\left\{V_{\downarrow}(\mathbf{A}(R)), V_{\rightarrow}(\mathbf{B}(R))\right\} + w_R, \\ V_{\downarrow}(a) &= V(S_a), \\ V_{\rightarrow}(b) &= V(S_b). \end{aligned}$$

To obtain an $O(n)$ algorithm, we just need to access the rectangles S_a and S_b in constant time. This is guaranteed by the hypothesis of the theorem. □

Note that the assumption made about M^P in this theorem is not completely unrealistic. It holds for example if the ones of each row and column of M^P are connected by a double linked list.

Convex posets

In [44], we show that the maximum weighted independent set of convex posets is equivalent to find the **maximum weighted point-interval set** of a collection of intervals. For the latter problem, described in Section 2.9, Lubiw shows a very simple polynomial time algorithm [29] that we now describe in our context.

Suppose that $P = (A \cup B, \mathscr{R})$ is a convex poset such that the points of A lie in the line $x + y = n$ and the point of B lie in the halfspace $x + y \geq n$ (see Lemma 2.4.2). Given two disjoint rectangles R and S in $\mathscr{R}$, let us say that R **precedes** S if either $R_x < S_x$ and $R_y \cap S_y \neq \emptyset$ or if $R_y < S_y$ and $R_x \cap S_x \neq \emptyset$. Since R precedes S implies that $A(R)_x < B(R)_x$, this relation induces no cycles. Therefore, every independent set must contain a rectangle not preceded by any other. This rectangle splits the independent set in two parts .

The algorithm finds the maximum weighted independent set $\mathscr{R}^*$ by guessing a rectangle $T \in \mathscr{R}^*$ that no one precedes, and then recursively finding the maximum weighted independent set of the rectangles lying below T and the rectangles lying to the left of T.

More precisely, for each point $q \in [n]^2$ in the halfplane $x + y \geq n$, let $V(q)$ be the weight of a maximum independent set using rectangles R having their defining vertices $A(R), B(R) \leq_{\mathbb{Z}^2} q$. The following recursion follows from the splitting property just mentioned:

$$V(q) = \max_{a \in A : a \leq q} \{V(q_x, a_y) + V(a_x, q_y) + w(a, q)\},$$

where $w(a,q)$ is the maximum weight of a rectangle $R \in \mathscr{R}$ contained in $\Gamma(a,q)$ (or 0 if there is no such rectangle).

Therefore, the cardinality of the maximum independent set $V((n,n))$, can be found in $O(n^3)$ by dynamic programming. The set itself can be found by backtracking the recursion.

Theorem 2.8.3 ([29], [44]). *The maximum weighted bump number of a convex poset can be solved in $O(n^3)$ time.*

2.9 The related work: point-interval pairs and set-pairs.

The combinatorial algorithm for 2D2C posets presented in Section 2.6 uses many ideas from previous research on seemingly unrelated problems. One of the main contributions of our work with Soto [44] is to provide a common setting where all these ideas become simple and intuitive. In this section we describe these connections.

Point-interval pairs

Györi and Lubiw studied two dual problems associated to a finite collection of intervals $\mathscr{I}$ in $\mathscr{R}$. To define them, we need to introduce some concepts first. A **point-interval pair** in $\mathscr{I}$ is a duple (p,I) where $I \in \mathscr{I}$ and $p \in I$. A collection $\mathscr{J}$ of point-interval pairs in $\mathscr{I}$ is **independent** if for every $(p,I) \neq (p',I') \in \mathscr{J}$ we have $p \notin I'$ or $p' \notin I$. A **basis** of $\mathscr{I}$ is a set of intervals $\mathscr{J}$ so that each interval in $\mathscr{I}$ can be written as union of intervals in $\mathscr{J}$.

Györi [18] showed that the maximum cardinality of an independent set in $\mathscr{I}$ equals the minimum cardinality of a basis of $\mathscr{I}$. This result is in fact equivalent to the equality $\mathbf{mis}_{\mathrm{IP}}(\mathscr{R}) = \mathbf{mhs}_{\mathrm{IP}}(\mathscr{R})$, when $\mathscr{R}$ is the family of rectangles arising from a convex poset. A simple algorithmic proof of this min-max result, using the concept of c.f.i. families, was given by Frank [13]. Our combinatorial algorithm for the jump number of 2D2C posets (see Section 2.6) heavily relies on the ideas of that paper.

Lubiw [29] also studied a weighted version of the maximum independent set of point-interval pairs. The algorithm she provided for this case is equivalent to the algorithm for the maximum weighted bump number of convex posets described in Section 2.8.

Set pairs

In a seminal paper, Frank and Jordán [14] extend Györi's result to **set-pairs**. For this result, we need to introduce some definitions.

A collection of pairs of sets $\{(S_i,T_i)\}$ is **half-disjoint** if for every $i \neq j$, $S_i \cap S_j$ or $T_i \cap T_j$ is empty. A directed edge (s,t) **covers** a set-pair (S,T) if $s \in S$ and $t \in T$. A family $\mathscr{S}$ of set-pairs is **crossing** if whenever (S,T) and (S',T') are in $\mathscr{S}$, so are $(S \cap T, S' \cup T')$ and $(S \cup T, S' \cap T')$.

Frank and Jordán prove that in every crossing family $\mathscr{S}$, the maximum size of a half-disjoint subfamily is equal to the minimum size of a collection of directed-edges covering $\mathscr{S}$. They also give a linear programming based algorithm to compute both optimizers. Later, combinatorial algorithms for this result were also given (e.g. [3]). See Végh's Ph.D. thesis [48] for related references.

Theorems 2.5.4 and 2.6.2 can be seen as non-trivial applications of Frank and Jordán's result. Given a 2D2C poset $G = (A \cup B, \mathscr{R})$, consider the family of set-pairs $\mathscr{S} = \{(R_x, R_y) : R \in \mathscr{R}_{\downarrow}\}$. It is easy to check that this family is crossing, that half-disjoint families of $\mathscr{S}$ correspond to independent sets in $\mathscr{R}_{\downarrow}$ and that coverings of $\mathscr{S}$ by directed-edges correspond to hitting sets for $\mathscr{R}_{\downarrow}$. We remark that this reduction relies heavily on the geometric interpretation of 2D2C posets we have presented in this thesis, and that our proofs are self-contained and simpler than the ones used to prove the broader result of Frank and Jordán.

The combinatorial algorithm presented in Section 2.6 also comprises algorithmic ideas from other applications of Frank and Jordan's result, such as the algorithm of Frank and Végh [15] for connectivity augmentation. Our description is tailored to the instances considered in this work, leading to a simpler description of the algorithm and a simpler running time analysis.

Chapter 3

The Joint Replenishment Problem

The Joint Replenishment Problem (**JRP**) is a fundamental model in inventory management which captures the trade-off between ordering and holding costs in a supply chain. The variant we consider in this chapter is the most basic version of a larger family of joint replenishment problems, but it captures well the essence of the difficulty of the problem.

In this chapter, we introduce the **JRP** and provide very simple approximation algorithms. The **JRP** is one of the main topic of the next chapter, although there it will be studied from the perspective of complexity. The results presented in this chapter are joint work with Andreas S. Schulz [41].

Before going into the mathematics, let us see the problem in action.

An introductory example

A manufacturing facility produces equipment using two commodities **A** and **B**. The production flow is such that the weekly demand for each commodity is constant, and the facility regularly order them from a single supplier. Each time the facility places an order for these commodities, the supplier decomposes the bill as follows:

- A constant shipping cost to deliver the order.
- A constant transaction cost if the order includes **A** and another constant transaction cost if the the order includes commodity **B**.

- The (variable) cost of the commodities themselves. We assume that the cost of each commodity is proportional to the amount of commodity acquired.

The facility takes the acquired inventory to a nearby warehouse that weekly charges each commodity proportionally to the volume stored. The problem of the facility is to minimize the average costs over time associated with the inventory replenishment.

Let us add some fictitious data to this example. Note that we are not including the variable cost of the commodities themselves: the constant weekly demand per commodity implies that this cost is constant on average.

	Commodity	
	A	**B**
Weekly demand	2	2
Weekly storage cost per unit of commodity (in $)	2	1
Transaction cost per order including commodity (in $)	8	9
Shipping cost per order (in $)	3	

Figure 3-1: Demand and cost information for the manufacturing example.

In the following table we propose several ways to replenish the inventory and their respective weekly average costs in the long run:

Replenishment strategy	Shipping	Transactional		Storage		Total
		A	**B**	**A**	**B**	
(1) Order **A** every 2 weeks **B** every 2 weeks (in $)	1.5	4	4.5	2	1	13
(2) Order **A** every 3 weeks **B** every 3 weeks (in $)	1	2.66	3	4	2	12.66
(3) Order **A** every 4 weeks **B** every 4 weeks (in $)	0.75	2	2.25	6	3	14
(4) Order **A** every 2 weeks **B** every 3 weeks (in $)	2	4	3	2	2	13
(5) Order **A** every 2 weeks **B** every 4 weeks (in $)	1.5	4	2.25	2	3	12.75

Figure 3-2: Some inventory replenishment alternatives for the manufacturing facility and their respective weekly average costs.

There are several ideas we can illustrate with this example. In all the strategies, each commodity places its orders periodically. We call such replenishment strategies **cyclic**. Strategy (5) is such that the smallest replenishment period is a divisor of the largest one, so it is called **nested**. In real life, these strategies are easy to implement and they may be preferred to more complex proposals. We will use the term **dynamic** when we consider strategies that are not periodic. All the terms in bold are standard in the literature.

Now let us consider the costs. There should be a trade-off between shipping plus transaction costs and storage costs, in the sense that a decrease in one of these costs implies an increase in the other one. Going a little bit further with this observation, it is possible to argue that extremely frequent or infrequent replenishments is never[1] a good strategy.

Another insight this example provides is the following. Consider strategies (4) and (5). Although the total cost of (5) is lower than the total cost of (4), the sum of transaction and storage costs of (5) are higher than those in (4). The triumph of (5) over (4) is due to the "benefit of coordinated replenishments" that reduces the transportation costs. This indicates that cyclic solutions that exploit this benefit must have replenishment periods that are "far" from being coprime, which is the key idea introduced in Chapter 4.

The **JRP** version we study in this chapter is easy to describe, yet the simplicity of its definition strongly contrasts with the difficulty of its solution. Being such a fundamental problem, the **JRP** has been thoroughly studied since the seventies, but it seems fair to say that our understanding of the problem is far from a satisfactory level. In fact, it is not known whether there are (theoretically) exact efficient algorithms for this problem. This work addresses two questions related to the existence of exact efficient algorithms for the **JRP**.

3.1 Mathematical formulations of the JRP

In this section, we describe several variants of the **JRP** that differ in the set of feasible solutions. Here we give precise definitions for most of the variants considered in the chapter, and also in Chapter 4.

[1] Except for trivial cases

Defining the JRP. Finite Horizon v/s Stationary.

Formally, in the **JRP** a facility has certain demand for n **commodities** labeled $\mathscr{I} = \{1, \ldots, n\}$ during a **time horizon** $[0, T)$, where T may be equal to $+\infty$. The facility acquires these commodities from a single supplier.

- For each commodity $i \in \mathscr{I}$, the facility faces a constant **demand rate** $d_i \in \mathbb{Z}_+$ per unit of time that must be fulfilled on time (this is usually called a **no backlogging** condition). But the facility can acquire commodity ahead of the time it is needed.

- The facility satisfies the demand by placing **orders** to the supplier. Each order may include multiple commodities at the same time. The **ordering cost**, is the sum of the **individual ordering costs** $K_i \in \mathbb{Z}_+$ associated with each commodity involved in the order plus a **joint ordering cost** $K_0 \in \mathbb{Z}_+$ that is independent of the specific commodities requested in the order.

- The acquired inventory is stored in a local warehouse until the time it is required. The warehouse charges a **holding cost** at a **holding cost rate** $h_i \in \mathbb{Z}_+$ per unit of commodity i and per unit of time.

- We assume that orders are delivered instantly. This is called a **no lead time** assumption.

The **JRP** asks for an ordering strategy in $[0, T)$, so that demands are fulfilled and the sum of ordering and holding cost is minimized. Before stating the mathematical model, there are some consequences of the assumptions above that we need to discuss. For simplicity, we only describe them for the case $T < \infty$, which is called the **finite horizon** model. As we observed in the manufacturing example, given that the demand is constant, the facility must order at least $d_i T$ specify interval units of commodity $i \in \mathscr{I}$. There is also no reason to order more than $d_i T$ units, as wasted commodity can be backtracked and eliminated from the source order, decreasing the total cost. This is why the minimization in the **JRP** does not include the costs of the commodities themselves, under the assumption that the per-unit price of commodity is constant. By a similar argument, the demand between

two consecutive o_1 and o_2 must be fulfilled entirely by o_1. Therefore, the times and the compositions of the orders uniquely define the quantities to order.

With these considerations in mind, in an optimal solution we can define orders and their associated costs in the finite horizon case. A **joint order** $(\mathscr{I}', t)$ is a request for a subset of commodities $\emptyset \neq \mathscr{I}' \subseteq \mathscr{I}$ at a specific point in time $t \in [0, T)$. A **schedule** $S = \{(\mathscr{I}'_1, t_1), (\mathscr{I}'_2, t_2), \ldots, (\mathscr{I}'_N, t_N)\}$ is a finite sequence of **joint orders** where $0 = t_1 < t_2 < \ldots < t_N < T$. The costs associated with a schedule $S = \{(\mathscr{I}'_1, t_1), (\mathscr{I}'_2, t_2), \ldots, (\mathscr{I}'_N, t_N)\}$ are defined as follows:

- The **total joint ordering cost** of S is $C_{\text{ord}}^{\text{joint}} \equiv NK_0$.
- The **individual ordering cost of a commodity** $i \in \mathscr{I}$ is equal to

$$C_{\text{ord}}^{\text{indiv}}(i) \equiv K_i \left| \{ j : i \in \mathscr{I}'_j \} \right|.$$

  In some sections of this chapter, we use n_i as a shortcut for $\left| \{ j : i \in \mathscr{I}'_j \} \right|$.

- The **individual holding cost of a commodity** $i \in \mathscr{I}$, denoted as $C_{\text{hold}}(i)$, is equal to

$$\frac{d_i h_i}{2} \sum_{j=1}^{n_i} (t_{j+1}^i - t_j^i)^2,$$

  where $0 = t_1^i < t_2^i < \ldots < t_{n_i}^i < T$ are the times where commodity i is ordered and $t_{n_i+1}^i = T$.

Finally, we also define the **total individual ordering cost** $C_{\text{ord}}^{\text{indiv}}$ as $\sum_{i \in \mathscr{I}} C_{\text{ord}}^{\text{indiv}}(i)$, and the **total holding cost** C_{hold} as $\sum_{i \in \mathscr{I}} C_{\text{hold}}(i)$. The objective of the **JRP** is to minimize the **total cost** $C[S] = C_{\text{ord}}^{\text{joint}} + C_{\text{ord}}^{\text{indiv}} + C_{\text{hold}}$. Note that all the costs defined above depend on the schedule S, but this dependence is not reflected in the notation. We will explicitly state the schedule only if it is not clear from the context.

We use the term **inter-replenishment period** to describe the interval between two consecutive orders of a particular commodity (or, in the context of joint replenishments, the interval between two consecutive joint orders). The term **inter-replenishment time** also means the same thing, although it may refer also to the length of the interval.

When $T = \infty$, we obtain a limit case of the finite horizon model usually called the **stationary** model. The main differences with respect to the finite horizon model is that a schedule S is now a **countable** sequence of orders, and the costs must be averaged over time in order to obtain a meaningful cost expression. For example, the **average** total cost per unit of time $\bar{C}[S]$ becomes

$$\bar{C}[S] = \lim_{T\to\infty} \frac{1}{T} C[\{(\mathscr{I}_i', t_i) \in S : t_i \leq T\}].$$

In a similar fashion, we can also define the average ordering cost per unit of time, the average holding cost per unit of time, etc. Note that these limits may not exist. However, we will always work with schedules where these limits exist and they are easy to compute.

Finally, there is another way to classify the orders according to the times they are allowed to be placed. We have defined the **JRP** in the **continuous time** model, where joint orders can be placed anywhere on $[0,T)$. But we can also define a **discrete** model where orders can be placed anywhere on $[0,T) \cap p\mathbb{Z}_+$ for some fixed value of p. Here, $p\mathbb{Z}_+ \equiv \{kp : k \in \mathbb{Z}_+\}$ denotes the set of non-negative multiples of p. The discrete model is more suitable for scenarios where orders are processed daily, weekly or with any fixed periodicity. We will use discrete models in Chapter 4.

The restriction to "simple" schedules: the general integer policy.

An arbitrary schedule is sometimes called a **dynamic** schedule. Since a highly irregular dynamic schedule may be difficult to implement in practice, it is natural to add some structure to the feasible schedules. In **JRP** jargon, it is customary to call any subset of the dynamic schedules a **policy**.

In the **JRP** with **general integer policies** we solve the stationary case, assuming that joint orders can be placed only at multiples of a **base period** p (to be determined), and that each commodity $i \in \mathscr{I}$ is periodically replenished every $k_i p$ units of time, for some $k_i \in \mathbb{Z}_+$.

An accurate mathematical description of the scenario above is the following formula-

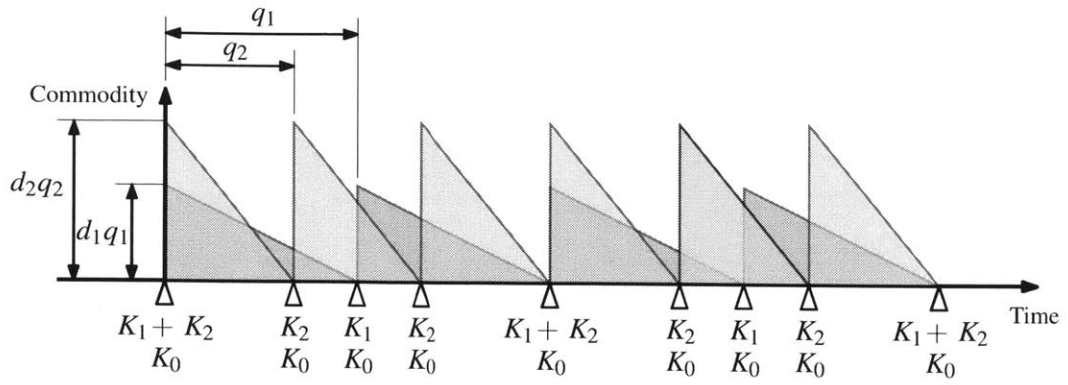


Figure 3-3: GICF model. The plot shows the amount of commodity in storage over time, when two commodities labeled 1 and 2 are replenished every q_1 and q_2 units of time, respectively. The ordering costs are shown below the horizontal axis.

tion, that corresponds to the **general integer model with correction factor (GICF)**

$$\begin{aligned} \min \quad & \frac{K_0\Delta\left(k_1,\ldots,k_{|\mathscr{I}|}\right)}{p} + \sum_{i\in\mathscr{I}}\left(\frac{K_i}{q_i}+\frac{1}{2}h_i d_i q_i\right) \\ \text{s.t.} \quad & q_i = k_i p \\ & k_i \in \mathbb{Z}_+ \\ & p > 0, \end{aligned} \tag{GICF}$$

where $\Delta\left(k_1,\ldots,k_{|\mathscr{I}|}\right)/p$ is the average per-unit-of-time number of joint orders effectively placed. This model is attributed to Dagpunar [9].

Let $\mathrm{lcm}(\cdot)$ denotes the least common multiple of its arguments. For completeness, we prove the following expression for the average number of joint orders.

Lemma 3.1.1.

$$\Delta\left(k_1,\ldots,k_{|\mathscr{I}|}\right) = \sum_{i=1}^{|\mathscr{I}|}(-1)^{i+1}\sum_{I\subseteq\mathscr{I}:|I|=i}\mathrm{lcm}(k_i : i\in I)^{-1}.$$

Proof. We can assume that $p=1$ by a scaling argument. Note that commodity i is replenished at times in the set $A_i \equiv k_i\mathbb{Z}_+$, and therefore the set of joint replenishment times is $A \equiv \bigcup_{i\in\mathscr{I}} A_i$. For $t>0$, let $A_i^t \equiv A_i\cap[0,t]$ and $A^t \equiv A\cap[0,t]$ be the restriction of those sets

to the interval $[0,t]$. Using the inclusion-exclusion formula

$$\frac{1}{t}\left|A^t\right| = \frac{1}{t}\sum_{i=1}^{|\mathscr{I}|}(-1)^{i+1}\sum_{I\subseteq\mathscr{I}:|I|=i}\left|\bigcap_{i\in I}A_i^t\right|.$$

Note that for any $I \subseteq \mathscr{I}$, $\bigcap_{i\in I}A_i^t = (\text{lcm}(k_i : i \in I)\cdot\mathbb{Z}_+)\cap[0,t]$, and therefore, $\frac{\left|\bigcap_{i\in I}A_i^t\right|}{t}$ converges to $\text{lcm}(k_i : i \in I)^{-1}$ as $t \to \infty$. Taking the limit in the equality above, we obtain

$$\Delta\left(k_1,\ldots,k_{|\mathscr{I}|}\right) = \lim_{t\to\infty}\frac{1}{t}\left|A^t\right| = \sum_{i=1}^{|\mathscr{I}|}(-1)^{i+1}\sum_{I\subseteq\mathscr{I}:|I|=i}\text{lcm}(k_i, i\in I)^{-1}.$$

□

It is easy to check now the correctness of the mathematical formulation (GICF). Fix a commodity $i \in \mathscr{I}$. The conditions $q_i = k_i p$, $k_i \in \mathbb{Z}_+$ and $p > 0$ are exactly the restrictions enforced by the general integer policy. According to the general integer model, the commodity is replenished every q_i units of time, raising its inventory to $d_i q_i$ just after replenishment and decreasing it linearly until the next replenishment at a rate $-q_i$ per unit of time. This means that the average individual holding cost is $\frac{1}{2}h_i d_i q_i^2$ every q_i units of time, and the average individual ordering cost is K_i every q_i units of time, which are exactly the costs in (GICF). Finally, Lemma 3.1.1 guarantees that the joint ordering costs are also correctly accounted for.

The **GICF** model is complicated to analyze because of the Δ term. Ignoring Δ (i.e. setting the joint ordering cost rate to K_0/p in (GICF)) defines the **general integer** model (**GI**). Note that this change is equivalent to assume that K_0 is paid at every multiple of the base period.

$$\begin{aligned} \min \quad & \frac{K_0}{p} + \sum_{i \in \mathscr{I}} \left(\frac{K_i}{q_i} + \frac{1}{2} h_i d_i q_i \right) \\ \text{s.t.} \quad & q_i = k_i p \\ & k_i \in \mathbb{Z}_+ \\ & p > 0. \end{aligned} \tag{GI}$$

We have just defined the **GI** and **GICF** formulations in the **variable base** model. Both formulations have a variant where the base p is restricted. In the **fixed base** version of the **GI** formulation, we require p to be a multiple of some constant B. In the **fixed base** version of the **GICF** formulation, we require p to be fixed. We could have also called these versions "discrete", but we decided to stick with the standard nomenclature for these variants.

Figure 3-4 summarizes the different models in the stationary case.

		Stationary model ($T = \infty$)	
	Dynamic	**GICF**	**GI**
Continuous	• Orders at any time • Average costs must converge	• Periodic replenishments • Joint orders restricted to multiples of p • Accurate joint ordering costs using the Δ term	• Periodic replenishments • Joint ordering costs paid at every multiple of p
Fixed Base	• Not considered here	• As above, but now p is a parameter	• As above, but now p is a multiple of some parameter B

Figure 3-4: Variants of the **stationary JRP** considered in chapters 3 and 4, with their main characteristics. We are not including the two finite horizon models (continuous and discrete).

3.2 Some remarks about approximability and complexity

Two concepts of approximation

In this work we focus on two different classes of approximations. The first one is the traditional notion of α-approximation algorithms: given an instance of **JRP** and a policy P, an α-approximation algorithm must run in polynomial time and must output a schedule satisfying P with cost at most α times the optimum within that class of policies. Of course, smaller values of α mean better approximations, with $\alpha = 1$ corresponding to an exact algorithm. In a fully polynomial time approximation scheme (FPTAS), for every $\varepsilon > 0$ there is an $1+\varepsilon$-approximation algorithm running in polynomial time on the input size and $1/\varepsilon$.

But in the **JRP** it is more important to understand the increase in the cost due to the use of a particular policy instead of the best dynamic one. We say that a policy P approximates the dynamic policy by a factor of α if the optimal schedule satisfying P has a cost at most α times the optimal cost with dynamic policies.

About complexity

In the **JRP** with continuous time and finite horizon, the input is a number T and list of $3|I|$ numbers (3 per commodity). If U is the maximum number of this list, then the input size is $O(\log T + |\mathscr{I}| \log U)$. It has not been ruled out that the optimal solution for this problem is a "highly irregular schedule" that needs a description of size $\Omega(T)$. This means that just describing the solution would take exponential time. The approach we take here to develop approximate solutions in polynomial time is to impose a periodicity condition on the solutions we report so that we can describe them using polynomial space.

A similar situation occurs in the stationary case with dynamic policies: the size of the optimal dynamic schedule could be exponential in the size of the input. This issue disappears when we restrict ourselves to general integer solutions, as they are defined by their (constant) inter-replenishment periods.

Finally, we want to point out that there is another version of the joint replenishment

problem that goes by the name "JRP" alone. This version is substantially different because demands can fluctuate over time: for each $j \in [0,T] \cap \mathbb{Z}_+$, there is a demand $d_i^j \in \mathbb{Z}_+$ for commodity i. The rest of the model is essentially defined in the same way as in the **JRP** with continuous time and finite horizon[2]. This problem is known to be **NP**-hard [2], but this result is unlikely to imply a similar one for any of the problems we consider here. One reason is that every **JRP** variant with constant demands and finite horizon has input size proportional to $O(\log T)$, while the input of the variable demand case is a list of $O(T)$ demand values per commodity. This strongly suggests that there is no polynomial time reduction between these two problems, as their inputs have incomparable sizes.

Finally, we want to remark that for all the variants of the **JRP** considered in this work, no hardness results were previously known. In particular, it is not known whether they can be solved in polynomial time.

3.3 Some differences between GI and GICF

We now describe certain differences between the two main stationary models that are critical for our work. We mainly focus on the continuous versions of both models.

The natural decision variables for the **JRP** with general integer policies are the inter-replenishment times per commodity. Once we know this set of values, the joint replenishments are uniquely defined, and therefore p is not really a parameter of the problem. How do we decide the parameter p then?

In the **GICF** model, given the inter-replenishment times per commodity $q_i = \frac{a_i}{b_i}$, $i \in \mathscr{I}$ where $a_i, b_i \in \mathbb{Z}^+$ are coprime[3], we need to find $p > 0$ such that $k_i b_i p = a_i$ for some k_i integer. It follows that $1/p$ must be a multiple of b_i and p must be a divisor of a_i, for each $i \in \mathscr{I}$. The largest value satisfying both properties is $p^* = \frac{\gcd(a_i : i \in \mathscr{I})}{\mathrm{lcm}(b_i : i \in \mathscr{I})}$, where $\gcd(\cdot)$ denotes the greatest common divisor of its arguments. But note that any integer divisor of p^* also satisfies the required properties. Under **GICF**, the cost remains the same no matter which of those values of p we pick.

[2]Except that since demands are discrete, the holding cost function is discretized.

[3]If the inter-replenishment times are irrational, it is not clear how to choose p in certain cases.

In the **GI** model, the situation is completely different. Under the same conditions we still need to find $p > 0$ such that $k_i b_i p = a_i$. But now it is easy to see that if we want the best possible estimate for the joint ordering cost we need to choose the largest possible value of p. In fact, we can make the cost artificially large by choosing p arbitrarily small. So in the **GI** model we must pick $p = \frac{\gcd(a_i : i \in \mathscr{I})}{\text{lcm}(b_i : i \in \mathscr{I})}$.

From the previous discussion, it is easy to build schedules where the **GI** model gives an average joint ordering cost completely different than then **GICF** model. For example, if $|\mathscr{I}| = 2$ and the inter-replenishment times $q_1 \neq q_2$ are two very large prime numbers, then p must be equal to 1. Therefore, the **GI** model reports an average joint ordering cost of K_0, while the **GICF** model reports an accurate average joint ordering cost of $K_0 O\left(\frac{1}{\min(q_1, q_2)}\right)$. This bad behavior, however, disappears when we are close to the optimal **GI** policy. This follows from the fact that the continuous **GI** model approximates the dynamic stationary model by a factor of ≈ 1.02 [32, 39].

While the good approximation achieved by **GI** policies is a strong reason to study it, researchers have focused on the **GI** model almost completely neglecting the **GICF** model. This lack of research on the **GICF** model seems to be the byproduct of a research trend. For example, most papers working on the **JRP** hardly, if at all, mention the **GICF** as a reasonable alternative to the **JRP** with general integer policies. Prior to our work in Chapter 4, there was no evidence forbidding an scenario where **GICF** is polynomial while **GI** is not.

3.4 A quick tour on the JRP

When we say "quick", we certainly really mean it. The **JRP** is one of the most well studied more studied in Inventory Management, and covering all the associated literature would be like writing a small book by itself. Here we focus on what has direct relevance with our work.

The single commodity case

When restricted to one commodity, the **JRP** is equivalent to the **Economic Lot Sizing (ELS)** problem. Introduced at the beginning of the twentieth century, the **ELS** models a simple trade-off between ordering and holding costs. The problem considers a facility facing a constant demand d for a single commodity that is met by placing orders to a supplier with unlimited stock. Assuming a fixed ordering cost K per order placed, and a holding cost rate h per unit of commodity stored per unit of time, the facility must choose the inter-replenishment period q so as to minimize the average cost rate

$$\frac{K}{q} + \frac{hdq}{2}.$$

Clearly, the optimal solution is $q^* = \sqrt{\frac{2K}{hd}}$ and the total cost is $\sqrt{2Khd}$.

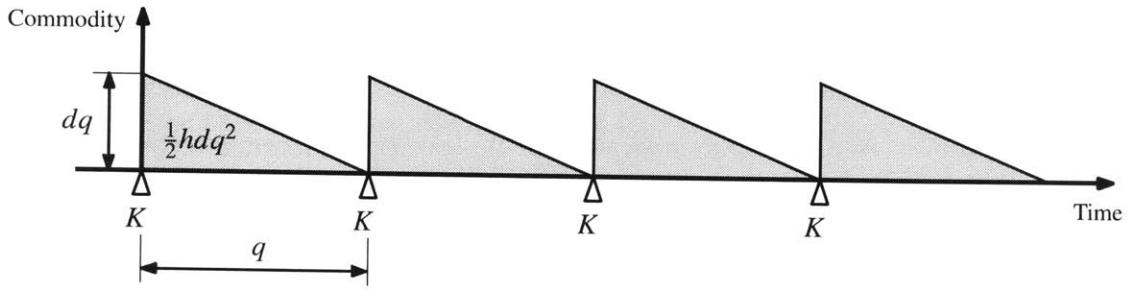


Figure 3-5: ELS model. The picture illustrates the amount of commodity in storage over time, if the commodity is replenished every q units of time. The total holding cost between replenishments is proportional to the area of each gray triangle.

It is easy to check that the Economic Lot Sizing is equivalent to the stationary and continuous **JRP** with a single commodity. It is also equivalent[4] to the stationary and continuous **JRP** with multiple commodities and no joint ordering costs. For convenience, we assume from now on that $\widetilde{K} \equiv K_0 + \sum_{i \in \mathscr{I}} K_i$ and $\widetilde{H} \equiv \sum_{i \in \mathscr{I}} d_i h_i$ are both strictly positive. If not, it is easy to check that the problem reduces to **ELS**.

An interesting result regarding this model is the performance of the **power-of-two policy**. In a power-of-two policy, the value q is restricted to be a power of two multiple of a base period p chosen beforehand, that is, $q = p2^k$ with $k \in \mathbb{Z}_+$. It can be shown that the optimal power-of-two policy has a cost within $\approx 6\%$ of the cost of the optimal solution [21].

[4]Under polynomial time reductions.

The stationary case

In a certain way, the **JRP** is the simplest non-trivial generalization of the **ELS** problem. However, even in the stationary case, optimizing the coordinated replenishments is so complicated that extra assumptions (policies) on the schedule are usually needed in order to obtain efficient and implementable solutions.

Among the strongest policies is the **rotation cyclic** policy, which forces every commodity to share the same constant inter-replenishment interval. This policy is very easy to implement in practical applications. Also, computing the optimal solution under this policy is equivalent to an ELS problem with ordering cost $\widetilde{K}$ and holding cost rate $\widetilde{H}$ per unit of time. Therefore, the optimal rotation cyclic policy has cost $\sqrt{2\widetilde{K}\widetilde{H}}$. This policy does not offer any theoretical guarantee of its performance..

A less restrictive policy is the **power-of-two** policy, that forces each commodity to have a constant inter-replenishment period which is a power of two multiple of some base value p chosen beforehand. Jackson et al. [21] prove that this policy approximates the dynamic ones within a 6%. A very important feature of this policy is that its inter-replenishment times are pairwise nested: for each pair of commodities, one of their inter-replenishment times is always an integer multiple of the other one. This property is highly desirable in practice.

The general integer policy relaxes the power-of-two policy by allowing every commodity to choose its own constant inter-replenishment interval. This policy is not pairwise nested anymore, but it is still "highly" periodic. Since the fixed base **GI** model includes power-of-two policies, the 1.06 approximation holds for this class as well. But the approximation can be improved to 1.02 in the continuous **GI** model [32]. Again, these approximations are taken with respect to the optimal dynamic solution.

Unfortunately, neither the power-of-two nor the general integer policy are known to be computable in polynomial time, so heuristics procedures have been developed to solve both of them efficiently. In the next paragraph we cover some of these heuristics.

Heuristics for continuous general integer policies. Many heuristics for the continuous **GI** model are based on a two step procedure where some values of p are decided first,

and then the inter-replenishment periods q_i are determined independently per commodity. Kaspi and Rosenblatt [23] proposed one of the first algorithms of this type, which **approximately** solves the continuous **GI** problem for several values of p and then picks the solution with minimum cost. They propose to pick values of p that are equispaced in a range $[p_{\min}, p_{\max}]$ known to contain the optimal p, but they do not specify how many values of p to use. For completeness, we now state and prove two possible values for $p_{\min}$ and $p_{\max}$.

Lemma 3.4.1. *Any optimal value of p^* for the **GI** model satisfies*

$$p_{\min} = \frac{K_0}{\sqrt{2\widetilde{K}\widetilde{H}}} \quad \leq \quad p^* \quad \leq \quad p_{\max} = 2\sqrt{\frac{2\widetilde{K}}{\widetilde{H}}}. \tag{3.1}$$

whenever $\widetilde{K} = K_0 + \sum_{i \in \mathscr{I}} K_i$ and $\widetilde{H} = \sum_{i \in \mathscr{I}} d_i h_i$ are strictly positive.

Proof. Note that any **GI** schedule with $p < p_{\min}$ has to pay at least the joint ordering costs $\sqrt{2\widetilde{K}\widetilde{H}}$ and any **GI** schedule with $p > p_{\max}$ has to pay at least the holding costs

$$\frac{1}{2} \sum_{i \in \mathscr{I}} d_i h_i k_i p \geq \frac{1}{2} \widetilde{H} p_{\max} = \sqrt{2\widetilde{K}\widetilde{H}}.$$

These costs are already larger than the total cost obtained with the optimal rotation cyclic policy. □

Back to our review, Wildeman et al. [49] transform the algorithm of Kaspi and Rosenblatt into an heuristic that produce a sequence of solutions whose cost converges to the optimal cost with continuous **GI** policies. The main two differences are that they **exactly** solve the problem for a sequence of values of p, and that this sequence is determined using a Lipschitz optimization procedure that ensures convergence to the optimal cost. While they claim that this procedure is more efficient than linear search in $[p_{\min}, p_{\max}]$, they do not establish any guarantee about the speed of convergence.

A completely different approach uses the convexity properties of the continuous **GI** model. Let $C(p)$ be the optimal cost of a continuous **GI** policy with a given base p. It is easy to see that $C(p)$ is piecewise convex and that the local optima between breakpoints can

be computed easily. This would lead to an efficient algorithm if the number of breakpoints is polynomial, which unfortunately is not known to be true. Yet, Lu and Posner [37] show that for every $\varepsilon > 0$ it is possible to slightly modify the cost function in the **GI** model so that the two properties above hold[5], the number of breakpoints is $O(n/\sqrt{\varepsilon})$ and the optimal schedule according to this modified cost function is within $1+\varepsilon$ of the optimal **GI** cost. This gives an FPTAS for continuous **GI** policies.

The GICF model. The joint replenishment problem with general integer policies was first introduced by Goyal [16]. He assumes periodic joint orders (some of them possibly empty), which essentially translates into the continuous **GI** formulation. This assumption was controversial at that time, as the consequences of using empty joint orders was not clear at all. Dagpunar [9] strongly criticizes Goyal's formulation and proposes the continuous **GICF** model. Goyal [17] replies back stating that it is hard to implement a solution complying with the **GICF** cost model, since joint replenishments will need follow a non-regular pattern. He also argues that finding a good solution for the **GICF** is harder than finding a good solution for **GI**, and therefore the model should be defined directly as in the **GI** model[6]. While both, Dagpunar and Goyal arguments are valid, the research has mainly focused on the **GI** model. But at the same time, it is rarely mentioned that empty replenishments are charged. Some papers, when defining the **GI** model, describe the problem with a sentence like "a processing cost is charged each time an order is placed", which is highly misleading. We find this very unusual, and somewhat confusing.

There are very few papers studying the **GICF** model in more detail. Most of them state simple facts, like those mentioned in Section 3.3. A more complete study is given by Porras and Dekker [36]. They show that the inclusion of the correction factor significantly changes the replenishment cycles k_i and the joint replenishment period p. Moreover, they prove that the optimal solution of **GICF** has cost strictly less than $\sum_{i\in\mathcal{I}}\sqrt{2(K_0+K_i)/h_i}$, which is the cost obtained assuming that individual orders cost K_0+K_i instead of just K_i, but joint ordering costs are eliminated. This property of **GICF** is very natural: if we pay K_0

[5] i.e., the modified cost function is piecewise convex in p and the local optima between breakpoints can be computed easily.

[6] Interestingly, with variable demands non-periodic inter-replenishment times are widely accepted.

for each individual order instead of for each joint order, the optimal cost should increase. Surprisingly, this natural property is not known to hold for **GI** policies.

The finite horizon case

In the finite horizon case the most interesting problem is the **JRP** with variable demands (as defined in Section 3.2). This problem admits several approximation algorithms [26, 22], with a current best approximation factor of 1.8 [27]. However, all these algorithms run in $\Omega(T)$ time, which is only pseudopolynomial when demands are constant. One exception is the algorithm of Joneja [22], which is designed for variable demands, but can run in polynomial time when demands are constant, achieving an approximation factor that converges to $11/10$ as $T \to \infty$. .

3.5 An approximation algorithm for the JRP

In this section we present a dynamic schedule for the finite horizon case in continuous time that approximates well the cost of the best dynamic policy. In certain sense, our schedule will be very close to be periodic, as we will discuss later.

We temporarily assume that the approximation algorithm has oracle access to N, the total number of joint orders in some optimal solution for the **JRP**. We do this assumption to make the description and analysis of the algorithm simpler. We show how to remove this assumption in the next section. The algorithm (see Algorithm 1) is a simple two-step process. In the first step, the algorithm places joint ordering points at every multiple of T/N, starting at $t = 0$. In the second step, each commodity places its orders on a subset of those joint orders in such a way that the individual ordering and holding costs are minimized. Note that this can be carried out separately for each commodity. This is one of the key ideas of the algorithm of Wildeman et al. [49] in the context of **GI** policies.

Algorithm 1

1: **Approx-JRP** (T, h_i, d_i, K_i, K_0)
2: Guess N, the number of joint orders in an optimal solution.
3: Set $p = T/N$ to be the joint inter-replenishment length.
4: Set $J = \{jp : j = 0, \ldots, N-1\}$, the set of joint order positions.
5: **for** $i \in \mathscr{I}$ **do**
6: Choose a subset of J to be the orders of commodity i such that $C_{\text{ord}}^{\text{indiv}}(i) + C_{\text{hold}}(i)$ is minimal.
7: **return** the schedule obtained.

Running time analysis

We have to be careful in how to execute the algorithm. The set J may have $\Omega(T)$ elements, while the input size is proportional to $\log T$. However, we can explicitly define this set by giving T and N, and it is easy to check that the size of N is polynomial in the input size. For example, a simple bound follows from the observation that the cost of the schedule that orders everything at time $t = 0$ must have cost at least NK_0, which is a trivial lower bound on the optimal cost. Therefore, $N \leq N_0$ where

$$N_0 \equiv \frac{1}{K_0}\left(K_0 + \sum_{i \in \mathscr{I}} (K_i + h_i d_i T^2/2)\right). \tag{3.2}$$

A similar difficulty arises in Step 6 of the algorithm, but a similar representation can be applied to keep the space polynomial. To see this, note that for each commodity i, the function $C_{\text{ord}}^{\text{indiv}}(i) + C_{\text{hold}}(i)$ is convex. If there are two renewal intervals of lengths q_1 and q_2 with $q_2 - q_1 \geq 2p$, then we can replace them with two renewal intervals of lengths $q_1 + p$ and $q_2 - p$ to obtain a schedule for that commodity with the same ordering cost $C_{\text{ord}}^{\text{indiv}}(i)$, but with a lower holding cost $C_{\text{hold}}(i)$. Therefore, we can specify each individual schedule in Step 6 by giving these two inter-replenishment lengths and their frequencies.

It follows that the only step where polynomiality can fail is Step 6. The following lemma establishes its complexity.

Lemma 3.5.1. *Suppose that commodity i can be ordered only at multiples of some fixed period p. Moreover, assume that T is a multiple of p. Then, it is possible to compute the schedule minimizing $C_{\text{hold}}(i) + C_{\text{ord}}^{\text{indiv}}(i)$ in polynomial time with respect to the input size.*

Proof. If $C_{\text{ord}}^{\text{indiv}}(i) + C_{\text{hold}}(i)$ is minimum, then the inter-replenishment lengths can take at most two values, and both are consecutive multiples of p. If these values are mp and $(m+1)p$, let a (resp. b) the number of orders of length mp (resp. $(m+1)p$). Clearly, we have that $ma + (m+1)b = T/p \equiv N$, which is a Diophantine equation in a, b with integral solutions of the form

$$a = -N + (m+1)r \qquad b = N - mr, \quad \text{for } r \in \mathbb{Z}. \tag{3.3}$$

The non-negativity of a and b implies the restriction $N/m \geq r \geq N/(m+1)$. We need to find m, a, b such that $[N/(m+1), N/m]$ contains an integer and $C_{\text{hold}}(i) + C_{\text{ord}}^{\text{indiv}}(i)$ is minimized.

Consider the cost rate function $\text{CR}(u) = K_i/u + h_i d_i u/2$, which coincides with the actual cost rate of the commodity when the inter-replenishment interval length u is constant. This function is minimized at $u^* = \sqrt{\frac{2K_i}{h_i d_i}}$. By convexity of the function $\text{CR}(u)$, it follows that

- If the optimal m satisfies $mp \geq u^*$, then it has to be the minimum m satisfying $[N/(m+1), N/m] \cap \mathbb{Z} \neq \emptyset$.
- If the optimal m satisfies $(m+1)p \leq u^*$, then it has to be the maximum m satisfying $[N/(m+1), N/m] \cap \mathbb{Z} \neq \emptyset$.
- If $mp < u^* < (m+1)p$, then $m = \lfloor u^*/p \rfloor$.

Therefore, m is one of these three values. They can be computed efficiently. For example, if $mp \geq u^*$, then m is at least $m^* \equiv \lceil u^*/p \rceil$, and we are interested in the smallest integer $m \geq m^*$ such that $[N/(m+1), N/m]$ contains an integer. The largest integer in $[0, N/m^*]$ is $r = \lfloor N/m^* \rfloor$, and therefore the smallest m we can use satisfies:

$$\frac{N}{r} - 1 \leq m \leq \frac{N}{r} \quad \text{and} \quad m \geq m^*.$$

So we set $m = \max\{m^*, \lceil N/r - 1 \rceil\}$.

Once m is fixed, the values of a and b are chosen in such a way that the inter-replenishment length with minimum cost rate appears the maximum number of times. It is not hard to see that a and b can be computed efficiently. Each of the triplets (m, a, b) is a candidate for

the optimal schedule, and we can find the actual optimum by picking the triplet defining an schedule with minimum cost. For completeness, we describe the procedure in Algorithm 2.

□

Algorithm 2 Computing the schedule for commodity i minimizing $C_{\text{ord}}^{\text{indiv}}(i) + C_{\text{hold}}(i)$, subject to replenishments in $\{jp : j = 0, \ldots, N-1\}$, where $p = T/N$.

Set $u^* = \sqrt{2K_i/(h_i d_i)}$.
Set $m_1^* = \lceil u^*/p \rceil$ and $m_2^* = \lfloor u^*/p \rfloor$.
Set $r_1 = \lfloor N/m_1^* \rfloor$ and $r_2 = \lceil N/m_2^* \rceil$.
Set $m_1 = \max\{m_1^*, \lceil N/r_1 - 1 \rceil\}$, $m_2 = \lfloor N/r_2 \rfloor$ and $m_3 = \lfloor u^*/p \rfloor$.
for $i = 1$ **to** 3
do
  if $\text{CR}(m_i p) \leq \text{CR}((m_i+1)p)$
    then set $a_i = -N + (m_i+1)\lfloor N/m_i \rfloor$ and $b_i = N - m_i \lfloor N/m_i \rfloor$.
    else set $a_i = -N + (m_i+1)\lceil N/(m_i+1) \rceil$ and $b_i = N - m_i \lceil N/(m_i+1) \rceil$.
Set $(m,a,b) = \arg\min \left\{ \frac{1}{a_i+b_i} \left(a_i \text{CR}(m_i p) + b_i \text{CR}((m_i+1)p)\right) \right\}$.
return the schedule that places a orders of length mp, followed by b orders of length $(m+1)p$.

Approximation analysis

Given an instance of **JRP**, let OPT be any optimal solution having exactly N joint orders, where N is the value guessed by Algorithm 1. For $i \in \mathscr{I}$, let n_i be the total number of individual orders of commodity i in OPT and let **OPT** be the optimal cost. In this section, we may emphasize the dependency on the schedule by including the schedule in brackets. For example, we may write $C[\text{OPT}] = \textbf{OPT}$.

If a commodity is ordered exactly m times, it is easy to show that its holding cost is minimized when the replenishments occur at $\{jT/m : j = 0, \ldots, m-1\}$. We say that m orders are **evenly distributed** when they are placed according to this configuration. This optimality property for the holding cost of evenly distributed orders is the basis for a lower bound on **OPT** we use to prove the approximation guarantee. Our first step in this direction is to define two feasible solutions for the problem:

- The virtual schedule (or VS) places exactly $(1+\beta_i)n_i$ evenly distributed orders of commodity i, for every $i \in \mathscr{I}$. Each β_i is a parameter to be defined.

- The real schedule (or RS) allows joint orders in $J = \{jp : j = 0,\ldots,N-1\}$. For each commodity i we place exactly $(1+\beta_i)n_i$ orders, that are obtained by shifting each individual order in the virtual schedule to the closest point in J. If there are two closest joint orders, we choose the closest one backwards in time.

Note that both schedules are not defined algorithmically. The real schedule is defined from the virtual schedule, and there is a one to one correspondence between their individual orders through the shifting process. We use the term **shifted order** to indicate this correspondence.

Loosely speaking, the cost of the real schedule is closely related to the cost of the schedule output by Algorithm 1, while the virtual schedule is related to a lower bound on **OPT**. Both are used as a bridge that relates **OPT** with the cost of the schedule returned by Algorithm 1.

Proposition 3.5.1. *If $\beta_i \leq 1/8$ for every $i \in \mathscr{I}$, then $C_{\text{hold}}[\text{RS}] \leq \frac{9}{8} C_{\text{hold}}[\text{VS}]$.*

Proof. Consider any commodity $i \in \mathscr{I}$. For simplicity, we omit subindices and write n instead of n_i and β instead of β_i. Let $q = T/(1+\beta)n$ be the inter-replenishment length of the commodity in the virtual schedule. Let $p = T/N$ be the joint inter-replenishment length for the real schedule. Note that $q \geq p/(1+\beta)$.

Suppose first that $p \geq q$. In RS, the commodity is replenished in every joint-order position. Directly evaluating the holding costs gives

$$C_{\text{hold}}[\text{RS}](i) = \frac{T^2 hd}{2N} \leq \frac{T^2 hd}{2n} = (1+\beta)C_{\text{hold}}[\text{VS}](i) \leq \frac{9}{8} C_{\text{hold}}[\text{VS}](i).$$

On the other hand, if $p < q$, let k be the only integer satisfying $kp \leq q < (k+1)p$. Clearly, the inter-replenishment lengths in the real schedule can only take the values kp or $(k+1)p$. Let a be the number of orders of length kp and let b the number of orders of length $(k+1)p$ in the real schedule. We have the relations:

$$a+b = (1+\beta)n \quad \text{and} \quad a(kp) + b(k+1)p = q(1+\beta)n,$$

from where we get, in particular, that $bp = (1+\beta)n(q-kp)$. Using these three relations,

and evaluating the holding cost, we obtain:

$$\frac{C_{\text{hold}}[\text{RS}](i)}{C_{\text{hold}}[\text{VS}](i)} = \frac{a(kp)^2 + b(k+1)^2p^2}{q^2(1+\beta)n} \leq \frac{(1+\beta)n(kp)^2 + b(2k+1)p^2}{q^2(1+\beta)n}.$$

which can be written after some additional manipulation as

$$\frac{C_{\text{hold}}[\text{RS}](i)}{C_{\text{hold}}[\text{VS}](i)} \leq (-k^2-k)\left(\frac{p}{q}\right)^2 + (2k+1)\frac{p}{q}.$$

To conclude, note that $-k(k+1)x^2 + (2k+1)x$, as a function of x, has maximum value $\frac{(2k+1)^2}{4k(k+1)}$, which is at most $9/8$ when $k \geq 1$. □

The next proposition shows that the individual ordering and holding costs in RS are within a constant factor of the respective costs in OPT. The proof uses Prop. 3.5.1 and some simple relations among RS, VS and OPT.

Proposition 3.5.2. *Let $\gamma = \sqrt{9/8}$. Then for every $\varepsilon > 0$ we can choose $\{\beta_i\}_{i\in\mathscr{I}}$ so that the real schedule satisfies*

- $C_{\text{hold}}[\text{RS}] \leq (1+\varepsilon)\gamma^2 \cdot C_{\text{hold}}[\text{OPT}]$
- $C_{\text{ord}}^{\text{indiv}}[\text{RS}] \leq (1+\varepsilon)\gamma^2 \cdot C_{\text{ord}}^{\text{indiv}}[\text{OPT}]$.

If T is sufficiently large, we also have

- $C_{\text{hold}}[\text{RS}] \leq (1+\varepsilon)\gamma \cdot C_{\text{hold}}[\text{OPT}]$
- $C_{\text{ord}}^{\text{indiv}}[\text{RS}] \leq (1+\varepsilon)\gamma \cdot C_{\text{ord}}^{\text{indiv}}[\text{OPT}]$

Proof. It is easy to see that $C_{\text{hold}}[\text{VS}](i) = \frac{T^2 h_i d_i}{2(1+\beta_i)n_i}$. Also, $C_{\text{hold}}[\text{OPT}](i) \geq \frac{T^2 h_i d_i}{2n_i}$, because the right hand side is equal to the holding cost of n_i evenly distributed orders. Combining these relations, we obtain

$$\min_i\{1+\beta_i\} C_{\text{hold}}[\text{VS}] \leq C_{\text{hold}}[\text{OPT}].$$

Additionally, if $\beta_i < 1/8$, we can combine this result with Prop. 3.5.1, obtaining

$$C_{\text{hold}}[\text{RS}] \leq \frac{9}{8\min_{i\in\mathscr{I}}\{1+\beta_i\}} C_{\text{hold}}[\text{OPT}].$$

Fix $\beta > 0$ and T sufficiently large such that for every commodity i there exists β_i satisfying $(1+\beta_i)n_i$ is integer and $(1-\varepsilon')(1+\beta) \leq 1+\beta_i \leq 1+\beta$, for some $\varepsilon' > 0$ to be chosen at the end of the proof. Such value of T must exist because the inter-replenishment length of the commodity in any optimal solution has an upper bound independent of T. Under these assumptions, we have that

$$C_{\text{ord}}^{\text{indiv}}[\text{RS}] \leq \max_i\{1+\beta_i\} C_{\text{ord}}^{\text{indiv}}[\text{OPT}] \leq (1+\beta) C_{\text{ord}}^{\text{indiv}}[\text{OPT}]$$

and

$$C_{\text{hold}}[\text{RS}] \leq \left(\frac{9}{8(1-\varepsilon')(1+\beta)}\right) C_{\text{hold}}[\text{OPT}].$$

Choosing the value of β that gives the best approximation bound, we obtain

$$C_{\text{ord}}^{\text{indiv}}[\text{RS}] \leq \frac{\gamma}{\sqrt{1-\varepsilon'}} C_{\text{ord}}^{\text{indiv}}[\text{OPT}] \quad \text{and} \quad C_{\text{hold}}[\text{RS}] \leq \frac{\gamma}{\sqrt{1-\varepsilon'}} C_{\text{hold}}[\text{OPT}].$$

We finish the proof by choosing ε' so that $(1-\varepsilon')^{-1/2} \leq (1+\varepsilon)$. In the case of arbitrary T, we can only choose $\beta_i = 0$, obtaining the bounds stated in the lemma. □

Let S be the schedule returned by Algorithm 1. Recall that its output is a schedule S that minimizes $C_{\text{ord}}^{\text{indiv}} + C_{\text{hold}}$ restricted to use N evenly distributed joint orders. This and Prop. 3.5.2 give the following inequalities for large T:

$$\left(C_{\text{ord}}^{\text{indiv}} + C_{\text{hold}}\right)[\text{S}] \leq \left(C_{\text{ord}}^{\text{indiv}} + C_{\text{hold}}\right)[\text{RS}] \leq (1+\varepsilon)\gamma\left(C_{\text{ord}}^{\text{indiv}} + C_{\text{hold}}\right)[\text{OPT}].$$

Since N is the number of joint orders in OPT, then $C_{\text{ord}}^{\text{joint}}[\text{S}] \leq C_{\text{ord}}^{\text{joint}}[\text{OPT}]$. Adding up, we obtain $C[\text{S}] \leq (1+\varepsilon)\gamma C[\text{OPT}]$ which is an approximation guarantee asymptotically equal to γ for Algorithm 1.

Theorem 3.5.1. *Algorithm 1 is a 9/8-approximation algorithm ($\sqrt{9/8}$ for large T) for dynamic policies in the finite horizon case.*

3.6 An FPTAS for the GI model

We can easily adapt the algorithm described in Sect. 3.5 to the **GI** model with variable base (see Algorithm 3). We now guess p, the optimal joint inter-replenishment length. Note that Step 5 is simpler, since q_i is always one of the two multiples of p closest to $\sqrt{K_i/h_i}$.

Algorithm 3 GI model algorithm

1: **Approx-JRP** (T, h_i, d_i, K_i, K_0)
2: Guess p, the optimal renewal interval in an optimal solution.
3: Set $J = \{jp : j = 0, \ldots, N-1\}$, the set of joint order positions.
4: **for** $i \in \mathscr{I}$ **do**
5: Choose q_i as a multiple of p such that $K_i/q_i + h_i q_i$ is minimum.
6: **return** the schedule obtained.

Note that Algorithm 3 finds the best value of q_i for the optimal p, and therefore computes the optimal **GI** policy. Since **GI** policies approximate unrestricted policies by a factor of $1/(\sqrt{2}\log 2) \approx 1.02$ [32, 46], our algorithm achieves these guarantees. The bound in Section 3.5 (≈ 1.06) is slightly worse since we are not using the powerful machinery available for the stationary case.

From this observation we can obtain a fully polynomial-time approximation scheme for **GI** policies by exhaustively searching p in powers of $(1+\varepsilon)$. The range of search can be $[p_{\min}, p_{\max}]$, which are the values defined in Equation 3.1. The total running time is polynomial in the size of the input and $\frac{1}{\log(1+\varepsilon)} = O(1/\varepsilon)$. The only thing we need to prove is that choosing p' in the range $p \le p' \le p(1+\varepsilon)$ is enough to get a $(1+\varepsilon)$-approximation. This follows from the fact that if $(p, \{k_i\}_{i\in\mathscr{I}})$ defines an optimal schedule with value **OPT**, then $(p/(1+\varepsilon), \{k_i\}_{i\in\mathscr{I}})$ has cost

$$\frac{K_0}{p(1+\varepsilon)} + \sum_{i\in\mathscr{I}} \frac{K_i}{(1+\varepsilon)k_i p} + \frac{1}{2} h_i d_i k_i (1+\varepsilon) \le (1+\varepsilon)\mathbf{OPT}.$$

Essentially the same idea can be used to remove the guessing assumption in Algorithm 1. We just exhaustively search N in (approximated) powers of γ.

Finally, Algorithm 3 can be extended to the fixed base **GI** model. The only difference is that we guess p assuming it is a multiple of the base B. The exhaustive search in powers of $(1+\varepsilon)$ has to carefully round the values of p to be multiples of B.

Theorem 3.6.1. *[41] Algorithm 3 (properly modified) is an FPTAS in the class of* ***GI*** *policies and in the class of fixed base* ***GI*** *policies.*

Chapter 4

Synchronization, periodicity and Integer Factorization

In the **Integer Factorization Problem (IFP)** we want to express a positive integer as a product of prime numbers. The decomposition into product of primes is called **prime factorization**, which is one of the building blocks of elementary number theory.

Researchers widely believe that the **IFP** does not admit polynomial time algorithms. But this belief is not justified by any of the traditional concepts from computational complexity. In this regard, the **IFP** stands as a unique problem in complexity theory.

In this chapter we establish the hardness of several scheduling problems in Operations Management based on the presumed hardness of the **IFP**. The problems for which we are able to prove these results share some elements of synchronization and periodicity, two properties that we link to integer factorization. The main results in this chapter are joint work with Andreas S. Schulz [41].

4.1 Some background on number theory

For convenience, in this subsection a, b and d denote positive integers.

We say that a is a **divisor** of b (or b is **multiple** of a) if the ratio b/a is an integer. We denote this relation as $a|b$. A basic property of this relation is the following.

Lemma 4.1.1 ([20]). *If $d|a$ and $d|b$ with $a > b$, then $d|(a+b)$ and $d|a-b$.*

Every positive integer has two **trivial** divisors, namely 1 and itself. Positive integers greater than or equal to 2 with only trivial divisors are called **primes**. Otherwise they are called **composite**. The Fundamental Theorem of Arithmetic [20] guarantees that every positive integer can be uniquely written[1] as a product of primes. For example, we write $36 = 2^2 \cdot 3^2$. Since every composite number is the product of at least two primes, we have the following property:

Lemma 4.1.2 ([20]). *Every composite number b has a prime divisor no larger than $\sqrt{b}$.*

We are interested in two particular integer functions. The **least common multiple** of two positive integers a and b, denoted by $\text{lcm}(a,b)$, is the smallest positive number that is simultaneously a multiple of a and b. The **greatest common divisor** $\gcd(a,b)$ is the largest positive number that is simultaneously a divisor of a and b. It is very easy to compute both functions from the prime factorization. For example, if $a = 2^4 \cdot 3^8 \cdot 5^9$ and $b = 2^7 \cdot 3^6$, then

$$\gcd(a,b) = 2^{\min\{4,7\}} \cdot 3^{\min\{8,6\}} \cdot 5^{\min\{9,0\}}$$

and

$$\text{lcm}(a,b) = 2^{\max\{4,7\}} \cdot 3^{\max\{8,6\}} \cdot 5^{\max\{9,0\}}.$$

In other words, the greatest common divisor takes the primes appearing in the prime factorization of a and b, and raises them to the minimum of their corresponding exponents. The least common multiple does the same, but raises the primes to the maximum of their corresponding exponents. From this property we easily obtain the next lemma.

Lemma 4.1.3 ([20]). $\gcd(a,b) \cdot \text{lcm}(a,b) = a \cdot b$.

We say that a and b are **coprimes** if $\gcd(a,b) = 1$. In other words, coprime numbers do not share any prime factor. It is possible to show that if $a > b$ are coprime numbers, then so are a and $a-b$. For example, this follows from the next result:

Lemma 4.1.4 ([20]). *For every $a > b$, $\gcd(a,b) = \gcd(b, a-b)$.*

[1] up to reordering of the factors.

Lemma 4.1.4 implies that $\gcd(a,b) = \gcd(b, a-kb)$ for any integer $k \geq 0$ such that $a - kb > 0$. From this we can express $\gcd(a,b)$ as the gcd of significantly smaller numbers, which leads to a simple iterative method to compute the greatest common divisor. For example,

$$\gcd(30,24) = \gcd(24, 30-1\cdot 24) = \gcd(24,6) = \gcd(6, 24-3\cdot 6) = \gcd(6,6) = 6.$$

Implemented properly, this algorithm satisfies:

Lemma 4.1.5 (Euclid's algorithm [8]). *The greatest common divisor of a and b can be computed in polynomial time (with respect to* $\log(a+b)$*).*

This result is important for us, as we will need to compute the greatest common divisor efficiently. It follows from Lemma 4.1.3 that the least common multiple can also be computed efficiently.

4.2 The complexity of factoring integer numbers

Given two positive integers $N < M$, the **decision version** of the **IFP** is to decide whether M has a non-trivial divisor d satisfying $d \leq N$. It is easy to show that a polynomial time algorithm for the decision version of **IFP** leads to a polynomial time algorithm for the **IFP** itself[2], so they are essentially equivalent for the discussion that follows.

What makes **IFP** very unique in terms of complexity starts with the following result:

Lemma 4.2.1. *The decision version of IFP is in* $\mathbf{NP} \cap co\mathbf{NP}$

Proof. Let $N < M$ be positive integer numbers. To show that **IFP** belongs to co**NP**, note that the prime factorization of M can be used to certify that M has no divisor d with $d \leq N$. To check the correctness of the prime factorization we can use the primality test of Agrawal et al. [1]. To show that **IFP** belongs to **NP**, we can just use any non-trivial divisor $d \leq N$ of M to certify that this divisor exists. □

[2]Basically, binary search in N allows us to find a non-trivial divisor in polynomial time.

It follows from this lemma that the **IFP** is unlikely to be **NP**-hard or co**NP**-hard. Moreover, most problems initially shown to be in **NP** $\cap$ co**NP** have been eventually shown to be in **P** (a classic example of this is linear programming).

Another particularity of the **IFP** is that it can be solved in polynomial time using quantum computers. This is a remarkable result, as it was the first famous problem that was shown to be efficiently solvable with a quantum computer, but not with a traditional one.

But against all this "evidence", researchers widely believe that the **IFP** is not polynomial time solvable with traditional computers. In fact, many cryptographic protocols used for transactions over the Internet depend on the hardness of integer factorization. If there were an efficient algorithm for factoring, then all those protocols would be breakable. The most well known of these protocols is RSA [38], which has been around since 1978 and is still used as of today. For this reason, and for its theoretical importance, the **IFP** has been heavily researched, but all attempts for finding an efficient algorithm have, so far, failed.

4.3 Using the IFP to prove hardness

We say that a problem π is **Integer Factorization-hard** (or simply **IF-hard**) if the existence of a polynomial time algorithm for π implies the existence of a polynomial time algorithm $\mathscr{A}$ for integer factorization. Usually, the way to prove this type of results is by using a **reduction**, that uses a fictitious polynomial time algorithm for π to algorithmically solve the **IFP** in polynomial time.

Since **NP**-hardness implies **IF**-hardness, we look for problems π that have an intrinsic connection with the **IFP**, but have not shown to be **NP**-hard. Indeed, we picked problems for which the complexity has been open for decades. To the best of our knowledge, this is the first time that an optimization problem is reduced to the **IFP**.

The problems we consider are optimization problems with discrete decision variables representing the frequency of certain events that are repeated **periodically** over time. The optimization function must model an incentive to **synchronize** the periods. By this, we mean that the model prefers to choose periods where multiple events are simultaneously executed.

In contrast to most non-trivial **NP-hard** reductions, there are no gadgets. We directly model the number M to factorize into the problem. To illustrate the technique, consider the following optimization problem having M as the only input:

$$\begin{aligned} \max \quad & \mathbf{U}(q_1, q_2) = Mq_1 + \gcd(q_1, q_2) \\ \text{s.t.} \quad & q_1 \leq M \\ & q_2 \leq M - 1 \\ & q_1, q_2 \in \mathbb{Z}_+. \end{aligned} \tag{4.1}$$

Note that if $q_1 \leq M-1$, then $\mathbf{U}(q_1,q_2) \leq M(M-1)+(M-1) = M^2-1$. If $q_1 = M$, we then have $\mathbf{U}(q_1,q_2) \geq M^2$. Therefore, q_1 must be equal to M in any optimal solution. In all the proofs of hardness we will eventually prove that some variable is equal to M in any optimal solution. We call this Step 1. Note that this argument critically depends on the discreteness of q_1 and q_2.

Now, given that $q_1 = M$, we can figure out what the value of q_2 is. In this case, we just want to maximize $\gcd(M,q_2)$ subject to $q_2 \leq M-1$. Therefore $\gcd(M,q_2)$ will be the largest non-trivial divisor of M, so the optimal objective value implicitly contains information to compute a non-trivial divisor. We call this Step 2.

In the final step we put the pieces together by describing an algorithm that finds a non-trivial divisor of M in polynomial time, assuming that problem 4.1 is solvable in polynomial time. In this case, we just need to substract M^2 from the optimal objective value. Finding a non trivial divisor of a composite number in polynomial time is all we need to solve the **IF** in polynomial time, as we can recursively factorize every divisor we find.

For the hardness results that follow, the functions involved in the optimization problem are more complicated, and so the proofs of Step 1 and Step 2 are lengthier. In Step 2 we may not be able to use the optimization problem to find a divisor of every positive integer M. However, the hardness proof still works as long as we can factorize every positive integer M either using the method just described, or using any other procedure that runs in polynomial time.

4.4 Application: The fixed base GI problem

We now prove a first hardness result for the Joint Replenishment Problem. The optimization problem, already introduced in Chapter 3, is the fixed base **GI** model:

$$\begin{aligned} \min \quad & \frac{K_0}{p} + \sum_{i \in \mathscr{I}} \left(\frac{K_i}{q_i} + \frac{1}{2} h_i d_i q_i \right) \\ \text{s.t.} \quad & \frac{q_i}{p} \in \mathbb{Z}_+ \quad \text{for all } i \in \mathscr{I} \\ & \frac{p}{B} \in \mathbb{Z}_+. \end{aligned}$$

In this problem, the base B is part of the input.

Setup

Starting from the fixed base **GI** model with two commodities, we set $B = 1$, $\frac{1}{2} h_2 d_2 = 1$ and $K_2 = K_0$. Renaming $\frac{1}{2} h_1 d_1 = H_1$, we obtain the following optimization problem:

$$\begin{aligned} \min \quad & \mathbf{U}(p, q_1, q_2) = K_0 \left(\frac{1}{p} + \frac{1}{q_2} \right) + \frac{K_1}{q_1} + H_1 q_1 + q_2 \\ \text{s.t.} \quad & \frac{q_1}{p}, \frac{q_2}{p} \in \mathbb{Z}_+ \\ & p \in \mathbb{Z}_+ \end{aligned} \tag{4.2}$$

We choose $B = 1$ because this guarantees the integrality of p. Setting $\frac{1}{2} h_2 d_2 = 1$ is just a normalization of the coefficients in the objective $\mathbf{U}$. The choice of $K_2 = K_0$ requires to understand the proof, but setting it at the beginning makes the proof simpler.

Step 1

In order to force $q_1 = M$ in any optimal solution of (4.2), the idea is to fix K_1 and H_1 so that the minimum of $L(q_1) \equiv \frac{K_1}{q_1} + H_1 q_1$ is attained at $q_1 = M$. We also need that H_1 and K_1 are

large enough compared to the other coefficients in $\mathbf{U}(p,q_1,q_2)$, so that $\mathbf{U}(p,q_1,q_2) \approx L(q_1)$ and therefore picking $q_1 \neq M$ is never optimal.

With this objective in mind, we set

$$H_1 = 4K_0 + (M+1)^2 \quad \text{and} \quad K_1 = M^2 H_1$$

It is easy to check that $L(q_1)$ is minimized at $q_1 = M$. To prove that $q_1 \neq M$ is never optimal for $\mathbf{U}(p,q_1,q_2)$, note that a little bit of algebra gives

$$\begin{aligned} L(M+1) - L(M) &= \frac{H_1}{M+1} \\ L(M-1) - L(M) &= \frac{H_1}{M-1}. \end{aligned}$$

Using these bounds and the convexity of L, we obtain that for all $q_1 \neq M$:

$$\mathbf{U}(p,q_1,q_2) \geq L(q_1) \geq \min\{L(M-1), L(M+1)\} = L(M) + \frac{4K_0}{M+1} + M + 1. \tag{4.3}$$

On the other hand,

$$\mathbf{U}(M,M,M) = L(M) + \frac{2K_0}{M} + M$$

is already smaller than the lower bound in (4.3), so q_1 must be equal to M in any optimal solution.

Step 2

From Step 1, we can set $q_1 = M$ and ignore the now constant terms in (4.2). If we also rename q_2 as q, and set $K_0 = M$, we obtain the following optimization problem

$$\begin{aligned} \min \quad & \mathbf{V}(p,q) = M\left(\frac{1}{p} + \frac{1}{q}\right) + q \\ \text{s.t.} \quad & \frac{M}{p}, \frac{q}{p} \in \mathbb{Z}_+ \\ & p \in \mathbb{Z}_+ \end{aligned} \tag{4.4}$$

In this step we assume that M is a composite number but not a multiple of 2 or 3, and we want to prove that every optimal solution (p^*, q^*) of (4.4) satisfies that p^* is a non-trivial divisor of M. This is almost implied by the constraints of (4.4), as they guarantee that any feasible p is already a divisor of M. We only need to prove the following lemma.

Lemma 4.4.1. *Let M be any composite number that is not a multiple of 2 or 3. Then every optimal solution (p^*, q^*) of (4.4) satisfies $p^* \neq 1, M$.*

Proof. It is easy to check that

$$\mathbf{V}(1,q) \geq M \text{ for every } q \in \mathbb{Z}_+$$

and

$$\mathbf{V}(M,q) \geq q \geq M, \text{ for every feasible } q \text{ in (4.4).}$$

Note that we used the constraint $\frac{q}{p} \in \mathbb{Z}_+$ in order to imply $q \geq M$.

On the other hand, let $\bar{p}$ be any non-trivial divisor of M. Since M is not divisible by 2 or 3, it follows that $5 \leq \bar{p} \leq \frac{M}{5}$. Using these bounds we can show that

$$\mathbf{V}(\bar{p},\bar{p}) = \frac{2M}{\bar{p}} + \bar{p} \leq \frac{2M}{5} + \frac{M}{5} = \frac{3M}{5}$$

Since $(\bar{p},\bar{p})$ is feasible for (4.4), and $\mathbf{V}(\bar{p},\bar{p})$ is smaller than both $\mathbf{V}(1,q)$ and $\mathbf{V}(M,q)$, for any feasible q, it follows that $p = 1$ or $p = M$ cannot be optimal for (4.4). □

Putting everything together

The algorithm $\mathscr{A}$ that finds a non-trivial divisor of a composite number M is simple. First, it tests whether M is multiple of 2 or 3. In those cases, $\mathscr{A}$ reports the non-trivial divisor 2 or 3 and terminates. Otherwise, it solves the following **GI** problem associated to M:

$$\begin{aligned} \min \quad & \mathbf{U}(p,q_1,q_2) = M\left(\frac{1}{p}+\frac{1}{q_2}\right)+(M^2+6M+1)\left(\frac{M^2}{q_1}+q_1\right)+q_2 \\ \text{s.t.} \quad & \frac{q_1}{p},\frac{q_2}{p} \in \mathbb{Z}_+ \\ & p \in \mathbb{Z}_+. \end{aligned} \tag{4.5}$$

If the optimal solution found is (p^*,q_1^*,q_2^*), it returns p^* and terminates. Steps 1 and 2 guarantee that $p^*|M$, and $p^* \neq 1,M$, therefore p^* is actually a non-trivial divisor of M. Since the size of the coefficients in (4.5) is polynomial in $\log M$, algorithm $\mathscr{A}$ could be implemented in polynomial time if there were a polynomial time algorithm to solve the **JRP** with **GI** policies. Therefore,

Theorem 4.4.1. *The **JRP** in the fixed base **GI** model is **IF**-hard.*

4.5 Application: The Clustering of Maintenance Jobs.

Many industrial processes ensure the correct function of its different **components** by doing regular maintenance jobs. In many cases, these jobs render the components unusable during the maintenance times, forcing the whole system to become idle for the duration of the maintenance.

The **Clustering of Frequency-Constrained Maintenance Jobs Problem (CMJ)** aims to reduce the cost of these downtimes. We describe here a special case, where the set of components $c \in \mathscr{C}$ are nested. These relations are given by a directed rooted tree $T = (\mathscr{C},E)$ with root r, where an arc from c to c' indicates that component c' is part of component c. The leaves of the tree (components with no outgoing arcs) are called **indivisible components**.

We assume that the maintenance of a component requires to disassemble all the components in which it is contained. In other words, if a component $c \in \mathscr{C}$ is under maintenance, all the components in the path from r to c must be under maintenance. We associate a constant cost K_c to the start of any maintenance job of a component c.

In the model we study here, we additionally assume that each indivisible component c works a constant integer amount of time q_c between maintenance jobs. Each q_c value, to be determined, is bounded from above by a given maximum operational time f_c, that is, $q_c \leq f_c$. This constraint implies that c cannot work for more than f_c units of time without maintenance.

For the following example, we refer to Figure 4-1. Consider an scenario with three components, where one component (labeled 0) contains two indivisible components (labeled 1 and 2). Suppose the maximum operational times of the indivisible components are $f_1 = 5$ and $f_2 = 6$. A valid maintenance schedule is $q_1 = 4$ and $q_2 = 6$, which is depicted in Figure 4-1. Our model does not consider the amount of time for each maintenance, so we arbitrarily set it to 1 in the timeline.

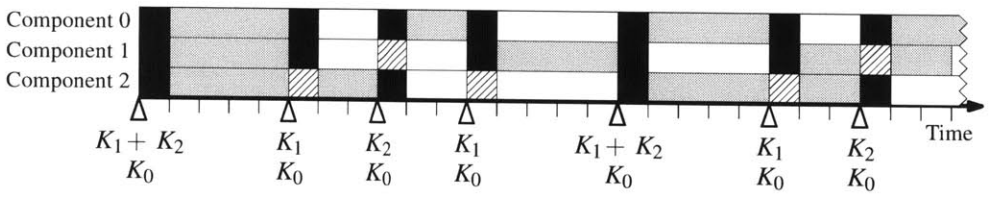


Figure 4-1: An example of the **CMJ** model. Components 1 and 2 are indivisible, and they are both contained in Component 0. Black areas denote maintenance jobs. Dashed areas denote idle time (where the component is not working and not under maintenance). Alternating gray and white areas delimit the times where a component is working without maintenance. Each maintenance job extends for one unit of time.

The objective of the **CMJ** is to minimize the maintenance cost per working unit of time. For indivisible components c, this cost is just K_c/q_c. For every other component (for example Component 0 in Figure 4-1) this cost is harder to compute, as maintenance jobs on these components are generated whenever one of its indivisible component starts a maintenance job. In order to give an explicit formula, let $\mathscr{I}_c$ be the set of indivisible components that c contains, plus the component c itself. Using the inclusion-exclusion formula (as in Lemma 3.1.1) we can write the maintenance cost of a component $c \in \mathscr{C}$ per unit of working time as:

$$K_c \sum_{i=1}^{|\mathscr{I}_c|} (-1)^{i+1} \sum_{I \subseteq \mathscr{I}_c : |I| = i} \operatorname{lcm}(q_c : c \in I)^{-1}.$$

This leads to the following formulation of the **CMJ** model:

$$\begin{aligned} \min \quad & \sum_{c\in\mathscr{C}} K_c \sum_{i=1}^{|\mathscr{I}_c|} (-1)^{i+1} \sum_{I\subseteq\mathscr{I}_c:|I|=i} \operatorname{lcm}(q_c : c\in I)^{-1} \\ & q_c \le f_c \text{ for all indivisible components } c, \qquad \textbf{(CMJ)} \\ & q_c \in \mathbb{Z}_+ \text{ for all } c\in\mathscr{C}. \end{aligned}$$

Levi et al. [25] generalize this model, applying it to the maintenance of aircrafts. In this generalization, the cost of performing a maintenance job of a collection of commodities at a given time is not the sum of the maintenance costs of each commodity in the collection, but a submodular function of the collection. They also assume that maintenance occur at times that are integral multiples of a fixed base period, which corresponds to our integrality assumption[3].

Other applications and results on this subject can be found in [35]. In the next section, we show that the model of **CMJ** presented above is **IF**-hard.

Setup

Starting from the **CMJ** with two indivisible components 1 and 2, and a third component 0 containing the other two, we obtain the following special case of **CMJ**:

$$\begin{aligned} \min \quad & \mathbf{U}(q_1,q_2) = \frac{K_0+K_1}{q_1} + \frac{K_0+K_2}{q_2} - \frac{K_0}{\operatorname{lcm}(q_1,q_2)} \\ \text{s.t.} \quad & q_1 \le f_1, \quad q_2 \le f_2 \\ & q_1,q_2 \in \mathbb{Z}_+ \end{aligned}$$

We initially set $K_0 = 1$ to normalize the coefficients. We will also set $K_2 = 0$ to make the proof simpler. The model becomes:

[3] after proper rescaling

$$\begin{aligned} \min \quad & \mathbf{U}(q_1,q_2) = \frac{1+K_1}{q_1} + \frac{1}{q_2} - \frac{1}{\mathrm{lcm}(q_1,q_2)} \\ \text{s.t.} \quad & q_1 \le f_1, \quad q_2 \le f_2 \\ & q_1, q_2 \in \mathbb{Z}_+ \end{aligned} \tag{4.6}$$

Step 1

In order to force $q_1 = M$ in any optimal solution of (4.6), we set $f_1 = M$ and choose K_1 large enough so that $\mathbf{U}(q_1,q_2) \approx \frac{1+K_1}{q_1}$. It turns out that setting $1+K_1 = M^2(M-1)$ is enough. Note that

$$\mathbf{U}(q_1,q_2) \ge \frac{1+K_1}{M-1} = M^2 \quad \text{whenever } q_1 < M. \tag{4.7}$$

On the other hand,

$$\mathbf{U}(M,f_2) \le \frac{1}{f_2} + \frac{1+K_1}{M} \le 1 - M + M^2.$$

Since $\mathbf{U}(M,f_2)$ is already smaller than the lower bound in (4.7) for any $M \ge 2$, it follows that q_1 must be equal to M in any optimal solution to (4.6).

Step 2

From Step 1, we can set $q_1 = M$ and ignore the now constant terms in (4.6). If we also rename q_2 as q, and f_2 as f, we obtain the following optimization problem

$$\begin{aligned} \min \quad & \mathbf{V}(q) = \frac{1}{q} - \frac{1}{\mathrm{lcm}(M,q)} \\ \text{s.t.} \quad & q \le f \\ & q \in \mathbb{Z}_+ \end{aligned} \tag{4.8}$$

In this step we assume that M is a composite number but not a multiple of 2, 3, 5 or 7. We write $M = ab$, where $a \le \sqrt{M}$. This factorization is guaranteed to exist by Lemma 4.1.2. Finally, we set f to be an arbitrary integer satisfying $M - a < f \le M - \frac{a}{2}$

Under these assumptions, we will conclude that any optimal solution q^* of (4.8) is not coprime with M. Before proving this result, we need a small lemma.

Lemma 4.5.1. *Let M be a composite number that is not a multiple of 2,3,5 or 7. Let $a \leq \sqrt{M}$ be a divisor of M. Then*

$$M - a - \frac{M}{a} > \frac{3}{4}M, \quad \textit{for all } M \geq 4$$

.

Proof. Since $11 \leq a \leq \sqrt{M}$, it follows that

$$M - a - \frac{M}{a} \geq \frac{10}{11}M - \sqrt{M}.$$

It is easy to see that the right-hand side of this inequality is greater than $\frac{3}{4}M$ for $M \geq 4$. □

Lemma 4.5.2. *If M is a composite number that is not a multiple of 2,3,5 or 7, then any optimal solution q^* to (4.8) is not coprime with M.*

Proof. If M and q^* were coprimes, then $\text{lcm}(M, q^*) = Mq^*$ and therefore

$$\mathbf{V}(q^*) \geq \min_{q \leq f} \left\{ \frac{1}{q} - \frac{1}{Mq} \right\} = \frac{1}{f} - \frac{1}{Mf} \tag{4.9}$$

Let $q \equiv a(b-1) = M - a$. Clearly, q is a feasible solution to (4.8). We will prove that $\mathbf{V}(q) < \mathbf{V}(q^*)$. Since $\text{lcm}(M, q) \leq M(b-1)$, it follows that

$$\mathbf{V}(q) = \frac{1}{q} - \frac{1}{\text{lcm}(M,q)} \leq \frac{1}{q} - \frac{1}{M(b-1)} \tag{4.10}$$

We now try to link (4.9) and (4.10). First, using that $q < f \leq M - \frac{a}{2}$ we obtain:

$$\frac{1}{f} - \frac{1}{q} = \frac{M - a - f}{fq} > \frac{-a/2}{q^2} = \frac{-1}{2a(b-1)^2}$$

On the other hand, using $M - a < f < M$ and Lemma 4.5.1 we get:

$$\frac{1}{(b-1)} - \frac{1}{f} = \frac{f - (b-1)}{f(b-1)} > \frac{M - a - b}{M(b-1)} > \frac{3}{4(b-1)}.$$

Finally, multiplying this last inequality by $\frac{1}{M}$ and adding it to the previous one, we obtain:

$$V(q^*) - V(q) > \frac{1}{a(b-1)}\left(\frac{3}{4b} - \frac{1}{2(b-1)}\right)$$

.

It is easy to chęck that the right hand side is positive, for every $b > 3$, and therefore, for every $M > 3^2 = 9$.

□

Putting everything together

Let P_i be the following **CMJ** problem:

$$\begin{array}{lll} \min & \mathbf{U}(q_1,q_2) = \dfrac{M^2(M-1)}{q_1} + \dfrac{1}{q_2} - \dfrac{1}{\text{lcm}(q_1,q_2)} & \\ \text{s.t.} & q_1 \leq M, \quad q_2 \leq M - 2^i & (P_i) \\ & q_1, q_2 \in \mathbb{Z}_+ & \end{array}$$

and denote by (q_1^i, q_2^i) an arbitrary optimal solution of P_i found by a hypothetical polynomial time algorithm for **CMJ**.

The algorithm $\mathscr{A}$ that finds a non-trivial divisor of a composite number M works as follows. First, it checks whether M is a multiple of 2, 3, 5 or 7. In those cases $\mathscr{A}$ reports the non-trivial divisor 2, 3, 5 or 7 and terminates. Otherwise, for each $i \in \{1,2,3,\ldots,\lfloor \log M \rfloor\}$, $\mathscr{A}$ solves P_i and computes $d_i = \gcd(M, q_2^i)$, reporting the first number d_i that is a non-trivial divisor of M.

We claim that if M is a composite number that is not a multiple of 2,3,5 or 7, then d_i is a non-trivial divisor for some i. To see this, note that if $M = ab$ with $11 \leq a \leq \sqrt{M}$, then there exists $i \in \{1,2,3,\ldots,\lfloor \log M \rfloor\}$ such that $M - a < M - 2^i \leq M - \frac{a}{2}$. Using Steps 1 and 2, it follows that d_i is a non trivial divisor of M.

Since we solve $O(\log M)$ instances of **CMJ**, each one having coefficients of size polynomial in $\log M$, it follows that algorithm $\mathscr{A}$ could be implemented in polynomial time

if there were a polynomial time algorithm to solve **CMJ**. Hence, we have the following theorem.

Theorem 4.5.1. *The **CMJ** problem is **IF**-hard, even with three components.*

4.6 Application: The fixed base GICF problem

We now prove another hardness result for the Joint Replenishment Problem, this time for the fixed base **GICF** model. The optimization problem, already introduced in Chapter 3, is the following:

$$\begin{aligned} \min \quad & \frac{K_0\Delta(k_1,\ldots,k_{|I|})}{p'} + \sum_{i\in\mathscr{I}} \frac{K_i}{q_i} + \frac{1}{2}h_i d_i q_i \\ \text{s.t} \quad & q_i = k_i p' \\ & k_i \in \mathbb{Z}_+. \end{aligned} \tag{GICF'}$$

Recall that in the fixed base model, p' is part of the input.

Setup

Starting from the **GICF** model with two commodities, we initially set the following parameters:

$$p' = 1, \quad K_2 = 0 \quad \text{and} \quad \frac{1}{2}h_2 d_2 = 1.$$

Renaming $\frac{1}{2}h_1 d_1 = H_1$, we obtain the following optimization problem:

$$\begin{aligned} \min \quad & \mathbf{U}(q_1,q_2) = K_0\left(\frac{1}{q_1} + \frac{1}{q_2} - \frac{1}{\operatorname{lcm}(q_1,q_2)}\right) + \frac{K_1}{q_1} + H_1 q_1 + q_2 \\ \text{s.t.} \quad & q_1, q_2 \in \mathbb{Z}_+ \end{aligned} \tag{4.11}$$

Note that, ignoring the term $K_0/\operatorname{lcm}(q_1,q_2)$, the objective $\mathbf{U}(q_1,q_2)$ is the sum of two functions of the form $f(x) = \frac{a}{x} + bx$. We will frequently use that the minimum of f on the real line is attained at $x = \sqrt{a/b}$.

Step 1

In order to force $q_1 = M$ in any optimal solution of (4.11), the idea is to fix K_1 and H_1 so that the minimum of $L(q_1) \equiv \frac{K_1}{q_1} + H_1 q_1$ is attained at $q_1 = M$. We also need that H_1 and K_1 are large enough compared to the other coefficients in $\mathbf{U}(q_1, q_2)$, so that $\mathbf{U}(q_1, q_2) \approx L(q_1)$ and therefore picking $q_1 \neq M$ is never optimal for $\mathbf{U}(q_1, q_2)$.

With this objective in mind, we set

$$K_1 = 2K_0M^3 \quad \text{and} \quad H_1 = 2K_0M.$$

It is easy to check that $L(q_1)$ is minimized at $q_1 = M$. We now prove that $q_1 \neq M$ is never optimal for $\mathbf{U}(q_1, q_2)$. A little bit of algebra gives

$$\begin{aligned} L(M+1) - L(M) &= \frac{H_1}{M+1} > \frac{K_0 M}{M} = K_0 \\ L(M-1) - L(M) &= \frac{H_1}{M-1} > K_0. \end{aligned}$$

Using these bounds and the convexity of L, we obtain that for all $q_1 \neq M$

$$L(q_1) \geq \min\{L(M-1), L(M+1)\} > L(M) + K_0,$$

and therefore

$$\mathbf{U}(q_1, q_2) \geq L(q_1) + 1 > L(M) + K_0 + 1 = 4K_0M^2 + K_0 + 1 \quad \text{for all } q_1 \neq M. \tag{4.12}$$

On the other hand,

$$\mathbf{U}(M, 1) = 4K_0M^2 + K_0 + 1$$

is already equal to the strict lower bound in (4.12), so q_1 must be equal to M in any optimal solution.

Step 2

From Step 1, we can set $q_1 = M$ and ignore the now constant terms in (4.11). If we also rename q_2 as q, and set $K_0 = M^2/4$, we obtain the following optimization problem

$$\begin{aligned} \min \quad & \mathbf{V}(q) \equiv \frac{M^2}{4}\left(\frac{1}{q} - \frac{1}{\text{lcm}(M,q)}\right) + q \\ \text{s.t} \quad & q \in \mathbb{Z}_+. \end{aligned} \tag{4.13}$$

For the algebra that follows, it is convenient to define

$$A(q) = \frac{M^2}{4q} + q, \quad B(q) = \frac{M^2}{4\text{lcm}(M,q)},$$

so that $\mathbf{V}(q) = A(q) - B(q)$.

In this step, we also need to assume that M is an odd composite number satisfying $M \geq 5$. What we prove is that any optimal solution q^* for (4.13) satisfies that the greatest common divisor of q^* and M is not equal to 1 or M.

The following lemma rules out the possibility that q^* is either 1 or a multiple of M. We would gain no information about the divisors of M in those cases.

Lemma 4.6.1. *If $M \geq 5$ is an odd number, then any optimal solution q^* for (4.13) is between 2 an $M-1$.*

Proof. Note that $A(q)$ is convex differentiable with real minimum $q = \frac{M}{2}$. This implies that $A(q)$ is increasing in $[M/2, \infty)$ and therefore $A(q) \geq A(M)$ for all $q \geq M$. Also, since $\text{lcm}(M,q) \geq M$, we have that $B(q)$ is maximized at $q = M$, and therefore $-B(q) \geq -B(M)$ for all $q \geq M$. Altogether, we obtain that $\mathbf{V}(q) \geq \mathbf{V}(M)$ for $q \geq M$.

Direct computation and bounding shows that

$$\mathbf{V}(M) = M, \quad \text{and also} \quad \mathbf{V}(1) = \frac{M^2}{4} + 1 - \frac{M}{4} \geq M \text{ for all } M \geq 5$$

and therefore $\mathbf{V}(q) \geq M$ for every $q \notin \{2, 3, \ldots, M-1\}$.

On the other hand, we can upper bound $\mathbf{V}(q)$ for $q = \lfloor M/2 \rfloor$ as follows. It is easy to

check that $A(q)$ is decreasing in $(0, M/2]$. This, and a little bit of algebra gives

$$A\left(\left\lfloor \frac{M}{2} \right\rfloor\right) \leq A\left(\frac{M}{2} - 1\right) = A\left(\frac{M}{2}\right) + \frac{1}{\frac{M}{2} - 1} = M + \frac{2}{M-2}.$$

Also, since $\operatorname{lcm}(a,b) \leq ab$ and $\left\lfloor \frac{M}{2} \right\rfloor = \frac{M-1}{2}$ for odd M, we obtain:

$$B\left(\left\lfloor \frac{M}{2} \right\rfloor\right) \geq \frac{M}{4\lfloor M/2 \rfloor} = \frac{M}{2(M-1)}.$$

Combining these two inequalities for A and B, and using that $\frac{2}{M-2} - \frac{M}{2(M-1)} < 0$ for $M \geq 5$ we obtain

$$\mathbf{V}\left(\left\lfloor \frac{M}{2} \right\rfloor\right) \leq M + \frac{2}{M-2} - \frac{M}{2(M-1)} < M,$$

and therefore q^* cannot be greater than M or equal to 1 in any optimal solution to (4.13) □

What we prove now is an analogous result to Lemma 4.5.2 for the **CMJ** problem.

Lemma 4.6.2. *If M is an odd composite number, then every optimal solution q^* to (4.13) is not coprime with M.*

Proof. Suppose that M and q^* were coprimes. Then $B(q^*) = \frac{M}{4q^*}$, and therefore

$$\mathbf{V}(q^*) \geq v \equiv \min_{q \in \mathbb{R}} \left\{ A(q) - \frac{M}{4q} \right\} = \min_{q \in \mathbb{R}} \left\{ \frac{M^2}{4q}\left(1 - \frac{1}{M}\right) + q \right\}.$$

Defining $u = 1 - \frac{1}{M}$, it is easy to see that $v = M\sqrt{u}$, and this minimum is attained at $\bar{q} = \frac{v}{2}$.

To obtain a contradiction, we prove that there exists an integer q such that $\mathbf{V}(q) < v$. The main two properties we need are quite similar to those we required in the Clustering Problem. We need that q is close to $\bar{q}$, and also that q is not coprime with M.

Since M is an odd composite number, Lemma 4.1.2 guarantees that M has a non-trivial divisor $3 \leq p \leq \sqrt{M}$. Let $q \in [\bar{q} - p/2, \bar{q} + p/2]$ be any multiple of p. It follows, using Lemma 4.1.3, that

$$B(q) = \frac{M^2}{4\operatorname{lcm}(M,q)} = \frac{M \cdot \gcd(M,q)}{4q} \geq \frac{Mp}{4q},$$

and therefore

$$\mathbf{V}(q) \leq \frac{M^2}{4q} + q - \frac{Mp}{4q}.$$

If we write $q = \bar{q}(1+\varepsilon)$ for some $1+\varepsilon > 0$, we can rewrite this upper bound as follows:

$$\mathbf{V}(q) \leq \frac{M^2 u}{4q} + q - \frac{M(p-1)}{4q} = \frac{M^2 u}{4\bar{q}(1+\varepsilon)} + \bar{q}(1+\varepsilon) - \frac{M(p-1)}{4\bar{q}(1+\varepsilon)}$$

Now, note that $\frac{M^2 u}{4\bar{q}} = \bar{q}$, and $v = 2\bar{q}$, so we can rewrite the last upper bound as follows:

$$\mathbf{V}(q) \leq v + \frac{\bar{q}}{(1+\varepsilon)} + \bar{q}(\varepsilon - 1) - \frac{M(p-1)}{4\bar{q}(1+\varepsilon)} = v + \frac{1}{4\bar{q}(1+\varepsilon)}\left(4\varepsilon^2\bar{q}^2 - M(p-1)\right) \quad (4.14)$$

Finally, note that $|\varepsilon\bar{q}| \leq \frac{p}{2}$. Using this and $3 \leq p \leq \sqrt{M}$ we obtain the following bound for one of the terms in (4.14):

$$4\varepsilon^2\bar{q}^2 - M(p-1) \leq p^2 - M(p-1) \leq M - M(p-1) < 0.$$

Plugging this bound into (4.14), we conclude that $V(q) < v$, achieving the desired contradiction. □

Putting everything together

The algorithm $\mathscr{A}$ that finds a non-trivial divisor of a composite number M is very simple. First, it tests whether M is even. In this case $\mathscr{A}$ reports the non-trivial divisor 2 and terminates. Otherwise, it solves the following **GICF** problem associated to M:

$$\begin{aligned} \min \quad & \mathbf{U}(q_1,q_2) = \frac{M^2}{4}\left(\frac{1}{q_1} + \frac{1}{q_2} - \frac{1}{\operatorname{lcm}(q_1,q_2)}\right) + \frac{M^5}{2q_1} + \frac{M^3}{2}q_1 + q_2 \\ \text{s.t} \quad & q_1, q_2 \in \mathbb{Z}_+. \end{aligned} \quad (4.15)$$

Using Euclid's algorithm, $\mathscr{A}$ returns the non-trivial divisor $\gcd(M,q_2)$ and terminates.

Step 1, together with Lemmas 4.6.1 and 4.6.2 guarantee that the optimal solution of (4.15) satisfies $\gcd(M,q_2) \neq 1, M$, and therefore this number is actually a non-trivial divisor

of M.

Since the size of the coefficients in (4.15) is polynomial in $\log M$, and Euclid's algorithm is polynomial, it follows that $\mathscr{A}$ could be implemented in polynomial time if there were a polynomial time algorithm to solve the **JRP** in the fixed base **GICF** model. Therefore,

Theorem 4.6.1. *The **JRP** in the fixed base **GICF** model is **IF**-hard, even with two commodities.*

Bibliography

[1] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, pages 781–793, 2004.

[2] E. Arkin, D. Joneja, and R. Roundy. Computational complexity of uncapacitated multi-echelon production planning problems. *Operations Research Letters*, 8:61–66, 1989.

[3] A. A. Benczúr. Pushdown-reduce: An algorithm for connectivity augmentation and poset covering problems. *Discrete Appl. Math.*, 129(2-3):233–262, 2003.

[4] A. Brandstädt. The jump number problem for biconvex graphs and rectangle covers of rectangular regions. In J. Csirik, J. Demetrovics, and F. Gécseg, editors, *Proceedings of the 18th symposium on Fundamentals of Computation Theory (FCT) 1989*, volume 380 of *Lecture Notes in Computer Science*, pages 68–77, Heidelberg, 1989. Springer.

[5] A. Brandstädt, V. B. Le, and J. Spinrad. *Graph classes: A survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[6] S. Ceroi. A weighted version of the jump number problem on two-dimensional orders is NP-complete. *Order*, 20(1):1–11, 2003.

[7] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164:51–229, 2006.

[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms. 3rd ed.* MIT Press, Cambridge, MA, 2009.

[9] J.S. Dagpunar. Formulation of a multi item single supplier inventory problem. *Journal of the Operational Research Society*, 33(3):285–286, 1982.

[10] E. Dahlhaus. The computation of the jump number of convex graphs. In Vincent Bouchitté and Michel Morvan, editors, *Proceedings of the International workshop on orders, algorithms and applications (ORDAL) 1994*, volume 831 of *Lecture Notes in Computer Science*, pages 176–185, Heidelberg, 1994. Springer.

[11] M. Dyer, A. Frieze, and R. Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM (JACM)*, 38(1):1–17, 1991.

[12] H. Fauck. Covering polygons with rectangles via edge coverings of bipartite permutation graphs. *J. Inform. Process. Cybernet.*, 27(8):391–409, 1991.

[13] A. Frank. Finding minimum generators of path systems. *Journal of Combinatorial Theory, Series B*, 75(2):237–244, 1999.

[14] A. Frank and T. Jordán. Minimal edge-coverings of pairs of sets. *Journal of Combinatorial Theory, Series B*, 65(1):73–110, 1995.

[15] A. Frank and L. A. Végh. An algorithm to increase the node-connectivity of a digraph by one. *Discrete Optimization*, 5(4):677–684, 2008.

[16] S. Goyal. Optimum ordering policy for a multi item single supplier system. *Operational Research Quarterly*, pages 293–298, 1974.

[17] S. Goyal. A note on formulation of the multi-item single supplier inventory problem. *Journal of the Operational Research Society*, 33(3):287–288, 1982.

[18] E. Györi. A minimax theorem on intervals. *Journal of Combinatorial Theory, Series B*, 37(1):1–9, 1984.

[19] M. Habib. Comparability invariants. *North-Holland Mathematics Studies*, 99:371–385, 1984.

[20] G.H. Hardy and E.M. Wright. *An introduction to the theory of numbers*. Clarendon press, 1984.

[21] P. Jackson, W. Maxwell, and J. Muckstadt. The joint replenishment problem with a powers-of-two restriction. *IIE Transactions*, 17(1):25–32, 1985.

[22] D. Joneja. The joint replenishment problem: New heuristics and worst case performance bounds. *Operations Research*, 38(4):711–723, 1990.

[23] M. Kaspi and M. Rosenblatt. On the economic ordering quantity for jointly replenished items. *International Journal of Production Research*, 29(1):107–114, 1991.

[24] B.H. Korte and J. Vygen. *Combinatorial optimization: theory and algorithms*, volume 21. Springer Verlag, 2006.

[25] R. Levi, T. Magnanti, and E. Zarybnisky. Maintenance Scheduling for Modular Systems. *MSOM 2010*, 2010.

[26] R. Levi, R. Roundy, and D. Shmoys. Primal-dual algorithms for deterministic inventory problems. In *Proceedings of the 36th annual ACM symposium on Theory of computing*, pages 353–362. ACM, 2004.

[27] R. Levi, R. Roundy, D. Shmoys, and M. Sviridenko. A constant approximation algorithm for the one-warehouse multiretailer problem. *Management Science*, 54(4):763–776, 2008.

[28] L. Lovász and S. Vempala. Simulated annealing in convex bodies and an o*(n4) volume algorithm. *Journal of Computer and System Sciences*, 72(2):392–417, 2006.

[29] A. Lubiw. A weighted min-max relation for intervals. *Journal of Combinatorial Theory, Series B*, 53(2):151–172, 1991.

[30] R.M. McConnell and J.P. Spinrad. Linear-time transitive orientation. In *Proceedings of the 8th annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 19–25. Society for Industrial and Applied Mathematics, 1997.

[31] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2004)*, pages 248–255, 2004.

[32] J. Muckstadt and R. Roundy. Analysis of multistage production systems. In A. Rinnooy and P. Zipkin, editors, *Handbooks in Operations Research and Management Science*, volume 4. North Holland, Amsterdam, 1993.

[33] H. Müller. Recognizing interval digraphs and interval bigraphs in polynomial time. *Discrete Applied Mathematics*, 78(1-3):189–205, 1997.

[34] Haiko Müller. Alternating cycle-free matchings. *Order*, 7:11–21, 1990.

[35] R.P. Nicolai and R. Dekker. Optimal maintenance of multi-component systems: A review. *Complex system maintenance handbook*, pages 263–286, 2008.

[36] E. Porras and R. Dekker. Generalized solutions for the joint replenishment problem with correction factor. *International Journal of Production Economics*, 113(2):834–851, 2008.

[37] M.E. Posner. Approximation procedures for the one-warehouse multi-retailer system. *Management Science*, 40(10):1305–1316, 1994.

[38] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[39] R. Roundy. 98%-effective integer-ratio lot-sizing for one-warehouse multi-retailer systems. *Management Science*, 31(11):1416–1430, 1985.

[40] A. Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, Berlin, 2003.

[41] A. Schulz and C. Telha. Approximation algorithms and hardness results for the joint replenishment problem with constant demands. In *Proceedings of the 19th Annual European Symposium on Algorithms (ESA 2011)*, volume 6942 of *Lecture Notes in Computer Science*, pages 628–639. Springer, 2011.

[42] A. M.S. Shrestha, S. Tayu, and S. Ueno. On two-directional orthogonal ray graphs. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS 2010)*, pages 1807–1810, 2010.

[43] J. A. Soto. *Contributions on Secretary Problems, Independent Sets of Rectangles and Related Problems*. PhD thesis, MIT, 2011.

[44] J.A. Soto and C. Telha. Jump number of two-directional orthogonal ray graphs. In *Proceedings of the 15th Conference on Integer Programming and Combinatorial Optimization (IPCO 2011)*, volume 6655 of *Lecture Notes in Computer Science*, pages 389–403. Springer, 2011.

[45] G. Steiner and L. K. Stewart. A linear time algorithm to find the jump number of 2-dimensional bipartite partial orders. *Order*, 3:359–367, 1987.

[46] C. Teo and D. Bertsimas. Multistage lot sizing problems via randomized rounding. *Operations Research*, 49(4):599–608, 2001.

[47] W.T. Trotter. *Combinatorics and partially ordered sets: Dimension theory*, volume 6. Johns Hopkins Univ Pr, 2001.

[48] L. A. Végh. *Connectivity Augmentation Algorithms*. PhD thesis, Eötvös Loránd University, 2010.

[49] R. Wildeman, J. Frenk, and R. Dekker. An efficient optimal solution method for the joint replenishment problem. *European Journal of Operational Research*, 99(2):433–444, 1997.