Applications of Integer Programming and Decomposition to Scheduling Problems: the Strategic Mine Planning Problem and the Bin Packing Problem with Time Lag

Orlando Rivera Letelier

CO-SUPERVISED THESIS PRESENTED

TO OBTAIN THE QUALIFICATION OF

DOCTOR OF THE UNIVERSITY OF BORDEAUX AND OF UNIVERSIDAD ADOLFO IBÁÑEZ

DOCTORAL SCHOOL OF MATHEMATICS AND COMPUTER SCIENCE
SPECIALIZATION IN APPLIED MATHEMATICS AND SCIENTIFIC CALCULATION
UNIVERSITY OF BORDEAUX

BUSINESS SCHOOL AND FACULTY OF ENGINEERING AND SCIENCE
DOCTORAL PROGRAM IN INDUSTRIAL ENGINEERING AND OPERATIONS RESEARCH
UNIVERSIDAD ADOLFO IBÁÑEZ

By Orlando Rivera Letelier

Applications of Integer Programming and Decomposition to Scheduling Problems: the Strategic Mine Planning Problem and the Bin Packing Problem with Time Lag

Under the direction of Ruslan Sadykov and Marcos Goycoolea,
and co-supervision of François Clautiaux

Defended on February 26th, 2021.

Members of the examination panel :

Jorge PEREIRA, Universidad Adolfo Ibáñez. President of the jury.
Alexandra NEWMAN, Colorado School of Mines.
Mourad BAÏOU, CNRS.
Michel GAMACHE, Polytechnique de Montréal.
Ruslan SADYKOV, University of Bordeaux, supervisor.
Marcos GOYCOOLEA, Universidad Adolfo Ibáñez, supervisor.

Title:

Applications of Integer Programming and Decomposition to Scheduling Problems: the Strategic Mine Planning Problem and the Bin Packing Problem with Time Lag

Abstract:

In scheduling problems, the goal is to assign time slots to a set of activities. In these problems, there are typically precedence constraints between activities that dictate the order in which they can be carried out and resource constraints that limit the number that can simultaneously be executed. In this thesis, we develop mixed integer programming methodologies, based on decomposition methods, for two very different classes of scheduling problems. These are the Strategic Open Pit Mine Planning Problem (SOPMP) and the Bin Packing Problem with Time Lags.

Given a discretized representation of an orebody known as a block model, the SOPMP that we consider consists of defining which blocks to extract, when to extract them, and how or whether to process them, in such a way as to comply with operational constraints and maximize net present value. These problems are known to be very difficult due to the large size of real mine planning problems (eg, millions of blocks, dozens of years). They are also very important in the mining industry. Every major mining operation in the world must solve this problem, at the very least, on a yearly basis.

In this thesis, we tackle the SOPMP in Chapters 2 and 3.

In Chapter 2 we begin by studying a lagrangean algorithm developed by Dan Bienstock and Mark Zuckerberg (henceforth, the BZ algorithm) in 2009 for solving the LP relaxation of large instances of SOPMP. In this study we generalize the classes of problems that can be solved with the BZ algorithm, and show that it can be cast as a special type of column generation algorithm. We prove, for general classes of mixed integer programming problems, that the BZ relaxation provides a bound that lies between the LP relaxation and Dantzig-Wolfe bounds. We further develop computational speed-ups that improve the performance of the BZ algorithm in practice, and test these on a large collection of data-sets.

In Chapter 3 we deal with the problem of computing integer-feasible solution to SOPMP. Using the BZ algorithm developed in Chapter 2, we develop heuristics for this. In addition, we develop pre-procesing algorithms that reduce problem size, and embed the BZ algorithm in a branch-and-cut framework that makes use of two new classes of cutting planes. When comparing the value of the heuristics to the LP relaxation bound, the average gap computed is close to 10%. However, when applying the pre-processing techniques and cutting planes, this is reduced to 1.5% at the root node. Four hours of branching further reduces this to 0.6%.

In Chapter 4, the BPPTL is presented. This is a generalization of the Bin

Packing Problem in which bins must be assigned to time slots, while satisfying precedence constraints with lags. Two integer programming formulations are proposed: a compact formulation that models the problem exactly, and an extended formulation that models a relaxation. For two special cases of the problem, the case with unlimited bins per period and the case with one bin per period and non-negative time lags, we strengthen the extended formulation with a special family of constraints. We propose a branch-cut-and-price (BCP) algorithm to solve this formulation, with separation of integer and fractional solutions, and a strong diving heuristic. Computational experiments confirm that the BCP algorithm outperforms solving the compact formulation with a commercial solver. Using this approach we were able to optimally solve 70% of a class of previously open instances of this problem.

Keywords:

Mix integer linear programming, strategic mine planning, Bienstock-Zuckerberg algorithm, branch-price-and-cut algorithm, bin packing problem with time lags

Unité de recherche:

Institut de Mathématiques de Bordeaux.
351, cours de la Libération - F 33 405 Talence, France.

Titre:

Applications de de la programmation en nombres entiers et la décomposition aux problèmes d'ordonnancement : le problème de la planification stratégique des mines et le problème de bin packing avec délais.

Resumé:

Dans les problèmes de planification, l'objectif est d'attribuer des plages horaires à un ensemble d'activités. Dans ces problèmes, il existe généralement des contraintes de précédence entre les activités qui dictent l'ordre dans lequel elles peuvent être exécutées et des contraintes de ressources qui limitent le nombre d'activités pouvant être exécutées simultanément. Dans cette thèse, nous développons des méthodologies de programmation en nombres entiers, basées sur des méthodes de décomposition, pour deux classes très différentes de problèmes d'ordonnancement. Il s'agit du problème de planification stratégique des mines à ciel ouvert (SOPMP) et du problème de bin packing avec délais (BPPTL).

Etant donné une représentation discrétisée d'un gisement appelé modèle de bloc, le SOPMP que nous considérons consiste à définir les blocs à extraire, quand les extraire, comment ou s'il faut les traiter, de manière à respecter les contraintes opérationnelles et maximiser la valeur actualisée nette. Ces problèmes sont connus pour être difficiles en raison de l'ampleur des problèmes réels de planification minière. Ils sont également importants dans l'industrie minière. Chaque grande opération minière dans le monde doit résoudre ce problème au moins une fois par an.

Dans le chapitre 2, nous commençons par étudier un algorithme lagrangien développé par Dan Bienstock et Mark Zuckerberg (qu'on appellera algorithme BZ) en 2009 pour résoudre la relaxation LP de grandes instances de SOPMP. Dans cette étude, nous généralisons les classes de problèmes qui peuvent être résolus avec l'algorithme BZ, et montrons qu'il peut être exprimé comme un algorithme de génération de colonnes. Nous prouvons, pour les classes générales de problèmes de programmation en nombres entiers mixtes, que la relaxation BZ fournit une borne qui se situe entre la relaxation LP et les bornes de Dantzig-Wolfe. Nous développons en outre des accélérations de calcul qui améliorent les performances de l'algorithme BZ dans la pratique.

Dans le chapitre 3, nous traitons le problème du calcul d'une solution entière réalisable pour SOPMP. En utilisant l'algorithme BZ développé au chapitre 2, nous développons des heuristiques pour ce problème. En outre, nous développons des algorithmes de prétraitement qui réduisent la taille du problème et intégrons l'algorithme BZ dans un algorithme de branch-and-cut qui utilise deux nouvelles classes de plans sécants. En comparant la valeur de l'heuristique à la borne de relaxation LP, l'écart moyen calculé est proche de 10%. Cependant, lors de

l'application des techniques de prétraitement et des plans sécants, il se réduit à 1,5% au nœud racine. Quatre heures de branchement réduisent encore ce pourcentage à 0,6%.

Le chapitre 4 présente le BPPTL. Il s'agit d'une généralisation du problème de bin packing, dans lequel les containers doivent être affectés à des créneaux horaires qui respectent des contraintes de délai entre les articles. Deux formulations de programmation en nombres entiers sont proposées : une formulation compacte qui modélise exactement le problème, et une formulation étendue qui modélise une relaxation. Pour deux cas particuliers du problème, le cas avec un nombre illimité de bin par période et le cas avec un bin par période et des délais non négatifs, nous renforçons la formulation étendue avec une famille spéciale de contraintes. Nous proposons un algorithme de *branch-and-cut-and-price* (BCP) pour résoudre cette formulation, avec séparation des solutions entières et fractionnaires, et une heuristique de *strong diving*. Les expérimentations confirment que l'algorithme BCP est plus performant que la résolution de la formulation compacte avec un solveur commercial. En utilisant cette approche, nous avons pu résoudre de manière optimale 70% d'une classe d'instances précédemment ouvertes de ce problème.

Mots clés:

Programmation linéaire mixte en nombres entiers, planification stratégique des mines, algorithme de Bienstock-Zuckerberg, algorithme de branch-price-and-cut, problème de bin packing avec délais

Título:

Aplicaciones de Programación Entera y Descomposición a Problemas de Scheduling: el Problema de Planificación Minera Estratégica y el Problema de Bin Packing con Lags de Tiempo

Resumen:

En problemas de agendamiento, el objetivo es asignar tiempos a un conjunto de actividades. En estos problemas, típicamente hay restricciones de precedencias entre actividades que dictan el orden en que deben ser ejecutadas y restricciones de recursos que limitan el número que pueden ser ejecutadas simultáneamente. En esta tesis desarrollamos metodologías de programación entera mixta, basada en métodos de descomposición, para dos clases de problemas de agendamiento. Estos son el problema de planificación minera a rajo abierto (SOPMP) y el problema de bin packing con *lags* de tiempo (BPPTL).

Dada una representación discretizada de un yacimiento minero, conocida como modelo de bloques, el SOPMP que consideramos consiste en definir qué bloques extraer, cuándo extraerlos, y el cómo procesarlos, de forma que se satisfagan restricciones operacionales y se maximice el valor presente neto. Es sabido que estos problemas son muy difíciles debido al gran tamaño de problemas de planificación minera reales (millones de bloques, docenas de años), si bien son muy importantes para la industria minera. Cada operación minera de gran tamaño en el mundo debe resolver este problema al menos una vez al año.

En el capítulo 2 empezamos estudiando un algoritmo lagrangeano desarrollado por Dan Bienstock y Mark Zuckerberg (el algoritmo de BZ) en 2009 para resolver la relajación LP de instancias grandes de SOPMP. En este estudio generalizamos la clase de problemas que pueden ser resueltos con el algoritmo de BZ, y mostramos que puede ser visto como un tipo especial de algoritmo de generación de columnas. Probamos, para la clase general de problemas de programación entera mixta, que la relajación de BZ provee una cota que está entre la de la relajación LP y la cota de Dantzig-Wolfe. Adicionalmente desarrollamos mejoras computacionales que mejoran el desempeño del algoritmo de BZ en la práctica, y las testeamos en una colección grande de instancias.

En el capítulo 3 lidiamos con el problema de computar soluciones enteras factibles de SOPMP. Usando el algoritmo de BZ desarrollado en el capítulo 2, desarrollamos heurísticas. Adicionalmente, desarrollamos algoritmos de preproceso que reducen el tamaño del problema, e insertamos el algoritmo de BZ dentro de un procedimiento de *branch-and-cut* que usa dos clases nuevas de planos cortantes. Al comparar el valor de las heurísticas con la cota de la relajación LP, el gap promedio computado es cercano a 10%. Sin embargo, al aplicar las técnicas de pre-proceso y planos cortantes, este gap se reduce a un 1.5% en el nodo raíz. Tras cuatro horas de branche este se reduce a un 0.6%.

En el capítulo 4, el problema de bin packing con lags de tiempo es presentado.

Este es una generalización del problema de bin packing, en el que los *bins* deben ser asignados a períodos de tiempo, satisfaciendo restricciones de precedencia con lags de tiempo. Proponemos dos formulaciones de programación entera: una formulación compacta que modela el problema de forma exacta, y una formulación extendida que modela una relajación. Para dos casos especiales del problema, el caso con un número ilimitado de bins por período y el caso en que sólo se permite un bin por período y los lags de tiempo son no-negativos, agregamos una familia espacial de restricciones a la formulación extendida para fortalecerla. Proponemos un un algoritmo de *branch-cut-and-price* (BCP) para resolver esta formulación, con separación de soluciones enteras y de soluciones fraccionarias, y una heurística de *strong diving*. Experimentos computacionales confirman que el algoritmo BCP supera el desempeño de resolver la formulación compacta con un software de optimización comercial. Usando este enfoque podemos resolver óptimamente el 70% de una clase de instancias previamente abiertas de este problema.

Palabras Clave:

Programación lineal entera mixta, planificación minera estratégica, algoritmo de Bienstock-Zuckerberg, algoritmo de branch-price-and-cut, problema de bin packing lags de tiempo

List of Papers

Chapter 2

Muñoz, G. and Espinoza, D. and Goycoolea, M. and Moreno, E. and Queyranne, M. and Rivera Letelier, O. "A study of the Bienstock-Zuckerberg algorithm: applications in mining and resource constrained project scheduling". In: *Computational Optimization and Applications*. Vol. 69, (2020), pp. 1425–1444. DOI: 10.1007/s10589-017-9946-1.

Chapter 3

Rivera Letelier, O. and Espinoza, D. and Goycoolea, M. and Muñoz, G. and Moreno, E. "Production Scheduling for Strategic Open Pit Mine Planning: A Mixed-Integer Programming Approach". In: *Operations Research*. Vol. 68, no. 2 (2018), pp. 501–534. DOI: 10.1287/opre.2019.1965.

Chapter 4

Rivera Letelier, O. and Clautiux, F. and Sadykov, R. "Bin Packing with Time Lags". *Submitted for publication.*

Contents

List of Papers **vii**

Contents **ix**

1 Introduction **1**
1.1 Introduction in english 1
1.2 Introduction en français 7
References 14

2 A study of the Bienstock-Zuckerberg algorithm: applications in mining and resource constrained project scheduling **17**

3 Production Scheduling for Strategic Open Pit Mine Planning: A Mixed-Integer Programming Approach **55**

4 Bin Packing with Time Lags **97**

5 Conclusions **133**
References 135

Chapter 1

Introduction

We present the introduction of this thesis in both english and french languages, and they constitute a substantial summary of this thesis.

1.1 Introduction in english

In this thesis we develop integer linear programming techniques for solving two classes of scheduling problems. The problems that we consider consist of scheduling activities that must be assigned to time periods, in such a way as to comply with precedence and resource consumption constraints. We consider two variants of this class of problems. The first, which we call the *General Production Scheduling Problem* (GPSP), is motivated by strategic mine planning problems. The GPSP generalizes the resource constrained project scheduling problem (RCPSP), and can be used to model many different problems in mine planning such as open pit phase design, open pit direct block scheduling and underground scheduling problems. The second, which we call the *Bin Packing Problem with Time Lags* (BPPTL) is a generalization of the classical bin packing problem (BPP), in which items are assigned to bins over times, and in which assignments must satisfy precedence constraints with time lags. By interpreting bin capacity constraints as resource consumption constraints, it can be seen that BPPTL and GPSP are actually very similar problems. The main difference is that in the BPPTL, in its general form, the number of bins used is not fixed. Thus, the BPPTL can be interpreted as a variant of the GPSP in which capacity can be purchased, and in which the objective is to minimize this cost. Chapters 2 and 3 focus on studying the GPSP, and chapter 4 studies BPPTL. We now describe this problems in more detail, emphasizing the state of the art prior to our work, the importance of these problems, and the papers published and expected from this research.

General Project Scheduling Problem

The GPSP can be described as follows. A set of activities can be scheduled at time periods, each activity has an associated duration that determines how long it consumes resource after starting. Precedence relationships with time lags determine how much time must elapse between the start time of pairs of activities. In multi-modal extensions of this problem, activities can be performed in different ways (or modes). Changing the mode of an activity affects its resource utilization, and its objective function value. We will assume, throughout this work, that changing the mode of an activity does not change the duration. In batch-processing extensions, activities are grouped together in clusters that

must be scheduled to start at the same time period. The general production scheduling problem that we study is the multi-modal batch-processing resource constrained project scheduling problem.

Formally, this class of problems can be described as follows. Consider a set of activities $\mathcal{A}$ and a discrete set of time periods $\mathcal{T} = \{1, \dots, T\}$. Let $\mathcal{C}$ be a partition of the activity set $\mathcal{A}$. We will say that each element (or set) in $\mathcal{C}$ is a cluster, or batch. These sets will represent activities that must begin simultaneously. We will assume that, unless specifically enforced otherwise, scheduling each cluster is optional. For each cluster $c \in \mathcal{C}$, let $P(c) \subseteq \mathcal{C}$ represent its set of predecessor clusters. That is, if c_2 is scheduled, then all clusters $c_1 \in P(c_2)$ must also be scheduled. For each $c_2 \in \mathcal{C}$ and each $c_1 \in P(c_2)$ let $l_{c_1 c_2}$ represent the corresponding precedence lag. This number represents the number of time periods that must elapse between the start time of c_1 and c_2. That is, if c_2 starts in time t or before, then c_1 must start in time $t - l_{c_1 c_2}$ or before. Note that if $l_{c_1 c_2} < 0$, then, even though c_1 is a predecessor of c_2, it is admissible that c_1 start after c_2, as long as it starts at most $|l_{c_1 c_2}|$ periods after. Let $\mathcal{M}_a$ represent the set of modes in which activity $a \in \mathcal{A}$ can be scheduled. Each scheduled activity must be assigned exactly one mode. The mode of an activity cannot be changed once it starts. For each activity $a \in A$, let d_a represent its duration. We assume that activities cannot be interrupted. Once they are scheduled they are running for d_a time periods or until the time horizon T is reached. Let $\mathcal{R}$ represent the set of necessary resources of the project, and for $r \in \mathcal{R}$ let $q^r_{a,m}$ be the amount of resource of type r consumed each time period by activity a when running in mode m. For each $t \in \mathcal{T}$ and $r \in \mathcal{R}$ let Q^t_r represent the amount of resources of type r available in time period t. Finally, let $v_{a,m,t}$ represent the profit obtained by scheduling activity a to start in time t, and run in mode m.

To model this problem we define binary variables $x_{c,t}$, indicating if cluster c is scheduled to start in time t or not, and binary variables $y_{a,m,t}$ indicating if activity a is scheduled to start running in time t with mode m. Using these variables the problem we consider is as follows:

$$\max \sum_{a \in A} \sum_{m \in M_a} \sum_{t \in T} v_{a,m,t} y_{a,m,t}, \tag{1.1a}$$

s.t. (1.1b)

$$\sum_{t \in \mathcal{T}} x_{c,t} \leq 1 \qquad \forall c \in \mathcal{C}, \tag{1.1c}$$

$$\sum_{m \in \mathcal{M}_a} y_{a,m,t} = x_{c,t} \qquad \forall t \in \mathcal{T}, \forall c \in \mathcal{C}, \forall a \in c, \tag{1.1d}$$

$$\sum_{a \in \mathcal{A}} \sum_{m \in \mathcal{M}_a} \sum_{s=\max\{1, t-d_a+1\}}^{t} q_{r,a,m} y_{a,m,s} \leq Q_{r,t} \qquad \forall r \in \mathcal{R}, \forall t \in \mathcal{T}, \tag{1.1e}$$

$$\sum_{s=1}^{t} x_{c_2,s} \leq \sum_{s=1}^{t-l_{c_1,c_2}} x_{c_1,s} \qquad \forall c_1 \in \mathcal{C}, \forall c_2 \in P(c_1), \forall t \in \mathcal{T}. \tag{1.1f}$$

In this formulation, constraints (1.1c) impose that a cluster is extracted at most once. Constraints (1.1d) impose that the time period an activity is scheduled is the same as the one of the cluster it belongs to. They also impose that each scheduled activity is assigned a mode. Constraints (1.1e) impose the resources capacity limit. In this constraints, if an activity a is scheduled at time period t, it consumes resources on time periods $\{t, \dots, t + d_a - 1\}$. Finally constraints (1.1f) impose the precedence with lags constraints. Here if cluster c_2 is scheduled at time period t, then cluster $c_2 \in P(c_1)$ must be scheduled at time period $t - l_{c_1,c_2}$ or before.

Note that the class of problems represented by formulation (1.1) is very broad, and includes some very difficult problems, including the job-shop and resource constrained project scheduling problems (RCPSPs). As reported by [VCB16], there are instances of RCPSP with as few as 90 jobs that to date have not been solved to optimality.

In this thesis we develop integer programming methodologies for solving model (1.1), with a focus on strategic open pit mine planning. In the strategic open pit mine planning problem (SOPMPP) the goal is to determine where, when and how to mine using open pit methods an ore body in order to maximize the value of the operation. A mining deposit or ore body is discretized into regular blocks of a three dimensional grid to define an economic block model. Time is discretized over time periods that usually represent years. The open pit methods impose spacial and geological restrictions requiring that to mine a given block a certain set in the shape of a cone over the block must be previously mined. Operational restrictions could require that the extraction is made in large groups of contiguous blocks. A mined block is sent to one of different possible destinations which could include processing plants, waste dumps, stockpiles, etc. Restrictions are imposed due to limited capacity for how much material can be mined or sent to specific destinations. The composition of the material in blocks determine the profit obtained for sending it to each specific destination.

The SOPMPP is a fundamental problem to solve in the mining industry, since it is necessary to evaluate the worthiness to exploit a mining deposit, and it is solved in practice on a yearly basis in order to re-evaluate current strategic plans with the updated information obtained of the block model. This problem has been studied since the 1960s. An integer programming formulation which is a particular case of (1.1) was proposed by T. B. Johnson [Joh68; Joh69], though the size of the problem made it impossible to solve it in practice, since real size instances can easily have hundred of thousands to several millions blocks, and many times that variables and constraints. Mining companies traditionally have used commercial software systems based on heuristics methods to obtain solutions for this problem. These software have been available since the 1980's. Two notable heuristics used to face the problem are the Lane's Algorithm's and the Nested Pit Methodology, that were combined in a methodology by the commercial software system *Whittle* which is in practice used until today [GEO19]. Integer programming approaches have been proposed since the 2000's [Ble+10; Bol+09], based in aggregation techniques to reduce the dimension of the problem and cutting planes to strength the formulation. However, the size

of the integer programming formulations makes it hard in practice to even solve the LP relaxation of the formulation.

The study of the SOPMPP in this thesis is based in two important developments that preceded it. First, a breakthrough in solving the LP relaxation of the problem was the algorithm proposed by Bienstock and Zuckerberg, the BZ algorithm [BZ09; BZ10]. This algorithm, based in a decomposition that takes advantage of the combinatorial structure of the GPSP formulation, iteratively solves much smaller linear programs and a combinatorial problem that can be efficiently solved. Second, a rounding heuristic proposed by Chicoisne et al. [Chi+12] produced solutions with small optimality gaps for a simplified version of the problem by rounding a solution of the LP relaxation of the problem.

The first part of this thesis, developed in chapter 2, studies deeply the BZ algorithm. We show that this algorithm can be extended to underground mine planning and other classes of scheduling problems. We also show that the algorithm can be cast as a column generation scheme, allowing us to compare it to the Dantzig-Wolfe decomposition, and to characterize the bounds it produces. Finally, computational speed-ups are presented, as well as a computational study that describes the performance of the algorithm in applications in strategic mine planning resource constraint project scheduling problems. The chapter is a pre-print of the following article:

A study of the Bienstock-Zuckerberg algorithm, Applications in Mining and Resource Constrained Project Scheduling. Munoz, G;. Espinoza, D.; Goycoolea, M.; Moreno, E.; Queyranne, M.; Rivera, O. Computational Optimization and Applications. Volume 69, Issue 2, pp 501 – 534. March, 2018.

In this paper it is confirmed that the BZ algorithm is an effective algorithm for solving large classes of mine planning problems. It extends the reformulation required for the algorithm to fit the more general GPSP formulation. Moreover, it develops techniques that should be taken into account when implementing the BZ algorithm that significantly improve the performance. It establishes the theoretical quality of the bounds afforded by the BZ algorithm, it develops the path-compression and k-step computational speed-ups, and it conducts a computational study to show that the BZ algorithm significantly outperforms solving the LP relaxation by means of Dantzig-Wolfe decomposition or other linear programming algorithms. My specific personal contribution to this paper was developing the more general reformulation, characterizing the quality of the bounds obtained, implementation of most of the algorithms and speed ups mentioned, and the execution of the computational study presented in the paper, including the recollection and manipulation of real mine instances, and the implementation of the platform that allowed to express the problem as an optimization problem suited for the BZ algorithm.

The second part of this thesis, developed in chapter 3, proposes a methodology to address the SOPMPP. It builds on a classical division of the SOPMPP into two subproblems, the phase design problem and the production scheduling problem, both of which that can be formulated as particular cases of the GPSP.

It uses the algorithm described in the chapter 2, but it extends this work by introducing a full methodology dealing with the newly introduced integrality constraints. The methodology combines customized heuristics, including the adaptation of the heuristic proposed in [Chi+12] to this framework, and it includes both well known and other newly proposed pre-processing techniques, and valid inequalities, as well as a branching scheme is presented. By combining all of the aforementioned techniques, we show that real-world problem instances of strategic mine planning problems can be solved to near optimality ($< 1\%$) in just a few hours. The chapter is a pre-print of the following article recently published:

Production scheduling for strategic open pit mine planning: A mixed integer programming approach. Rivera, O; Espinoza, D; Goycoolea, M; Moreno, E.; Munoz, G. Operations Research. Published Online. Aug 27, 2020.

The importance of this paper is that it first presents a methodology to solve the SOPMPP by means of exact methods of integer programming. It shows a way to model the SOPMPP diving the problem into two stages, both of them being possible to formulate as different GPSPs that can be solved to small optimality gaps with the methodology presented in the paper. A particularly important contribution of this paper is the introduction of the hourglass cuts. The hourglass cuts are novel in using a part of the structure of the formulation that has not been used before to develop valid inequalities. These cuts produced an important impact in closing the gap. My specific personal contribution in this paper was that I developed most of the implementation used for the computational experiments (with the exception of the branching algorithm). I also came up with all of the new hourglass cutting planes, the proposed new heuristics, and adapted and implemented most of the pre-processing schemes, heuristics and older classes of cutting planes originally proposed for simpler variants of the problem, to the general class of problems considered in the paper. I also carried out all of the computational experiments.

The methodology proposed for the strategic open pit mining problem has been tested in practice by mining companies, leading to satisfactory solutions and production scheduling plans.

Bin Packing Problem with Time Lags

The bin packing problem with time lags (BPPTL) is a generalization of the bin packing problem. In the bin packing problem a set of weighted items must be assigned to the minimum possible number of equally capacitated bins. In the BPPTL the bins must additionally be scheduled in time, and, in such a way as to satisfy a given set of precedence constraints with time lags. These time lags can be either positive or negative.

The BPPTL can be used to model tasks that must be performed repeatedly, using limited resources that must be paid for. Positive and negative lags allow fixing the minimum and maximum time that must elapse between consecutive

executions of each task. An application of this is the phytosanitary treatments in a vineyard, in which areas of the vineyard require periodic treatment that should be performed with an ideal, given frequency. There is flexibility on the frequency of the tasks, though consecutive treatments should not be performed too close or too far apart in time. Tasks consume resources that must be rented per day, and the same resources could be used to perform tasks in different areas of the vineyard if they must be performed the same day.

The BPPTL is a new problem that extends several other generalizations of the bin packing problem. The bin packing problem with precedences (BPP-P) [DDI12; Per16] corresponds to the specific case of BPPTL in which all lags are equal to one. The simple assembly line balancing problem (SALBP) [Per15] corresponds to the specific case of BPPTL in which all lags are equal to zero. The bin packing problem with generalized precedence constraints (BPP-GP) [KDI17] also generalizes the bin packing problem and considers precedence constraints with time lags. However, BPP-GP considers a different objective function than BPPTL and assumes lags are non-negative. Despite this the BPP-GP coincides with the BPPTL when when lags are zero or one.

Formally, in the BPPTL, we are given a directed valued graph $G = (V, A, l)$, where V is a set of items that must be assigned to bins, A is a set of arcs defining precedence relationships, and $l : A \longrightarrow \mathbb{Z}$ represents the time lags of arcs. We are also given a weight $w_i \in \mathbb{Z}^+$ for each item $i \in V$, and a bin capacity $W \in \mathbb{Z}^+$. The problem consists in partitioning the items into bins, and scheduling each item $i \in V$ to a time period $p(i) \in \mathbb{Z}^+$, in such a way that: (a) items assigned to a bin are scheduled in the same time period, (b) the cumulative weight of items assigned to each bin does not exceed the prescribed capacity W, and (c) for each $(i, j) \in A$, it holds that $p(i) + l_{i,j} \leq p(j)$. Additionally, we can be given a constraint on the maximum number of bins L that can be assigned to the same time period. The objective of the problem is to find such an assignment that uses the minimum possible number of bins.

The BPPTL is presented in Chapter 4, where we study the problem from an integer programming perspective. We propose a compact formulation based on a time-index formulation, and propose different variants, comparing their performance. The time-index formulations proposed is very similar to that used for formulating the GPSP (1.1), in that the same variables and precedence constraints are considered. We also propose an extended formulation that defines a relaxation of the BPPTL. We develop a branch-cut-and-price algorithm based on this formulation, that allows to solve to optimality two specific cases of the BPPTL: (a) the case with unlimited bin per periods (BPPTL$^\infty$), and (b) the case in which time lags are non-negative and in which only one bin is allowed per period (BPPTL$^1_+$). For both BPPTL$^\infty$ and BPPTL$^1_+$ we characterize the feasible solutions of the extended formulation, and develop separation algorithms for both integer and fractional solutions. Finally, we develop a strong diving heuristic, and conduct a computational study to compare the performance of the branch-cut-and-price algorithm and a commercial software over the compact formulation. We show that for the instances inspired in the phytosanitary treatments problem the branch-cut-and-price approach outperforms the use of a

CPLEX on the compact formulation. We also show that for the instances of BPP-P, SALBP and BPP-GP, we are able to improve both the lower and upper bounds known in the literature. Combining both known and improved bounds, the optimality status is reached for 70% of instances with less than 100 items that were previously open problems in the literatue. This chapter is a pre-print of a submitted paper:

Bin Packing with Time Lags. Rivera, O; Clautiaux, F; Sadykov, R. Manuscript submitted for publication.

My specific personal contributions to this paper were the mathematical characterization of feasible solutions, the separation algorithm for fractional solutions, the generation of instances inspired in real applications, the model and implementation of the compact formulation and its variants, the implementation of the branch-cut-and-price algorithm using the generic branch-and-price academic software BapCod and the computational study. The implementation of the algorithm includes the model of the problem, the integer and fractional separation procedures for both BPPTL^1_+ and BPPTL^∞, and the separator for the diving heuristic.

1.2 Introduction en français

Dans cette thèse, nous développons des techniques de programmation linéaire en nombres entiers pour résoudre deux classes de problèmes d'ordonnancement. Les problèmes que nous considérons consistent à programmer des activités qui doivent être affectées à des périodes de temps, de manière à respecter des contraintes de priorité et de consommation de ressources. Nous considérons deux variantes de cette classe de problèmes. La première, que nous appelons le *problème général de planification de la production* (GPSP), est motivée par des problèmes de planification stratégique dans le contexte minier. Le GPSP généralise le problème de planification de projets sous contraintes de ressources (RCPSP), et peut être utilisé pour modéliser de nombreux problèmes différents de planification minière tels que la conception de la phase d'exploitation à ciel ouvert, la planification directe des blocs à ciel ouvert et les problèmes de planification souterraine. Le second, que nous appelons le *problème de bin packing avec délais* (BPPTL) est une généralisation du problème classique de bin packing (BPP), dans lequel les articles sont affectés aux bins au fil du temps, et dans lequel les affectations doivent satisfaire à des contraintes de délai entre les articles. En interprétant les contraintes de capacité des bins comme des contraintes de consommation de ressources, on peut voir que le BPPTL et le GPSP sont en fait des problèmes très similaires. La principale différence est que dans le BPPTL, sous sa forme générale, le nombre de bins utilisés n'est pas fixé. Ainsi, le problème BPPTL peut être interprété comme une variante du GPSP dans laquelle la capacité peut être achetée et dont l'objectif est de minimiser ce coût. Les chapitres 2 et 3 se concentrent sur l'étude du GPSP, et le chapitre 4

étudie le BPPTL. Nous décrivons maintenant ces problèmes plus en détail, en rappelant l'état de l'art, l'importance de ces problèmes et les articles publiés et les retombées de cette recherche.

General Project Scheduling Problem

Le GPSP peut être décrit comme suit. Un ensemble d'activités peut être programmé par périodes, chaque activité a une durée associée qui détermine la durée pendant laquelle elle consomme des ressources. Les relations de précédence avec délais déterminent le temps qui doit s'écouler entre le début de paires d'activités. Dans les extensions multimodales de ce problème, les activités peuvent être réalisées de différentes manières (ou modes). Le changement de mode d'une activité affecte sa consommation de ressources et la valeur de sa fonction objective. Nous supposerons, tout au long de ce travail, que le fait de changer le mode d'une activité ne change pas sa durée. Dans les extensions de traitement par lots, les activités sont regroupées en clusters qui doivent être programmés pour commencer à la même période. Le problème général d'ordonnancement de la production que nous étudions est le problème d'ordonnancement de projets multimodal de traitement par lots à ressources limitées.

Formellement, cette catégorie de problèmes peut être décrite comme suit. Considérons un ensemble d'activités $\mathcal{A}$ et un ensemble discret de périodes de temps $\mathcal{T} = \{1, \ldots, T\}$. Soit $\mathcal{C}$ une partition de l'ensemble d'activités $\mathcal{A}$. Nous dirons que chaque élément (ou ensemble) de $\mathcal{C}$ est un groupe, ou un lot. Ces ensembles représenteront les activités qui doivent commencer simultanément. Nous supposerons que, sauf indication contraire, l'ordonnancement de chaque groupe est facultatif. Pour chaque grappe $c \in \mathcal{C}$, soit $P(c) \subseteq \mathcal{C}$ son ensemble de clusters prédécesseur. Autrement dit, si c_2 est ordonnancé, alors tous les clusters $c_1 \in P(c_2)$ doivent également être ordonnancés. Pour chaque $c_2 \in \mathcal{C}$ et chaque $c_1 \in P(c_2)$, soit $l_{c_1 c_2}$ le délai (lag) correspondant. Ce nombre représente le nombre de périodes de temps qui doivent s'écouler entre le début de c_1 et celui de c_2. Autrement dit, si c_2 commence au temps t ou avant, alors c_1 doit commencer au temps $t - l_{c_1 c_2}$ ou avant. Notez que si $l_{c_1 c_2} < 0$, alors, même si c_1 est un prédécesseur de c_2, il est admissible que c_1 commence après c_2, à condition qu'il commence au maximum $|l_{c_1 c_2}|$ périodes après. Soit $\mathcal{M}_a$ l'ensemble des modes dans lesquels l'activité $a \in \mathcal{A}$ peut être programmée. Chaque activité programmée doit se voir attribuer exactement un mode. Le mode d'une activité ne peut pas être modifié une fois qu'elle a commencé. Pour chaque activité $a \in\in A$, soit d_a sa durée. Nous supposons que les activités ne peuvent pas être interrompues (pas de préemption). Une fois qu'elles sont programmées, elles fonctionnent pendant des durées d_a ou jusqu'à ce que l'horizon temporel T soit atteint. Soit $\mathcal{R}$ l'ensemble des ressources du projet, et pour $r \in \mathcal{R}$ soit $q^r_{a,m}$ la quantité de ressource de type r consommée à chaque période par l'activité a lorsqu'elle est exécutée en mode m. Pour chaque $t \in \mathcal{T}$ et $r \in \mathcal{R}$ soit Q^t_r la quantité de ressources de type r disponible dans la période de temps t. Enfin, soit $v_{a,m,t}$ le profit obtenu en programmant l'activité a pour démarrer au temps t, et s'exécuter dans le mode m.

Pour modéliser ce problème, nous définissons des variables binaires $x_{c,t}$, indiquant si le cluster c commence au temps t ou non, et des variables binaires $y_{a,m,t}$ indiquant si l'activité a est programmée pour commencer au temps t avec le mode m. En utilisant ces variables, le problème que nous considérons est le suivant :

$$\max \sum_{a\in A}\sum_{m\in M_a}\sum_{t\in T} v_{a,m,t}y_{a,m,t}, \tag{1.2a}$$

$$\text{s.t.} \tag{1.2b}$$

$$\sum_{t\in\mathcal{T}} x_{c,t} \leq 1 \qquad \forall c\in\mathcal{C}, \tag{1.2c}$$

$$\sum_{m\in\mathcal{M}_a} y_{a,m,t} = x_{c,t} \qquad \forall t\in\mathcal{T}, \forall c\in\mathcal{C}, \forall a\in c, \tag{1.2d}$$

$$\sum_{a\in\mathcal{A}}\sum_{m\in\mathcal{M}_a}\sum_{s=\max\{1,t-d_a+1\}}^{t} q_{r,a,m}y_{a,m,s} \leq Q_{r,t} \qquad \forall r\in\mathcal{R}, \forall t\in\mathcal{T}, \tag{1.2e}$$

$$\sum_{s=1}^{t} x_{c_2,s} \leq \sum_{s=1}^{t-l_{c_1,c_2}} x_{c_1,s} \qquad \forall c_1\in\mathcal{C}, \forall c_2\in P(c_1), \forall t\in\mathcal{T}. \tag{1.2f}$$

Dans cette formulation, les contraintes (1.2c) imposent qu'un cluster soit extrait au maximum une fois. Les contraintes (1.2d) imposent que la période de temps pendant laquelle une activité est programmée soit la même que celle du cluster auquel elle appartient. Elles imposent également qu'un mode soit attribué à chaque activité programmée. Les contraintes (1.2e) imposent la limite de capacité des ressources. Dans cette contrainte, si une activité a est programmée à la période t, elle consomme des ressources sur les périodes $\{t,\ldots,t+d_a-1\}$. Enfin, les contraintes (1.2f) imposent les contraintes de décalage. Ici, si le cluster c_2 est programmé à la période t, alors le cluster $c_2\in P(c_1)$ doit être programmé à la période $t-l_{c_1,c_2}$ ou avant.

Notez que la classe de problèmes représentée par la formulation (1.2) est très grande et inclut certains problèmes très difficiles, notamment les problèmes d'atelier et les problèmes de RCPSP. Comme le rapporte [VCB16], il existe des cas de RCPSP ne comportant pas plus de 90 tâches qui, à ce jour, n'ont pas été résolus de manière optimale.

Dans cette thèse, nous développons des méthodologies de programmation en nombres entiers pour résoudre le modèle (1.2), en se concentrant sur la planification stratégique de mines à ciel ouvert. Dans le problème de la planification stratégique de mines à ciel ouvert (SOPMPP), l'objectif est de déterminer où, quand et comment exploiter un gisement de minerai à l'aide de méthodes d'excavation à ciel ouvert, afin de maximiser le gain de l'opération. Un gisement minier est discrétisé en blocs réguliers d'une grille tridimensionnelle pour l'exprimer comme un modèle de bloc économique. Le temps est discrétisé par périodes qui représentent généralement des années. Les méthodes d'exploitation

à ciel ouvert imposent des restrictions spatiales et géologiques qui exigent que pour exploiter un bloc donné, un certain ensemble en forme de cône au-dessus du bloc doit être exploité au préalable. Les restrictions opérationnelles peuvent exiger que l'extraction soit effectuée dans de grands groupes de blocs contigus. Un bloc exploité est envoyé vers l'une des différentes destinations possibles, qui peuvent comprendre des usines de traitement, des décharges, des stocks, etc. Des restrictions sont imposées en raison de la capacité limitée de la quantité de matière pouvant être extraite ou envoyée vers des destinations spécifiques. La composition des blocs détermine le profit obtenu en l'envoyant à chaque destination spécifique.

Le SOPMPP est un problème fondamental à résoudre dans l'industrie minière, puisqu'il est nécessaire d'évaluer l'opportunité d'exploiter un gisement minier, et il est résolu en pratique chaque année afin de réévaluer les plans stratégiques actuels avec les informations actualisées obtenues du modèle de bloc. Ce problème est étudié depuis les années 1960. Une formulation de programmation en nombres entiers, qui est un cas particulier de (1.2), a été proposée par T. B. Johnson [Joh68; Joh69], mais la taille du problème rend impossible sa résolution dans la pratique, puisque les instances de taille réelle peuvent facilement avoir des centaines de milliers à plusieurs millions de blocs, et bien plus de variables et de contraintes. Les sociétés minières ont traditionnellement utilisé des systèmes logiciels commerciaux basés sur des méthodes heuristiques pour obtenir des solutions à ce problème. Ces logiciels sont disponibles depuis les années 1980. Deux heuristiques notables utilisées pour faire face au problème sont l'algorithme de *Lane* et l'algorithme *Nested Pit Methodology*, qui ont été combinés dans une méthodologie par le système logiciel commercial *Whittle* qui est utilisé en pratique jusqu'à ce jour [GEO19]. Des approches de programmation en nombres entiers ont été proposées depuis les années 2000 [Ble+10; Bol+09], basées sur des techniques d'agrégation pour réduire la dimension du problème et des plans sécants pour renforcer la formulation. Cependant, la taille des formulations de programmation en nombres entiers rend difficile en pratique de résoudre même leur relaxation LP.

L'étude de la SOPMPP dans cette thèse est basée sur deux développements importants de la littérature. Premièrement, une percée dans la résolution de la relaxation LP du problème a été l'algorithme proposé par Bienstock et Zuckerberg, l'algorithme BZ [BZ09; BZ10]. Cet algorithme, basé sur une décomposition qui tire parti de la structure combinatoire de la formulation du GPSP, résout de manière itérative des programmes linéaires beaucoup plus petits et un problème combinatoire qui peut être résolu efficacement. Deuxièmement, une heuristique d'arrondi proposée par Chicoisne et al. [Chi+12] a produit des solutions avec de petits écarts d'optimalité pour une version simplifiée du problème en arrondissant une solution de la relaxation LP du problème.

La première partie de cette thèse, développée dans le chapitre 2, étudie en profondeur l'algorithme BZ. Nous montrons que cet algorithme peut être étendu à la planification des mines souterraines et à d'autres classes de problèmes d'ordonnancement. Nous montrons également que l'algorithme peut être exprimé comme un schéma de génération de colonnes, ce qui nous permet de le comparer

à la décomposition de Dantzig-Wolfe et de caractériser ses limites. Enfin, des techniques d'accélération sont présentées, ainsi qu'une étude expérimentale qui étudiue les performances de l'algorithme dans des applications de planification stratégique des mines et des problèmes de planification de projets à contraintes de ressources. Les résultats de ce chapitre ont été publiés dans l'article suivant :

A study of the Bienstock-Zuckerberg algorithm, Applications in Mining and Resource Constrained Project Scheduling. Munoz, G;. Espinoza, D.; Goycoolea, M.; Moreno, E.; Queyranne, M.; Rivera, O. Computational Optimization and Applications. Volume 69, Issue 2, pp 501 – 534. March, 2018.

Dans cet article, nous confirmons que l'algorithme BZ est un algorithme efficace pour résoudre de grandes classes de problèmes de planification d'exploitation minière. Nous généralisons la reformulation nécessaire pour que l'algorithme s'adapte à celle plus générale du GPSP. En outre, nous développons des techniques à utiliser lors de la mise en œuvre de l'algorithme BZ et qui améliorent considérablement les performances. Nous établissons la qualité théorique des limites de l'algorithme BZ, et développons des techniques de compression de chemin et des accélérations de calcul. Nous montrons aussi numériquement que l'algorithme BZ est nettement plus performant pour résoudre la relaxation LP que la décomposition de Dantzig-Wolfe ou d'autres algorithmes de programmation linéaire. Ma contribution personnelle spécifique à cet article a été de développer la reformulation plus générale, de caractériser la qualité des bornes obtenues, l'implémentation de la plupart des algorithmes et des accélérations mentionnés, et la réalisation de l'étude expérimentale présentée dans l'article, y compris le recueil et la manipulation de données réelles de mines, et l'implémentation de la plate-forme qui a permis d'exprimer le problème comme un problème d'optimisation adapté à l'algorithme BZ.

La deuxième partie de cette thèse, développée dans le chapitre 3, propose une méthodologie pour aborder le SOPMPP. Elle s'appuie sur une division classique du SOPMPP en deux sous-problèmes, le problème de conception des phases et le problème d'ordonnancement de la production, qui peuvent tous deux être formulés comme des cas particuliers du GPSP. Elle utilise l'algorithme décrit dans le chapitre 2, mais elle prolonge ce travail en introduisant une méthodologie complète traitant des contraintes d'intégralité. La méthodologie combine des heuristiques ad-hoc, y compris l'adaptation de l'heuristique proposée dans [Chi+12], et elle inclut des techniques de prétraitement bien connues et d'autres que nous proposons, ainsi que des inégalités valables et un nouveau schéma de branchement. En combinant toutes ces techniques, nous montrons que des cas réels de problèmes de planification stratégique des mines peuvent être résolus de manière quasi optimale ($< 1\%$) en quelques heures seulement. Ce chapitrecorresponda à l'article suivant :

Production scheduling for strategic open pit mine planning: A mixed integer programming approach. Rivera, O; Espinoza, D; Goycoolea, M; Moreno, E.; Munoz, G. Operations Research. Published Online. Aug 27, 2020.

L'importance de ce document est qu'il présente d'abord une méthodologie pour résoudre le SOPMPP au moyen de méthodes exactes de programmation des nombres entiers. Il montre une manière de modéliser le SOPMPP en décomposant le problème en deux étapes, les deux pouvant être formulées comme différents GPSP qui peuvent être résolus avec de faibles écarts d'optimalité avec la méthodologie présentée dans le document. Une contribution particulièrement importante de ce document est l'introduction des coupes *hourglass* (sablier). Les coupes hourglass sont nouvelles en ce sens qu'elles utilisent une partie de la structure de la formulation qui n'a pas été utilisée auparavant pour développer des inégalités valides. Ces coupes ont eu un impact important pour réduier l'écart d'optimalité. Ma contribution personnelle spécifique dans cet article est que j'ai développé la plupart des implémentations utilisées pour les expériences de calcul (à l'exception de l'algorithme de branchement). J'ai également conçu tous les nouvelles coupes hourglass, ainsi que les nouvelles heuristiques, et adapté et mis en œuvre la plupart des schémas de prétraitement, des heuristiques et des anciennes classes d'inégalités valides initialement proposés pour des variantes plus simples du problème, à la classe générale des problèmes examinés dans le document. J'ai également réalisé toutes les expérimentations.

La méthodologie proposée pour le problème stratégique de l'exploitation minière à ciel ouvert a été testée en pratique par les sociétés minières, ce qui a permis d'aboutir à des solutions satisfaisantes et à des plans de programmation de la production.

Bin Packing Problem with Time Lags

Le problème de bin packin avec contraintes de délais (Bin-Packing With Time Lags, BPPTL) est une généralisation du problème de bin packing. Dans ce problème, un ensemble d'articles doit être affecté au nombre minimum possible de bins de même capacité. Dans le BPPTL, les bins doivent en outre être programmés dans le temps, de manière à satisfaire à un ensemble de contraintes de délais entre les articles. Ces délais peuvent être positifs ou négatifs.

Le problème BPPTL peut être utilisé pour modéliser des tâches qui doivent être exécutées de manière répétitive, en utilisant des ressources limitées qui doivent être payées. Les délais positifs et négatifs permettent de fixer le temps minimum et maximum qui doit s'écouler entre les exécutions consécutives de chaque tâche. Une application de cette méthode est le traitement phytosanitaire dans un vignoble, dans lequel des zones du vignoble nécessitent un traitement périodique qui doit être effectué avec une fréquence idéale donnée. La fréquence des tâches est souple, bien que les traitements consécutifs ne doivent pas être effectués trop près ou trop loin dans le temps. Les tâches consomment des ressources qui doivent être louées par jour, et les mêmes ressources peuvent être utilisées pour effectuer des tâches dans différentes zones du vignoble si elles doivent être effectuées le même jour.

Le BPPTL est un nouveau problème qui étend plusieurs autres généralisations du problème de bin packing. Le problème de bin packing avec priorités (BPP-P)

[DDI12; Per16] correspond au cas spécifique du BPPTL dans lequel tous les délais sont égaux à un. Le problème d'équilibrage de charge (SALBP) [Per15] correspond au cas spécifique de BPPTL dans lequel tous les délais sont égaux à zéro. Le problème de bin packing avec contraintes de priorité généralisées (BPP-GP) [KDI17] généralise également le problème de bin packing et prend en compte les contraintes de priorité avec des décalages dans le temps. Cependant, le problème BPP-GP considère une fonction objectif différente de BPPTL et ne considère que des délais positifs. Le problème BPP-GP coïncide avec BPPTL lorsque les délais sont tous égaux à zéro ou un.

Formellement, dans le problème BPPTL, on dispose d'un graphe dirigé valué $G = (V, A, l)$, où V est un ensemble d'éléments qui doivent être affectés à des bins, A est un ensemble d'arcs définissant des délais, et $l : A \longrightarrow \mathbb{Z}$ représente les délais associés aux arcs. On considère également un poids $w_i \in \mathbb{Z}^+$ pour chaque élément $i \in V$, et une capacité de bin $W \in \mathbb{Z}^+$. Le problème consiste à partitionner les articles en bins, et à programmer chaque article $i \in V$ sur une période $p(i) \in \mathbb{Z}^+$, de telle sorte que : (a) les articles affectés à un bin soient ordonnancés dans la même période, (b) le poids cumulé des articles affectés à chaque bin ne dépasse pas la capacité prescrite W, et (c) pour chaque $(i, j) \in A$, on a $p(i) + l_{i,j} \leq p(j)$. En outre, il peut exister une contrainte sur le nombre maximum de bacs L qui peuvent être affectés à la même période. L'objectif du problème est de trouver une telle affectation qui utilise le nombre minimum possible de bacs.

Le BPPTL est présenté au chapitre 4, où nous étudions le problème sous l'angle de la programmation des nombres entiers. Nous proposons une formulation compacte indicée sur le temps, et proposons différentes variantes, en comparant leurs performances. Les formulations indicée sur le temps que nous proposons sont très similaires à celles utilisées pour la formulation du GPSP (1.2), dans la mesure où les mêmes variables et contraintes de priorité sont prises en compte. Nous proposons également une formulation étendue qui définit une relaxation du problème BPPTL. Nous développons un algorithme de branch-and-cut-and-price basé sur cette formulation, qui permet de résoudre de manière optimale deux cas spécifiques de la BPPTL : (a) le cas avec un nombre illimité de bins par période (BPPTL$^\infty$), et (b) le cas dans lequel les décalages temporels sont positifs ou nuls et dans lequel un seul bin est autorisé par période (BPPTL$^1_+$). Pour BPPTL$^\infty$ et BPPTL$^1_+$, nous caractérisons les solutions réalisables de la formulation étendue et développons des algorithmes de séparation pour les solutions entières et fractionnaires. Enfin, nous mettons au point une heuristique de strong diving et menons une étude numérique pour comparer les performances de l'algorithme de branch-and-cut-and-price et d'un logiciel commercial appliqué à la formulation compacte. Nous montrons que pour les cas inspirés par le problème des traitements phytosanitaires, l'approche branch-cut-and-price est plus performante que l'utilisation de CPLEX sur la formulation compacte. Nous montrons également que pour les cas de BPP-P, SALBP et BPP-GP, nous sommes capables d'améliorer les bornes inférieures et supérieures de la littérature. En combinant les limites connues et améliorées, le statut d'optimalité est atteint pour 70% des cas avec des articles de moins

de 100 qui étaient auparavant des problèmes ouverts dans la littérature. Ce chapitre reprend les élements décrits dans le document suivant, qui a été soumis pour publication :

Bin Packing with Time Lags. Rivera, O; Clautiaux, F; Sadykov, R. Manuscript submitted for publication.

Mes contributions personnelles spécifiques à cet article ont été la caractérisation mathématique des solutions réalisables, l'algorithme de séparation pour les solutions fractionnaires, la génération d'instances inspirées d'applications réelles, l'implémentation de la formulation compacte et de ses variantes, l'implémentation de l'algorithme de branch-and-cut-and-price en utilisant la bibliothèque BapCod, ainsique l'étude expérimentale. L'implémentation de l'algorithme comprend le modèle, les procédures de séparation entière et fractionnaire pour BPPTL^1_+ et BPPTL^∞, et la séparation pour l'heuristique de diving.

References

[Ble+10] Bley, A. et al. "A strengthened formulation and cutting planes for the open pit mine production scheduling problem". In: *Computers & Operations Research* vol. 37, no. 9 (2010), pp. 1641–1647.

[Bol+09] Boland, N. et al. "LP-based disaggregation approaches to solving the open pit mining production scheduling problem with block processing selectivity". In: *Computers & Operations Research* vol. 36 (4 2009), pp. 1064–1089.

[BZ09] Bienstock, D. and Zuckerberg, M. "A new LP algorithm for precedence constrained production scheduling". Unpublished. Columbia University, BHP Billiton. Available at Optimization Online. Aug. 2009.

[BZ10] Bienstock, D. and Zuckerberg, M. "Solving LP Relaxations of Large-Scale Precedence Constrained Problems". In: *Proceedings from the 14th conference on Integer Programming and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science 6080* (2010), pp. 1–14.

[Chi+12] Chicoisne, R. et al. "A new algorithm for the open-pit mine production scheduling problem". In: *Operations Research* vol. 60, no. 3 (2012), pp. 517–528.

[DDI12] Dell'Amico, M., Díaz, J. C. D., and Iori, M. "The Bin Packing Problem with Precedence Constraints". In: *Operations Research* vol. 60, no. 6 (2012), pp. 1491–1504.

[GEO19] GEOVIA Whittle. *Dassault Systèmes.* `https://www.3ds.com/products-servic` Last visited August 2019.

[Joh68] Johnson, T. B. “Optimum open pit mine production scheduling”. PhD thesis. Operations Research Department, Universiry of California, Berkeley, May 1968.

[Joh69] Johnson, T. B. “Optimum open-pit mine production scheduling”. In: *A decade of digital computing in the mining industry*. Ed. by Weiss, A. AIME, New York, 1969. Chap. 4.

[KDI17] Kramer, R., Dell’Amico, M., and Iori, M. “A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints”. In: *International Journal of Production Research* vol. 55, no. 21 (2017), pp. 6288–6304.

[Per15] Pereira, J. “Empirical evaluation of lower bounding methods for the simple assembly line balancing problem”. In: *International Journal of Production Research* vol. 53, no. 11 (2015), pp. 3327–3340.

[Per16] Pereira, J. “Procedures for the bin packing problem with precedence constraints”. In: *European Journal of Operational Research* vol. 250, no. 3 (2016), pp. 794–806.

[VCB16] Vanhoucke, M., Coelho, J., and Batselier, J. “An Overview of Project Data For Integrated Project Management and Control”. In: *The Journal of Modern Project Management* vol. 03, no. 03, January - April (2016), pp. 6–21.

Chapter 2

A study of the Bienstock-Zuckerberg algorithm: applications in mining and resource constrained project scheduling

Gonzalo Muñoz, Daniel Espinoza, Marcos Goycoolea, Eduardo Moreno, Maurice Queyranne, Orlando Rivera Letelier

Published in Comput. Optim. Appl. DOI: 10.1007/s10589-017-9946-1.

A study of the Bienstock-Zuckerberg algorithm*

Applications in Mining and Resource Constrained Project Scheduling

Gonzalo Muñoz[1], Daniel Espinoza[2], Marcos Goycoolea[3], Eduardo Moreno[4], Maurice Queyranne[5], and Orlando Rivera[6]

[1]*Industrial Engineering and Operations Research, Columbia University*
[2]*Gurobi Optimization*
[3]*School of Business, Universidad Adolfo Ibañez*
[4]*Faculty of Engineering, Universidad Adolfo Ibañez*
[5]*School of Business, University of British Columbia*
[6]*School of Business and Faculty of Engineering, Universidad Adolfo Ibañez*

April 11, 2017

We study a Lagrangian decomposition algorithm recently proposed by Dan Bienstock and Mark Zuckerberg for solving the LP relaxation of a class of open pit mine project scheduling problems. In this study we show that the Bienstock-Zuckerberg (BZ) algorithm can be used to solve LP relaxations corresponding to a much broader class of scheduling problems, including the well-known Resource Constrained Project Scheduling Problem (RCPSP), and multi-modal variants of the RCPSP that consider batch processing of jobs. We present a new, intuitive proof of correctness for the BZ algorithm that works by casting the BZ algorithm as a column generation algorithm. This analysis allows us to draw parallels with the well-known Dantzig-Wolfe decomposition (DW) algorithm. We discuss practical computational techniques for speeding up the performance of the BZ and DW algorithms on project scheduling problems. Finally, we present computational experiments independently testing the effectiveness of the BZ and DW algorithms on different sets of publicly available test instances. Our computational experiments confirm that the BZ algorithm significantly outperforms the DW algorithm for the problems considered. Our computational experiments also show that the proposed speed-up techniques can have a significant impact on the solve time. We provide some insights on what might be explaining this significant difference in performance.

Keywords: Column generation, Dantzig-Wolfe, Optimization, RCPSP

*The authors of this article would like to acknowledge support by grants Fondecyt 1151098 (MG and ORL), Fondecyt 1130681 (EM), Fondecyt 1150046 (DE), Conicyt PIA Anillo ACT 1407 (DE, MG, EM, and ORL), NSERC RGPIN 5837-08 (MQ) and Conicyt BCH 72130388 (GM).

1. Introduction

Resource constrained project scheduling problems (RCPSPs) seek to optimally schedule activities over time in such a way as to comply with precedence and resource usage constraints. These problems can be notoriously difficult. Despite great progress in optimization methodologies during the last fifty years, there are instances of RCPSP involving as few as sixty activities that cannot be solved with today's most effective algorithms [24]. In multi-modal extensions of RCPSP, jobs can be processed in different ways (or modes). Changing the mode of a job affects its resource utilization, and its processing costs. In some applications, changing the mode of a job changes the duration of its processing time. In batch-processing extensions of RCPSP jobs are grouped together in clusters that must be processed together. For simplicity of notation, we will henceforth refer to multi-modal batch resource constrained project scheduling problems, simply as General Production Scheduling Problems, or GPSPs.

To date, the most effective methods for solving GPSPs, especially modal and batch variants, are based on integer programming methods that use the so-called time index formulations [39, 3, 7, 9]. These formulations define a binary variable for each job-execution time combination. An important limitation of these formulations is that the linear programming (LP) relaxations are very difficult to solve. This is because they tend to be large and degenerate, even for small problem instances.

While classical decomposition techniques and column generation approaches can be used for addressing some of these issues (specially complications related to the large number of variables), they often suffer from slow convergence rate. In this context, an important contribution was made by Bienstock and Zuckerberg [5, 6], where the authors presented an alternative to tackle these limitations effectively. They developed a new algorithm that can considerably outperform classical methods in a broad class of problems, thus providing a novel set of tools with a high practical impact and a wide range of potential extensions.

In this paper we study the Bienstock-Zuckerberg (BZ) algorithm [5, 6] in depth, and find that it can be used to overcome the stated limitations on a wide range of scheduling problems. We provide additional evidence to that in [5, 6], advocating for the efficacy of the algorithm in practice, and we provide new insights on it that allow further extensions. Specifically, we study the BZ algorithm as an approach to batch, multi-modal production scheduling problems where jobs can have arbitrary durations, but where these durations are not affected by changes of mode. The application of the BZ algorithm to this class of problems requires us to extend the algorithmic template as it was originally proposed. We are specifically concerned about this class of problems because, in addition to generalizing the well-known RCPSP, it generalizes three very important problems that arise in the context of mine planning: Underground Production Scheduling Problems (UPSPs), Open Pit Phase Design Problems (OPPDPs), and Open Pit Production Scheduling Problems (OPPSPs).

We present a new proof of algorithm correctness by casting the BZ algorithm as a column generation method, discuss algorithmic speed-ups, computationally test it on a variety of problem instances, and compare the methodology to the more traditional Dantzig-Wolfe decomposition (DW) method. As part of our analysis we prove that the BZ algorithm is closely related to a decomposition scheme that produces a bound somewhere in between that of the DW method, and that of the LP relaxation. This study allows us to conclude that the BZ algorithm is an effective

way of solving the LP relaxation of large time index formulations, significantly outperforming the DW method on all considered problem classes, and provides insights on the effectiveness of the methodology.

This article is organized as follows. In Section 2 we present a literature review of related scheduling work in mine planning applications and mathematical programming methodologies. In Section 3 we present an integer programming model that generalizes the classes of problems we are interested in studying. In Section 4 we present a reformulation of this model that fits the algorithmic framework we will be analyzing. In Section 5 we present a general column generation framework that can be used to solve the problem. We use this column generation approach to motivate the BZ algorithm, and facilitate a comparison to the DW method. We also introduce important speed ups for the BZ algorithm. Computational results analyzing the performance of BZ and comparing this performance to that of DW are presented in Section 6.

2. Background

Scheduling applications in Mine Planning

In this article we are mainly interested in addressing scheduling problems that are of relevance to planning applications in the mining industry.

The class of mining problems we are interested in include open pit and underground mine planning problems. In these problems, deposits are discretized into three-dimensional arrays known as block models. The problem to solve consists of deciding which blocks should be extracted, when they should be extracted, and what to do with the blocks once they are extracted. In this context, jobs correspond to extraction activities, modes correspond to different processing options and batches correspond to groups of blocks that must be extracted concurrently in order to comply with equipment extraction requirements. Resource constraints would be used to impose extracting, milling and refining capacity constraints. In an open-pit mine, precedence constraints would be used to impose that a mine must be extracted from the surface on downwards. In an under-ground mine, specifically in a stopping operation, precedence constraints would be used to impose that selected blocks branch out from a network of tunnels descending from the surface.

In all of these mining problems the objective is to maximize net-present-value of the extracted minerals. This is different from traditional scheduling problems, in which the objective is typically to minimize completion time metrics such as makespan, maximum tardiness, or total throughput time. Another difference between mine planning problems and traditional scheduling problem is that in mining it is not a strict requirement to execute all jobs. In all other ways, the Underground Production Scheduling Problems (UPSPs) is identical to the RCPSP. The Open Pit Phase Design Problems (OPPDPs) is a multi-modal RCPSP in which all jobs take a single-time period to complete, regardless of the selected mode. What the mode affects is the objective function value and the resource consumption of each activity. The Open Pit Production Scheduling Problems (OPPSPs) is just like the OPPDP, with the addition that there are batch constraints that force groups of blocks to be extracted simultaneously. These batch constraints are used to enforce equipment operability constraints, with each batch corresponding to contiguous set of

blocks typically called a *bench-phase* or *increment* in the mining industry.

For an introduction to optimization in underground mining see Alford et al. [1]. For related work in underground mining see Martinez and Newman [27], Newman and Kuchta [32] and O'Sullivan et al. [35, 36]. The user manuals of Deswik Scheduler [15] and MineMax iGantt [28] illustrate how UPSP is solved in practical mining applications. For an introduction to Open Pit Mining see Hustrulid and Kuchta [22], and the user manuals of Dassault Whittle [13], MineMax Scheduler [30], and MineMax Planner [29]. For a discussion on OPPSP see Goycoolea et al. [18]. For a general survey on OR applications in mine planning see Newman et al. [33].

Mathematical programming methodologies

To our knowledge, the first mathematical programming formulations of GPSPs dates back to Johnson [23] and Pritsker et al. [38], in the late 1960s. Each of these articles spawned its own track of academic articles on production scheduling problems. The first track, following the work of Johnson, mostly focused on strategic open pit mine planning problems. The second track, following the work of Pritsker et al., took a more general approach. Surprisingly, though many of the results found in these two tracks coincide, there are few, if any, citations connecting the two literatures.

The academic literature on exact optimization methods for GPSPs is immensely rich. Well-known surveys from the scheduling track include those of Graham et al. [19], Brucker et al. [8] and Hartmann and Briskorn [20]. In a recent book by Artigues et al. [2] a very thorough survey of GPSP is presented, including a computational study that compares existing methodologies on benchmark instances. Surveys from the mining track include those of Newman et al. [33], Espinoza et al. [16] and Osanloo et al. [34].

A brief summary of advances on solving the LP relaxation of time-index formulations is as follows. Since as early as the 1960s, most methods have been based on some form of decomposition that reduces the problem to a sequence of maximum closure or minimum cut problems. Johnson [23], in the context of mining (single mode OPPDPs), was the first to attempt such an approach with a Dantzig-Wolfe decomposition. Shortly after, and independently, Fisher [17], in the context of scheduling, proposed a Lagrangian Relaxation decomposition approach. Since then, a number of decomposition algorithms, primarily Lagrangian Relaxation decomposition algorithms, have been proposed for the problem in both literatures. Important examples include Dagdelen and Johnson [11] and Lambert and Newman [26] in mining (single mode OPPDPs), and Christofides et al. [10] and Mohring et al. [31] in scheduling.

Some recent approaches are as follows. Chicoisne et al. [9] consider single-mode OPPDPs with a single renewable resource, and propose a decomposition algorithm that solves the continuous relaxation in polynomial time. Boland et al. [7] consider a multi-modal variant of the same problem with two renewable resources, and propose a decomposition algorithm for the continuous relaxation that, while not provably polynomial, is very effective in practice. The BZ algorithm [5], developed shortly after, can be considered a generalization of this last algorithm that extends to multi-modal OPPMPs with an arbitrary number of renewable or non-renewable resources. This algorithm proved to be very efficient, even for extremely large instance sizes; it reduced the solving times drastically and it was able to tackle instances that could not be solved before. Berthold et al. [3] developed a branch-and-bound algorithm that combines mathematical

programming and constraint programming methods to close a large number of open benchmark instances of RCPSP. Zhu et al. [39] developed a mathematical programming based algorithm for solving multi-modal variants of RCPSP without batch constraints, but where mode changes affect duration times. In his work he closes a large percentage of open benchmark instances.

3. Integer programming formulation

We now describe an integer programming formulation for a class of production scheduling problems that generalizes the RCPSP, UPSP, OPPDP and OPPSP. As mentioned before, we simply refer to this class of problems as the General Production Scheduling Problem (GPSP).
Sets:

- $\mathscr{A}$: activities that must be scheduled in the problem.
- $\mathscr{C}$: clusters of activities that define a partition of $\mathscr{A}$.
- $prec(c)$: clusters that must be initiated no later than cluster $c \in \mathscr{C}$.
- $\mathscr{R} = \{1,\ldots,R\}$: resources that are consumed when carrying out activities.
- $\mathscr{T} = \{1,\ldots,T\}$: time periods in which it is possible to initiate activities.
- $\mathscr{M}_a = \{1,\ldots,M_a\}$: possible modes for activity $a \in \mathscr{A}$.

Parameters:

- t_a^- : release date for activity $a \in \mathscr{A}$.
- t_a^+ : due date for activity $a \in \mathscr{A}$.
- $d_{a,m}$: duration (number of time periods) of activity $a \in \mathscr{A}$ when executed in mode $m \in \mathscr{M}_a$.
- $p_{a,m,t}$: profit obtained if activity $a \in \mathscr{A}$ is initiated in period $t \in \mathscr{T}$ using mode $m \in \mathscr{M}_a$.
- $l(c_1,c_2)$: lag (number of time periods) that must elapse before activities in cluster $c_2 \in \mathscr{C}$ can be initiated, after activities in cluster $c_1 \in prec(c_2)$ have been initiated.
- $Q_{r,t}$: amount of resource $r \in \mathscr{R}$ available in time period $t \in \mathscr{T}$.
- $q_{r,a,m}$: amount of resource $r \in \mathscr{R}$ consumed by activity $a \in \mathscr{A}$ in each time period it is being executed, when executed in mode $m \in \mathscr{M}_a$.

Variables:

$$x_{c,t} = \begin{cases} 1 & \text{if the activities in cluster } c \text{ all start in time period } t \\ 0 & \text{otherwise} \end{cases}$$

$$y_{a,m,t} = \begin{cases} 1 & \text{if activity } a \text{ starts in time period } t \text{ using mode } m \\ 0 & \text{otherwise} \end{cases}$$

Objective function:

$$\text{maximize} \sum_{a\in\mathscr{A}} \sum_{m\in\mathscr{M}_a} \sum_{t\in\mathscr{T}} p_{a,m,t} y_{a,m,t} \tag{1}$$

Note that we express the objective function only in terms of the y variables. There is no loss of generality in this due to constraints (3), below.

Constraints:

Clusters can only be initiated once over the time horizon [$\forall c \in \mathscr{C}$]:

$$\sum_{t\in\mathscr{T}} x_{c,t} \leq 1. \tag{2}$$

Activities in a cluster must start simultaneously, and must be carried out using a unique mode [$\forall c \in \mathscr{C}$, $\forall a \in c$, $\forall t \in \mathscr{T}$]:

$$x_{c,t} = \sum_{m\in\mathscr{M}_a} y_{a,m,t}. \tag{3}$$

In order to initiate the activities in a cluster, all activities in preceding clusters must be initiated early enough so as to satisfy the lag requirement [$\forall c_2 \in \mathscr{C}$, $\forall c_1 \in prec(c_2)$, $\forall t \in \mathscr{T}$]:

$$\sum_{s\leq t} x_{c_2,s} \leq \sum_{s\leq t-l(c_1,c_2)} x_{c_1,s}. \tag{4}$$

The amount of resources consumed cannot exceed the amount of resources available each period [$\forall r \in \mathscr{R}$, $t \in \mathscr{T}$]:

$$\sum_{a\in\mathscr{A}} \sum_{m\in\mathscr{M}_a} q_{r,a,m} \sum_{s=\max\{1,t-d_{a,m}+1\}}^{t} y_{a,m,s} \leq Q_{r,t}. \tag{5}$$

Activities can only be scheduled after their release dates [$\forall a \in \mathscr{A}$]:

$$\sum_{m\in\mathscr{M}_a} \sum_{t=1}^{t_a^- -1} y_{a,m,t} = 0. \tag{6}$$

Activities must be terminated no later than due dates and exactly one mode must be chosen for each executed activity [$\forall a \in \mathscr{A} : t_a^+ < \infty$]:

$$\sum_{m\in\mathscr{M}_a} \sum_{t=1}^{t_a^+ - d_{a,m}+1} y_{a,m,t} = 1. \tag{7}$$

Whenever $t_a^+ = \infty$, there is no due date for activity a and we do not include constraint (7). Note, however, that due to (2) and (3) we always have:

$$\sum_{m\in\mathscr{M}_a} \sum_{t\in\mathscr{T}} y_{a,m,t} \leq 1.$$

Thus, even when $t_a^+ = \infty$, there is an implicit constraint enforcing the choice of one mode and one time period at most.

It should be noted that the GPSP described by Formulation (1)-(7) generalizes the scheduling formulations found in Bienstock and Zuckerberg [6] in that it allows activities to have durations that span multiple time periods. This allows us to consider instances of RCPSP and UPSP which fell outside the scope of the Bienstock and Zuckerberg studies [5, 6].

Though this formulation is intuitive, and also the most commonly used formulation in the academic literature, it must be reformulated so as to fit the algorithmic schemes that will be presented.

4. Reformulation

In this section we present a reformulation of the GPSP formulation described in Section 3. As we will see in Section 5, this reformulation is key for the decomposition algorithms that we will discuss in this article.

For each $c \in \mathscr{C}, a \in \mathscr{A}, m \in \mathscr{M}_a$ and $t \in \mathscr{T}$ define,

$$w_{c,t} = \sum_{s=1}^{t} x_{c,s} \qquad \text{and} \qquad z_{a,m,t} = \sum_{i \in \mathscr{M}_a} \sum_{s=1}^{t-1} y_{a,i,s} + \sum_{i=1}^{m} y_{a,i,t}.$$

In this way, $w_{c,t}$ is a binary variable that takes value one if and only if cluster c is initiated "by" time t (i.e., no later than t). Likewise, $z_{a,m,t}$ is a binary variable that takes value one if and only if activity a is initiated by time $t-1$, or, if it is initiated in time period t, and its mode i is such that $i \leq m$.

To see that it is possible to formulate the production scheduling problem given by (1)-(7), using the (z,w) variables, note that we can map between the two variable spaces by means of the following linear sets of equations:

$$y_{a,m,t} = z_{a,m,t} - z_{a,m-1,t} \qquad \forall a \in \mathscr{A},\ m = 2,\ldots,M_a,\ t \in \mathscr{T} \tag{8a}$$

$$y_{a,1,t} = z_{a,1,t} - z_{a,M_a,t-1} \qquad \forall a \in \mathscr{A},\ t = 2,\ldots,T \tag{8b}$$

$$y_{a,1,1} = z_{a,1,1} \qquad \forall a \in \mathscr{A} \tag{8c}$$

$$x_{c,t} = w_{c,t} - w_{c,t-1} \qquad \forall c \in \mathscr{C},\ \forall t = 2,\ldots,T \tag{8d}$$

$$x_{c,1} = w_{c,1} \qquad \forall c \in \mathscr{C} \tag{8e}$$

Using this mapping we can substitute out the variables in (1)-(7), to trivially obtain the following equivalent formulation:

Objective function:

$$\max \sum_{a \in \mathscr{A}} \sum_{m \in \mathscr{M}_a} \sum_{t \in \mathscr{T}} \tilde{p}_{a,m,t} z_{a,m,t} \tag{9}$$

where,

$$\tilde{p}_{a,m,t} = \begin{cases} p_{a,m,t} - p_{a,m+1,t} & \text{if } m < M_a \\ p_{a,m,t} - p_{a,1,t+1} & \text{if } m = M_a,\ t < T \\ p_{a,m,t} & \text{if } m = M_a,\ t = T \end{cases}$$

Constraints:

For $a \in \mathscr{A}$, $m \in \{1,\ldots,M_a-1\}$, $t \in \mathscr{T}$,:

$$z_{a,m,t} \leq z_{a,m+1,t}. \tag{10}$$

For $a \in \mathscr{A}$, $m = M_a$, $t \in \{1,\ldots,T-1\}$,:

$$z_{a,m,t} \leq z_{a,1,t+1}. \tag{11}$$

For $c \in \mathscr{C}$, $a \in c$, $t \in \mathscr{T}$,

$$w_{c,t} = z_{a,M_a,t}. \tag{12}$$

For $c_2 \in \mathscr{C}$, $c_1 \in prec(c_2)$, $t \in \{1+l(c_1,c_2),\ldots,T\}$,

$$w_{c_2,t} \leq w_{c_1,t-l(c_1,c_2)}. \tag{13}$$

For $r \in R$, $t \in \mathscr{T}$,

$$\sum_{a\in\mathscr{A}} \sum_{m\in\mathscr{M}_a} \sum_{s\in\mathscr{T}} \tilde{q}^{r,t}_{a,m,s} z_{a,m,s} \leq Q_{r,t}. \tag{14}$$

where

$$\tilde{q}^{r,t}_{a,m,s} = \begin{cases} q_{r,a,m}\mathbf{1}_{[t-d_{a,m}+1,t]}(s) - q_{r,a,m+1}\mathbf{1}_{[t-d_{a,m+1}+1,t]}(s) & \text{if } m < M_a \\ q_{r,a,M_a}\mathbf{1}_{[t-d_{a,M_a}+1,t]}(s) - q_{r,a,1}\mathbf{1}_{[t-d_{a,1},t-1]}(s) & \text{if } m = M_a \end{cases}$$

and,

$$\mathbf{1}_{[x,y]}(s) = \begin{cases} 1 & \text{if } x \leq s \leq y \\ 0 & \text{otherwise.} \end{cases}$$

(The derivation of this last constraint from (5) can be found in Appendix A).
For $a \in \mathscr{A}$, $m \in \mathscr{M}_a$, $t < t_a^-$:

$$z_{a,m,t} = 0. \tag{15}$$

For $a \in \mathscr{A}$, $m \in \mathscr{M}_a$, $t \geq t_a^+$:

$$z_{a,m,t} = 1. \tag{16}$$

After the reformulation, if we substitute out the w variables by using linear equalities (12), we obtain what Bienstock and Zuckerberg [6] call a General Precedence Constrained Problem (GPCP) :

$$\begin{aligned} Z^* = \quad & \max \quad && c'z && (17) \\ & \text{s.t.} \quad && z_i \leq z_j \qquad \forall (i,j) \in I, && (18) \\ & && Hz \leq h, && (19) \\ & && z \in \{0,1\}^n && (20) \end{aligned}$$

In this problem constraints (18) correspond to constraints (10), (11), and (13), and constraints (19) correspond to constraints (14), (15) and (16).

It is interesting that despite the fact that the General Production Scheduling Problem (GPSP) formulated in Section 3 is more general than the scheduling problem formulations presented by Bienstock and Zuckerberg [5], both problems can be reformulated as an instance of GPCP.

5. Methodology.

In this section we describe a generalized version of the Bienstock-Zuckerberg (BZ) algorithm, originally introduced in [5, 6]. More specifically, we describe a decomposition algorithm well suited for solving the LP relaxation of large mixed integer programming problems having form,

$$\begin{aligned} Z^{IP} = \max \quad & c'z + d'u \\ \text{s.t.} \quad & z_i \leq z_j, \qquad \forall (i,j) \in I, \\ & Hz + Gu \leq h, \\ & z \in \{0,1\}^n. \end{aligned} \tag{21}$$

Like Bienstock and Zuckerberg [6] before us, we will call this problem the General Precedence Constrained Problem (GPCP). It should be noted, however, that in our definition of GPCP we consider the presence of the extra u variables. These variables can be used for other modeling purposes. For example, they can be used to model variable capacities, or, in the context of mining problems, they can be used to model the use of stockpiles or other requirements.

Our presentation of the BZ algorithm is different than the original presentation in two respects: first, we cast it as a column generation method, and second, as stated before, we consider the presence of extra variables (the u variables in (21)). These differences require a new proof of algorithm correctness that we present below. The advantage of presenting the algorithm this way is that it is easier to compare to existing decomposition algorithms, and thus, in our opinion, becomes easier to understand and extend it. It also opens up the possibility of developing new classes of algorithms that might be useful in other contexts.

Before we present the BZ algorithm, we present a generic column generation (GCG) algorithm that can be used as a framework for understanding the BZ algorithm. This column generation scheme can be used to solve a relaxation of mixed integer problems having form:

$$\begin{aligned} Z^{IP} = \max \quad & c'z + d'u \\ \text{s.t.} \quad & Az \leq b, \\ & Hz + Gu \leq h, \\ & z \in \{0,1\}^n. \end{aligned} \tag{22}$$

Much like DW algorithm, this relaxation can yield bounds that are tighter than those obtained by solving the LP relaxation of the same problem. Throughout this section we will assume, without loss of generality, that $Az \leq b$ includes $0 \leq z \leq 1$. Other than this, we make no additional assumption on problem structure.

5.1. A general column generation algorithm

In this section we introduce a General Column Generation (GCG) algorithm that will later motivate the BZ algorithm. This column generation algorithm is presented as a decomposition scheme for computing upper bounds of (22) that are no weaker than those provided by the LP relaxation.

Given a set $S \subseteq R^n$, let $\texttt{lin.hull}(S)$ denote the linear space spanned by S. That is, let $\texttt{lin.hull}(S)$ represent the smallest linear space containing S. Let $P = \{z \in \{0,1\}^n : Az \leq b\}$. Define problem,

$$
\begin{array}{rll}
Z^{LIN} = \max & c'z + d'u & \\
\text{s.t.} & Az \leq b, & \\
& Hz + Gu \leq h, & \\
& z \in \texttt{lin.hull}(P), &
\end{array}
\tag{23}
$$

The GCG algorithm that we present computes a value Z^{UB} such that $Z^{IP} \leq Z^{UB} \leq Z^{LIN}$. Observe that the optimal value Z^{LIN} of problem (23) is such that $Z^{IP} \leq Z^{LIN} \leq Z^{LP}$, where Z^{LP} is the value of the linear relaxation of problem (22).

The key to solving this problem is the observation that

$$
\texttt{lin.hull}(P) = \{z : z = \sum_{i=1}^{d} \lambda_i v^i, \text{ for some } \lambda \in \mathbb{R}^n\}, \tag{24}
$$

where $\{v^1, \ldots, v^d\}$ is a basis for $\texttt{lin.hull}(P)$. If V is a matrix whose columns $\{v^1, \ldots, v^d\}$ define a basis of $\texttt{lin.hull}(P)$, it follows that problem (23) is equivalent to solving,

$$
\begin{array}{rlll}
Z(V) = & \max & c'V\lambda + d'u & \\
& \text{s.t.} & AV\lambda \leq b & (\alpha) \\
& & HV\lambda + Gu \leq h & (\pi).
\end{array}
\tag{25}
$$

In order for the optimal solution of (25) to be optimal for (23), it is not necessary for the columns of V to to define a basis of $\texttt{lin.hull}(P)$. It suffices for the columns of V to span a subset of $\texttt{lin.hull}(P)$ containing an optimal solution of (23). This weaker condition is what the column generation seeks to achieve. That is, starting with a matrix V with columns spanning at least one feasible solution of (23), the algorithm iteratively computes new columns until it computes the optimal value Z^{LIN}. As we will see, in some cases it is possible for the algorithm to terminate before having computed Z^{LIN}. In these cases the algorithm will terminate having computed a value Z^{UB} such that $Z^{UB} \leq Z^{LIN}$.

Henceforth, let α, π denote the dual variables corresponding to the constraints of problem (25). To simplify notation, we will henceforth assume that α and π are always row vectors. Note that $\alpha, \pi \geq 0$.

To solve problem (23) with column generation we begin each iteration with a set of points $\{v^1, \ldots, v^k\}$ such that $\texttt{lin.hull}(\{v^1, \ldots, v^k\}) \subseteq \texttt{lin.hull}(P)$ contains at least one feasible solution of problem (23). At each iteration we add a linearly independent point v^{k+1} to this set, until we can prove that we have generated enough points so as to generate the optimal solution of (23).

The first step of each iteration consists in solving problem (25), with a matrix V^k having columns $\{v^1, \ldots, v^k\}$. Let $Z_k^L = Z(V^k)$. Let (λ^k, u^k) and (α^k, π^k) be the corresponding optimal primal and dual solutions. Note that $Z_k^L = \alpha^k b + \pi^k h$.

Define $z^k = V^k \lambda^k$. It is clear that (z^k, u^k) is feasible for (23); hence, $Z_k^L \leq Z^{LIN}$. However, is (z^k, u^k) optimal for (23)? To answer this question, observe that (α, π) is dual-feasible for problem (25) if and only if $\pi G = d'$ and,

$$
\bar{c}(v, \alpha, \pi) \equiv c'v - \alpha Av - \pi Hv = 0, \qquad \forall v \in V.
$$

Let V^P represent a matrix whose columns $\{v^1,\ldots,v^{|P|}\}$ coincide with set P. Since V^P is clearly a generator of $\texttt{lin.hull}(P)$, we know that the optimal solution of (25) (for V^P) corresponds to an optimal solution of (23). Thus, if $\bar{c}(v,\alpha^k,\pi^k)=0$ for all $v\in V^P$ (or equivalently, $v\in P$), we conclude that (z^k,u^k) is optimal for (23), since we already know that $\pi^k G=d'$. On the other hand, if (z^k,u^k) is not optional for (23), we conclude that there must exist $v\in P$ such that $\bar{c}(v,\alpha^k,\pi^k)\neq 0$.

This suggests how the column generation scheme for computing an optimal solution of (23) should work. Solve (25) with V^k to obtain primal and dual solutions (z^k,u^k) and (α^k,π^k). Keeping with traditional column generation schemes, we refer to problem (25) as the *restricted master problem*. If $\bar{c}(v,\alpha^k,\pi^k)=0$ for all $v\in P$, then (z^k,u^k) is an optimal solution of (23). If this condition does not hold, let $v^{k+1}\in P$ be such that $\bar{c}(v^{k+1},\alpha^k,\pi^k)\neq 0$. Note that v^{k+1} must be linearly independent of the columns in V^k. In fact, every column v of V^k must satisfy $\bar{c}(v,\alpha^k,\pi^k)=0$, hence so must any vector that can be generated with this set. We refer to the problem of computing a vector v^{k+1} with $\bar{c}(v^{k+1},\alpha^k,\pi^k)\neq 0$ as the *pricing problem*. Obtain a new matrix V^{k+1} by adding column v^{k+1} to V^k, and repeat the process. Given that the rank of V^k increases strictly with each iteration, the algorithm must finitely terminate.

The pricing problem can be tackled by solving:

$$\begin{array}{lll} L(\pi)= & \max & c'v-\pi(Hv-h) \\ & \text{s.t.} & Av\leq b, \\ & & v\in\{0,1\}^n \end{array} \tag{26}$$

For this, at each iteration compute $L(\pi^k)$, and let $\hat{v}$ be the corresponding optimal solution. Observe that $L(\pi^k)\geq Z^{IP}$ for all $k\geq 1$. In fact, since the u variables are free in problem (25), and since (α^k,π^k) is dual-feasible in problem (25), we have that $d'-\pi^k G=0$. This implies that problem (26), for π^k, is equivalent to

$$\begin{array}{lll} \max & c'v+d'u-\pi^k(Hv+Gu-h) \\ \text{s.t.} & Av\leq b, \\ & v\in\{0,1\}^n. \end{array} \tag{27}$$

Given that $\pi^k\geq 0$, problem (27) is a relaxation of problem (22), obtained by penalizing constraints $Hz+Gu\leq h$. Hence $L(\pi^k)\geq Z^{IP}\ \forall k\ \geq 1$.

Define $Z_k^U=\min\{L(\pi^i):i=1,\ldots,k\}$ and note that $Z_k^U\geq Z^{IP}$, by the argument above. If $Z_k^U\leq Z_k^L$, we can define $Z^{UB}=Z_k^U$, and terminate the algorithm, as we will have that $Z^{IP}\leq Z^{UB}\leq Z^{LIN}$. In fact, since $Z_k^L\leq Z^{LIN}$, we have $Z^{IP}\leq Z^{UB}=Z_k^U\leq Z_k^L\leq Z^{LIN}$.

On the other hand, if $Z_k^U>Z_k^L$, then the optimal solution $\hat{v}$ of (26) is such that $\bar{c}(\hat{v},\alpha^k,\pi^k)\neq 0$, so we obtain an entering column by defining $v^{k+1}=\hat{v}$. To see this, note that

$$\begin{aligned} Z_k^U-Z_k^L>0 &\Leftrightarrow c'\hat{v}-\pi^k(H\hat{v}-h)-\alpha^k b-\pi^k h>0 \\ &\Leftrightarrow c'\hat{v}-\pi^k H\hat{v}-\alpha^k b>0 \\ &\Rightarrow c'\hat{v}-\pi^k H\hat{v}-\alpha^k A\hat{v}>0 \quad (\text{since } \alpha^k b\geq\alpha^k A\hat{v}) \\ &\Rightarrow \bar{c}(\hat{v},\alpha^k,\pi^k)>0. \end{aligned}$$

Observe that, to be precise, we do not need to know a starting feasible solution in order to solve problem (23). In fact, if we do not have a feasible solution, we can add a variable to the right-hand-side of each constraint $Hz+Gu \leq h$, and heavily penalize it to proceed as in a Phase-I simplex algorithm (see [4]). The first iteration only needs a solution v^1 such that $Av^1 \leq b$.

5.2. Comparison to the Dantzig-Wolfe decomposition algorithm.

When solving problems with form (22) a common technique is to use the Dantzig-Wolfe decomposition [12] to compute relaxation values. The Dantzig-Wolfe decomposition (DW) algorithm is very similar to the General Column Generation (GCG) algorithm presented in the previous section. In fact, the DW algorithm computes the optimal value of problem,

$$\begin{array}{rll} Z^{DW} = \max & c'z + d'u & \\ \text{s.t.} & Az \leq b, & \\ & Hz + Gu \leq h, & \\ & z \in \texttt{conv.hull}(P). & \end{array} \tag{28}$$

Given that $\texttt{conv.hull}(P) \subseteq \{z : Az \leq b\}$, this problem is typically written as,

$$\begin{array}{rll} Z^{DW} = \max & c'z + d'u & \\ \text{s.t.} & z \in \texttt{conv.hull}(P), & \\ & Hz + Gu \leq h, & \end{array} \tag{29}$$

or equivalently, as

$$\begin{array}{rll} Z^{DW} = \max & c'V\lambda + d'u & \\ \text{s.t.} & \lambda \cdot 1 = 1, & \\ & HV\lambda + Gu \leq h, & \\ & \lambda \geq 0. & \end{array} \tag{30}$$

In this formulation, the columns $\{v^1, \ldots, v^k\}$ of V correspond to the extreme points of $\texttt{conv.hull}(P)$.

Given that $\texttt{conv.hull}(P) \subseteq \texttt{lin.hull}(P)$, it follows that $Z^{IP} \leq Z^{DW} \leq Z^{LIN} \leq Z^{LP}$. When $\texttt{conv.hull}(P) = \{z : Az \leq b\}$ it is easy to see that $Z^{DW} = Z^{LIN} = Z^{LP}$. However, the following example shows that all of these inequalities can also be strict:

$$\begin{array}{rll} Z^{IP} = \max & -x_1 + 2x_2 + x_3 & \\ \text{s.t.} & -x_1 + x_2 \leq 0.5, & \\ & x_1 + x_2 \leq 1.5, & \\ & x_3 \leq 0.5, & \\ & x \in \{0,1\}^3. & \end{array} \tag{31}$$

By decomposing such that $Hz+Gu \leq h$ corresponds to $-x_1+x_2 \leq 0.5$ we get,

$$\{z : Az \leq b\} = \{z \in [0,1]^3 : x_1 + x_2 \leq 1.5,\ x_3 \leq 0.5\},$$

and,

$$P = \{z \in \{0,1\}^n : Az \leq b\} = \{(0,0,0),(0,1,0),(1,0,0)\},$$

and so,

$$\begin{aligned}\texttt{lin.hull}(P) &= \{(x_1,x_2,x_3) : x_3 = 0\},\\ \texttt{conv.hull}(P) &= \{(x_1,x_2,x_3) : 0 \le x_1,\ 0 \le x_2,\ x_1 + x_2 \le 1,\ x_3 = 0\}.\end{aligned}$$

From this it can be verified that $z^{IP} = 0.0$, $z^{DW} = 1.25$, $z^{LIN} = 1.5$, and $z^{LP} = 2.0$.

The DW algorithm is formally presented in Algorithm 1. The proof of correctness is strictly analogous to the proof presented in Section 5.1 for the generic column generation algorithm.

Observe that the DW pricing problem (see (35)) is exactly the same as the GCG pricing problem. However, since optimizing over $\{v : Av \le b,\ v \in \{0,1\}\}$ is exactly the same as optimizing over $\texttt{conv.hull}(P)$, we will have at every iteration $j \ge 1$ that $L(\pi^j) \ge Z^{DW}$. Thus, the bounds will never cross as they might in GCG, and there will be no early termination condition.

The main difference between the algorithms is in the master problem, where both solve very different linear models. One would expect that solving the master problem for the DW decomposition method would be much faster. In fact, let r_1 and r_2 represent the number of rows in the A and H matrices, respectively. The DW master problem has $r_2 + 1$ rows, compared to the $r_1 + r_2$ rows in the GCG master problem. However, this is only the case because of the way in which problem (22) is presented. If the first system of constraints, $Az \le b$, instead has form $Az = b$, the algorithm can be modified to preserve this property. This equality form version of GCG, which we call GCG-EQ, is presented in Appendix B.

The discussion above suggests that for the GCG algorithm to outperform the DW algorithm, it would have to do less iterations. It is natural to expect that this would happen. After all, given a common set of columns, the feasible region of the GCG master problem is considerably larger than the corresponding feasible region of the DW master problem. That is, the dual solutions obtained by solving the GCG master problem should be "better" than those produced by the DW master problem, somehow driving the pricing problem to find the right columns quickly throughout the iterative process. This improved performance, of course, would only compensate on problems in which the DW algorithm has a tough time converging, and could come at the cost of a worse upper bound to the underlying integer programming problem. In fact, this slow convergence rate is exactly what happens in General Production Scheduling Problems (GPSPs), and is what motivates the BZ algorithm in the first place.

As column generation algorithms, the size of the master problem will be increasing by one in each iteration, both for the GCG and DW. At some point the number of columns could grow so large that solving the master problem could become unmanageable. An interesting feature of the DW algorithm is that this can be practically managed by removing columns that are not strictly necessary in some iterations. The key is the following Lemma, which is a direct result of well-known linear programming theory (Bertsimas and Tsisiklis [4]).

Lemma 1. *Let (λ^*, u^*) represent an optimal basic solution of problem*

$$\begin{array}{rlll} Z(V) = & \max & c'V\lambda + d'u \\ & s.t. & 1 \cdot \lambda = 1 \\ & & HV\lambda + Gu \le h \\ & & \lambda \ge 0. \end{array}$$

Then, $|\{i : \lambda_i^* \ne 0\}| \le r_2 + 1$, where r_2 is the number of rows in matrix H.

Algorithm 1: The DW Algorithm.

Input: A feasible mixed integer programming problem of form

$$\begin{array}{rll} Z^* = & \max & c'z + d'u \\ & \text{s.t.} & Az \leq b \\ & & Hz + Gu \leq h \\ & & z \in \{0,1\}. \end{array} \tag{32}$$

A matrix V whose columns are feasible 0-1 solutions of $Az \leq b$.

Output: An optimal solution (z^*, u^*) to problem

$$\begin{array}{rll} Z^{DW} = \max & & c'z + d'u \\ & \text{s.t.} & z \in \texttt{conv.hull}(P), \\ & & Hz + Gu \leq h. \end{array} \tag{33}$$

1 $j \leftarrow 1$ and $Z_0^U \leftarrow \infty$.

2 $V^1 \leftarrow V$.

3 Solve the *restricted DW master problem*,

$$\begin{array}{rll} Z_j^L = & \max & c'V^j\lambda + d'u \\ & \text{s.t.} & 1 \cdot \lambda = 1 \\ & & HV^j\lambda + Gu \leq h \\ & & \lambda \geq 0 \end{array} \tag{34}$$

Let (λ^j, u^j) be an optimal solution to problem (34), and let (z^j, u^j), be the corresponding feasible solution of (33), where $z^j = V^j\lambda^j$. Let π^j be an optimal dual vector corresponding to constraints $HV^j\lambda + Gu \leq h$.

4 Solve the *DW pricing problem*. For this, consider the problem,

$$\begin{array}{rll} L(\pi) = & \max & c'v - \pi'(Hv - h) \\ & \text{s.t.} & Av \leq b, \\ & & v \in \{0,1\}^n \end{array} \tag{35}$$

and let v^j represent an optimal solution of $L(\pi^j)$. Let $Z_j^U \leftarrow \min\{L(\pi^j), Z_{j-1}^U\}$.

5 **if** $Z_j^L = Z_j^U$ **then**

6 $\quad z^* \leftarrow z^j$. Stop.

7 **else**

8 $\quad V^{j+1} \leftarrow [V^j, v^j]$.

9 $\quad j \leftarrow j + 1$. Go to step 3.

In fact, what this Lemma says is that after m iterations, no matter how large m, we can always choose to remove all but $r_2 + 1$ columns, thus making the problem smaller. As a means of ensuring that the resulting column generation algorithm does not cycle, a reasonable technique

would be to remove columns only after the primal bound in the DW algorithm changes. That is, only in those iterations k such that $Z^L_k > Z^L_{k-1}$.

It should be noted that a variant of this Lemma is also true for the GCG-EQ algorithm (see Appendix B). However, there is no similar result for the GCG algorithm. That is, while unused columns can be discarded in GCG, there is no guarantee that the columns remaining in the master problem will be few in number.

5.3. The BZ algorithm

The BZ algorithm, originally proposed by Bienstock and Zuckerberg [5, 6] is a variant of the GCG algorithm that is specialized for General Precedence Constrained Problems (GPCPs). That is, the BZ algorithm assumes that constraints $Az \leq b$ correspond to precedence constraints having form $z_i - z_j \leq 0$, for $(i, j) \in I$, and bound constraints, having form $0 \leq z \leq 1$. This assumption allows the BZ to exploit certain characteristics of the optimal master solutions in order to speed up convergence. As we will see, the BZ algorithm is very effective when the number of rows in H and the number of u variables is small relative to the number of z variables. It should be noted, however, that the optimal value of the problem solved by the BZ algorithm will have value $Z^{BZ} = Z^{LP}$. That is, the bound will be no tighter than that of the LP relaxation. This follows directly from the fact that $\{z : z_i \leq z_j \quad \forall (i, j) \in I\}$ defines a totally unimodular system.

The BZ algorithm is exactly as the GCG algorithm, with the exception of two changes:

First, the BZ algorithm uses a very specific class of generator matrices V^k. These matrices have two main properties. The first is that the linear space spanned by V^{k+1} will always contain z^k, the optimal solution of the master problem in the previous iteration. The second is that the columns of V^k are orthogonal 0-1 vectors. That is, for every column v^q of matrix V^k define $I^q = \left\{i \in \{1, \ldots, n\} : v^q_i \neq 0\right\}$ (note that I^q describes the *support* of v^q, for $q = 1, \ldots, k$). Then, 0-1 vectors v^q and v^r are orthogonal if and only if their supports are disjoint, i.e., $I^q \cap I^r = \emptyset$.

An intuitive explanation for why this would be desirable is that, when V^k has orthogonal 0-1 columns, problem (25) is equivalent to

$$\begin{array}{rlll} v^* = & \max & c'z + d'u & \\ & \text{s.t.} & Az \leq b & \\ & & Hz + Gu \leq h & \\ & & z_i = z_j & \forall i, j \in I^q,\ \forall q = 1, \ldots, k. \end{array} \tag{36}$$

That is, restricting the original feasible region to the linear space spanned by V^k is equivalent to equating the variables corresponding to the non-zero entries of each column in V^k. In a combinatorial optimization problem, this resembles a contraction operation, which is desirable because it can lead to a significant reduction of rows and variables, while yet preserving the original problem structure.

The matrix V^k used in the BZ algorithm is obtained by the following recursive procedure. Let $\hat{v}$ be the 0-1 vector obtained from solving the pricing problem. Let $[V^k, \hat{v}]$ be the matrix obtained by appending column $\hat{v}$ to V^k. We would like to compute a matrix V^{k+1} comprised of orthogonal 0-1 columns such that, first, $Span(V^{k+1}) \supseteq Span([V^k, \hat{v}])$; and second, such that V^{k+1} does not have too many columns. This can be achieved as follows. For two vectors $x, y \in \{0, 1\}^n$, define $x \wedge y \in \{0, 1\}^n$ such that $(x \wedge y)_i = 1$ if and only if $x_i = y_i = 1$. Likewise, define $x \setminus y \in \{0, 1\}^n$

such that $(x \setminus y)_i = 1$ if and only if $x_i = 1$ and $y_i = 0$. Assume that V^k is comprised of columns $\{v^1, \ldots, v^r\}$. Let V^{k+1} be the matrix made of the non-zero vectors from the collection:

$$\{v^j \wedge \hat{v} : 1 \leq j \leq r\} \cup \{v^j \setminus \hat{v} : 1 \leq j \leq r\} \cup \left\{ \hat{v} \setminus \left(\sum_{j=1}^{k} v^j \right) \right\}.$$

We call this procedure of obtaining the matrix V^{k+1} from V^k and $\hat{v}$ the *refining* procedure. Note that in each iteration k, the refining procedure will produce a matrix V^{k+1} with at most $2r+1$ columns.

The second difference between BZ and GCG is that it introduces a mechanism for preventing an unmanageable growth in the number of columns used in the master problem.

Consider any nonzero vector $z \in \mathbb{R}^n$. Let $\lambda_1, \ldots, \lambda_d$ denote the distinct non-zero values of the components of z, and for $i = 1, \ldots, d$ let $v^i \in \mathbb{R}^n$ denote the indicator vector of the components of z equal to λ_i, that is, v^i has components $v^i_j = 1$ if $z_j = \lambda_i$, and 0 otherwise. Note that $1 \leq d \leq n$ and $v^1, \ldots, v^d$ are orthogonal 0-1 vectors. Thus we can write $z = \sum_{i=1}^{d} \lambda_i v^i$ and we say that $v^1, \ldots, v^d$ define the *elementary basis* of z.

If at some iteration the number of columns in matrix V^k is too large, the BZ algorithm will replace matrix V^k with a matrix whose columns make up an elementary basis of the incumbent master solution z^k.

Again, there are two possible problems with this. On the one-hand, it is possible that such a coarsification procedure leads to cycling. On the other-hand, it might still be the case that after applying coarsification procedure the number of columns remains prohibitively large.

To alleviate the first concern, the BZ algorithm applies the coarsification procedure in exactly those iterations in which the generation of a new column leads to a strict improvement of the objective function. That is, when $Z^L_k > Z^L_{k-1}$. The second concern is alleviated by the fact that under the BZ algorithm assumptions, the elementary basis associated to a basic feasible solution of problem (36) will always have at most r_2 elements, where r_2 is the number of rows in matrix H (see Bienstock and Zuckerberg [5] for the original proof, or Appendix C for the proof adapted to our notation and extra variables).

A formal description of the column generation algorithm that results from the previous discussion is summarized in Algorithm 2.

5.4. BZ algorithm speedups

In this section we describe a number of computational techniques for improving the performance of the BZ algorithm. The computational techniques that we present for speeding up the master problem are generic, and can be used more generally in the GCG algorithm. However, the speedups that we present for the pricing problem are specific for solving generalized production scheduling problems (GPSPs), and thus, are specific for the BZ algorithm.

Algorithm 2: The BZ Algorithm.

Input: A feasible linear programming problem of form

$$\begin{array}{lll} Z^* = & \max & c'z + d'u \\ & \text{s.t.} & Az \leq b \\ & & Hz + Gu \leq h \end{array} \tag{37}$$

where constraints $Az \leq b$ correspond to precedence constraints having form $z_i - z_j \leq 0$, for $(i, j) \in I$, and bound constraints, having form $0 \leq z \leq 1$. A matrix V whose columns are orthogonal 0-1 vectors spanning at least one feasible solution of (37).

Output: An optimal solution (z^*, u^*) to problem (37).

1 $j \leftarrow 1$ and $Z_0^U \leftarrow \infty$.

2 $V^1 \leftarrow V$.

3 Solve the *restricted BZ master problem*,

$$\begin{array}{lll} Z_j^L = & \max & c'V^j\lambda + d'u \\ & \text{s.t.} & AV^j\lambda \leq b \\ & & HV^j\lambda + Gu \leq h \end{array} \tag{38}$$

Let (λ^j, u^j) be an optimal solution to problem (38), and let (z^j, u^j), be the corresponding feasible solution of (37), where $z^j = V^j\lambda^j$. Let π^j be an optimal dual vector corresponding to constraints $HV^j\lambda + Gu \leq h$.

4 Solve the *BZ pricing problem*. For this, consider the problem,

$$\begin{array}{lll} L(\pi) = & \max & c'v - \pi'(Hv - h) \\ & \text{s.t.} & Av \leq b, \end{array} \tag{39}$$

and let v^j represent an optimal solution of $L(\pi^j)$. Let $Z_j^U \leftarrow \min\{L(\pi^j), Z_{j-1}^U\}$.

5 **if** $Z_j^L = Z_j^U$ **then**

6 $\quad z^* \leftarrow z^j$. Stop.

7 **else**

8 $\quad$ Obtain V^{j+1}, a matrix of orthogonal 0-1 columns, by refining V^j with v^j.

9 $\quad j \leftarrow j + 1$. Go to step 3.

5.4.1. Speeding up the BZ pricing algorithm

The pricing problem consists of solving a problem of the form

$$\begin{array}{lll} \max & \bar{c}'z & \\ \text{s.t.} & z_i \leq z_j & \forall (i, j) \in I \\ & 0 \leq z \leq 1 & \end{array} \tag{40}$$

for some objective function $\bar{c}$, and a set of arcs I. This class of problems are known as *maximum closure* problems. In order to solve (40), we use the Pseudoflow algorithm of [21]. We use the following two features to speed-up this algorithm in the present context.

Pricing hot-starts [PHS]. The pricing problem is solved once per iteration of the BZ algorithm, each time with a different objective function. An important feature of the Pseudoflow algorithm is that it can be hot-started. More precisely, the Pseudoflow algorithm uses an internal data structure called a *normalized tree* that can be used to re-solve an instance of a maximum closure problem after changing the objective function vector. Rather than solve each pricing problem from scratch, we use the normalized tree obtained from the previous iteration to hot-start the algorithm. Because the changes in the objective function are not componentwise monotonous from iteration to iteration, we need to re-normalize the tree each time. For more information on the detailed working of the Pseudoflow algorithm, with indications on how to incorporate hot-starts, see [21].

Path contractions [PC]. Consider problem (40), and define an associated directed acyclic graph $G = (\mathscr{V}, \mathscr{E})$ as follows. For each variable z_i define a vertex v_i. For each precedence constraint $z_i \leq z_j$ with $(i, j) \in I$, define a directed arc (v_i, v_j) in $\mathscr{E}$. We say that a directed path $P = (v(1), v(2), \ldots, v(k))$ in G is *contractible* if it is a maximal path in G such that (i) $k \geq 3$, and (ii) every internal vertex $v(2), \ldots, v(k-1)$ has both in-degree and out-degree equal to one.

Observe that the variables associated to this path can only take $k+1$ different combinations of 0-1 values: either (i) $z_{v(i)} = 0$ for $i = 1, \ldots, k$ and the total contribution of the nodes in P is zero; or else, (ii) there exists an index $j \in \{1, \ldots, k\}$ such that $z_{v(i)} = 0$ for all $i = 1, \ldots, j-1$ and $z_{v(i)} = 1$ for all $i = j, \ldots, k$.

Since (40) is a maximization problem, in an optimal integer solution to (40) the contribution of the path will either be 0 (this will happen when $z_{v(k)} = 0$), or the contribution will be $\max_{1 \leq j \leq k} \sum_{i=j}^{k} \bar{c}_{v(i)}$ (which will happen when $z_{v(k)} = 1$).

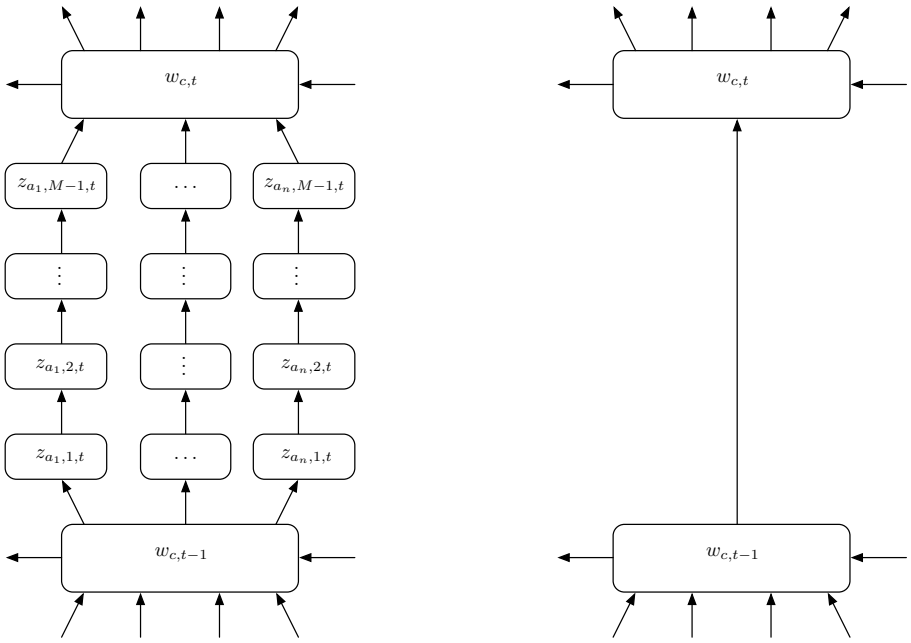
This suggests the following arc-contracting procedure.

Let $j(P, \bar{c}) = \operatorname{argmax}_{1 \leq j \leq k} \sum_{i=j}^{k} \bar{c}_{v(i)}$ and define a new graph $\hat{G} = (\hat{\mathscr{V}}, \hat{\mathscr{E}})$ from G by eliminating the vertices $v(2), \ldots, v(k-1)$ from $\mathscr{V}$ and the arcs incident to them, and then adding an arc that connects $v(1)$ and $v(k)$. Define $\hat{c}$ with $\hat{c}_v = \bar{c}_v$ for all vertices $v \notin \{v_1, \ldots, v_k\}$, $\hat{c}_{v(k)} = \sum_{i=j(P,\bar{c})}^{k} \bar{c}_{v(i)}$ and $\hat{c}_{v(1)} = \sum_{i<j(P,\bar{c})} \bar{c}_{v(i)}$.

Solving the pricing problem in this smaller graph $\hat{G}$ with the objective $\hat{c}$ gives an optimal solution to the original problem on graph G with objective $\bar{c}$, with the same optimal objective value.

This procedure can be used to contract all contractible paths in G in order to obtain, in some cases, a significantly smaller graph. In fact, problem instances with multiple destinations and batch constraints induce many contractible paths in the pricing problem graph. This is illustrated with an example in Figure 1.

It should be noted that identifying and contracting paths only needs to be done in the first iteration. In all subsequent runs of the pricing problem, it is possible to use the same contracted graph, after updating the indices $j(P, \bar{c})$ and the objective function $\hat{c}$.



(a) Before path contraction

(b) After path contraction

Figure 1: Example illustrating the possible effect of the path contraction speed-up. This example assumes a cluster c comprised of activities $\{a_1, \ldots, a_n\}$. Each activity is assumed to have M possible modes. The graph includes an arc between all pairs of variables for which there is a precedence relationship.

5.4.2. Speeding up the BZ master algorithm

The following ideas can be used to speed up solving the master problem (38) in the BZ algorithm:

Starting columns [STCOL]. As input, the BZ algorithm requires an initial set of columns inducing a (possibly infeasible) solution of the problem. Such columns can be obtained by computing an elementary basis associated to a solution obtained with heuristics.

If, when starting the algorithm, we do not have an initial set of columns describing a feasible the problem, we can proceed as in the Phase-I simplex algorithm. For this, start with some arbitrary set of columns, and add "artificial variables" for each row.

Master hot-starts [MHS]. Because of the refining procedure, the restricted BZ master problem (38) in an iteration may be quite different from that in the previous iteration. To reduce its solve time, we can feed any simplex-based solver the solution from the previous iteration so that it can be used to attempt and identify a good feasible starting basis.

k-Step column management [k-Step]. In the original BZ algorithm, the coarsification procedure is applied in every iteration where there is a strict increase in the primal objective function value. Sometimes, however, it is better to keep the existing columns so as to improve the convergence rate. To avoid having too many columns in each iteration, we use what we refer to as

a k-step column management rule. This rule, which requires as input an integer parameter k, works as follows: assume that we are at the m-th iteration of the BZ algorithm, where $m > k$. Further assume that in the previous iteration (iteration $m-1$) there was a strict increase in the primal objective function value. The column matrix V^m used in the m-th iteration is built from scratch, by first obtaining an elementary basis associated to solution z^{m-k}, and then successively refining this basis with the vectors $v^{m-k+1}, v^{m-k+2}, \ldots, v^{m-1}$ (see Section 5.3).

6. Computational results

Our computational tests consider solving the linear relaxation of three different classes of problems. Specifically, we consider instances of OPPSP, OPPDP and RCPSP. The OPPSP and OPPDP instances are based on both real and hypothetical mine-planning instances obtained from industry partners, and from the MineLib website [16]. In order to protect the confidentiality of the data sets obtained from our industry partners, we have renamed all of the instances using the names of Chilean volcanoes. In Table 1 we present some characteristics of these instances. Note that for each instance we have a specification regarding the maximum number of time periods to consider. For each of the OPPSP problems we count with two versions: one with with less, and one with more clusters. We refer to these as the fine and course versions of each problem. These clusters correspond to what mining engineers refer to as a bench-phase, or increment (see [22] for formal definitions). We do not have cluster definition data for all of the problem instances, thus, the set of instances considered for OPPSP is a subset of the instances considered for OPPDP. The RCPSP instances that we consider are obtained from PSPlib [25] (datasets *j30*, *j60*, *j90* and *j120*) and from [14] (dataset *RG300*). It should be noted that these problems have a different objective function than the OPPDP and OPPSP problems. That is, these problems seek to minimize Makespan, rather than maximize net present value (NPV).

We evaluate the performance of the BZ algorithm, comparing it to the DW algorithm, and to a version of DW that incorporates the dual-stabilization techniques (DW-S) proposed by Pessoa et al. [37]. For each of the algorithms we attempt to compute the linear relaxation solution of the test problems, stopping early if the relative difference between the master problem bound and the pricing problem bound is less than 10^{-6} (the default reduced cost tolerance used by the CPLEX simplex algorithm). We do not report the bounds obtained by each algorithm since they will all coincide with the value of the LP relaxation for each problem.

We also evaluate the performance of the speed-up techniques described in this paper. Specifically, we analyze the performance of Pricing hot-starts (PHS), Path contractions (PC), Starting columns (STCOL), Master hot-starts (MHS), and k-step column management (k-Step), with $k = 10$. All of these speed-up techniques, with the exception of k-step (which is not applicable) are also implemented for the DW and DW-S algorithms. In fact, our code uses the exact same implementations of these speed-up techniques for the BZ, DW and DW-S algorithms.

To assess the value of our proposed speed-up techniques we defined a *default* set of speed-up-techniques to activate for each class of problems considered. For the OPPDP and OPPSP we defined the default speed-up-techniques to be PHS, MHS and PC. We found these to be the

Table 1: Description of the different instances of the test set used in our computational study.

Instance	Blocks (UPIT)	Modes	Periods	Resources	Clusters	
					Course	Fine
antuco	3 525 317	2	45	2	—	—
calbuco	198 248	3	21	1	324	548
chaiten	287 473	2	20	2	273	475
dospuntas	897 609	2	40	4	—	—
guallatari	57 958	3	21	3	272	460
kd	12 154	2	12	1	53	93
lomasblancas	1 492 024	3	45	3	—	—
marvin	8 516	2	20	2	56	98
mclaughlin_limit	110 768	2	15	1	166	290
palomo	97 183	2	40	2	44	93
ranokau	304 977	2	81	2	186	296
sm2	18 388	2	30	2	—	—
zuck_large	96 821	2	30	2	—	—
zuck_medium	27 387	2	15	2	—	—
zuck_small	9 399	2	20	2	—	—

features that improved performance of the algorithm, without requiring a heuristic to generate a solution. For the RCPSP class of problems we use a different set of default speed-up options. Since we have to compute a feasible solution to each problem in order to define the number of time periods, we change the default settings so as to use this as a starting solution (STSOL). In addition, we observe that these problem instances have signficantly more time periods per activities when compared to the OPPSP and OPPDP instances. This resulted in a very different algorithm behavior that prompted us to include k-Step column management as a default option, as it improved problem performance.

In order to assess the contribution of each individual speed-up technique in the algorithm, we turned each of these off (or on) to measure the impact from the change relative to that of the default settings.

All algorithms were implemented using C programming language, using CPLEX® 12.6 as optimization solver. The machines were running Linux 2.6.32 under x86_64 architecture, with four eight-core Intel® Xeon® E5-2670 processors and with 128 Gb of RAM. For all results, we present normalized geometric means comparing the performance versus BZ algorithm under default configuration.

6.1. Results for OPPDP

Table 2 shows the time required to solve each problem using the different proposed algorithms. The table also shows the number of iterations required by each algorithm, as well as the final number of columns generated by BZ. We do not report the bound generated by each relaxation, since they all generate the same bound for each instance.

Table 2: Comparison between the different algorithms for OPPDP instances

Instance	Time (sec)				Iterations			BZ cols
	BZ	DW	DW+S	CPLEX	BZ	DW	DW+S	
antuco	208 000	1 542 712	353 313	-	97	2 539	559	31 640
calbuco	1 480	6 715	3 602	-	34	243	132	1 876
chaiten	2 620	35 339	9 625	-	34	884	259	5 800
dospuntas	36 100	451 515	108 105	-	52	1 542	385	112 943
guallatari	143	1 668	494	-	30	444	158	2 421
kd	7	22	16	15 560	23	115	90	365
lomasblancas	86 600	521 279	208 256	-	75	773	360	5 635
marvin	9	28	17	34 601	27	154	99	206
mclaughlin_limit	167	1 096	539	-	27	208	102	1 273
palomo	1 290	9 550	3 637	-	45	587	264	2 089
ranokau	192 000	1 526 274	210 506	-	108	7 261	1 025	124 406
sm2	44	256	109	254	52	874	281	3 594
zuck_large	634	3 564	1 761	-	46	416	194	3 195
zuck_medium	40	112	93	954 405	28	107	82	229
zuck_small	11	37	24	56 165	29	150	113	294
Norm. G. Mean	1	5.98	2.42	-	1	11.8	4.9	

As can be seen, DW is nearly six times slower than BZ. Using stabilization techniques, the DW-S is able to signficantly reduced the time required to reach the optimality condition. In fact, stabilization techniques reduce the time of DW in a 60%, but it is still 2.42 times slower than BZ. This can be explained by the number of iterations required to converge by each algorithm. Even though a BZ iteration is, in average, 2.1 times slower than a DW iteration, DW and DW+S require 11.8 and 4.9 times the number of iterations of BZ to converge. This is possible because the BZ algorithm can produce a large number of useful columns in few iterations. In fact, the number of columns generated by BZ is considerably larger than the number of columns produced by DW and DW-S (the number of columns for these algorithms is equal to the number of iterations). Finally, note that CPLEX is only able to solve the five smallest instances.

In Table 3 we show what happens when we change the tolerance used to define optimality in all of the algorithms. It can be seen that BZ algorithm still outperforms the DW and DW-S algorithms after reducing the target optimality gap to 10^{-4} and 10^{-2}. However, the performance difference narrows as the target gap becomes smaller. The resulting times, normalized to the time of BZ under default setting, are presented in Table 3.

Finally, we study the impact of the different speed-up techniques on BZ. From the default settings, we disable the speed-ups PHS, MHS and PC, one-by-one, and also, we disable all three of them together. We also run BZ under our default setting after adding a starting column

Table 3: Normalized time required for different optimality gaps for OPPDP instances

Instance	10^{-6}			10^{-4}			10^{-2}		
	BZ	DW	DW+S	BZ	DW	DW+S	BZ	DW	DW+S
antuco	1	7.42	1.70	0.96	6.28	1.49	0.93	4.63	1.20
calbuco	1	4.54	2.43	0.92	3.49	1.94	0.74	1.55	1.08
chaiten	1	13.49	3.67	0.95	10.53	2.92	0.79	4.30	1.57
dospuntas	1	12.51	2.99	0.83	7.77	2.16	0.67	2.40	1.15
guallatari	1	11.66	3.45	0.86	7.35	2.44	0.70	2.60	1.37
kd	1	3.05	2.25	0.90	2.40	1.78	0.76	1.13	1.18
lomasblancas	1	6.02	2.40	0.95	5.04	2.05	0.87	3.26	1.38
marvin	1	3.19	1.99	0.91	2.26	1.47	0.81	0.94	0.81
mclaughlin_limit	1	6.56	3.23	0.86	4.75	2.47	0.68	1.89	1.47
palomo	1	7.40	2.82	0.98	6.10	2.33	0.88	2.69	1.50
ranokau	1	7.95	1.10	0.82	4.79	0.86	0.63	1.87	0.50
sm2	1	5.88	2.50	0.95	3.77	1.89	0.88	2.03	1.35
zuck_large	1	5.62	2.78	0.96	4.42	2.28	0.84	2.21	1.45
zuck_medium	1	2.80	2.31	0.87	2.15	1.78	0.66	1.23	0.97
zuck_small	1	3.22	2.13	0.86	2.20	1.52	0.77	1.24	0.85
Norm. G. Mean	1	5.98	2.42	0.90	4.36	1.89	0.77	2.04	1.14

(STCOL) generated using the Critical Multiplier algorithm (See [9]), and after enabling the k-Step feature. The resulting times, normalized to the time of BZ under default setting, are presented in Table 4.

We can see that all speeding features provide, in average, an improvement on the time required by BZ to converge. However, the individual performance of each feature differs instance by instance. The most important speed-up technique is path contraction (PC), as disabling this feature increases the time required to converge by 60%. Since the reduction in number of variables of this speed-up in OPPDP is proportional to the number of modes of the problem, this feature is particularly important for the Guallatari instance, which has 3 different modes. Disabling these three features, the total time required to converge more than doubles. On the other hand, if we start with a preliminary set of starting columns, the time required is decreased by 32%. Finally, we see that k-Step does not improve the convergence time, making the algorithm run 35% slower. In fact, it makes every problem converge slower, with the exception of the ranokau instance.

6.2. Results for OPPSP

For these instances we use the same default settings as those used for OPPDP. We note that these problems are considerably smaller than the OPPSP problems. This is both in the number of variables and precedence constraints. All algorithms are able to solve these problems in a few

Table 4: Normalized time required for BZ algorithm with/without features for OPPDP instances

Instance	Default	No PHS	No PC	No MHS	No PHS/PC MHS	Default + STCOL	Default + k-Step
antuco	1	1.57	1.60	0.98	2.50	0.57	2.56
calbuco	1	1.48	1.77	1.10	2.78	0.85	1.09
chaiten	1	1.38	1.49	1.01	1.83	0.62	1.20
dospuntas	1	1.36	1.57	1.03	1.75	0.73	1.52
guallatari	1	1.47	2.18	1.21	2.88	1.00	0.99
kd	1	1.30	1.63	1.00	2.22	0.69	1.51
lomasblancas	1	1.84	1.68	1.01	2.94	0.25	1.98
marvin	1	1.32	1.27	0.97	2.47	0.91	1.49
mclaughlin_limit	1	1.27	1.78	0.94	2.06	0.77	1.20
palomo	1	1.60	1.47	0.98	2.21	0.94	1.53
ranokau	1	0.93	1.04	0.97	1.15	0.22	0.57
sm2	1	1.33	1.93	1.16	2.35	1.11	2.06
zuck_large	1	1.15	1.60	1.00	2.08	0.62	1.31
zuck_medium	1	1.75	1.59	1.20	2.74	0.84	1.13
zuck_small	1	1.32	1.72	0.95	2.41	0.92	1.27
Norm. G. Mean.	1	1.39	1.60	1.03	2.24	0.68	1.35

hours, with the exception of the *ranokau* instance, which CPLEX fails to due to the memory limit (128Gb). We present the results in Table 5.

We can see that in this class of problems the performance of DW and DW-S is more similar to that of BZ. This is probably explained by the fact that the number of iterations required by DW and DW-S is much smaller. Note also that stabilization techniques for DW only marginally reduce the number of iterations required by DW to converge, resulting in that DW-S is 18% slower than DW.

Comparing the impact of the different features on BZ (see Table 6), we see that the most important feature is, again, path contraction (PC). Disabling this feauture makes the algorithm run almost 10 times slower. Similarly to the OPPDP problems, providing starting columns to BZ reduces the time by 23%, and introducing k-Step column management makes the problem run slower (83%).

6.3. Results for RCPSP instances

In order to formulate the RCPSP instances we need a maximum number of time periods to consider. Since the objective of these problems is to minimize Makespan, the number of time

Table 5: Comparison between the different algorithms for OPPSP instances

Instance	Time (sec)				Iterations			BZ cols
	BZ	DW	DW+S	CPLEX	BZ	DW	DW+S	
calbuco	88	90	122	56 102	32	96	95	167
chaiten	86	174	184	4 595	34	189	149	392
guallatari	29	39	50	11 757	30	117	102	188
kd	1	1	1	5	18	44	40	70
marvin	2	2	2	17	27	70	73	82
mclaughlin_limit	14	19	22	2 310	25	78	73	100
palomo	49	59	95	429	44	138	137	160
ranokau	2 034	5 102	4084	-	186	1 461	781	2 108
Norm. G. Mean	1	1.38	1.63	54.57	1	3.64	3.17	

Table 6: Normalized time required for BZ algorithm with/without features for OPPSP instances

Instance	Default	No PHS	No PC	No MHS	Default + STCOL	Default + k-Step
calbuco	1	0.92	9.96	1.03	0.82	2.10
chaiten	1	0.95	14.63	1.04	0.81	2.06
guallatari	1	1.01	4.91	1.05	0.73	1.77
kd	1	1.04	5.63	1.05	0.65	1.59
marvin	1	0.90	3.51	0.92	0.68	1.26
mclaughlin_limit	1	0.99	14.65	0.93	0.98	1.91
palomo	1	0.87	13.33	0.97	0.77	1.83
ranokau	1	1.03	19.48	1.03	0.78	2.37
Norm. G. Mean	1	0.96	9.25	1.00	0.77	1.83

periods should be an upper bound on the number of periods required to schedule all of the activities. Such a bound can be computed by running any heuristic to compute any feasible integer solution. For this purpose, we use a greedy TOPOSORT heuristic [9], which takes fractions of a second to solve for all of our instances.

Table 7 describes the performance of the different algorithms on our RCPSP test instances. Note that we only consider instances that are solved by CPLEX in more than 1 second, obtaining a total of 1575 instances. The running times presented in the table are geometric means over instances in the same dataset.

Table 7: Comparison between the different algorithms for RCPSP instances

		Time (sec)				Iterations		
Dataset †		BZ	DW	DW+S	CPLEX	BZ	DW	DW+S
j30	(51 inst.)	1.23	4.71	1.91	1.61	141.4	650.4	312.9
j60	(152 inst.)	0.98	8.19	2.55	3.77	68.5	477.0	232.1
j90	(293 inst.)	0.54	2.27	1.07	5.33	23.3	115.4	73.4
j120	(599 inst.)	3.95	33.13	11.94	31.78	58.2	440.5	230.5
RG300	(480 inst.)	22.86	43.76	24.87	240.51*	91.1	393.9	260.5
Norm. G. Mean		1	4.76	2.00	7.25	1	5.8	3.3

†: We only consider instances which took CPLEX more than a second to solve.

∗: We only consider the 438 instances which were solved within 48 hours.

Table 7 shows that BZ is again faster than the other algorithms when solving RCPSP instances. In fact, it is 2 times faster than DW+S and 4.7 faster than DW. Note that this difference is particularly large for the *j120* instances from PSPLIB repository, where DW and DW+S run 8.4 and 3 times slower, respectively. It would seem that the performance of these algorithms is greatly dependent on problem structure. This can be seen when considering the *RG300* instances, which are generated in a different way. In these instances, DW with stabilization is only marginally slower than BZ.

Table 8 shows how much the performance of the BZ algorithm is affected by turning off each of the default speed up features. It is interesting to note that on RCPSP instances the k-Step column management rule is very important. Turning it off makes the problem run, in average, 2.51 times slower. This is in stark contrast to what happens with the OPPSP and OPPDP instances, where activating the k-Step feature actually makes the problem run slower. We speculate that this is due to the fact that the RCPSP problems have a significantly greater number of time periods per activity than the OPPSP and OPPDP instances. This results in a problem with significantly more resource consumption constraints per variable. It is also interesting to note that disabling the PC and MHS features actually makes BZ run faster in the RCPSP instances. We speculate that the MHS feature does not improve performance because, having enabled the k-Step feature as well, succesive master problem formulations significantly differ from each other. We speculate that the PC feature does not improve performance because

in RCPSP instances there are not as many paths to contract due to the structure of the precedence graphs. This is due to the fact that there are no clusters and that there is just a single mode per activity.

Table 8: Normalized time required for BZ algorithm with/without features for RCPSP instances

Dataset †	Default	No PHS	No PC	No MHS	No k-Step	No STCOL
j30 (51 inst.)	1	1.30	0.78	0.80	0.74	1.44
j60 (152 inst.)	1	1.53	0.99	1.03	1.44	1.91
j90 (293 inst.)	1	1.15	0.86	0.91	1.51	1.80
j120 (599 inst.)	1	1.40	0.92	0.95	2.39	1.23
RG300 (453 inst.)	1	1.22	0.87	0.89	5.78	1.04
Norm. G. Mean	1	1.30	0.90	0.93	2.51	1.33

†: We only consider instances for which the BZ algorithm finished in 48 hours in all settings.

6.4. Concluding remarks

We summarize with three important conclusions. First, the BZ algorithm significantly outperforms DW and DW+S on all GPCP problem classes. Second, the speed-up features proposed in this article significantly improve the performance of the BZ algorithm. Third, the algorithm and speed-up performances greatly depend on the problem classes that are being considered. These conclusions suggest that the BZ algorithm with the proposed speed-ups is a technique that can be used to effectively be used to compute the LP relaxation solution of different classes of scheduling problems. Such an algorithm might be useful as a means to provide upper bounds for scheduling problems, or as a part of the many rounding heuristics that have recently been proposed, or eventually, as part of a branch-and-bound solver. In addition, they suggest that the BZ algorithm's potential use for other classes of problems should also be further studied.

References

[1] Alford, C., Brazil, M., Lee, D.: Optimisation in underground mining. In: Handbook of operations research in natural resources, pp. 561–577. Springer (2007)

[2] Artigues, C., Demassey, S., Neron, E.: Resource-constrained project scheduling: models, algorithms, extensions and applications, vol. 37. John Wiley & Sons (2010)

[3] Berthold, T., Heinz, S., Lübbecke, M., Möhring, R., Schulz, J.: A constraint integer programming approach for resource-constrained project scheduling. In: Integration of AI and OR techniques in constraint programming for combinatorial optimization problems, pp. 313–317. Springer (2010)

[4] Bertsimas, D., Tsitsiklis, J.: Introduction to linear optimization, vol. 6. Athena Scientific Belmont, MA (1997)

[5] Bienstock, D., Zuckerberg, M.: A new LP algorithm for precedence constrained production scheduling. Optimization Online (2009)

[6] Bienstock, D., Zuckerberg, M.: Solving LP relaxations of large-scale precedence constrained problems. Proceedings from the 14th conference on Integer Programming and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science 6080 pp. 1–14 (2010)

[7] Boland, N., Dumitrescu, I., Froyland, G., Gleixner, A.: LP-based disaggregation approaches to solving the open pit mining production scheduling problem with block processing selectivity. Computers & Operations Research **36**, 1064–1089 (2009)

[8] Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research **112**(1), 3–41 (1999)

[9] Chicoisne, R., Espinoza, D., Goycoolea, M., Moreno, E., Rubio, E.: A new algorithm for the open-pit mine production scheduling problem. Operations Research **60**(3), 517–528 (2012)

[10] Christofides, N., Alvarez-Valdés, R., Tamarit, J.: Project scheduling with resource constraints: A branch and bound approach. European Journal of Operational Research **29**(3), 262–273 (1987)

[11] Dagdelen, K., Johnson, T.: Optimum open pit mine production scheduling by lagrangian parameterization. In: Proceedings of the 19th International Symposium on the Application of Computers and Operations Research in the Mineral Industry (APCOM) (1986)

[12] Dantzig, G., Wolfe, P.: Decomposition principle for linear programs. Operations research **8**(1), 101–111 (1960)

[13] Dassault Systèmes: GEOVIA Whittle (2015). URL `http://www.gemcomsoftware.com/products/whittle`

[14] Debels, D., Vanhoucke, M.: A decomposition-based genetic algorithm for the resource-constrained project-scheduling problem. Operations Research **55**(3), 457–469 (2007)

[15] Deswik: Deswik.sched (2015). URL `https://www.deswik.com/product-detail/deswik-scheduler/`

[16] Espinoza, D., Goycoolea, M., Moreno, E., Newman, A.: Minelib: A library of open pit mining problems. Annals of Operations Research **206**, 93–114 (2013)

[17] Fisher, M.: Optimal solution of scheduling problems using lagrange multipliers: Part i. Operations Research **21**(5), 1114–1127 (1973)

[18] Goycoolea, M., Espinoza, D., Moreno, E., Rivera, O.: Comparing new and traditional methodologies for production scheduling in open pit mining. In: Proceedings of the 37th International Symposium on the Application of Computers and Operations Research in the Mineral Industry (APCOM), pp. 352–359 (2015)

[19] Graham, R., Lawler, E., Lenstra, J., Kan, A.: Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics **5**, 287–326 (1979)

[20] Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research **207**(1), 1–14 (2010)

[21] Hochbaum, D.: The pseudoflow algorithm: A new algorithm for the maximum-flow problem. Operations Research **56**, 992–1009 (2008)

[22] Hustrulid, W., Kuchta, K. (eds.): Open Pit Mine Planning and Design. Taylor and Francis, London, UK (2006)

[23] Johnson, T.: Optimum open pit mine production scheduling. Ph.D. thesis, Operations Research Department, Universiry of California, Berkeley (1968)

[24] Kolisch, R., Sprecher, A.: PSP-Library. Online at `http://www.om-db.wi.tum.de/psplib/datamm.html` (1997). [Online; accessed March-2015]

[25] Kolisch, R., Sprecher, A.: PSPLIB - a project scheduling problem library. European Journal of Operational Research **96**(1), 205 – 216 (1997)

[26] Lambert, W.B., Newman, A.M.: Tailored lagrangian relaxation for the open pit block sequencing problem. Annals of Operations Research **222**(1), 419–438 (2014)

[27] Martinez, M., Newman, A.: A solution approach for optimizing long-and short-term production scheduling at LKAB's Kiruna mine. European Journal of Operational Research **211**(1), 184–197 (2011)

[28] MineMax: iGantt. `https://www.minemax.com/solutions/products/igantt` (2015)

[29] MineMax: Planner. `http://www.minemax.com/solutions/requirements/strategic-planning` (2015)

[30] MineMax: Scheduler (2015). URL `http://www.minemax.com/solutions/requirements/strategic-pl`

[31] Möhring, R., Schulz, A., Stork, F., Uetz, M.: Solving project scheduling problems by minimum cut computations. Management Science **49**(3), 330–350 (2003)

[32] Newman, A., Kuchta, M.: Using aggregation to optimize long-term production planning at an underground mine. European Journal of Operational Research **176**(2), 1205–1218 (2007)

[33] Newman, A., Rubio, E., Caro, R., Weintraub, A., Eurek, K.: A review of operations research in mine planning. Interfaces **40**, 222–245 (2010)

[34] Osanloo, M., Gholamnejad, J., Karimi, B.: Long-term open pit mine production planning: a review of models and algorithms. International Journal of Mining, Reclamation and Environment **22**(1), 3–35 (2008)

[35] O'Sullivan, D., Newman, A.: Extraction and backfill scheduling in a complex underground mine. Interfaces **44**(2), 204–221 (2014)

[36] O'Sullivan, D., Newman, A., Brickey, A.: Is open pit production scheduling 'easier' than its underground counterpart? Mining Engineering **67**(4), 68–73 (2015)

[37] Pessoa, A., Sadyvok, R., Uchoa, E., Vanderbeck, F.: In-out separation and column generation stabilization by dual price smoothing. In: 12th International Symposium on Experimental Algorithms(SEA), Rome, Lecture Notes in Computer Science, vol. 7933, pp. 354–365 (2013)

[38] Pritsker, A., Waiters, L., Wolfe, P.: Multiproject scheduling with limited resources: A zero-one programming approach. Management science **16**(1), 93–108 (1969)

[39] Zhu, G., Bard, J., Yu, G.: A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. INFORMS Journal on Computing **18**(3), 377–390 (2006)

Appendix

A. Resource constraints in GPSP reformulation

In this section we provide a detailed proof on how to derive (14) from (5). This is needed in order to reformulate an instance of a General Production Scheduling Problem (GPSP), described in Section 3, as an instance of a General Precedence Constrained Problem (GPCP), presented in Section 4. For the benefit of the reader, we recall (14) and (5).

Proposition 2. *Consider inequality* (5)

$$\sum_{a\in\mathscr{A}}\sum_{m\in\mathscr{M}_a} q_{r,a,m} \sum_{s=\max\{1,t-d_{a,m}+1\}}^{t} y_{a,m,s} \le Q_{r,t},$$

as defined in Section 3. If we apply the variable substitution scheme introduced in Section 4:

$$\begin{aligned} y_{a,m,t} &= z_{a,m,t} - z_{a,m-1,t} && \forall a\in\mathscr{A},\ m=2,\dots,M_a,\ t\in\mathscr{T},\\ y_{a,1,t} &= z_{a,1,t} - z_{a,M_a,t-1} && \forall a\in\mathscr{A},\ t=2,\dots,T,\\ y_{a,1,1} &= z_{a,1,1} && \forall a\in\mathscr{A}, \end{aligned}$$

we obtain inequality (14)*:*

$$\sum_{a\in\mathscr{A}}\sum_{m\in\mathscr{M}_a}\sum_{s\in\mathscr{T}} \tilde{q}^{r,t}_{a,m,s} z_{a,m,s} \le Q_{r,t},$$

where

$$\tilde{q}^{r,t}_{a,m,s} = \begin{cases} q_{r,a,m}\mathbf{1}_{[t-d_{a,m}+1,t]}(s) - q_{r,a,m+1}\mathbf{1}_{[t-d_{a,m+1}+1,t]}(s) & \text{if } m < M_a\\ q_{r,a,M_a}\mathbf{1}_{[t-d_{a,M_a}+1,t]}(s) - q_{r,a,1}\mathbf{1}_{[t-d_{a,1},t-1]}(s) & \text{if } m = M_a \end{cases}$$

Proof.

$$
\begin{aligned}
Q_{r,t} \;\geq\; & \sum_{a\in\mathscr{A}} \sum_{m\in\mathscr{M}_a} q_{r,a,m} \sum_{s=\max\{1,t-d_{a,1}+1\}}^{t} y_{a,m,s} \\
=\; & \sum_{a\in\mathscr{A}} \left(\sum_{m=2}^{M_a} \sum_{s=\max\{1,t-d_{a,m}+1\}}^{t} q_{r,a,m} y_{a,m,s} + \sum_{s=\max\{1,t-d_{a,1}+1\}}^{t} q_{r,a,1} y_{a,1,s} \right) \\
=\; & \sum_{a\in\mathscr{A}} \left(\sum_{m=2}^{M_a} \sum_{s=\max\{1,t-d_{a,m}+1\}}^{t} q_{r,a,m} (z_{a,m,s} - z_{a,m-1,s}) \right. \\
& \left. + \sum_{s=\max\{1,t-d_{a,1}+1\}}^{t} q_{r,a,1} z_{a,1,s} - \sum_{s=\max\{2,t-d_{a,1}+1\}}^{t} q_{r,a,1} z_{a,M_a,s-1} \right) \\
=\; & \sum_{a\in\mathscr{A}} \left(\sum_{m=2}^{M_a} \sum_{s=\max\{1,t-d_{a,m}+1\}}^{t} q_{r,a,m} z_{a,m,s} - \sum_{m=1}^{M_a-1} \sum_{s=\max\{1,t-d_{a,m+1}+1\}}^{t} q_{r,a,m+1} z_{a,m,s} \right. \\
& \left. + \sum_{s=\max\{1,t-d_{a,1}+1\}}^{t} q_{r,a,1} z_{a,1,s} - \sum_{s=\max\{1,t-d_{a,1}\}}^{t-1} q_{r,a,1} z_{a,M_a,s} \right) \\
=\; & \sum_{a\in\mathscr{A}} \left(\sum_{m=1}^{M_a} \sum_{s=\max\{1,t-d_{a,m}+1\}}^{t} q_{r,a,m} z_{a,m,s} - \sum_{m=1}^{M_a-1} \sum_{s=\max\{1,t-d_{a,m+1}+1\}}^{t} q_{r,a,m+1} z_{a,m,s} \right. \\
& \left. - \sum_{s=\max\{1,t-d_{a,1}\}}^{t-1} q_{r,a,1} z_{a,M_a,s} \right).
\end{aligned}
$$

□

B. Equality form version of the GCG algorithm

In this section we present the GCG algorithm when, instad of the form (22), the mixed integer problem to be solved has form,

$$
\begin{aligned}
Z^{IP} = \max \quad & c'z + d'u \\
\text{s.t.} \quad & Az = b, \\
& Hz + Gu \leq h, \\
& z \in \{0,1\}^n.
\end{aligned}
\tag{41}
$$

That is, when the first set of inequalities have been replaced with equalities. As before, define

$$P = \{z : Az = b,\ z \in \{0,1\}\}.$$

We present a column generation algorithm for computing the optimal solution of

$$
\begin{aligned}
Z^{LIN} = \max \quad & c'z + d'u \\
\text{s.t.} \quad & Az = b, \\
& Hz + Gu \leq h, \\
& z \in \texttt{lin.hull}(P).
\end{aligned}
\tag{42}
$$

Let v^o be such that $Av^o = b$. For each $k \geq 1$ let V^k be a matrix comprised of linearly independent points $\{v^1, \ldots, v^k\}$ such that $Av^i = 0$ for $i = 1, \ldots, k$. Define,

$$\begin{array}{rlll} Z(V) = & \max & c'v^o + c'V\lambda + d'u & \\ & \text{s.t.} & HV\lambda + Gu \leq h - Hv^o & (\pi) \end{array} \tag{43}$$

Begin each iteration by solving problem (43) with $V = V^k$. Define $Z_k^L = Z(V^k)$ and let (λ^k, u^k) and π^k be the corresponding optimal primal and dual solutions respectively. Define $z^k = v^o + V^k\lambda^k$ and let,

$$\begin{array}{rlll} L(\pi) = & \max & c'v - \pi(Hv - h) & \\ & \text{s.t.} & Av = b, & \\ & & v \in \{0,1\}^n. & \end{array} \tag{44}$$

Observe that since $\pi^k G = d'$, we have:

$$\begin{array}{rlll} L(\pi^k) = & \max & c'v + d'u - \pi^k(Hv + Gu - h) & \\ & \text{s.t.} & Av = b, & \\ & & v \in \{0,1\}^n. & \end{array} \tag{45}$$

Thus, $L(\pi^k)$ is a relaxation of (41) obtained by penalizing constraints $Hz + Gu \leq h$ with $\pi^k \geq 0$. This implies $L(\pi^k) \geq Z^{IP}$ for all iterations k.

Solve problem (44) with $\pi = \pi^k$ and let $\hat{v}$ be the corresponding optimal solution. Define $Z_k^U = \min\{L(\pi^i) : i = 1, \ldots, k\}$. As before, we will have that $Z_k^U \geq Z^{IP}$ and $Z_k^L \leq Z^{LIN}$.

If, at any iteration, we have that $Z_k^L \geq Z_k^U$ we can halt the algorithm. By defining $Z^{UB} = Z_k^U$ we will have that $Z^{IP} \leq Z^{UB} \leq Z^{LIN}$.

On the other hand whenever $Z_k^L < Z_k^U$ we will have that column $\hat{v} - v^o$ has positive reduced cost in the master problem. Thus, letting $v^{k+1} = \hat{v} - v^o$ and $V^{k+1} = [V^k, v^{k+1}]$ we can continue iterating the algorithm.

C. Distinct fractional values Lemma

In this section we show that the BZ master problem will have bounded number of distinct fractional values. This fact is important in Section 5.3, in order to argue why the use of the *elementary basis* can reduce the number of columns in the BZ algorithm.

Consider the setting for the BZ algorithm, i.e, a problem of the form

$$\begin{array}{rlll} Z^{BZ} = & \max & c'z + d'u & \\ & \text{s.t.} & Az \leq b & \\ & & Hz + Gu \leq h & \end{array} \tag{46}$$

where $\{z : Az \leq b\} = \{z : z_i \leq z_j \; \forall \; (i,j) \in I, \; 0 \leq z \leq 1\}$. It is well known that A defines a totally unimodular matrix in such case, thus all extreme points of $\{z : Az \leq b\}$ are 0-1 vectors.

Recall that z is an n-dimensional vector, and let m be the dimension of u. In this section we prove the following result:

Lemma 3. *Let $(\bar{z},\bar{u})$ be an extreme point of* (46)*, and let q be the number of rows of H (and G). Then the number of distinct fractional values of $\bar{z}$ is at most $q-m$.*

Note that by assuming an extreme point $(\bar{z},\bar{u})$ of (46) exists, we are implicitly assuming $q \geq m$. In fact, if $(\bar{z},\bar{u})$ is an extreme point of (46), then $\bar{u}$ must be an extreme point of $\{u : Gu \leq h - H\bar{z}\}$. This, however, requires that G have at least m rows, thus implying that $q-m \geq 0$ holds.

An almost identical result is originally proved in [5]. We present such proof here, adapted to our notation and including the u variables, which modifies the final result. Note that this theorem also applies to every *restricted BZ master problem* (38), since the latter is obtained by simply equating sets of variables of (37), thus maintaining the same structure.

To prove Lemma 3 we will make use of the *elementary basis* introduced in Section 5.3, that is, we write

$$\bar{z} = \sum_{i=1}^{k} \lambda_i \bar{v}^i$$

where $\{\lambda_1,\ldots,\lambda_k\}$ are the distinct non-zero values of $\bar{z}$, and each $\bar{v}^i$ is the corresponding indicator vector for $\bar{z} = \lambda_i$. Without loss of generality we assume $\lambda_1 = 1$, so the fractional values of $\bar{z}$ are given by $\{\lambda_2,\ldots,\lambda_k\}$.

Lemma 4. *Let $(\bar{z},\bar{u})$ be an extreme point of* (46)*, and decompose $\bar{z}$ as above. Additionally, denote $\bar{A}$ the sub-matrix of A corresponding to binding constraints at $(\bar{z},\bar{u})$. Then $\bar{v}^i \in null(\bar{A})$ for all $2 \leq i \leq k$, and they are linearly independent.*

Proof. The $\bar{v}^i$ vectors are clearly linearly independent since they have disjoint support. As for the first claim, consider a precedence constraint $z_i \leq z_j$ $(i,j) \in I$. If such constraint is binding in $(\bar{z},\bar{u})$ then $\bar{z}_i = \bar{z}_j$, which implies

$$\bar{v}^l_i = 1 \iff \bar{v}^l_j = 1, \quad l = 1,\ldots,k$$

thus, $\bar{v}^l_i = \bar{v}^l_j \; \forall l$. On the other hand, if a constraint $z_i \leq 1$ or $z_i \geq 0$ is binding at $(\bar{z},\bar{u})$ then clearly $\bar{v}^l_j = 0$ for $l \geq 2$. This proves $\bar{A}\bar{v}^l = 0$ for $2 \leq l \leq k$. □

Lemma 5. *Let $(\bar{z},\bar{u})$ and q be as in Lemma 3. Additionally, let $\bar{A}$ be the sub-matrix of A corresponding to binding constraints at $(\bar{z},\bar{u})$. Then $\dim\big(null(\bar{A})\big) \leq q-m$.*

Proof. Let $\bar{H},\bar{G}$ be the set of rows of H and G corresponding to active constraints in $(\bar{z},\bar{u})$. Since this is an extreme point, the matrix

$$\begin{bmatrix} \bar{A} & 0 \\ \bar{H} & \bar{G} \end{bmatrix}$$

must contain at least $n+m$ linearly independent rows. Suppose $[\bar{H}\ \bar{G}]$ has $\bar{q}$ linearly independent rows. This implies $\bar{A}$ must have $n+m-\bar{q}$ linearly independent rows, and in virtue of the rank-nullity theorem,

$$\dim\big(\text{null}(\bar{A})\big) = n - \text{rank}(\bar{A}) = \bar{q} - m \leq q - m$$

□

Lemma 3 is then obtained as a direct corollary of Lemmas 4 and 5, since these two prove $k-1 \leq q-m$, and $k-1$ is exactly the number of fractional values of $\bar{z}$.

Chapter 3

Production Scheduling for Strategic Open Pit Mine Planning: A Mixed-Integer Programming Approach

Orlando Rivera Letelier, Daniel Espinoza, Marcos Goycoolea, Gonzalo Muñoz, Eduardo Moreno

Published in Operations Research. DOI: 10.1287/opre.2019.1965

II

Production scheduling for strategic open pit mine planning

A mixed-integer programming approach*

Orlando Rivera Letelier[1], Daniel Espinoza[2], Marcos Goycoolea[3], Eduardo Moreno[4], and Gonzalo Muñoz[5]

[1]*Doctoral Program in Industrial Engineering and Operations Research, Universidad Adolfo Ibáñez (orlando.rivera@uai.cl)*
[2]*Gurobi Optimization, (espinoza@gurobi.com*
[3]*School of Business, Universidad Adolfo Ibáñez, (marcos.goycoolea@uai.cl)*
[4]*Faculty of Engineering and Sciences, Universidad Adolfo Ibáñez, (eduardo.moreno@uai.cl)*
[5]*Department of Industrial Engineering and Operations Research, Columbia University, (gonzalo@ieor.columbia.edu)*

October 28, 2020

Given a discretized representation of an ore body known as a block model, the open-pit mining production scheduling problem that we consider consists of defining which blocks to extract, when to extract them, and how or whether to process them, in such a way as to comply with operational constraints, and maximize net present value. While it has been established that this problem can be modeled with mixed integer programming, the number of blocks used to represent real-world mines (millions) has made solving large instances nearly impossible in practice. In this article, we introduce a new methodology for tackling this problem and conduct computational tests using real problem sets ranging in size from 20,000 to 5,000,000 blocks and spanning 20 to 50 time periods. We consider both direct block scheduling and bench-phase scheduling problems, with capacity, blending, and minimum production constraints. Using new preprocessing and cutting planes techniques, we are able to reduce the LP relaxation value by up to 33%, depending on the instance. Then, using new heuristics, we are able to compute feasible solutions with an average gap of 1.52% relative the previously computed bound. Moreover, after four hours of running a customized branch-and-bound algorithm on the problems with larger gaps, we are able to further reduce the average from 1.52% to 0.71%.
Keywords: Column generation, Dantzig-Wolfe, Optimization, RCPSP

*This work was partially funded by the government of Chile through CONICYT grants FONDECYT 1151098 (M.G., O.R.), FONDECYT 1161064 (E.M.), Basal CMM U. Chile, ANILLO ACT 1407 (M.G., E.M., O.R.) and BCH 72130388 (G.M.).

1 Introduction

The central concern of strategic open-pit mine planning is the construction of a tentative production schedule. This is a life-of-mine plan (20-50 years) specifying which part of a mineral deposit should be extracted, as well when and how, so as to maximize the value of the mining project while satisfying operational and economic constraints. In the early stages of a mining endeavor, a strategic production schedule provides insight into the cash flows of a project (e.g., capacity investments and production goals), and, as such, it is critical to investors. Moreover, early strategic decisions in a mine planning project are binding for and critical to long-term profits. It is widely acknowledged that amongst all stages in a mining project, strategic mine planning has the most significant impact on the overall costs ([28]).

Strategic mine planning begins with the construction of a discretized mathematical representation of the ore body, known as a *block model*. This is typically a three-dimensional array of units of equal size, called *blocks*. The horizontal levels of this array are referred to as *benches*. To each block, a number of geological attributes are assigned based on data retrieved from exploration drill holes, including mineral and contaminant grades, rock types, and rock density.

A strategic mine plan, or production schedule, specifies which blocks are extracted, and also their time of extraction and their destination once they have been extracted. Destinations represent not only physical locations, but also block processing and treatment options. Examples of block destinations include mills, waste-dumps, leech pads and stockpiles.

Amongst others, production schedules must comply with capacity and spatial constraints. While the former limit the amount of material which can be mined and sent to each destination, the latter ensure the accessibility of blocks and the safety of mining operations. The most common spatial constraints consist of imposing minimal wall slope angles (to prevent cave-ins, or wall slope failures) and ensuring that extraction regions are wide and contiguous so as to accommodate ramps and roads for large equipment. Additional constraints are often imposed to address blending, stockpiling, and other concerns.

In practice, given the complexity of holistic approaches, the strategic open pit mine planning problem is sub-divided into two more manageable problems. The first sub-problem, which we term the Open Pit Phase Design Problem (OPPDP), involves sub-dividing the deposit into large contiguous spatial units known as *phases* or *pushbacks*. Phases define the stages in which the deposit is to be excavated, and they are designed such that profitable parts of the mine are quickly reached while maintaining operational feasibility. The second sub-problem, which we call the Open Pit Production Scheduling Problem (OPPSP), consists of computing a production schedule detailing when and how the blocks in each phase should be extracted. Phases are extracted bench-by-bench, beginning with the blocks in Phase 1. The extraction of blocks in Phase 2 should begin during or after the extraction of the blocks in Phase 1, and so on. For a more detailed description of this scheme, see [28].

Starting with the doctoral dissertation of [30], a majority of academic papers regarding mathematical programming methods for strategic open pit mine planning have focused on the OPPDP. In addition, most of them have considered a simplification of the OPPDP known as the C-PIT Problem, which considers predefined destinations for each block and simple classes of constraints. Even these simplified problems, however, have proved very challenging considering the scale of practical problem instances and the difficulty of solving even their LP relaxations.

The works of [7] and, shortly afterward, [3, 4] mark a significant departure from previous results in two regards. First, they introduce a mathematical programming formulation that generalizes both the OPPDP and OPPSP, with block-destination selectivity, block clustering, and arbitrary constraints. Second, they present an effective decomposition method for solving the LP relaxation for problems of realistic size.

In this article, we build on the work of Bienstock and Zuckerberg (BZ) by proposing a framework for solving the problem with integral variables. This framework includes preprocessing, cutting-plane, heuristic, and specialized branching techniques. A novel aspect of our work is that we discuss both the phase-design (OPPDP) and production scheduling (OPPSP) steps of the strategic mine planning problem, and propose problem-specific techniques for each. Another novel aspect is that we consider realistic, but more complicated, constraints in our computations, including blending, flow-balance, and minimum production. Extensive computational tests show that we can solve both the OPPDP and OPPSP to near-optimality in a very reasonable amount of time. To our knowledge, this is the first time this is achieved.

2 Formulating the phase design and production scheduling problems

In this section, we describe the Precedence-Constrained Project Scheduling Problem with Clusters (PCPSP-C), along with an integer programming formulation of it. In addition, we show how one can formulate both the OPPDP and OPPSP as a PCPSP-C and survey different methodologies proposed for solving this and similar problems.

2.1 Definitions and Notation

Let $\mathscr{B}$ represent the set of blocks of a discretized ore body and $\mathscr{D}$ the set of possible destinations to which a block can be sent. For each $b \in \mathscr{B}$ and $d \in \mathscr{D}$, let $p_{b,d}$ represent the estimated value obtained by sending b to d, and let $p_{b,*} = \max\{p_{b,d} : d \in \mathscr{D}\}$, bearing in mind that these values can be negative. Let $\mathscr{T} = \{1, \ldots, T\}$ represent the time periods in which decisions can be made. Let $p_{b,d,t}$ represent the value obtained from sending b to d within time period t. We assume that $p_{b,d,t}$ is obtained from $p_{b,d}$ by applying a discount factor $r > 0$ over time, that is, $p_{b,d,t} = p_{b,d}/(1+r)^t$.

Consider $b_1, b_2 \in \mathscr{B}$. Define b_1 to be a *precedence* of b_2 – formally, $b_1 \prec b_2$ – if b_1 must be extracted no later than b_2. Precedence relationships between blocks are imposed to ensure minimum wall-slope angles and thus avoid wall-slope failures or cave-ins. See [31] for a description of how one determines these precedence relationships in practice.

Given sets $B_1, B_2 \subseteq \mathscr{B}$, we write $B_1 \prec B_2$ if there exist $b_1 \in B_1$ and $b_2 \in B_2$ such that $b_1 \prec b_2$.

Given a set $P \subseteq \mathscr{B}$, we say that P is a *pit* in $\mathscr{B}$ if $b_1 \prec b_2 \wedge b_2 \in P \Rightarrow b_1 \in P$. Also, a sequence of pits $P_1, P_2, \ldots, P_n \subseteq \mathscr{B}$ is *nested* if $P_1 \subseteq P_2 \subseteq \ldots \subseteq P_n$.

Let $\mathscr{C}$ be a partition of the set of blocks $\mathscr{B}$. That is, the sets in $\mathscr{C}$ are pairwise disjoint, and they cover the entire set $\mathscr{B}$. We henceforth refer to the elements of $\mathscr{C}$ as *clusters*. Assume that $\prec$ induces a partial order over the elements of $\mathscr{C}$. Let $\mathscr{A} \subseteq \mathscr{C} \times \mathscr{C}$ be the arcs representing the directed acyclic graph induced by $\prec$ in $\mathscr{C}$. That is, $(c_2, c_1) \in \mathscr{A}$ if and only if $c_1 \prec c_2$.

Note that we chose the arc directions such that a cluster's *out-neighbors* are its precedents. For computational purposes, we assume $\mathscr{A}$ to be the transitive reduction of this arc set. In other words, $\mathscr{A}$ is composed only of *immediate precedence relationships*: if $(c_1, c_2) \in \mathscr{A}$ and $(c_2, c_3) \in \mathscr{A}$, then $(c_1, c_3) \notin \mathscr{A}$.

Given $c \in \mathscr{C}$, we define the *closure* of c, or $cl(c)$, and the *reverse closure* of c, or $rcl(c)$, as follows:

$$cl(c) = \{c\} \cup \{c' \in \mathscr{C} : c' \prec c\},$$
$$rcl(c) = \{c\} \cup \{c' \in \mathscr{C} : c \prec c'\}.$$

Although each $c \in \mathscr{C}$ is a *set*, we use lower-case letters to represent these elements (as opposed to the usual upper-case notation). This is mainly because we define variables using each $c \in \mathscr{C}$ as a subscript. For each $b \in \mathscr{B}$, let $c(b)$ represent the unique cluster $c \in \mathscr{C}$ such that $b \in c$. We use clusters to indicate that sets of blocks should be extracted simultaneously in a mine-planning solution. That is, all the blocks in a single cluster are extracted in exactly the same proportion in each time period. These clusters only enforce an extraction-related constraint, as the destination of each block can be different from the destination of other blocks in the same cluster.

For each block $b \in \mathscr{B}$, let q_b be its weight (or amount of material). For each $t \in \mathscr{T}$, let U_t represent the maximum amount of material that can be extracted in time period t, and let U_t^d represent the maximum amount of material that can be sent to destination $d \in \mathscr{D}$ in time period $t \in \mathscr{T}$.

Let variables $x_{c,t}$ indicate the proportion of cluster $c \in \mathscr{C}$ extracted in time $t \in \mathscr{T}$. For each $b \in \mathscr{B}$, $d \in \mathscr{D}$, and $t \in \mathscr{T}$, let the variable $y_{b,d,t}$ indicate the proportion of block b sent to destination d in time t. In (1), we describe a mixed-integer programming formulation proposed by [3] which generalizes both the OPPSP and OPPDP. Although these authors originally refer to this problem as the Parcel Assignment Problem (PAP), we henceforth refer to it as the Precedence Constrained Production Scheduling Problem with Clusters (PCPSP-C).

$$\max \sum_{b \in \mathscr{B}} \sum_{d \in \mathscr{D}} \sum_{t \in \mathscr{T}} p_{b,d,t} y_{b,d,t} \tag{1a}$$
$$\text{s.t. } x_{c,t} = \sum_{d \in \mathscr{D}} y_{b,d,t} \qquad \forall c \in \mathscr{C},\ b \in c,\ t \in \mathscr{T} \tag{1b}$$
$$\sum_{t \in \mathscr{T}} x_{c,t} \leq 1 \qquad \forall c \in \mathscr{C} \tag{1c}$$
$$\sum_{t'=1}^{t} x_{c,t'} \leq \sum_{t'=1}^{t} x_{c',t'} \qquad \forall (c, c') \in \mathscr{A},\ t \in \mathscr{T} \tag{1d}$$
$$Gy \leq g \tag{1e}$$
$$y_{b,d,t} \geq 0 \qquad \forall b \in \mathscr{B}, d \in \mathscr{D}, t \in \mathscr{T} \tag{1f}$$
$$x_{c,t} \text{ satisfies an integrality condition.} \qquad \forall c \in \mathscr{C}, t \in \mathscr{T} \tag{1g}$$

Here, the objective function (1a) maximizes the net present value of the solution. Constraints (1b) impose consistency between the x and y variables, ensuring that in each period, all blocks within a cluster are extracted and processed in the same proportion. Constraints (1c) impose that a cluster can be extracted at most once and (1d) impose the precedence relationships. Constraints

(1e) represent general operational requirements, (1f) bound the variables, and (1g) impose integrality conditions on the x variables. The exact nature of constraints (1e) and (1g) depends on specific considerations of the model, and we show some common choices below. Nevertheless, we should point out that, regardless of the application, constraints (1e) typically include conditions of the form:

$$\sum_{b\in\mathscr{B}}\sum_{d\in\mathscr{D}} q_b y_{b,d,t} \leq U_t \qquad \forall t \in \mathscr{T}, \tag{2a}$$

$$\sum_{b\in\mathscr{B}} q_b y_{b,d,t} \leq U_t^d \qquad \forall d \in \mathscr{D},\, t \in \mathscr{T}, \tag{2b}$$

where (2a) and (2b) impose mining and destination capacity limits, respectively.

We denote by LP-PCPSP-C the linear programming (LP) relaxation of the PCPSP-C, that is, the relaxation obtained from (1) by removing constraints (1g). Note that due to the general constraints (1e), the PCPSP-C, and even its LP relaxation, may be infeasible. However, if $G \geq 0$, then the null solution $(x,y) = (0,0)$ is feasible.

In this work, we consider two different integrality conditions (1g):

- The following conditions define *full integrality* constraints,

$$x_{c,t} \in \{0,1\} \qquad \forall c \in \mathscr{C}, t \in \mathscr{T}. \tag{3}$$

  These ensure that each block must be extracted entirely in a single time period.

- The following conditions define *partial integrality* constraints,

$$\sum_{t'=1}^{t} x_{c,t'} > 0 \text{ implies } \sum_{t'=1}^{t} x_{c',t'} = 1 \qquad \forall (c,c') \in \mathscr{A},\, \forall t \in \mathscr{T}. \tag{4}$$

  These constraints allow the duration of cluster extraction to span multiple time periods; however, they specify that a cluster must be entirely extracted before any of its successors can, in turn, be extracted.

Partial integrality conditions (4) are often imposed in commercial software packages, such as [23]), which, to our knowledge, were first identified in [24]. Note that by using either (3) or (4), the PCPSP-C has a bounded feasible set. Thus, there exists an optimal solution whenever the set is non-empty.

From this point onward, we refer to PCPSP-C whenever a technique is independent of the exact nature of (1g) (e.g., when the discussion concerns the LP relaxation); otherwise, we use PCPSP-F or PCPSP-P if the technique applies to the PCPSP-C under full integrality conditions (3) or partial integrality conditions (4), respectively. Similarly, the OPPDP and OPPSP can be formulated as special cases of the PCPSP-C.

2.2 The phase design problem (OPPDP)

The objective of solving an Open-Pit Phase Design Problem (OPPDP) is to compute a sequence of nested pits, which a mining engineer can subsequently use to define phases. The OPPDP

is the specific case of the PCPSP-F in which all the clusters are singletons (i.e., each cluster corresponds to a single block), and in which $G, g \geq 0$. Instances of the OPPDP with a single destination per block are sometimes known as Capacitated Pit Problems (C-PITs), as in [10], whereas those instances with multiple destinations per block are sometimes referred to as Open Pit Mine Production Scheduling Problems (OPMPSPs), as in [7], or PCPSPs, as in [3]. In general, instances of the OPPDP are often just referred to as *direct block scheduling* problems.

Observe that feasible solutions of the PCPSP-C always correspond to a sequence of nested pits, regardless of how the clusters are defined. In fact, for every $t \in \mathscr{T}$ and every feasible solution x of the PCPSP-C, the set $\mathscr{B}(x;t) = \{b \in \mathscr{B} : \sum_{t'=1}^{t} x_{c(b),t'} > 0\}$ defines a pit. Furthermore, the sequence $\mathscr{B}(x;1), \ldots, \mathscr{B}(x;2), \ldots, \mathscr{B}(x;T)$ is nested. By defining clusters as singletons, it is possible to generate the best possible sequence of nested pits using the OPPDP, in terms of net present value.

Traditionally, nested pits for phase design have heuristically been computed in the mining community using parameterized maximum closure algorithms. See [36], [27] and [28] for details.

Ideally, pits should be few in number, and the volumetric difference between consecutive pits should define wide contiguous areas, where one can easily accommodate ramps and where equipment (shovels, trucks) can work and move around easily; however, neither the pits generated by solving the OPPDP nor those obtained via parametric analysis necessarily meet these conditions. Moreover, it is not clear how one can modify these methodologies so that these conditions are met. In practice, this issue is addressed through the pits computed by the OPPDP, or by parametric analysis, as "guides" in a manual design process. This process begins with a sequence of pits $P_1, P_2, \ldots, P_n$ and, by using specialized computer-aided design (CAD) software systems such as [16] or [38], it constructs a new sequence $P'_1, P'_2, \ldots, P'_k$ such that the required operational conditions hold. These new pits, in turn, are used to compute a sequence of phases $\varphi_1, \varphi_2, \ldots, \varphi_n$, where $\varphi_1 = P'_1$ and $\varphi_j = P'_j \setminus P'_{j-1}$ for all $j > 1$.

Although ultimately, both the OPPDP and parametric analysis are heuristic mechanisms by which to generate nested pits, the use of the OPPDP makes it easier to control the size and number of pits. See [39] for a discussion of the limitations of parametric analysis.

2.3 The production scheduling problem (OPPSP)

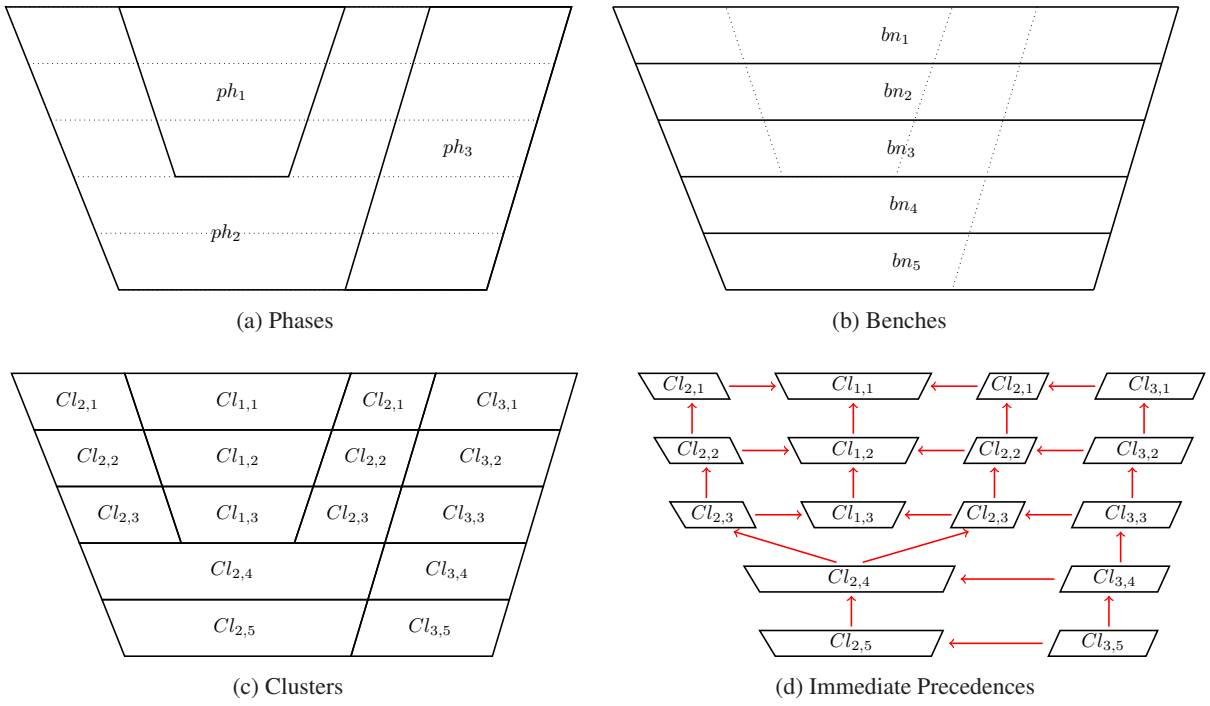
The Open-Pit Production Scheduling Problem (OPPSP) is a specific case of the PCPSP-P in which all clusters correspond to bench-phases, defined as the intersection of a bench and a phase. As mentioned above, each bench corresponds to a horizontal level of the block model, and each phase is the output of the phase design process described in Section 2.2. The OPPSP is very closely related to the Cutoff Grade Optimization Problem, originally proposed by [35]. See [3] for a discussion of how linear programming duality can be used to demonstrate the relationship between the OPPSP and cutoff grades.

As discussed in Section 2, given two clusters c_1, c_2, we have that $c_1 \prec c_2$ whenever there exist $b_1 \in c_1$ and $b_2 \in c_2$ such that $b_1 \prec b_2$. In practical applications, it is common to also use additional precedence relationships. For example, it is often required that clusters in the same bench be extracted in the order prescribed by the phases [e.g. 48]. This requirement is easy to

model in the OPPSP, as it is simply a matter of adding arcs to set $\mathscr{A}$.

Figure 1 illustrates how a particular phase design could determine the definition of clusters used and the precedence relationships between them. In this figure, we let ph_i be the i-th phase and bn_i be the i-th bench. For phase ph_i and bench bn_j, let $Cl_{i,j}$ be the corresponding cluster obtained from the intersection of ph_i and bn_j.

OPPSP formulations tend to be more detailed than OPPDP formulations. In addition to (2), inequalities (1e) can include other mining constraints, such as those related to flow-balance, material-blending, and stockpiles.



(a) Phases (b) Benches

(c) Clusters (d) Immediate Precedences

Figure 1: Example illustrating the definition of phases, benches, clusters and the precedences between clusters. Each arrow represents an immediate precedence relationship. The head of each arrow corresponds to the predecessor in the corresponding precedence relationship.

3 Previous methodological work and contributions of this article

A typical mine planning problem is composed of 1–5 million blocks, 20–50 time periods, and 2–3 destinations, which results in a huge number of decisions, leading to models that are impossible to solve with commercial integer-programming solvers; even solving the LP-PCPSP-C has proved elusive. To date, optimization approaches are varied and, to a large extent, work by

exploiting problem-specific characteristics, such as the type of problem being solved (C-PIT, PCPSP or PCPSP-C), and/or the characteristics of the matrix G, e.g., $G \geq 0$, or G general.

The earliest work attempting to solve PCPSP-C dates back to [30], who proposed using the decomposition of [15] for the LP relaxation of the PCPSP but without any methods for computing integer-feasible solutions; he only managed to solve instances with up to 36 blocks. After Johnson's work, most optimization research regarding the PCPSP-C has focused on studying C-PIT, i.e., the variant of the PCPSP-C in which each cluster is a single block with a single destination per block, full integrality conditions are assumed, and there are only positive (upper-bound) capacity constraints. Important milestones in this direction include:

- [13] and [14] study problems with up to $5,400$ blocks by solving the LP relaxation with Lagrangian relaxation. The authors argue that in practical mining problems, primal feasibility is not so important, and they propose using the solution of the Lagrangian pricing problem as the primal output. Even though this solution is integer-feasible, it can violate capacity constraints.

- [9] prove that a common preprocessing technique based on computing ultimate pit limits is correct for C-PIT. They also claim to solve problems with up to $209,664$ blocks to optimality with a customized branch-and-bound approach. Their methodology is not described, citing commercial interests.

- [22] addresses problems with up to $25,620$ blocks by combining Lagrangian relaxation and customized rounding heuristics, and by introducing the notion of "early starts" and "late starts" for preprocessing and LP bound strengthening.

- [12] and [33] handle problems with up to $25,620$ blocks by combining tailored Lagrangian relaxation and a sliding time window heuristic. Their results are the first to consider instances with lower-bound capacity constraints. Solutions within 4% of optimality are obtained.

- [19] make a collection of OPPDP instances available by means of the MineLib website. These instances range in size from $1,060$ to $2,140,342$ blocks.

- [10] develop a customized decomposition algorithm for solving the LP relaxation of instances with only a single capacity constraint per period. To obtain integer-feasible solutions, they propose a heuristic called *TopoSort*, and a local search heuristic for further refinement. Solutions within 2% of optimality are reported for instances with up to 4 million blocks. Using the algorithm for variants of the instances with two capacity constraints, solutions with a provable gap of 4% are computed.

- [49] develop a customized branch-and-bound method using Benders' Decomposition and customized heuristics. The approach considers lower bound constraints and a novel type of integrality. Problems with up to $25,620$ blocks are solved to 1% of optimality.

- To date, the best-known solutions for the C-PIT are those produced by three recent heuristics. [29] propose a heuristic combining aggregation and disaggregation, [37] propose a topological sorting heuristic which does not take as input an LP relaxation solution, and

[46] extend TopoSort by introducing a diving component guided by genetic algorithms. All of these papers improve the best gaps reported for MineLib instances.

Complementary works related to optimization methodologies for C-PIT are:

- [20], [5] and [18] study clique, cover and other inequalities on precedence-constrained knapsack problems and derive cuts and separation algorithms.
- A number of articles have proposed aggregating variables to obtain smaller problems which are directly solvable with commercial integer programming solvers. These methods vary depending on how clusters are defined. Important examples encompass the *fundamental-trees* of [43], the *mining-blocks* of [47], the *mining-cuts* of [1, 2], and the *clumps* of [21].
- A number of articles have studied stochastic variants of C-PIT. For reference, see the work of [6], [44], and [45].

The work of [7] represents a significant departure from earlier research, as they abandon the study of C-PIT and instead tackle the more general PCPSP (i.e., with multiple destinations and general classes of constraints). In their work, they develop a customized decomposition method that works by aggregating and disaggregating blocks. In computational experiments, they solve instances with up to $96,000$ blocks to 1% of optimality.

- [3, 4] generalize the formulation and decomposition method of [7] to the more general PCPSP-C (which they call the PAP). Though their formulation generalizes both the OPPDP and OPPSP, their computational experiments focus on instances of the former, discussing neither model in detail and studying only the LP relaxation of the problem.
- [41] build on the work of [3], extending its applicability to broader classes of scheduling problems, comparing the extension with other decomposition approaches, and introducing a number of computational enhancements. Similar to [3], this article only focuses on LP relaxations.
- [25, 26] discuss how solutions generated by the PCPSP-C formulation compare to those generated by traditional commercial software, both from modeling and computational viewpoints.
- [32] extend the PCPSP-C to model open-pit-to-under-ground transition problems, and [40] extend the PCPSP-C to model stockpiles.

Also, a different approach, based on network flows and aimed at a simultaneous, aggregated analysis of multiple mine sites is that of [17].

In this article, we extend many of the aforementioned methodologies, originally developed in the context of the C-PIT, to the context of the PCPSP-C. We emphasize how these methodologies can be used to solve both the OPPDP and OPPSP, each of which requires some problem-specific techniques. Overall, our extensions mostly focus on: the decisions which affect clusters of blocks rather than individual blocks, the multiplicity of destinations which a block can have,

and the possibility of admitting partial integrality. In addition, we propose a number of new methodologies. We are especially concerned with techniques that are useful for the OPPSP, which has received much less attention in academic literature. Our work introduces new classes of cutting planes and heuristics designed to reduce the large integrality gaps observed in OPPSP instances. We also implement a specialized branch-and-bound algorithm.

Finally, we show that all of these methodologies work well when integrated. Most, if not all, academic papers have studied these techniques individually, rather than in concert. This integration is key to showing, for the first time, that it is possible to solve to near optimality instances of both the OPPDP and OPPSP on real mine planning data sets, such as those handled by commercial mine planning software systems.

4 Preprocessing

As discussed above, PCPSP-C can be very large in terms of the number of variables and constraints. In this section, we present techniques for transforming this problem into a mathematically equivalent one which possesses fewer variables. In other words, we reduce the dimension of the problem without affecting the optimal objective function value.

4.1 Ultimate Pit Limit Preprocessing

In this subsection, we describe a preprocessing scheme that can be used to eliminate variables from an instance of PCPSP-C by identifying clusters that provably do not appear in an optimal solution of the problem. The scheme is a generalization of one proposed by [9] in the context of C-PIT, which we extend to general instances of PCPSP-C, in which $G, g \geq 0$.That is, the generalization works for both partial and full integrality, and also for instances with clusters which are not necessarily singleton blocks.

In general, given an optimization problem defined over a region $S \subseteq \mathbb{R}^n$, we say that an optimal solution $x \in S$ is *minimal* if every other optimal solution $x' \in S$ is such that $x' \leq x$ implies $x' = x$.

For each $c \in \mathscr{C}$, define $\bar{p}_c = \sum_{b \in c} p_{b,*}$, where $p_{b,*} = \max_{d \in \mathscr{D}} p_{b,d}$ (as defined in Section 2). We define the Ultimate Pit Limit Problem (U-PIT) of a PCPSP-C instance (see formulation (1)) as

$$\max \sum_{c \in \mathscr{C}} \bar{p}_c x_c \tag{5a}$$

$$\text{s.t. } x_c \leq x_{c'} \qquad \forall (c, c') \in \mathscr{A}, \tag{5b}$$

$$x_c \in [0,1] \qquad \forall c \in \mathscr{C}. \tag{5c}$$

Note that because this problem does not consider limited resources, there is no need to define variables over multiple time periods.

U-PIT is a Maximum Closure Problem. As such, (a) its formulation (5) is totally unimodular, and hence, its optimal basic solutions are binary; (b) it admits a unique minimal optimal solution; and (c) it can be solved efficiently. For more information about the Maximum Closure Problem, see [27].

Given a vector $x \in [0,1]^{\mathscr{C}}$ that is feasible for U-PIT, we say that $P = \{c \in \mathscr{C} : x_c > 0\}$ is the *pit-limit* associated with x. Let (x^*, y^*) represent a minimal optimal solution to PCPSP-C. For

each $c \in \mathscr{C}$, let $x^1_c = \sum_{t \in \mathscr{T}} x^*_{c,t}$. Since x^* satisfies constraints (1d), it follows that x^1 is feasible for U-PIT. Let us define P^{OPT} as the pit limit associated with x^1. Slightly abusing notation, let P^{OPT} be the pit limit associated with the minimal solution (x^*, y^*).

The following theorem generalizes a result of [9]. The proof is presented in the e-companion of this article.

Theorem 1. *Consider an instance of PCPSP-C, as described in formulation* (1)*, such that $G \geq 0$ and $g \geq 0$. Let P^{OPT} and $P^{U\text{-}PIT}$ be the pit limits of the minimal optimal solutions to PCPSP-C and U-PIT, respectively. Then, $P^{OPT} \subseteq P^{U\text{-}PIT}$.*

Theorem 1 allows us to preprocess as follows when $G \geq 0$ and $g \geq 0$. For each $c \in \mathscr{C}$, compute $\bar{p}_c$ and solve U-PIT, as described in (5). We can eliminate all variables associated with clusters $c \notin P^{\text{U-PIT}}$ from the problem, as we know that there exists an optimal solution to PCPSP-C which does not use them. Note that the proof of Theorem 1 holds for both integrality conditions (3) and (4). Because of this $P^{\text{U-PIT}}$ is typically referred to as the *ultimate pit limit* of PCPSP-C.

4.2 Dominated Triplet Elimination Preprocessing

In nearly all strategic mine planning problems, most blocks are typically "waste" blocks, i.e., blocks whose grade is so low that the profit of selling the recovered metal does not cover the processing costs. In most, though not all, problems, the option of sending such blocks to the processor can be removed from the model (by eliminating a destination of these blocks), thus making the problem smaller. Below we present a new, yet simple, preprocessing algorithm that can be used to detect and eliminate destinations which are never used in an optimal solution for cases such as these.

For each triplet $(b,d,t) \in \mathscr{B} \times \mathscr{D} \times \mathscr{T}$, let $G_{b,d,t}$ be the column of matrix G corresponding to variable $y_{b,d,t}$. We say that triplet (b,d_1,t) dominates triplet (b,d_2,t) if $G_{b,d_1,t} \leq G_{b,d_2,t}$ (component-wise) and $p_{b,d_1,t} \geq p_{b,d_2,t}$. Intuitively, (b,d_1,t) dominates (b,d_2,t) if the proportion of block b sent to destination d_1 in time t consumes the same or fewer resources and provides at least the same value than sending b to d_2 in t. If this is the case, then there is no need to actually define variable $y_{b,d_2,t}$, as we may assume that there exists an optimal solution such that $y_{b,d_2,t} = 0$.

4.3 Aggregation Preprocessing

In many instances of PCPSP-C, blocks in close proximity to each other are identical. In such cases, it is possible to group these blocks to reduce the number of variables. We describe this new preprocessing idea below.

Consider an instance of PCPSP-C and two blocks b_1, b_2 within the same cluster such that for some $\lambda \geq 0$, the following holds:

$$p_{b_1,d} = \lambda p_{b_2,d} \qquad \forall d \in \mathscr{D}, \tag{6}$$

$$G_{b_1,d,t} = \lambda G_{b_2,d,t} \qquad \forall d \in \mathscr{D},\ t \in \mathscr{T}. \tag{7}$$

In such a case, we can aggregate the two blocks into one block $\bar{b}$ with $G_{\bar{b},d,t} = G_{b_1,d,t} + G_{b_2,d,t}$, and $p_{\bar{b},d,t} = p_{b_1,d,t} + p_{b_2,d,t}$. The validity of this preprocessing comes from the fact that given

any feasible solution (x^*, y^*) to the PCPSP-C satisfying (6) and (7), there exists another solution (x, y) to the new aggregated problem with the same objective value as (x^*, y^*). We can take $y_{\bar{b},d,t} = \frac{\lambda}{1+\lambda} y^*_{b_1,d,t} + \frac{1}{1+\lambda} y^*_{b_2,d,t}$ for each $d \in \mathscr{D}, t \in \mathscr{T}$, take $y_{b,d,t} = y^*_{b,d,t}$ for each $b \in \mathscr{B} \setminus \{b_1, b_2\}, d \in \mathscr{D}, t \in \mathscr{T}$, and take $x = x^*$. We can observe that $p_{\bar{b},d,t} y_{\bar{b},d} = p_{b_1,d,t} y_{b_1,d} + p_{b_2,d,t} y_{b_2,d}$ and $G_{\bar{b},d,t} y_{\bar{b},d,t} = G_{b_1,d,t} y_{b_1,d,t} + G_{b_2,d,t} y_{b_2,d,t}$ for each $d \in \mathscr{D}, t \in \mathscr{T}$.

Note that this scheme is not affected by precedence relationships given that only blocks within the same cluster are aggregated and precedences in PCPSP-C are only defined between clusters.

5 Cutting planes

[4] claim that formulation (1) of PCPSP-C empirically presents low integrality gaps. In Section 8, computational experiments show that this is largely true for OPPDP. However, we do not observe the same behavior in OPPSP. To obtain low integrality gaps, we require problem-specific cutting planes. In this section, we first extend clique and early-start inequalities from the context of C-PIT to PCPCP-C. The first family of cuts that we obtain ignores the destination of each block, and we refer to them as *extraction cuts*. We also derive two new classes of cuts which exploit the problem structure resulting from block destination selectivity. We call these *production cuts*.

In what follows, we use a new set of variables describing the accumulated value of x variables:

$$w_{c,t} = \sum_{t'=1}^{t} x_{c,t'} \qquad \forall d \in \mathscr{D}, t \in \mathscr{T}. \tag{8}$$

The value of $w_{c,t}$ in a feasible solution is the fraction of cluster c extracted in time period t or earlier. Observe that the precedence constraints (1d) and mining capacity constraints (2a) on the x variables imply that

$$w_{c,t} \leq w_{c,t+1}, \qquad \forall c \in \mathscr{C},\ \forall t \in 1, \ldots, T-1, \tag{9a}$$
$$w_{c,t} \leq w_{c',t}, \qquad \forall (c, c') \in \mathscr{A},\ t \in \mathscr{T}, \tag{9b}$$
$$\sum_{c \in \mathscr{C}} q_c w_{c,t} \leq \sum_{t'=1}^{t} U_{t'}, \qquad \forall t \in \mathscr{T}. \tag{9c}$$

where $q_c = \sum_{b \in c} q_b$ is the total weight of the cluster. The full integrality conditions (3) and partial integrality conditions (4) translate, respectively, as follows to the w variables:

$$w_{c,t} \in \{0, 1\}, \qquad \forall c \in \mathscr{C},\ \forall t \in \mathscr{T}, \tag{10a}$$
$$w_{c,t} > 0 \text{ implies } w_{c',t} = 1, \qquad \forall (c, c') \in \mathscr{A},\ \forall t \in \mathscr{T}. \tag{10b}$$

All cuts are presented using the $w_{c,t}$ variables for simplicity; nonetheless, it is evident that they can be expressed using only variables $x_{c,t}$. Thus, slightly abusing notation, we discuss these *valid inequalities* for PCPSP-C using variables $w_{c,t}$.

For the reader's sake, let us recall relevant definitions and assumptions. We assume $\mathscr{C}$ to be such that $\prec$ defines a partial order, and, therefore, $\mathscr{A}$ defines a directed acyclic graph. For each cluster $c \in \mathscr{C}$, we define the *closure* of c as $cl(c) = \{c\} \cup \{c' \in \mathscr{C} : c' \prec c\}$ and the *reverse closure*

of c as $rcl(c) = \{c\} \cup \{c' \in \mathscr{C} : c \prec c'\}$. For a block $b \in \mathscr{B}$, we denote by $c(b)$ the unique cluster containing it. Additionally, for a set of clusters $C \subseteq \mathscr{C}$, we denote by $b(C) := \{b \in \mathscr{B} : b \in c, c \in C\}$ the set of blocks on C. Given $S \subseteq \mathscr{C}$ and a vector $q \in \mathbb{R}^{\mathscr{B}}$, we define $q(S) = \sum_{c \in S} q_c = \sum_{c \in S} \sum_{b \in c} q_b$.

5.1 Extraction cuts

We next describe a class of cuts which exploits extraction capacity limits (2a). Since (9a)-(9b) are precedence constraints and (9c) is a knapsack constraint (i.e., non-negative coefficients and a positive right-hand side), (9) is a precedence-constrained knapsack relaxation of PCPSP-C.

For a discussion of cuts derived from the precedence-constrained knapsack problem, in addition to separation techniques, see [5] and [18]. The precedence-constrained knapsack cuts that we analyze differ from those treated traditionally in academic literature in two aspects. First, the time indexing of the variables gives these relaxations a nested structure that can be exploited. Second, we not only consider the full integrality constraint (10a), but also the partial integrality constraint (10b).

We now move on to describing three families of extraction cuts: early start cuts, diamond cuts, and clique and lifted clique cuts.

5.1.1 Early Start Cuts.

As noted by [22], [34] and others, one can use early start cuts to eliminate a number of variables from PCPSP. We extended this preprocessing technique to the more general PCPSP-C, including the case of partial integrality condition (4). This can reduce the size of the problem, possibly speeding up computations, and it can also improve the LP relaxation bound of the problem.

Theorem 2. *Let $t \in \mathscr{T}$ and $Q_t = \sum_{t'=1}^{t} U_{t'}$, and consider a cluster $c \in \mathscr{C}$ such that $q(cl(c)) > Q_t$. Then,*

- *equality $w_{c,t} = 0$ is valid for the PCPSP-F*
- *whenever $q_c > 0$, inequality $w_{c,t} \leq \frac{(Q_t - q(cl(c)\setminus\{c\}))^+}{q_c}$ is valid for PCPSP-P.*

Proof.
The result for the PCPSP-F is known; thus, we only prove this result for the PCPSP-P. We use integrality condition (10b) obtained from (4). If $w_{c,t} > 0$, then $w_{c',t} = 1$ for all $c' \in cl(c) \setminus \{c\}$. From this and inequality (9c), it follows that $q_c w_{c,t} \leq Q_t - q(cl(c) \setminus \{c\})$. If $Q_t \leq q(cl(c) \setminus \{c\})$, we obtain a contradiction with $w_{c,t} > 0$. Hence, $Q_t \leq q(cl(c) \setminus \{c\})$ implies $w_{c,t} = 0$. From these conditions, we conclude that $w_{c,t} \leq \frac{(Q_t - q(cl(c)\setminus\{c\}))^+}{q_c}$ is valid for PCPSP-P, where $(Q_t - q(cl(c) \setminus \{c\}))^+ = \max\{0, (Q_t - q(cl(c) \setminus \{c\}))\}$.

Note that early start cuts can also be used as a preprocessing technique, eliminating variables in PCPSP-F or PCPSP-P when the cut sets them to 0.

5.1.2 Diamond Cuts.

The following inequalities are similar to those used by [50] in the context of multimode resource-constrained project problems. These cuts use the set $cl(c_2) \cap rcl(c_1)$ for $c_1, c_2 \in \mathscr{C}$ such that $c_1 \prec c_2$, which intuitively can be pictured as the intersection of two cones facing each other.

Theorem 3. *Let $t_1, t_2 \in \mathscr{T}$, $t_1 \leq t_2$, and $Q_{t_1,t_2} = \sum_{t'=t_1}^{t_2} U_{t'}$, and consider clusters $c_1, c_2 \in \mathscr{C}$ such that $c_1 \prec c_2$. Then,*

- *whenever $q(cl(c_2) \cap rcl(c_1)) > Q_{t_1,t_2}$, inequality $w_{c_2,t_2} \leq w_{c_1,t_1}$ is valid for PCPSP-F.*
- *whenever $q(cl(c_2) \cap rcl(c_1) \setminus \{c_1, c_2\}) > Q_{t_1,t_2}$, inequality $w_{c_2,t_2} \leq w_{c_1,t_1}$ is valid for the PCPSP-P.*

Proof.
First, consider full integrality condition (10a) obtained from (3), and as a contradiction, suppose that $w_{c_2,t_2} > w_{c_1,t_1}$. This implies that $w_{c_2,t_2} = 1$ and $w_{c_1,t_1} = 0$. Consequently, all clusters in $cl(c_2) \cap rcl(c_1)$ are extracted in time periods t_1 through t_2. Nevertheless, this is impossible, as $q(cl(c_2) \cap rcl(c_1)) > Q_{t_1,t_2}$.

Now, assume partial integrality condition (10b) obtained from (4) and that $w_{c_2,t_2} > w_{c_1,t_1}$. Consequently, $w_{c_1,t_1} < 1$ and $w_{c_2,t_2} > 0$. These conditions imply that every cluster in $cl(c_2) \cap rcl(c_1) \setminus \{c_1, c_2\}$ is fully extracted in time periods t_1 through t_2. Nevertheless, this is impossible, as $q(cl(c_2) \cap rcl(c_1) \setminus \{c_1, c_2\}) > Q_{t_1,t_2}$.

5.1.3 Clique and Lifted Clique Cuts.

Clique and lifted clique inequalities have been studied for precedence-constrained knapsack problems since [8]. We now introduce a natural generalization for the context of the PCPSP-C which can be used with partial integrality.

Consider two clusters $c_1, c_2 \in \mathscr{C}$ such that neither $c_1 \prec c_2$ nor $c_2 \prec c_1$. We define the *incompatibility* of c_1 and c_2 as follows:

- We say that c_1 and c_2 are *f-incompatible* in time t if $q(cl(\{c_1, c_2\})) > Q_t$.
- We say that c_1 and c_2 are *p-incompatible* in time t if $q(cl(\{c_1, c_2\}) \setminus \{c_1, c_2\}) > Q_t$.

Intuitively, incompatible clusters are such that neither can be extracted before a certain time period. Using this information, we obtain the following result:

Theorem 4. *Let $\{c_1, c_2, \ldots, c_k\}$ be a set of clusters, and consider the following inequality:*

$$\sum_{i=1}^{k} w_{c_i,t} \leq 1. \tag{11}$$

Then,

- *if $\{c_1, c_2, \ldots, c_k\}$ are pairwise f-incompatible, then inequality* (11) *is valid for PCPSP-F.*

- *if* $\{c_1, c_2, \ldots, c_k\}$ *are pairwise p-incompatible, then inequality* (11) *is valid for PCPSP-P.*

In addition, if a cluster $c \in \mathscr{C}$ *is such that* $c \prec c_i$ *for* $i = 1, \ldots, k$*, then constraint* $\sum_{i=1}^{k} w_{c_i,t} \leq w_{c,t}$ *is also valid for PCPSP-F (PCPSP-P) if* $\{c_1, c_2, \ldots, c_k\}$ *are pairwise f-incompatible (p-incompatible).*

Proof.

To prove that (11) is valid, suppose $\sum_{i=1}^{k} w_{c_i,t} > 1$. This assumption implies that there exist $i_1, i_2 \in 1, \ldots, k$ such that $w_{i_1,t} > 0$ and $w_{i_2,t} > 0$.

Assume that $\{c_1, c_2, \ldots, c_k\}$ are pairwise f-incompatible. Using integrality condition (10a), we would have that $w_{i_1,t} = w_{i_2,t} = 1$. This would imply that all clusters in $cl(\{c_{i_1}, c_{i_2}\})$ were extracted at time t or earlier, which would, in turn, contradict $q(cl(\{c_{i_1}, c_{i_2}\})) > Q_t$. This proves the result for the PCPSP-F.

Now, assume that $\{c_1, c_2, \ldots, c_k\}$ are pairwise p-incompatible. Using integrality condition (10b), $w_{i_1,t} > 0$ and $w_{i_2,t} > 0$ would imply that all clusters in $cl(\{c_{i_1}, c_{i_2}\}) \setminus \{c_{i_1}, c_{i_2}\}$ were extracted at time t or earlier, which would, in turn, contradict $q(cl(\{c_{i_1}, c_{i_2}\}) \setminus \{c_{i_1}, c_{i_2}\}) > Q_t$. From here, we conclude the result for PCPSP-P.

5.2 Production cuts

In this subsection, we describe new classes of cuts for the PCPSP-C, which we term *production cuts* because they exploit the capacity limits of each destination (2b) that are typically used to impose limits on production.

5.2.1 Production cuts in a simplified case.

For the reader's sake, in this subsection, we present the *production cuts* in a simplified version of the problem which considers two destinations: blocks are either processed (P) or wasted (W). We assume no clusters (or equivalently, single-block clusters) and no time index (i.e., $T = 1$). Consider an acyclic directed graph $G = (\mathscr{B}, \mathscr{A})$, a vector of positive coefficients $q \in \mathbb{R}_+^{\mathscr{B}}$ and a capacity $U > 0$. We define the *mode-knapsack mixed-integer set* (MK) as follows. A vector $(x, y^P, y^W) \in \mathbb{R}^{\mathscr{B}} \times \mathbb{R}^{\mathscr{B}} \times \mathbb{R}^{\mathscr{B}}$ belongs to the set MK if and only if the following conditions are met:

$$\sum_{b \in \mathscr{B}} q_b y_b^P \leq U, \tag{12}$$

$$y_b^P + y_b^W = x_b \qquad \forall b \in \mathscr{B}, \tag{13}$$

$$x_b \leq 1 \qquad \forall b \in \mathscr{B}, \tag{14}$$

$$x_b > 0 \implies x_{b'} = 1 \qquad \forall (b, b') \in \mathscr{A}, \tag{15}$$

$$y_b^P, y_b^W \geq 0 \qquad \forall b \in \mathscr{B}. \tag{16}$$

We can derive the following cuts for MK.

Simplified Variable-Right-Hand Side (VRHS) Cuts.

This family of cuts combines the precedence relationships in the extraction of blocks and the production limit (12). In other words, they ensure that if a solution sends a large number of blocks to be processed, then all predecessors of those blocks must be fully extracted as well.

Theorem 5. *Let $b \in \mathscr{B}$ be such that $q_b \leq U$. Then, the following inequality is valid for MK:*

$$\sum_{b' \in rcl(b)} q_{b'} y^P_{b'} \leq U x_b.$$

Proof. Proof. If $x_b = 1$, the inequality holds because of (12). If $x_b < 1$, then $x_{b'} = 0$ for each $b' \in rcl(b) \setminus \{b\}$, and then, the inequality holds because $y^P_b \leq x_b$ and $q_b \leq U$. □

Note that although we assume that $q_b \leq U$, we can easily relax this assumption by changing the coefficient of the variable on the right-hand side.

Simplified Hourglass Cuts.

Contrary to the VRHS cuts, the *hourglass* cuts derive valid inequalities involving W, the *waste* destination in our interpretation. In other words, we exploit the fact that for a block to be extracted, it is necessary to send blocks comprising an aggregate minimum weight to destination W.

Theorem 6. *Let $b \in \mathscr{B}$ and $S \subseteq cl(b) \setminus \{b\}$ be such that $q(S) \geq U$. Then, the following inequality is valid for MK:*

$$(q(S) + q_b - U)x_b \leq \sum_{b' \in S \cup \{b\}} q_{b'} y^W_{b'}.$$

Proof. Proof. The inequality is clearly valid when $x_b = 0$. Suppose that $x_b > 0$; then, for each $b' \in S$, we have $x_{b'} = y^P_{b'} + y^W_{b'} = 1$. Hence,

$$\begin{aligned}
\sum_{b' \in S \cup \{b\}} q_{b'} y^P_{b'} \leq U &\implies U - \sum_{b' \in S} q_{b'} y^P_{b'} \geq q_b y^P_b, \\
&\implies U - \sum_{b' \in S} q_{b'} (1 - y^W_{b'}) \geq q_b y^P_b, \\
&\implies \sum_{b' \in S} q_{b'} y^W_{b'} \geq q_b y^P_b + q(S) - U, \\
&\implies \sum_{b' \in S \cup \{b\}} q_{b'} y^W_{b'} \geq q_b (y^P_b + y^W_b) + q(S) - U, \\
&\implies \sum_{b' \in S \cup \{b\}} q_{b'} y^W_{b'} \geq q_b x_b + q(S) - U \geq (q_b + q(S) - U)x_b.
\end{aligned}$$

In the above, the last inequality holds because $q(S) \geq U$ and $x_b \leq 1$. Therefore, whenever $x_b > 0$, we have:

$$(q(S) + q_b - U)x_b \leq \sum_{b' \in S \cup \{b\}} q_{b'} y^W_{b'}.$$

□

These cuts can be lifted using the extraction variables for the nodes in $rcl(b)$ to obtain the following valid inequality for MK:

$$(q(S)+q_b-U)x_b+\sum_{b'\in rcl(b)\setminus\{b\}} q_{b'}y^P_{b'} \leq \sum_{b'\in S\cup\{b\}} q_{b'}y^W_{b'}.$$

The proof of the validity of this inequality for MK is identical to the previous one. Because this last expression uses variables for blocks in $cl(b)$ and in $rcl(b)$, we call this family *hourglass cuts*.

Now, we introduce both families in the general PCPSP-C context. The intuition is the same; however, additional analysis is necessary to include clusters, arbitrary numbers of destinations and multiple time periods.

5.2.2 Variable-Right-Hand Side (VRHS) Cuts.

First, we observe that the following conditions are valid for PCPSP-C for both full and partial integrality conditions:

$$w_{c,t} < 1 \Rightarrow y_{b,d,t} = 0, \quad \forall c \in \mathscr{C}, \forall b \in \{b \in \mathscr{B} : c(b) \in rcl(c) \setminus \{c\}\}, \forall d \in \mathscr{D}, \forall t \in \mathscr{T}, \quad (17a)$$

$$y_{b,d,t} \leq w_{c,t}, \qquad \forall c \in \mathscr{C}, \forall b \in c, \forall d \in \mathscr{D}, \forall t \in \mathscr{T}, \quad (17b)$$

$$\sum_{b\in\mathscr{B}} q_b y_{b,d,t} \leq U^d_t, \qquad \forall d \in \mathscr{D}, \forall t \in \mathscr{T}. \quad (17c)$$

with $w_{c,t}$ defined in (8). These can be used to derive a new family of valid inequalities, generalizing the cuts discussed above.

Theorem 7. *Consider a destination $d \in \mathscr{D}$, a time period $t \in \mathscr{T}$ and a family of clusters $c_1, c_2, \ldots, c_n \in \mathscr{C}$ such that the following conditions hold:*

1. $c_1 \prec c_2 \prec \ldots \prec c_n$,
2. $q(rcl(c_1) \setminus rcl(c_n)) < U^d_t$,
3. $q(rcl(c_1)) > U^d_t$.

Additionally, define $\Delta_k = rcl(c_k) \setminus rcl(c_{k+1})$ and $\delta_k = q(\Delta_k)$ for $k = 1, \ldots, n-1$, and $\Delta_n = rcl(c_n)$ and $\delta_n = U^d_t - q(rcl(c_1) \setminus rcl(c_n))$. Consider the inequality

$$\sum_{k=1}^{n-1}\left(\sum_{c\in\Delta_k}\sum_{b\in c} q_b y_{b,d,t}\right) + \sum_{b\in c_n} \alpha_b q_b y_{b,d,t} + \sum_{c\in rcl(c_n)\setminus\{c_n\}}\sum_{b\in c} q_b y_{b,d,t} \leq \sum_{k=1}^{n} \delta_k w_{c_k,t} \quad (18)$$

for some constants $0 \leq \alpha_b \leq 1$, $b \in c_n$. Then, (18) is valid for PCPSP-F. Additionally, if constants α_b satisfy

$$\sum_{b\in c_n} \alpha_b q_b \leq U^d_t - q(rcl(c_1) \setminus rcl(c_n)), \quad (19)$$

then (18) is valid for PCPSP-P.

Proof. Proof. The proof is divided in two cases: (i) the case in which $\forall k \in \{1,\dots,n\}\ w_{c_k,t} = 1$, and (ii) the case in which $\exists k \in \{1,\dots,n\}\ w_{c_k,t} < 1$.

Case 1: $w_{c_k,t} = 1$ for $k = 1,\dots,n$. Note that $\{\Delta_1,\dots,\Delta_{n-1}\}$ is a partition of $rcl(c_1) \setminus rcl(c_n)$. Thus, $\sum_{k=1}^{n-1} \delta_k = q(rcl(c_1) \setminus rcl(c_n))$, which implies $\sum_{k=1}^{n} \delta_k = U_t^d$. Then

$$\begin{aligned}
&\sum_{k=1}^{n-1}\left(\sum_{c\in\Delta_k}\sum_{b\in c} q_b y_{b,d,t}\right) + \sum_{b\in c_n} \alpha_b q_b y_{b,d,t} + \sum_{c\in rcl(c_n)\setminus\{c_n\}}\sum_{b\in c} q_b y_{b,d,t} \\
&\leq \sum_{k=1}^{n-1}\left(\sum_{c\in\Delta_k}\sum_{b\in c} q_b y_{b,d,t}\right) + \sum_{b\in rcl(c_n)} q_b y_{b,d,t} \\
&\leq \sum_{b\in\mathscr{B}} q_b y_{b,d,t} \leq U_t^d = \sum_{k=1}^{n} \delta_k = \sum_{k=1}^{n} \delta_k w_{c_k,t},
\end{aligned}$$

where the first inequality comes from the fact that $\alpha_b \leq 1$, and the second inequality ensues, as each block is being counted at most once on the left-hand side. Thus, in this case, (18) is valid.

Case 2: Suppose now that $\exists k$ such that $w_{c_k,t} < 1$, and let $k_o = \min\{k \in 1,\dots,n : w_{c_k,t} < 1\}$. Observe that $w_{c_k,t} = y_{b,d,t} = 0$ for $k > k_o$ and $b \in b(\Delta_k)$. Additionally, since this case implies $w_{c_n,t} < 1$, for each $c \in rcl(c_n) \setminus \{c_n\}$, we have $w_{c,t} = 0$. This makes inequality (18) equivalent to

$$\sum_{k=1}^{n-1}\left(\sum_{c\in\Delta_k}\sum_{b\in c} q_b y_{b,d,t}\right) + \sum_{b\in c_n} \alpha_b q_b y_{b,d,t} \leq \sum_{k=1}^{n} \delta_k w_{c_k,t}. \tag{20}$$

We now analyze two sub-cases regarding k_o:

- If $k_o < n$, since $c_n \in \Delta_n$ and $rcl(c_n) \setminus \{c_n\} \subseteq \Delta_n$, the second term on the left-hand side of inequality (20) is 0. Moreover, this inequality simplifies to

  $$\sum_{k=1}^{k_o}\left(\sum_{c\in\Delta_k}\sum_{b\in c} q_b y_{b,d,t}\right) \leq \sum_{k=1}^{k_o} \delta_k w_{c_k,t}.$$

  This inequality is valid since

  $$\sum_{c\in\Delta_k}\sum_{b\in c} q_b y_{b,d,t} \leq \sum_{c\in\Delta_k}\sum_{b\in c} q_b w_{c,t} \leq \left(\sum_{c\in\Delta_k}\sum_{b\in c} q_b\right) w_{c_k,t} = \delta_k w_{c_k,t} \qquad \forall k \in \{1,\dots,n-1\}. \tag{21}$$

- If $k_o = n$ and we use full integrality conditions, then $w_{c_n,t} = y_{b,d,t} = 0$ for all $b \in c_n$. Thus, the validity of (20) for the PCPSP-F follows from (21).
  On the other hand, if $k_o = n$ and we use the partial integrality condition, from (17a) and (19), we obtain

  $$\sum_{b\in c_n} \alpha_b q_b y_{b,d,t} \leq \left(\sum_{b\in c_n} \alpha_b q_b\right) w_{c_n,t} \leq (U_t^d - q(rcl(c_1) \setminus rcl(c_n)))w_{c_n,t} = \delta_n w_{c_n,t}. \tag{22}$$

  Thus, the validity of (20) for the PCPSP-P derives from (21) and (22).

□

5.2.3 The Hourglass Cuts

We now generalize the hourglass cuts for the PCPSP-C. These are a variant of the VRHS inequalities described above.

In its simplified form, each hourglass cut considers a cluster $\bar{c}$, a set of blocks S contained in $b(cl(\bar{c}))$, and the set of blocks in $b(rcl(\bar{c}))$, as the name of the inequality suggests.

Theorem 8. *Let $1 \leq t_1 \leq t \leq T$, $\bar{c} \in \mathscr{C}$, and $S \subseteq b(cl(\bar{c}) \setminus \{\bar{c}\})$ be such that*

$$q(S) \geq \sum_{s=t_1}^{t} U_s^d.$$

Let $R = rcl(\bar{c}) \setminus \{\bar{c}\}$. Then, the inequality

$$\left(q(S \cup \{\bar{c}\}) - \sum_{s=t_1}^{t} U_s^d\right) w_{\bar{c},t} + \sum_{b \in b(R)} q_b \sum_{s=t_1}^{t} y_{b,d,s} \leq \sum_{b \in S \cup \{\bar{c}\}} q_b \left(w_{c(b),t} - \sum_{s=t_1}^{t} y_{b,d,s}\right) \tag{23}$$

is valid for PCPSP-C.

Proof. Proof. If $w_{\bar{c},t} = 0$, then the left-hand side is equal to 0, and the inequality holds true. Henceforth, assume $w_{\bar{c},t} > 0$. Then, for each $b \in S$, $w_{c(b),t} = 1$. We know that

$$\sum_{b \in S} q_b y_{b,d,s} + \sum_{b \in \bar{c}} q_b y_{b,d,s} + \sum_{b \in b(R)} q_b y_{b,d,s} \leq U_s^d \quad \forall s \leq t$$

Combining these inequalities, we obtain:

$$\sum_{b \in S} q_b \sum_{s=t_1}^{t} y_{b,d,s} + \sum_{b \in \bar{c}} q_b \sum_{s=t_1}^{t} y_{b,d,s} + \sum_{b \in b(R)} q_b \sum_{s=t_1}^{t} y_{b,d,s} \leq \sum_{s=t_1}^{t} U_s^d$$

Moving the left-most term to the right-hand side, and adding $\sum_{b \in S} q_b w_{c(b),t}$ on both sides yields:

$$\sum_{b \in S} q_b \underbrace{w_{c(b),t}}_{1} + \sum_{b \in \bar{c}} q_b \sum_{s=t_1}^{t} y_{b,d,s} + \sum_{b \in b(R)} q_b \sum_{s=t_1}^{t} y_{b,d,s} \leq \sum_{s=t_1}^{t} U_s^d + \sum_{b \in S} q_b w_{c(b),t} - \sum_{b \in S} q_b \sum_{s=t_1}^{t} y_{b,d,s},$$

$$\implies q(S) - \sum_{s=t_1}^{t} U_s^d + \underbrace{\sum_{b \in \bar{c}} q_b \sum_{s=t_1}^{t} y_{b,d,s}}_{\sum_{b \in \bar{c}} q_b \left(w_{\bar{c},t} - w_{\bar{c},t} + \sum_{s=t_1}^{t} y_{b,d,s}\right)} + \sum_{b \in b(R)} q_b \sum_{s=t_1}^{t} y_{b,d,s} \leq \sum_{b \in S} q_b \left(w_{c(b),t} - \sum_{s=t_1}^{t} y_{b,d,s}\right).$$

$$\implies q(S) - \sum_{s=t_1}^{t} U_s^d + \underbrace{\sum_{b \in \bar{c}} q_b w_{\bar{c},t}}_{q(\bar{c}) w_{\bar{c},t}} + \sum_{b \in b(R)} q_b \sum_{s=t_1}^{t} y_{b,d,s} \leq \sum_{b \in S \cup \{\bar{c}\}} q_b \left(w_{c(b),t} - \sum_{s=t_1}^{t} y_{b,d,s}\right),$$

$$\implies \left(q(S) - \sum_{s=t_1}^{t} U_s^d + q(\bar{c})\right) w_{\bar{c},t} + \sum_{b \in b(R)} q_b \sum_{s=t_1}^{t} y_{b,d,s} \leq \sum_{b \in S \cup \{\bar{c}\}} q_b \left(w_{c(b),t} - \sum_{s=t_1}^{t} y_{b,d,s}\right).$$

This proves the validity of (23). □

Given the flexibility in the choice of the set S above, we are also interested in finding *the best* possible S. It turns out that there is a simple expression for the set S with maximum violation. Specifically, given a solution (w^*, y^*) of the relaxation of the PCPSP-C, a cluster $\bar{c} \in \mathscr{C}$ and time periods $1 \leq t_1 \leq t \leq T$, the set S^* which maximizes the violation in (23) is

$$S^* = \left\{ b \in \mathscr{B} \,:\, c(b) \prec \bar{c},\; w^*_{\bar{c},t} > w^*_{c(b),t} - \sum_{s=t_1}^{t} y^*_{b,d,s} \right\}. \tag{24}$$

If a block $\bar{b}$ such that $c(\bar{b}) \prec \bar{c}$ satisfies $w^*_{\bar{c},t} > w^*_{c(b),t} - \sum_{s=t_1}^{t} y^*_{b,d,s}$ and it is not in S^*, then adding it to S^* would increase the violation of the inequality. If a block $\bar{b} \in S^*$ is such that $w^*_{\bar{c},t} < w^*_{c(b),t} - \sum_{s=t_1}^{t} y^*_{b,d,s}$, removing it from S^* would decrease it. Expression (24) allows us to quickly compute the most-violated hourglass cut, thus efficiently finding a deep cut in this family.

6 Heuristics

To quickly compute good feasible solutions to instances of PCPSP-C (and, consequently, strong lower bounds), we rely on a combination of heuristic techniques and a local search method. The first technique is a generalization of the TopoSort heuristic described by [10] originally for C-PIT, to PCPSP-C. That is, this generalization considers arbitrary clusters and both full and partial integrality. The heuristic requires all the coefficients in constraint matrix G to be non-negative; thus, it is well suited for all instances of the OPPDP and for instances of OPPSP with $G \geq 0$. The second technique that we consider is based on converting an instance of PCPSP-C to an instance of PCPSP by fixing the destination of each block before the optimization. This technique potentially allows us to use the many different algorithms that have been proposed for C-PIT, although it is more general since it does not require G to be non-negative. Furthermore, if the number of clusters that define the problem is not too large, as is often the case, one can solve the resulting PCPSP directly with a mixed integer programming solver. This makes the technique well-suited for the OPPSP but not very effective for the OPPDP. We call this second technique the *1-DEST Heuristic*. Finally, given an integer-feasible solution of PCPSP-C, it is possible to perform a simple local-search procedure to improve the destination assignment of the blocks. One can apply this to any integer-feasible solution of PCPSP-C simply by solving a linear programming problem, which we call the *Optimize-Destinations* heuristic. We now describe these in more detail.

6.1 The Generalized TopoSort Heuristic [TopoSort]

The generalized TopoSort heuristic, presented in Algorithm 1, is a modified version of the expected-time TopoSort heuristic of [10] that expands the functionality of the algorithm from instances of the C-PIT to instances of the PCPSP-C. It takes as input a solution (x^*, y^*) obtained by solving the LP-PCPSP-C and outputs a feasible solution $(\bar{x}, \bar{y})$ of the PCPSP-P satisfying the partial integrality constraint, as defined in (4). The heuristic assumes that matrix G, appearing in constraints (1e), is such that $G \geq 0$ (i.e., all its components are non-negative). It is straightforward to modify this heuristic such that its output satisfies full integrality (3).

Given a directed acyclic graph (DAG), a topological ordering is such that if a node u comes before a node v, then there cannot exist an arc going from u to v. It is well known that every DAG admits a topological ordering and that computing it can be done in linear time (see [11]). In terms of the Open Pit Production Scheduling Problem, a topological ordering corresponds to an extractable ordering of the blocks. That is, the position of every block b in the ordering is such that all its predecessors appear before it.

The TopoSort heuristic presented in [10] works by first sorting all blocks in topological order. Following this order, each block is scheduled as early as possible, that is, in the first time period with sufficient resources and no earlier than its predecessors. Once a block is scheduled, the available resources are updated, and the algorithm continues by scheduling the next block. Because many topological orderings are available, the heuristic uses the optimal solution of the LP relaxation to guide the sorting process. See [10] for details.

The generalized TopoSort algorithm that we present in Algorithm 1 exhibits several important differences from the TopoSort heuristic described by [10]. First, it schedules clusters rather than blocks. Second, it can handle partial integrality condition (4). Third, it forces clusters to be extracted no earlier than the first time period in which the corresponding variables in the LP relaxation have nonzero values. This last modification, implemented in Step 8 of Algorithm 1, prevents negative-valued extractions from occurring too early in the schedule. Fourth, we break ties with the coefficients in the objective function.

6.2 The 1-DEST Heuristic.

This heuristic proceeds in three steps. First, it uses the optimal solution (x^*, y^*) of LP-PCPSP-C to assign a fixed destination to each block for each period. Second, it constructs a small instance of PCPSP with a single destination (and no clusters). This is done by substituting out all of the y variables and obtaining a problem only in terms of the x variables. Third, the resulting problem is solved directly by using a mixed-integer programming solver.

Specifically, for each $b \in \mathscr{B}$, $d \in \mathscr{D}$, let us define:

$$\bar{y}_{b,d} = \frac{\sum_{t \in \mathscr{T}} y^*_{b,d,t}}{\sum_{d' \in \mathscr{D}} \sum_{t \in \mathscr{T}} y^*_{b,d',t}}.$$

Given these values, we impose the following condition for each $c \in \mathscr{C}$, $b \in c, d \in \mathscr{D}$ and $t \in \mathscr{T}$:

$$y_{b,d,t} = x_{c,t} \cdot \bar{y}_{b,d}. \tag{25}$$

Substitute the equations (25) in constraints (1c)-(1f) of the PCPSP-C formulation to obtain:

$$\begin{aligned}
\max \ & \sum_{c\in\mathscr{C}}\sum_{t\in\mathscr{T}} \bar{p}_{c,t}x_{c,t} & \\
\text{s.t.} \ & \sum_{t\in\mathscr{T}} x_{c,t} \leq 1 & \forall c \in \mathscr{C}, \\
& \sum_{t'=1}^{t} x_{c,t'} \leq \sum_{t'=1}^{t} x_{c',t'} & \forall (c,c') \in \mathscr{A},\ t \in \mathscr{T}, \\
& \bar{G}x \leq g & \\
& x_{c,t} \geq 0 & \forall c \in \mathscr{C}, t \in \mathscr{T}, \\
& x_{c,t} \text{ satisfies an integrality condition} & \forall c \in \mathscr{C}, t \in \mathscr{T},
\end{aligned}$$

where $\bar{p}$ and $\bar{G}$ are naturally defined from the substitution.

Solve this problem to optimality using a mixed-integer programming solver, and use (25) to determine the value of the associated y variables. Depending on the integrality conditions used in the last optimization problem, we construct a solution to either PCPSP-F or PCPSP-P.

6.3 The Optimize-Destinations Heuristic

Consider a feasible mixed-integer solution (x,y) of PCPSP-C. A simple method to obtain an improved solution (x,y') is to fix the x variables and re-optimize the y variables with a linear programming solver (e.g., with the BZ algorithm proposed by [3, 4]). This is a natural process after running the generalized TopoSort or 1-DEST heuristics, which can only improve the solution.

7 Branch-and-Bound

We now introduce a simple branch-and-bound algorithm to solve PCPSP-C to optimality. In our description of the algorithm, we assume that we are interested in computing a solution which satisfies the partial integrality constraint (4). Imposing integrality constraint (3) instead is analogous. This algorithm does not make any assumptions about the matrix G appearing in the constraints (1e). For background about the branch-and-bound algorithm, see [42]. The main characteristics of our branch-and-bound algorithm are:

- The nodes of the branch-and-bound tree can be explored in a depth-first-search or a best-bound-first-search. In our computational runs, we used the best-bound-first-search rule. We use strong branching at every node of the tree to select variables.
- The LP at each node of the tree is solved with the BZ algorithm, using the features described by [41].
- When using the partial integrality condition (4), a solution x^* is considered "fractional" if there exists an arc $(c^1,c^2) \in \mathscr{A}$ and a time period $t \in \mathscr{T}$, such that:

$$\sum_{t'=1}^{t} x^*_{c^1,t'} > 0 \text{ and } \sum_{t'=1}^{t} x^*_{c^2,t'} < 1.$$

In this case, $(c^1, c^2) \in \mathscr{A}$ and $t \in \mathscr{T}$ define a possible branching point. The two branches to consider for this point are:

$$\sum_{t'=1}^{t} x_{c^1,t'} = 0 \quad \text{(the down-branch)}, \qquad \text{and} \qquad \sum_{t'=1}^{t} x_{c^2,t'} = 1 \quad \text{(the up-branch)}. \tag{26}$$

If, after branching on (c^1, c^2) and t, we find that one of the branches is pruned, the parent node can be strengthened as follows.

- If the down-branch is pruned, all integral solutions at the parent node must satisfy $\sum_{t'=1}^{t} x_{c^1,t'} > 0$. This, in turn, implies that all integral solutions at the parent node must satisfy the following constraint, which can be added to the node:

$$\sum_{t'=1}^{t} x_{c,t'} = 1 \qquad \text{for all } c \text{ such that } (c^1, c) \in \mathscr{A}. \tag{27}$$

- If the up-branch is pruned, integral solutions at the parent node must satisfy $\sum_{t'=1}^{t} x_{c^2,t'} < 1$. This, in turn, implies that all integral solutions at the parent node must satisfy the following constraint, which can be added to the node:

$$\sum_{t'=1}^{t} x_{c,t'} = 0 \qquad \text{for all } c \text{ such that } (c, c^2) \in \mathscr{A}. \tag{28}$$

- The solver only adds cutting planes to the root node.

8 Computational study

The primary goal of our computational study is to determine if the proposed methodology can be used to solve real problem instances of OPPDP and OPPSP. The secondary goal of our study is to determine the contribution of each feature to the overall effectiveness of the methodology. To conduct our study, we implemented the methodology described in this paper, including all the features covered in Sections 4 - 7, and collected ten mine planning instances which could be used to solve instances of both OPPDP and OPPSP.

Table 1 summarizes the numerical characteristics of our test instances. These instances all correspond to real mines, with the exception of *marvin*. Moreover, four of these instances (*kd, marvin, mclaughlin* and *mclaughlin_limit*) are publicly available in the MineLib repository (see [19]). The remaining instances were provided by industry partners, whose names we have changed for confidentiality reasons.

Using each of these datasets, we created one instance of OPPDP and two instances of OPPSP. We refer to the latter two as the *OPPSP base instance* and the *OPPSP extended instance*.

To build each OPPSP base instance, we computed clusters for the corresponding problems as follows. A mining engineer used the Dassault Whittle [23] strategic mine planning software to compute four phases for each instance using the 'Automatic Pushback Generator' feature (Milawa NPV algorithm). The clusters used for our models consist of the bench phases obtained by using these phases. Each instance has between 50 and 300 clusters. To build each OPPSP extended instance, each OPPSP base instance was modified in one of four ways:

- For instances *calbuco, kd*, and *palomo*, the mining engineer computed clusters using seven phases, rather than four.

- For instances *chaiten, marvin* and *mclaughlin*, we added minimum processing constraints,

$$\sum_{b\in\mathscr{B}} q_b y_{b,d,t} \geq L_t^d \qquad \forall t \in \mathscr{T},$$

  where $d \in \mathscr{D}$ is the mill, or the concentrator destination. These constraints are fairly common and seek to ensure a constant yearly production and usage of the concentrator.

- For instances *guallatari, mclaughlin_limit,* and *ranokau*, we imposed what is known as a flow balance constraint, that is, a constraint of the form:

$$\sum_{b\in\mathscr{B}}\sum_{d\in\mathscr{D}} q_b y_{b,d,t+1} \leq (1+\alpha)\sum_{b\in\mathscr{B}}\sum_{d\in\mathscr{D}} q_b y_{b,d,t} \qquad \forall t \in \{1,\ldots,T-1\}. \tag{29}$$

  These constraints are fairly common in industry since, in practice, it is not feasible to radically change the size of the workforce and fleet of trucks from one year to another. For example, Dassault Whittle offers a heuristic called Milawa Balance which imposes this condition. In our runs, we used $\alpha = 0.0$, which produces solutions with characteristics similar to those produced by the default Milawa Balance algorithm.

- In the case of instance *tronador*, we imposed a blending constraint. These constraints are typically used to limit the weighted average of a given contaminant sent to a fixed destination $d \in \mathscr{D}$. For each $b \in B$, we let q'_b represent the tons of prohibited material in b, and since we wish to impose that the weighted average grade of this material is less than β_t in each period $t \in \mathscr{T}$, we add the following constraint:

$$\frac{\sum_{b\in\mathscr{B}} q'_b y_{b,d,t}}{\sum_{b\in\mathscr{B}} q_b y_{b,d,t}} \leq \beta_t \qquad \forall t \in \mathscr{T}\setminus\{1\}.$$

  By multiplying the denominator, this constraint becomes linear and can be added to the model. In the specific case of *tronador*, the constraint is used to limit the weighted average amount of arsenic sent to the mill.

The constraints added to the OPPSP extended instances made the OPPSP base instance solutions infeasible. Moreover, with the exception of the "7 phases" cases, the TopoSort and UPIT methods cannot be used in any of the OPPSP extended instances.

All our algorithms were implemented using the C programming language, with CPLEX® 12.6 as the optimization solver. The computer servers employed for computations use Linux 2.6.32 on x86_64 architecture, with four eight-core Intel® Xeon® E5-2670 processors and 128 GB of RAM. Results analyze the performance with different sets of activated features and are given as normalized geometric means.

8.1 Effectiveness of the methodology on instances of the OPPDP.

We began by testing the effectiveness of our three proposed preprocessing techniques: ultimate pit limit preprocessing (UPIT), dominated triplet elimination (DTE), and early start cuts (ES), as described in Sections 4.1, 4.2 and 5.1, respectively. We ran the preprocessing algorithm four times on each instance: one with all our preprocessing techniques (we call this our default preprocessing methodology), and then we preprocessed each instance by using our default preprocessing with exactly one of the features disabled. Table 2 describes the results of these experiments. To facilitate comparisons, we present the resulting number of variables as a percentage of the original number of variables (originally given in Table 1). As can be appreciated from the fifth column, the effect of preprocessing can be significant. This is illustrated by the *calbuco* instance, where only 3.19% of the variables remained after preprocessing. However, in the other instances, the effect is only moderate. By observing columns two through four, it is noticeable that all three techniques (UPIT, DTE, and ES) are effective. However, disabling UPIT has the most detrimental effect. After preprocessing these instances, we solved the linear programming relaxation of the corresponding PCPSP-C instance using the BZ algorithm, as described in [41]. Then, we used the TopoSort heuristic, as described in Section 6.1, to compute integer-feasible solutions for each instance and the Optimize-Destinations heuristic, as described in Section 6.3, to improve them.

The results are presented in Table 3. To facilitate comparisons, in the second column, we show the objective function value of the LP relaxation as a percentage of the solution obtained via the TopoSort heuristic. The total time taken by the algorithm, including the time to solve the LP relaxation, is presented in column three. It should be further noted that the time required for the TopoSort and Optimize-Destination heuristics is negligible (less than one second). In Table 3, we observe that with the exception of the *ranokau* instance, it is possible to compute very high-quality solutions to these problems in just minutes. For example, consider the *guallatari* instance. In just over two minutes (2 m 16 s), it is possible to obtain an integer-feasible solution within 1.3% of optimality. The LP relaxation of *ranokau* is difficult to solve owing to the number of time periods it contains. However, the quality of the solution obtained by the heuristics is still very good, showing a gap of 2.22%. Overall, the geometric mean of gaps is only 1.25%, confirming that the bounds provided by LP relaxation of OPPDP instances tend to be very tight (as observed by [3]).

8.2 Effectiveness of the methodology on instances of OPPSP.

In addition to testing the UPIT, DTE and ES techniques, as we did in Section 8.1, we also tested the aggregation (AGG) technique, as described in Section 4.3. Note that the ES technique is considered for both the preprocessing and cutting planes analysis. Our default preprocessing option consists of applying all four techniques. The results of our experiments are summarized in Table 4. In the sixth column, we can observe that the effect of preprocessing is also significant for the OPPSP; however, the most efficient techniques change with respect to OPPDP.

Next, we analyze the tightness of the LP relaxation bounds for these instances and the effectiveness of the cutting planes described in Section 5 for improving these bounds. To do so, we compare the value of the LP relaxation to the value of the best known integer-feasible solution

of each instance by taking the ratio of these two values. This ratio is computed in six different manners for each instance: once with no cutting planes, once with our default options (early start, lifted clique, hourglass and VRHS cuts), and once with our default options minus one of the cutting plane classes. Our cutting-plane approach is simple. In each round, we generate all possible cuts but only add the $1,000$ most violated ones. We run no more than ten rounds of cuts. Cuts are eliminated in subsequent rounds if the corresponding dual variables are null. As is disclosed later, the objective function value of these integer-feasible solutions is within one percent of optimality for all instances. Thus, these ratios provide a very good estimate of the tightness of the bounds and the effectiveness of the cutting planes in improving these bounds. See Table 5.

Note that the ratios have been multiplied by 100 to facilitate readability. Thus, if the value of a ratio is 100, the LP relaxation has the same objective function value as the best known integer-feasible solution (i.e., the bound is tight). The values in the second column of Table 5 provide evidence regarding how weak the LP relaxation bounds computed without cutting planes can be. On average, this bound is within 10% of optimality, but in some instances, it can be significantly worse (more than 30% in both *ranokau* instances and almost 20% in the *chaiten* instances), which stands in contrast to what we observe for the OPPDP instances. Nonetheless, the seventh column shows a significant tightening of the bounds after adding the cutting planes. In general, all cutting planes help improve the bounds (columns three through six); however, it seems that the hourglass cuts have the most significant impact. It is also worth pointing out that the behaviors of the base and extended instances are very similar. As would be expected, the gaps of the extended instances are marginally worse, but the cuts seem to help in both types of problems.

Table 6 demonstrates the impact which branching for four hours has upon the upper bounds provided by the linear programming relaxation as computed with and without cutting planes. We do not add cuts at nodes of the branch-and-bound tree in any of the instances. When comparing the third and fourth columns, in addition to the seventh and eighth columns, we can see that in most instances, branching for four hours is less effective than adding cuts. On the other hand, by comparing columns four and five, in addition to columns eight and nine, we can see that branching does improve the bound computed by the cutting planes. In fact, branching reduces the average ratio to 100.47 for the extended instances and 100.64 for the base ones.

Table 7 depicts the effectiveness of our different techniques for computing integer-feasible solutions to OPPSP (i.e., lower bounds). Specifically, we analyze the performance of the TopoSort heuristic (Topo), running the 1-DEST Reduction Heuristic, and branching with a time limit of four hours (BB4). All values are normalized relative to the best upper bound computed for each instance (which would have a value of 100) using branch-and-bound with cutting planes. Note that we cannot use the TopoSort heuristic on the extended problem sets (since the matrix G appearing in the PCPSP-C has negative coefficients).

From the second column, as in the OPPDP instances, we observe that the TopoSort heuristic is very effective. However, the 1-DEST heuristic seems to compute slightly better solutions and has the advantage of being able to run for both the base and extended problem instances. Table 7 also shows that using the solutions computed from the LP relaxation with cutting planes results in better feasible solutions for all methods. Finally, the table shows that the best solutions are typically computed with the 1-DEST heuristic, although branching sometimes yields slightly

better solutions. This is significant because it shows that provably near-optimal solutions can be computed without the need for customized branch-and-bound algorithms, which are difficult to implement.

A summary of the performance of our default features on instances of OPPSP is presented in Table 8. Here, the gaps are computed as $gap = \frac{ub-lb}{ub}$, where lb is the best bound computed by the branch-and-bound algorithm and ub is the value of the best integer-feasible solution amongst those computed by the 1-DEST and BB4 algorithms. We refer to this as the *proved gap*, since it is the best gap that can be computed using the information obtained from the algorithm execution and not using the best-known feasible solution to the problem (possibly computed using some other method). As Table 8 indicates, the algorithms perform very well, obtaining integer-feasible solutions that, on average, are below one percent. Although we failed to compute solutions within one percent of optimality for some instances, the results were still within four percent throughout.

Finally, a summary of the times required to compute these solutions is presented in Table 9. In practice, 1-DEST often takes significantly less than four hours to compute high-quality feasible solutions. In fact, in most problems, solutions can be computed in just minutes. Table 9 also shows that in just under a third of the instances, we were able to terminate the branch-and-bound run within the four hours. Though this suggests that there is room to improve these results through further research on the branch-and-bound methodology, by combining these overall results with those presented in Table 8, it is possible to see that our methodology is effective in addressing the proposed problem and efficient in terms of the overall time required.

8.3 Tables

Algorithm 1: The TopoSort Heuristic.

Input: An instance of the PCPSP-P, as defined in (1) with integrality constraints (4), such that $G \geq 0$ and $g \geq 0$. A solution (x^*, y^*) of the LP relaxation.

Output: A solution $(\bar{x}, \bar{y})$ of a PCPSP-P instance.

Let $\mathscr{T}^* = \mathscr{T} \cup \{T+1\}$.

For each $c \in \mathscr{C}$, define:

$$\begin{aligned} \delta(c) &= |\{c' \in \mathscr{C} : (c,c') \in \mathscr{A}\}|, \\ S_c &= \{c' \in \mathscr{C} : (c',c) \in \mathscr{A}\}, \\ x^*_{c,T+1} &= 1 - \sum_{t \in \mathscr{T}} x^*_{c,t}, \\ es(c) &= \min\{t \in \mathscr{T}^* : x^*_{c,t} > 0\}, \\ E(c) &= \sum_{t \in \mathscr{T}^*} t x^*_{c,t}. \end{aligned}$$

$(\bar{x}, \bar{y}) \leftarrow (0,0)$.

while $\mathscr{C} \neq \emptyset$ **do**

  Choose $c \in \mathscr{C}$ that solves $\min\{E(c) : \delta(c) = 0,\ es(c) \leq T\}$.

  for $b \in c$ **do**

    $d(b) \leftarrow \operatorname{argmax}\left\{ \sum_{t \in \mathscr{T}} y^*_{b,d,t} : d \in \mathscr{D} \right\}$.

  $t \leftarrow es(c)$.

  $w \leftarrow 0$.

  while $w < 1$ *and* $t \leq T$ **do**

    $\bar{\alpha} \leftarrow \max\left\{\alpha : G\bar{y} \leq g,\ w + \alpha \leq 1,\ \bar{y}_{b,d(b),t} = \alpha,\ \forall b \in c\right\}$.

    $\bar{y}_{b,d(b),t} \leftarrow \bar{\alpha}$ for all $b \in c$.

    $\bar{x}_{c,t} \leftarrow \bar{\alpha}$.

    $w \leftarrow (w + \bar{\alpha})$.

    $t \leftarrow (t+1)$.

  for $a \in S_c$ **do**

    $\delta(a) \leftarrow \delta(a) - 1$.

    $es(a) \leftarrow \max\{es(a), t\}$.

  $\mathscr{C} \leftarrow \mathscr{C} \setminus \{c\}$.

Table 1: Description of instances used for computational tests.

Instance	$\lvert\mathscr{D}\rvert$	$\lvert\mathscr{T}\rvert$	mcap	OPPDP		OPPSP base			OPPSP extended		
				$\lvert\mathscr{B}\rvert$	nvars	$\lvert\mathscr{C}\rvert$	$\lvert\mathscr{B}\rvert$	nvars	$\lvert\mathscr{C}\rvert$	$\lvert\mathscr{B}\rvert$	nvars
calbuco	3	21	n	5,016,971	316,069,173	324	200,241	8,416,926	548	200,241	12,615,183
chaiten	2	25	y	339,199	16,959,950	273	288,073	7,208,650	273	288,073	7,208,650
guallatari	3	21	y	1,672,198	105,348,474	272	57,527	2,421,846	272	57,527	2,421,846
kd	2	12	n	14,153	339,672	53	10,128	122,172	93	10,128	243,072
marvin	2	20	y	53,271	2,130,840	56	8,515	171,420	56	8,515	171,420
mclaughlin	2	20	n	2,140,342	85,613,680	173	180,749	3,618,440	173	180,749	3,618,440
mclaughlin_limit	2	15	n	112,687	3,380,610	166	110,768	1,664,010	166	110,768	1,664,010
palomo	2	40	y	772,800	61,824,000	74	190,319	7,615,720	123	190,319	15,225,520
ranokau	2	81	y	1,873,035	303,431,670	186	317,907	25,765,533	186	317,907	25,765,533
tronador	2	20	y	329,859	18,801,963	220	30,099	1,805,940	220	30,099	1,805,940

Note: mcap indicates the presence of mining capacity constraints (2a). All instances have destination capacity constraints (2b), and nvars indicates the number of variables in the problem before preprocessing.

Table 2: Percentage of variables in OPPDP instances remaining after preprocessing.

Instance	Default w/o ES	Default w/o DTE	Default w/o UPIT	Default
calbuco	3.19	3.95	68.46	3.19
chaiten	53.73	84.06	61.04	53.16
guallatari	2.35	3.43	43.46	2.32
kd	85.88	85.88	100.00	85.88
marvin	12.07	15.88	53.67	11.96
mclaughlin	6.24	8.44	52.22	6.24
mclaughlin_limit	77.76	98.30	78.62	77.76
palomo	7.46	12.39	46.51	7.31
ranokau	11.89	15.31	46.33	11.01
tronador	29.21	32.24	70.33	28.75

Table 3: Objective values of OPPDP instances as a percentage of TopoSort objective value.

Instance	LP relaxation	Total time
calbuco	102.06	13m 5s
chaiten	100.33	26m 55s
guallatari	101.22	2m 16s
kd	100.87	2.8s
marvin	102.49	4.5s
mclaughlin	100.21	4m 55s
mclaughlin_limit	100.16	1m 36s
palomo	101.10	12m 12s
ranokau	102.22	9h 39m 13s
tronador	102.47	3m 13s
Geo. Mean	101.31	-

Table 4: Percentage of variables in OPPSP instances remaining after preprocessing.

Instance	Default w/o ES	Default w/o DTE	Default w/o UPIT	Default w/o AGG	Default
calbuco	30.56	50.25	30.56	70.94	30.56
chaiten	23.21	24.09	15.96	17.96	15.96
guallatari	51.09	80.58	41.61	42.20	41.61
kd	42.97	42.97	42.97	100.00	42.97
marvin	51.29	54.93	43.41	43.41	43.41
mclaughlin	8.45	10.59	8.45	47.94	8.45
mclaughlin_limit	12.06	14.96	12.06	58.28	12.06
palomo	1.36	1.44	1.46	6.04	1.26
ranokau	45.10	33.26	21.46	21.47	21.46
tronador	67.04	74.43	50.55	68.84	50.55

Table 5: Effect of cutting planes on upper bound obtained from LP relaxation in OPPSP instances.

Instance	No cuts	No Early-Start	No Lifted Clique	No VRHS	No Hourglass	Ext. cuts	Pro. cuts	All cuts
Base Instances								
calbuco	108.28	102.42	102.42	102.86	104.83	108.28	102.42	102.42
chaiten	117.26	102.01	101.34	100.03	100.29	100.88	109.23	100.00
guallatari	102.02	101.09	100.54	100.54	100.71	100.87	101.09	100.54
kd	101.75	100.21	100.21	100.22	101.10	101.75	100.21	100.21
marvinml	105.75	101.10	100.61	100.63	101.46	103.06	101.10	100.61
mclaughlin	102.52	100.34	100.34	100.37	101.37	102.52	100.34	100.34
mclaughlinlimit	102.39	100.25	100.25	100.18	101.43	102.39	100.25	100.25
palomo	114.87	103.62	101.26	101.33	103.55	111.37	103.62	101.26
ranokau	131.48	104.76	101.82	101.87	102.10	104.96	105.20	101.82
tronador	108.84	101.94	100.83	100.80	100.83	100.90	104.00	100.80
Geo. Mean	109.17	101.76	100.96	100.88	101.76	103.65	102.71	100.82
Extended Instances								
calbuco	105.31	101.87	101.87	102.20	103.31	105.31	101.87	101.87
chaiten	117.86	101.73	101.68	100.14	100.48	100.90	109.90	100.12
guallatari	102.47	102.01	101.34	101.36	101.43	101.54	102.01	101.34
kd	100.70	100.25	100.25	100.28	100.66	100.70	100.25	100.25
marvinml	110.56	104.62	103.78	103.79	104.70	107.20	104.62	103.78
mclaughlin	103.24	100.85	100.85	100.90	101.95	103.24	100.85	100.85
mclaughlinlimit	103.16	100.75	100.75	100.77	102.09	103.16	100.75	100.75
palomo	111.53	103.68	101.75	101.88	103.04	108.07	103.68	101.75
ranokau	131.10	104.92	101.82	100.33	100.31	104.86	105.17	100.15
tronador	117.07	105.51	104.11	103.39	103.59	103.60	109.25	103.39
Geo. Mean	109.95	102.60	101.81	101.50	102.15	103.83	103.78	101.42

Table 6: Effect of Branch-and-Bound on the upper bound after 4 hours of branching in OPPSP instances.

Instance	Base Instances				Extended Instances			
	No Cuts		All Cuts		No Cuts		All Cuts	
	Root	BB4	Root	BB4	Root	BB4	Root	BB4
calbuco	108.28	103.76	102.42	102.12	105.31	103.03	101.87	101.61
chaiten	117.26	103.83	100.00	100.00	117.86	109.77	100.12	100.04
guallatari	102.02	101.18	100.54	100.26	102.47	101.92	101.34	100.67
kd	101.75	100.00	100.21	100.00	100.70	100.00	100.25	100.00
marvinml	105.75	100.01	100.61	100.00	110.56	101.07	103.78	100.01
mclaughlin	102.52	100.27	100.34	100.08	103.24	100.88	100.85	100.45
mclaughlinlimit	102.39	100.00	100.25	100.01	103.16	100.62	100.75	100.29
palomo	114.87	102.75	101.26	100.14	111.53	106.52	101.75	100.76
ranokau	131.48	126.43	101.82	101.82	131.10	123.41	100.15	100.15
tronador	108.84	101.41	100.80	100.32	117.07	110.02	103.39	102.44
Geo. Mean	109.17	103.71	100.82	100.47	109.95	105.51	101.42	100.64

Table 7: Effectiveness of heuristics and branch-and-bound computing feasible integer solutions in OPPSP instances.

Instance	Base Instances						Extended Instances			
	No Cuts			All Cuts			No Cuts		All Cuts	
	Topo	1-DEST	BB4	Topo	1-DEST	BB4	1-DEST	BB4	1-DEST	BB4
calbuco	97.68	97.68	97.68	94.65	98.07	98.07	97.92	97.92	98.38	98.38
chaiten	95.67	99.53	99.53	99.15	100.00	100.00	98.18	98.18	99.96	99.96
guallatari	99.66	99.66	99.66	99.66	99.66	99.66	99.24	99.24	99.33	99.33
kd	99.86	99.97	100.00	99.95	99.96	100.00	99.98	100.00	99.97	100.00
marvinml	98.00	99.63	100.00	99.37	99.89	100.00	99.03	99.03	99.46	99.99
mclaughlin	98.75	98.75	99.34	99.62	99.62	99.62	98.63	98.63	99.40	99.40
mclaughlinlimit	99.47	99.47	100.00	99.88	99.88	99.99	99.30	99.30	99.56	99.56
palomo	96.55	98.61	98.61	93.77	98.67	98.67	99.24	99.24	99.01	99.01
ranokau	92.63	98.21	98.21	95.30	97.94	97.94	99.78	99.78	99.69	99.69
tronador	98.12	99.69	99.69	99.45	99.79	99.79	94.93	94.93	99.94	99.94
Geo. Mean	97.62	99.12	99.27	98.05	99.35	99.37	98.61	98.62	99.47	99.52

Note: Topo corresponds to the value obtained from using the TopoSort heuristic, 1-DEST to the value of the 1-DEST heuristic, and BB4 correspond to the values obtainmed by using the Branch-and-Bound algorithm with a time limit of four hours.

Table 8: Effect of Branch-and-Bound on the proved gap in OPPSP instances.

Instance	Base Instances		Extended Instances	
	Before BB4	After BB4	Before BB4	After BB4
calbuco	2.70%	2.37%	1.87%	1.63%
chaiten	0.00%	0.00%	0.13%	0.04%
guallatari	0.63%	0.36%	1.33%	0.67%
kd	0.26%	0.00%	0.31%	0.00%
marvinml	0.71%	0.00%	4.16%	0.01%
mclaughlin	0.66%	0.41%	0.99%	0.60%
mclaughlinlimit	0.37%	0.01%	0.90%	0.44%
palomo	2.43%	1.33%	1.95%	0.99%
ranokau	2.06%	2.06%	0.31%	0.31%
tronador	0.80%	0.32%	3.34%	2.44%
Geo. Mean	1.06%	0.68%	1.52%	0.71%

Table 9: Relevant times for algorithms in OPPSP instances.

Instance	Base Instances				Extended Instances			
	LP No cuts	LP + Cuts	1-DEST	BB	LP No cuts	LP + Cuts	1-DEST	BB
calbuco	10s	4m 42.9s	1m 3.9s	$>$ 4h	8.5s	4m 4.6s	7m 51.2s	$>$ 4h
chaiten	9.9s	1m 26.4s	5m 41.4s	8.1s	17s	2m 23.7s	1m 28.2s	$>$ 4h
guallatari	3.5s	23.4s	5m 26.7s	$>$ 4h	6.4s	38.6s	7m 13.9s	$>$ 4h
kd	0.2s	0.9s	0.7s	38.5s	0.1s	0.7s	1.3s	51.5s
marvinml	0.4s	2s	2.4s	15m 54.1s	0.4s	3.5s	1.6s	3h 50m 50s
mclaughlin	2.1s	12.4s	6.3s	$>$ 4h	3.4s	22.2s	7.7s	$>$ 4h
mclaughlinlimit	1.1s	5.2s	5.9s	2h 19m 44.3s	1.5s	5.6s	26.6s	$>$ 4h
palomo	3.4s	29.6s	21.7s	$>$ 4h	3.7s	31.7s	1m 19s	$>$ 4h
ranokau	9m 19.8s	6m 12.6s	13m 9.3s	$>$ 4h	10m 36.2s	14m 55.5s	41m 12.2s	$>$ 4h
tronador	2.9s	9.8s	17.5s	$>$ 4h	33.8s	1m 45.4s	6m 58.2s	$>$ 4h

References

[1] Askari-Nasab, H., K. Awuah-Offei, H. Eivazy. 2010. Large-scale open pit production scheduling using mixed integer linear programming. *International Journal of Mining and Mineral Engineering* **2**(3) 185–214.

[2] Askari-Nasab, H., Y. Pourrahimian, E. Ben-Awuah, S. Kalantari. 2011. Mixed integer linear programming formulations for open pit production scheduling. *Journal of Mining Science* **47**(3) 338–359.

[3] Bienstock, D., M. Zuckerberg. 2009. A new LP algorithm for precedence constrained production scheduling. Unpublished. Columbia University, BHP Billiton. Available at Optimization Online.

[4] Bienstock, D., M. Zuckerberg. 2010. Solving LP relaxations of large-scale precedence constrained problems. *Proceedings from the 14th conference on Integer Programming and Combinatorial Optimization (IPCO). Lecture Notes in Computer Science 6080* 1–14.

[5] Bley, A., N. Boland, C. Fricke, G. Froyland. 2010. A strengthened formulation and cutting planes for the open pit mine production scheduling problem. *Computers & Operations Research* **37**(9) 1641–1647.

[6] Boland, N., I. Dumitrescu, G. Froyland. 2008. A multistage stochastic programming approach to open pit mine production scheduling with uncertain geology. Unpublished. Available at Optimization Online.

[7] Boland, N., I. Dumitrescu, G.Froyland, A. M. Gleixner. 2009. LP-based disaggregation approaches to solving the open pit mining production scheduling problem with block processing selectivity. *Computers & Operations Research* **36** 1064–1089.

[8] Boyd, E. Andrew. 1993. Polyhedral results for the precedence-constrained knapsack problem. *Discrete Applied Mathematics* **41** 185–201.

[9] Caccetta, L., S. P. Hill. 2003. An application of branch and cut to open pit mine scheduling. *Journal of Global Optimization* **27** 349–365.

[10] Chicoisne, R., D. Espinoza, M. Goycoolea, E. Moreno, E. Rubio. 2012. A new algorithm for the open-pit mine production scheduling problem. *Operations Research* **60**(3) 517–528.

[11] Cook, W. J., W.H. Cunningham, W.R. Pulleyblank, A. Schrijver. 1998. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization, Wiley-Interscience, New York.

[12] Cullenbine, C., K. Wood, A. M. Newman. 2011. A sliding time window heuristic for open pit mine block sequencing. *Optimization Letters* **5**(3) 365–377.

[13] Dagdelen, K. 1985. Optimum multi-period open pit mine production scheduling. Ph.D. thesis, Colorado School of Mines, Golden, Colorado.

[14] Dagdelen, K., T. B. Johnson. 1986. Optimum open pit mine production scheduling by Lagrangian parameterization. *19th APCOM Symposium of the society of mining engineers* .

[15] Dantzig, G. B., P. Wolfe. 1960. Decomposition principle for linear programs. *Operations Research* **8**(1) 101–111.

[16] Deswik.CAD. 2017. Deswik.cad. `https://www.deswik.com/product-detail/deswik-cad/`. Accessed: 2017-12-21.

[17] Epstein, R., M. Goic, A. Weintraub, J. Catalán, P. Santibáñez, R. Urrutia, R. Cancino, S. Gaete, A. Aguayo, F. Caro. 2012. Optimizing long-term production plans in underground and open-pit copper mines. *Operations Research* **60**(1) 4–17.

[18] Espinoza, D., M. Goycoolea, E. Moreno. 2015. The precedence constrained knapsack problem: Separating maximally violated inequalities. *Discrete Applied Mathematics* **194** 65–80.

[19] Espinoza, D., M. Goycoolea, E. Moreno, A. M. Newman. 2012. Minelib: a library of open pit mining problems. *Annals of Operations Research* **206** 1–22.

[20] Fricke, C. 2006. Applications of integer programming in open pit mining. Ph.D. thesis, Department of Mathematics and Statistics, The University of Melbourne.

[21] Froyland, G. A., M. Menabde. 2011. System and method (s) of mine planning, design and processing. US Patent 7,957,941.

[22] Gaupp, M. 2008. Methods for improving the tractability of the block sequencing problem for an open pit mine. Ph.D. thesis, Division of Economics and Business, Colorado School of Mines.

[23] GEOVIA Whittle. 2019. Dassault Systèmes. `https://www.3ds.com/products-services/geovia/produc`

[24] Gershon, M. E. 1983. Mine scheduling optimization with mixed integer programming. *Mining Engineering* **35** 351–354.

[25] Goycoolea, M., E. Moreno, O. Rivera. 2013. Direct optimization of an open cut scheduling policy. *Proceedings of APCOM. Porto Alegre, Brazil. November, 2013*. 434–432.

[26] Goycoolea, M., E. Moreno, O. Rivera. 2015. Comparing New and Traditional Methodologies for Production Scheduling in Open Pit Mining. *Proceedings of APCOM. Fairbanks, Alaska. May, 2015*. 352–359.

[27] Hochbaum, D. S. 2008. The pseudoflow algorithm: A new algorithm for the maximum-flow problem. *Operations Research* **56** 992–1009.

[28] Hustrulid, W., K. Kuchta, eds. 2006. *Open Pit Mine Planning and Design*. Taylor and Francis, London, UK.

[29] Jelvez, E., N. Morales, P. Nancel-Penard, J. Peypouquet, P. Reyes. 2016. Aggregation heuristic for the open-pit block scheduling problem. *European Journal of Operational Research* **249**(3) 1169 – 1177.

[30] Johnson, T. B. 1968. Optimum open pit mine production scheduling. Ph.D. thesis, Operations Research Department, Universiry of California, Berkeley.

[31] Khalokakaie, R., P. A. Dowd, R. J. Fowell. 2000. Lerchs–Grossmann algorithm with variable slope angles. *Mining Technology* **109**(2) 77–85.

[32] King, B., M. Goycoolea, A. Newman. 2017. Optimizing the open pit-to-underground mining transition. *European Journal of Operational Research* **257**(1) 297–309.

[33] Lambert, W. B., A. M. Newman. 2014. Tailored lagrangian relaxation for the open pit block sequencing problem. *Annals of Operations Research* **222**(1) 419–438.

[34] Lambert, W. M., A. Brickey, A. M. Newman, K. Eurek. 2014. Open pit block sequencing formulations: a tutorial. *Interfaces* **44**(2) 127–142.

[35] Lane, K. F. 1964. Choosing the optimum cutoff grade. *Colorado School of Mines Quarterly* **59**(4) 811–829.

[36] Lerchs, H., I. F. Grossmann. 1965. Optimum design of open-pit mines. *Transactions* **LXVIII** 17–24.

[37] Liu, S. Q., E. Kozan. 2016. New graph-based algorithms to efficiently solve large scale open pit mining optimisation problems. *Expert Systems with Applications* **43** 59–65.

[38] Maptek Vulcan. 2017. Maptek vulcan. `http://www.maptek.com/products/vulcan/`. Accessed: 2017-12-21.

[39] Meagher, C, R Dimitrakopoulos, D Avis. 2014. Optimized open pit mine design, pushbacks and the gap problema review. *Journal of Mining Science* **50**(3) 508–526.

[40] Moreno, E., M. Rezakhah, A. Newman, F. Ferreira. 2017. Linear models for stockpiling in open-pit mine production scheduling problems. *European Journal of Operational Research* **260**(1) 212 – 221.

[41] Muñoz, G., D. Espinoza, M. Goycoolea, E. Moreno, M. Queyranne, O. Rivera. 2018. A study of the Bienstock-Zuckerberg algorithm, Applications in Mining and Resource Constrained Project Scheduling. *Computational Optimization and Applications* **69** 501–534.

[42] Nemhauser, G. L., L. A. Wolsey. 1989. Integer programming. G. L. Nemhauser, A. H. G. Rinnooy Kan, M. J. Todd, eds., *Handbook in Operations Research and Management Science 1: Optimization*, chap. 6. North-Holland, Amsterdam, 447–527.

[43] Ramazan, S., K. Dagdelen, T. B. Johnson. 2005. Fundamental tree algorithm in optimising production scheduling for open pit mine design. *Mining Technology (Trans. Inst. Min. Metall. A)* **114** 45–54.

[44] Ramazan, S., R. Dimitrakopoulos. 2013. Production scheduling with uncertain supply: a new solution to the open pit mining problem. *Optimization and Engineering* **14**(2) 361–380.

[45] Rimélé, Adrien. 2016. Méthode heuristique doptimisation stochastique de la planification minière et positionnement des résidus miniers dans la fosse. Ph.D. thesis, École Polytechnique de Montréal.

[46] Samavati, Mehran, Daryl Essam, Micah Nehring, Ruhul Sarker. 2018. A new methodology for the open-pit mine production scheduling problem. *Omega* doi:10.1016/j.omega.2017.10.008. To appear.

[47] Smith, M. L., M.J. Wicks. 2013. Medium-term production scheduling of the Lumwana mining complex using MIP with a rolling horizon. *Interfaces* **44**(2) 176–194.

[48] Tolwinski, B. 1998. Scheduling production for open pit mines. *Proceedings of the 28th International Symposium on the Application of Computers and Mathematics (APCOM) in the Mineral Industries*. 651–662.

[49] Vossen, T. W.M., Kevin R.K. Wood, A.M. Newman. 2016. Hierarchical benders decomposition for open-pit mine block sequencing. *Operations Research* **64**(4) 771–793.

[50] Zhu, G., J. Bard, G. Yu. 2006. A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. *INFORMS Journal on Computing* **18**(3) 377–390.

9 Appendix

In this section, we provide a proof of Theorem 1, presented in Section 4. We begin with a simple lemma and then move on to the proof itself.

9.1 Proof of Theorem 1

Lemma 9. *Consider a finite sequence $\mu_1, \ldots, \mu_T$ of real numbers with nonpositive sum, $\sum_{t=1}^{T} \mu_t \leq 0$. Then, for every discount factor $r > 0$, there exists $t_r \in \{1, \ldots, T\}$ such that the discounted partial sum $\sum_{t=t_r}^{T} \frac{\mu_t}{(1+r)^t} \leq 0$.*

Proof. Proof. By contradiction, assume that for some $r > 0$, we have $\sum_{t=s}^{T} \mu_t (1+r)^{-t} > 0$ for all $s \in \{1, \ldots, T\}$. We show by reverse induction on $s = T$ down to 1 that

$$\sum_{t=s}^{T} \mu_t \geq \sum_{t=s}^{T} \mu_t (1+r)^{s-t} > 0. \tag{30}$$

Note that the second inequality in (30) follows from the contradiction assumption and $(1+r)^s > 0$. The base case $s = T$ of (30) trivially holds. Thus, for the inductive step assume that $(1+r)^{-(s+1)} \sum_{t=s+1}^{T} \mu_t \geq \sum_{t=s+1}^{T} \mu_t (1+r)^{-t} > 0$. Then,

$$\begin{aligned} \sum_{t=s}^{T} \mu_t &= \mu_s + \left(1 - \frac{1}{1+r}\right) \sum_{t=s+1}^{T} \mu_t + \frac{1}{1+r} \sum_{t=s+1}^{T} \mu_t > \mu_s + \frac{1}{1+r} \sum_{t=s+1}^{T} \mu_t \\ &\geq \mu_s + \frac{1}{1+r} \sum_{t=s+1}^{T} \mu_t (1+r)^{s+1-t} = \sum_{t=s}^{T} \mu_t (1+r)^{s-t}, \end{aligned}$$

where the first inequality follows from $1 > 1/(1+r)$ and the inductive assumption, and the second inequality from the inductive assumption. This completes the inductive proof of (30). However, (30) with $s = 1$ then implies $\sum_{t=1}^{T} \mu_t > 0$, which contradicts the lemma's assumption that $\sum_{t=1}^{T} \mu_t \leq 0$. □

Before moving on to the theorem, and for the reader's convenience, let us recall some important definitions:

- For $b \in \mathscr{B}$ and $d \in \mathscr{D}$, $p_{b,d}$ represents the value that would be obtained by sending b to d.
- For $b \in \mathscr{B}$, $p_{b,*} = \max\{p_{b,d} : d \in \mathscr{D}\}$.
- For $c \in \mathscr{C}$, define $\bar{p}_c = \sum_{b \in c} p_{b,*}$.
- Let $p_{b,d,t} = p_{b,d}/(1+r)^t$, i.e, the value obtained after sending b to d within time period t, which is obtained from $p_{b,d}$ by applying a discount factor $r > 0$.
- Given a vector $x \in [0,1]^{\mathscr{C}}$ that is feasible for the U-PIT, we say that $P = \{c \in \mathscr{C} : x_c > 0\}$ is the *pit-limit* associated with x.
- Let (x^*, y^*) represent a minimal optimal solution to the PCPSP-C.

- For each $c \in \mathscr{C}$, let $x^1_c = \sum_{t\in\mathscr{T}} x^*_{c,t}$. We know x^1 is feasible for the U-PIT (discussed in Section 4.1). We say that P^{OPT} is the pit-limit associated with (x^*, y^*) if it is the pit-limit associated with the U-PIT feasible solution given by x^1.

Theorem 10 (Theorem 1.)**.** *Consider an instance of the PCPSP-C, as described in formulation* (1)*, such that $G \geq 0$ and $g \geq 0$. Let P^{OPT} and $P^{U\text{-}PIT}$ be the pit limits of minimal optimal solutions to the PCPSP-C and U-PIT, respectively. Then, $P^{OPT} \subseteq P^{U\text{-}PIT}$.*

Proof. Proof. Assume that the result does not hold. That is, assume that $P^{\text{OPT}} \setminus P^{\text{U-PIT}} \neq \emptyset$. The minimality of (x^*, y^*) is contradicted by constructing a different solution $(x^{**}, y^{**}) \leq (x^*, y^*)$ that is also optimal for the PCPSP-C. To achieve this, proceed as follows.

Let x^2 represent a minimal optimal solution to the U-PIT with $P^{\text{U-PIT}}$ its corresponding pit-limit. Define x^3 such that $x^3_c = \max\{x^1_c, x^2_c\}$ for all $c \in \mathscr{C}$. Observe that x^3 is feasible for the U-PIT. Because x^2 is an optimal solution of the U-PIT, $\bar{p}'(x^3 - x^2) \leq 0$.

For each $t \in \mathscr{T}$, define $\mu_t = \sum_{c\in P^{\text{OPT}}\setminus P^{\text{U-PIT}}} \bar{p}_c x^*_{c,t}$. Note that $\sum_{t\in\mathscr{T}} \mu_t = \bar{p}'(x^3 - x^2) \leq 0$.

Observe that for every $t \in \mathscr{T}$, since $p_{b,d,t} \leq p_{b,*}/(1+r)^t$, we have that:

$$\sum_{c\in P^{\text{OPT}}\setminus P^{\text{U-PIT}}} \sum_{b\in c} \sum_{d\in\mathscr{D}} p_{b,d,t} y^*_{b,d,t} \leq \frac{\mu_t}{(1+r)^t}.$$

Given that P^{OPT} is minimal and $G \geq 0$, we know that:

$$\sum_{t\in\mathscr{T}} \sum_{c\in P^{\text{OPT}}\setminus P^{\text{U-PIT}}} \sum_{b\in c} \sum_{d\in\mathscr{D}} p_{b,d,t} y^*_{b,d,t} > 0.$$

According to Lemma 9, there exists a $t_r \in \mathscr{T}$ such that $\sum_{t=t_r}^{T} \frac{\mu_t}{(1+r)^t} \leq 0$.

Define x^{**} such that:

$$x^{**}_{c,t} = \begin{cases} x^*_{c,t} & \text{if } c \in P^{\text{OPT}} \cap P^{\text{U-PIT}}, \\ x^*_{c,t} & \text{if } c \in P^{\text{OPT}} \setminus P^{\text{U-PIT}} \text{ and } t < t_r, \\ 0 & \text{otherwise.} \end{cases}$$

Likewise, define y^{**} such that

$$y^{**}_{b,d,t} = \begin{cases} y^*_{b,d,t} & \text{if } b \in c, \text{ and } c \in P^{\text{OPT}} \cap P^{\text{U-PIT}}, \\ y^*_{b,d,t} & \text{if } b \in c, \text{ and } c \in P^{\text{OPT}} \setminus P^{\text{U-PIT}}, \text{ and } t < t_r, \\ 0 & \text{otherwise.} \end{cases}$$

Since $G \geq 0$, we know that (x^{**}, y^{**}) is feasible for the PCPSP-C. By definition, we have that $(x^{**}, y^{**}) \leq (x^*, y^*)$. Finally, as $\sum_{t=t_r}^{T} |\mu_t| > 0$, we know that $(x^{**}, y^{**}) \neq (x^*, y^*)$. It remains to show that (x^{**}, y^{**}) is also optimal for the PCPSP-C.

Let $\mathscr{B}^1 = \{b \in c : \text{for some } c \in P^{\text{OPT}} \cap P^{\text{U-PIT}}\}$ and $\mathscr{B}^2 = \{b \in c : \text{for some } c \in P^{\text{OPT}} \setminus P^{\text{U-PIT}}\}$. It

follows that:

$$\begin{aligned}
\sum_{b\in\mathscr{B}}\sum_{d\in\mathscr{D}}\sum_{t\in\mathscr{T}} p_{b,d,t}y^{*}_{b,d,t} &= \sum_{b\in\mathscr{B}^1}\sum_{d\in\mathscr{D}}\sum_{t\in\mathscr{T}} p_{b,d,t}y^{*}_{b,d,t} + \sum_{b\in\mathscr{B}^2}\sum_{d\in\mathscr{D}}\sum_{t<t_r} p_{b,d,t}y^{*}_{b,d,t} + \sum_{b\in\mathscr{B}^2}\sum_{d\in\mathscr{D}}\sum_{t\geq t_r} p_{b,d,t}y^{*}_{b,d,t} \\
&= \sum_{b\in\mathscr{B}}\sum_{d\in\mathscr{D}}\sum_{t\in\mathscr{T}} p_{b,d,t}y^{**}_{b,d,t} + \sum_{b\in\mathscr{B}^2}\sum_{d\in\mathscr{D}}\sum_{t\geq t_r} p_{b,d,t}y^{*}_{b,d,t} \\
&= \sum_{b\in\mathscr{B}}\sum_{d\in\mathscr{D}}\sum_{t\in\mathscr{T}} p_{b,d,t}y^{**}_{b,d,t} + \sum_{t\geq t_r}\sum_{c\in P^{\text{OPT}}\setminus P^{\text{U-PIT}}}\sum_{b\in c}\sum_{d\in\mathscr{D}} p_{b,d,t}y^{*}_{b,d,t} \\
&\leq \sum_{b\in\mathscr{B}}\sum_{d\in\mathscr{D}}\sum_{t\in\mathscr{T}} p_{b,d,t}y^{**}_{b,d,t} + \sum_{t\geq t_r}\frac{\mu_t}{(1+r)^t} \\
&\leq \sum_{b\in\mathscr{B}}\sum_{d\in\mathscr{D}}\sum_{t\in\mathscr{T}} p_{b,d,t}y^{**}_{b,d,t}.
\end{aligned}$$

□

Chapter 4

Bin Packing with Time Lags

Orlando Rivera Letelier, François Clautiaux, Ruslan Sadykov

Manuscript Submitted

III

Bin Packing Problem with Time Lags

Orlando Rivera Letelier[1], François Clautiaux[2,3], and Ruslan Sadykov[3,2]

[1]Doctoral Program in Industrial Engineering and Operations Research, Universidad Adolfo Ibáñez, Av. Diagonal Las Torres 2640, Peñalolén, Santiago, Chile. 7941169
[2]IMB, Université de Bordeaux, 351 cours de la Libération, 33405 Talence, France
[3]Inria Bordeaux – Sud-ouest, 200 avenue de la Vieille Tour, 33405 Talence, France

June 30, 2021

Abstract

We introduce and motivate several variants of the bin packing problem where bins are assigned to time slots, and minimum and maximum lags are required between some pairs of items. We suggest two integer programming formulations for the general problem: a compact one, and a stronger formulation with an exponential number of variables and constraints. We propose a branch-cut-and-price approach which exploits the latter formulation. For this purpose, we devise separation algorithms based on a mathematical characterization of feasible assignments for two important special cases of the problem: when the number of possible bins available at each period is infinite, and when this number is limited to one, and time lags are non-negative. Computational experiments are reported for instances inspired from a real-case application of chemical treatment planning in vineyards, as well as for literature instances for special cases of the problem. The experimental results show the efficiency of our branch-cut-and-price approach, as it outperforms the compact formulation of newly proposed instances, and is able to obtain improved lower and upper bounds for literature instances.

1 Introduction

The bin packing problem is one of the most widely studied combinatorial optimization problems and has been known since the 1930s (Kantorovich, 1960). The classical bin packing problem (BPP) consists in assigning a set of items to a minimum possible number of identical capacitated bins. Formally, we are given a set of items $V = \{1, \dots, n\}$, and an unlimited quantity of identical bins with a positive capacity W. Each item $i \in V$ has a positive weight $w_i \leq W$. The goal is to find a packing of items into the minimum number of bins, such that the total weight of items packed in each bin does not exceed its capacity.

In this paper, we introduce the *bin packing problem with time lags* (BPPTL), which is a generalization of the BPP. Here the assignment of items to bins is performed over a discretized time horizon, and there are generalized precedence relations between items. In the BPPTL, in addition to the set of items, we have a directed valued graph $G = (V, A, l)$ representing precedence constraints between items, as well as a positive integer value L which defines the upper bound on the number of bins that can be used in each time period. If the value L is large enough, *i.e* if in any feasible solution the number of bins is always sufficient at each time period, for instance $L \geq n$, we simply write $L = \infty$. Each arc $(i, j) \in A$ has an associated time lag $l_{i,j} \in \mathbb{Z}$, which can be either positive or negative. Positive time lags serve to define the minimum possible distance (expressed in number of periods) between items, and negative time lags serve to define the maximum possible distance. The BPPTL consists in assigning each item $i \in V$ to bin $\bar{b}_i \in \{1, \dots, L\}$ and a time period $\bar{p}_i \in \mathbb{Z}_+$, such that bin capacity (1a) and time lag (1b) constraints are satisfied:

$$\sum_{i \in V:\, \bar{b}_i = b,\, \bar{p}_i = p} w_i \leq W \quad \forall b \in \{1, \dots, L\},\ \forall p \in \mathbb{Z}_+, \tag{1a}$$

$$\bar{p}_i + l_{i,j} \leq \bar{p}_j \quad \forall (i, j) \in A. \tag{1b}$$

The objective is to find a feasible assignment that minimizes the number of bin-period pairs (b, p) which have at least one item assigned to them, or to determine that such an assignment does not exist.

Our motivation for studying the BPPTL stems from applications in which some tasks should be performed repeatedly using resources that are available on a pay-per-use basis. For example, a given task should be performed six times in a given time period. A possible way to deal with recurrent tasks is to determine an a priori fixed time between two consecutive occurrences of the same task. This over-constrained setting limits the sharing of ressources, and may lead to expensive solutions. On the other hand, in many applications the decision maker does not impose a fixed frequency of task occurrences. Only a desired frequency is specified, which may be altered to some extent if this leads to a cost reduction. Thus, a more flexible approach is to impose positive and negative time lags between two consecutive occurrences of the same task. Positive lags determine the minimum elapsed time between two consecutive occurrences of a task, whereas negative lags determine the maximum elapsed time between them. This allows the decision maker to ensure a better usage of the resource. At the same time, solutions in which tasks are not distributed evenly over the time horizon are forbidden.

A specific case of such a problem can be observed when one performs the planning of phytosanitary treatments in a vineyard. The vineyard is divided into sectors, i.e. well-defined portions of the property that correspond to contiguous geographical areas. One is given a set of meta-requests for treatment, i.e. diseases against which the vineyard must be protected using phytosanitary products. For each appropriate sector-disease pair, a periodic treatment has to be performed. Each treatment consists of a sequence of tasks of a certain duration. A phytosanitary treatment protects the vines only for a certain period of time, so the same task has to be repeated over the time horizon. On the other hand, treatments should not be repeated too often due to regulations on the spread of phytosanitary products. Practically speaking, the same chemical product cannot be spread twice in a given period, whose length depends in the toxicity of the product. Therefore, two consecutive occurrences of the same treatment should be neither too far nor too close in time. Treatments are performed by identical rented vehicles with a fixed cost per day of usage. Vehicles are limited in the total duration of work in a day. A vehicle can perform several treatments in a day as long as their total duration does not exceed the maximum total duration. The time needed for a vehicle to go from one sector to another is negligible in comparison to the duration of treatments. Given the strict regulation on chemical product spreading, tasks are never fragmented and cannot be processed over two time periods.

This application can be modelled as the bin packing problem with time lags. Here, capacitated bins represent vehicles each having a maximum total duration. Items represent treatment tasks of different durations. The time lags graph consists of double-linked chains, each chain representing one periodic treatment. Two consecutive items in the chain represent two consecutive occurrences of a phytosanitary treatment. There are two arcs between consecutive items: one with a positive time lag representing the minimum number of days between treatment occurrences, and the other with a negative time lag representing the maximum number of days between occurrences. Two special source and sink nodes together with arcs between them serve to fix the planning time horizon. An example of such graph is depicted in Figure 1. The optimal solution for this example is depicted in Figure 2. It happens that this solution uses at most one bin per time period. In general, however, more than one bin can be used in the same period.

The BPPTL has some similarities with batch scheduling problems, in which jobs are first packed in batches, and then batches are scheduled within a time horizon. There are two main reasons why we attribute our problem to the class of bin packing problems. First, batches in scheduling are typically of different durations, whereas in our case the duration of a bin is always one period. Second, our objective function is typical for bin packing problems (minimizing the number of bins), whereas the total schedule length (makespan) objective is typical for scheduling problems. The difference here is that time periods without items or jobs are penalized in scheduling, but not penalized in the BPPTL.

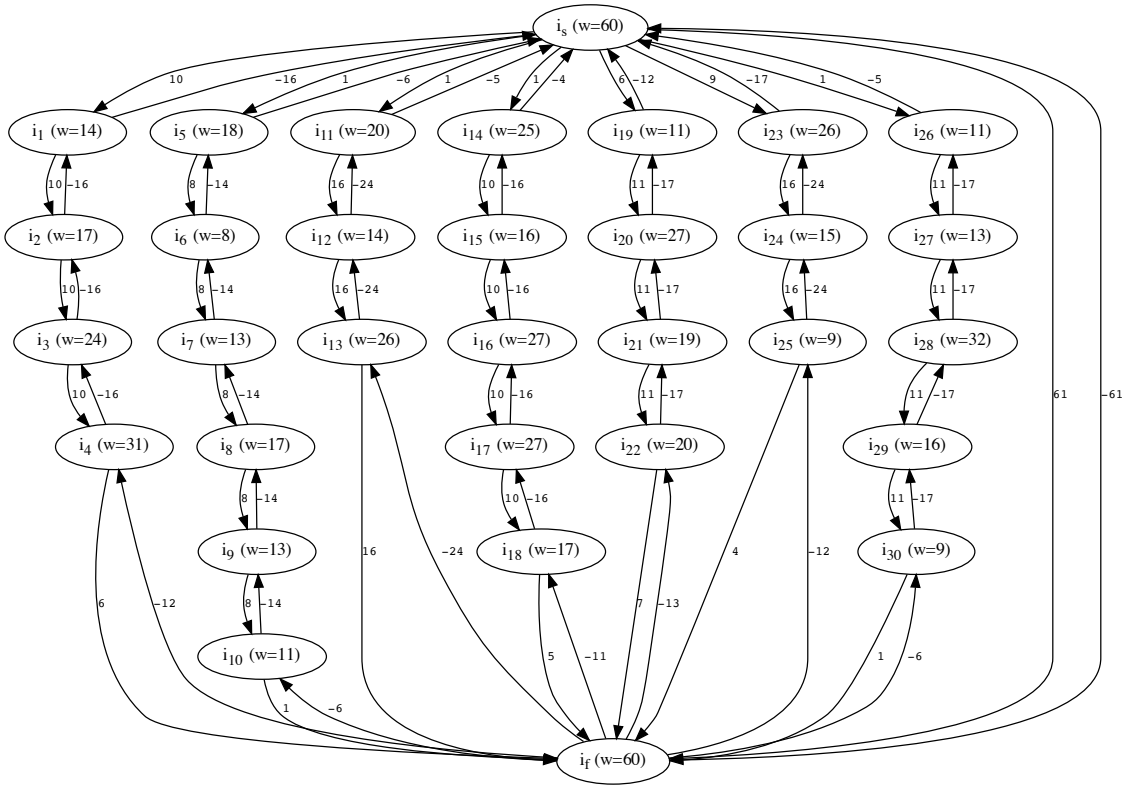


Figure 1: An example of the time lags graph for an instance of the phytosanitary treatments planning application

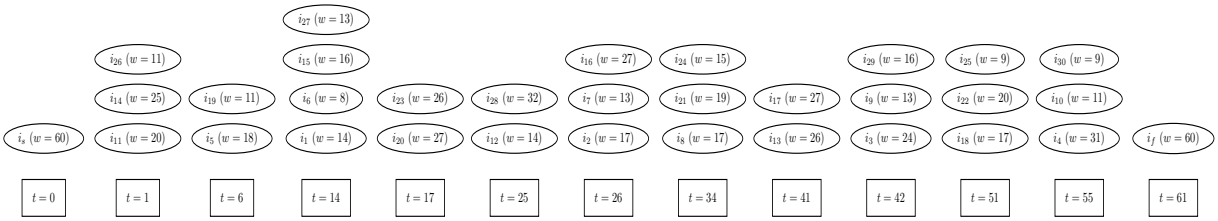


Figure 2: Solution of instance in Figure 1

1.1 Literature review

We now review the literature on the bin packing problems related to the BPPTL. The literature on the classical BPP is vast. Here we only present recent exact approaches to this problem. A comprehensive survey on exact methods for the BPP, including an original comparative computational study, was published by (Delorme et al., 2016). After that, other exact specialized algorithms for the BPP were proposed by (Delorme and Iori, 2020) and by (Wei et al., 2020). A generic BCP solver capable of solving exactly the BPP among other problems was proposed by (Pessoa et al., 2020).

The BPPTL is closely related to some generalizations of the BPP known in the literature. The bin packing problem with precedence constraints (BPP-P) is a special case of the BPPTL in which all time lags are unitary and $L = 1$. The first exact approach for the BPP-P was developed by (Dell'Amico et al., 2012). They proposed a large set of lower bounds, a variable neighborhood search, and a branch-and-bound algorithm. Later, (Pereira, 2016) suggested a dynamic programming-based heuristic and an exact enumeration procedure which uses several lower bounds and dominance rules to solve the BPP-P.

The simple assembly line balancing problem (SALBP-1) is a special case of the BPPTL in which all time lags are equal to zero and $L = 1$. The survey on simple assembly line balancing problems was presented by (Scholl and Becker, 2006). (Morrison et al., 2014) proposed an exact branch-bound-and-remember algorithm for the SALBP-1. (Pereira, 2015) analyzed different families of lower bounds for the SALBP-1 and also presented new lower bounds which improved on the best known bounds for open instances of the problem.

Recently a generalization of both the BPP-P and the SALBP-1 was considered by (Kramer et al., 2017). They introduced the bin packing problem with generalized precedence constraints (BPP-GP). In the BPP-GP considered by (Kramer et al., 2017), time lags can only be non-negative, $L = 1$, and the objective is to minimize the makespan, i.e. the latest time period that has non-empty bins. As shown in Section 4, the number of used bins and the makespan objectives are equivalent for the case when $L = 1$ and every time lag is either one or zero. However, if some time lags are strictly greater than one, then the BPP-GP is not a special case of the BPPTL. In (Kramer et al., 2017), the authors proposed an effective iterated local search algorithm for the BPP-GP with the makespan objective. They also applied preprocessing techniques and lower bounding procedures in order to estimate the quality of the solutions they obtained. Thus, many instances were solved to optimality.

Another related problem is the bin packing problem with conflicts (BPPC), in which a given undirected graph represents conflicts between items. Two adjacent items in the conflict graph cannot be put together in one bin. The objective function is the same as in the standard BPP, i.e. minimization of the number of bins used. The BPPC was introduced by (Gendreau et al., 2004). (Khanafer et al., 2010, 2012) proposed improved lower bounds for the BPPC based on dual-feasible functions, as well as tree-decomposition-based heuristics. Exact branch-and-price algorithms for the BPPC have been proposed in several papers (Fernandes Muritiba et al., 2010; Elhedhli et al., 2011; Sadykov and Vanderbeck, 2013; Wei et al., 2020).

In branch-and-price algorithms for the BPPC, the pricing problem is the knapsack problem with conflicts (KPC). This problem is interesting to us as it is a special case of the pricing problem considered in Section 3.2.2. The KPC has been solved either by MIP in (Fernandes Muritiba et al., 2010; Elhedhli et al., 2011), by dynamic programming and branch-and-bound algorithms in (Sadykov and Vanderbeck, 2013), or by a labelling algorithm as a resource-constrained shortest path problem in (Wei et al., 2020). Recently, improved combinatorial branch-and-bound algorithms for the KPC were proposed by (Bettinelli et al., 2017) and (Coniglio et al., 2020).

One more related class of problem is p-batch (or parallel batch) scheduling. In these problems, jobs have processing time and size. A set of jobs can form a batch if their total size does not exceed the batch capacity. When jobs have identical processing times, the p-batch scheduling problem with the makespan objective is equivalent to the standard bin packing problem, as shown by (Uzsoy, 1994). (Uzsoy, 1995) considered the p-batch scheduling problem with incompatible job families, which corresponds to the BPPC. However, only the polynomially solvable variant with identical item sizes was considered. The p-batch scheduling problem with different job processing times and sizes and the makespan objective was considered by (Dupont and Dhaenens-Flipo, 2002), who proposed an exact branch-and-bound algorithm for it. A branch-and-price algorithm for the same problem with parallel batching machines was suggested by (Rafiee Parsa et al.,

2010). Among recent papers dealing with batch scheduling problems with precedence constraints we can mention works by (Bilyk et al., 2014) and by (Emde et al., 2020). We are not aware of works on batch scheduling with generalized time lags. However, in the case of identical job processing times, release dates and dealines of jobs can be modelled with positive and negative time lags. We would like to conclude this part with a reminder that the main difference between our problem and batch scheduling is the objective function employed. Minimizing the number of bins corresponds to minimizing the number of batches in scheduling. In the presence of precedence constraints, the latter objective is different from ones traditionally used in scheduling like the makespan or tardiness-related criteria.

1.2 Contribution and outline

In this paper, as well as introducing and motivating the BPPTL, we present integer programming (IP) formulations for the problem. The first formulation is a compact one which involves one binary variable for every triple item-bin-time period. The second IP formulation with an exponential number of variables gives a relaxation to the BPPTL, similar to the one used in (Pereira, 2016) for the BPP-P. An exponential set of constraints is then introduced allowing us to turn the second formulation into an exact one for two important special cases of the BPPTL. The first one is when $L = \infty$. The second one is when $L = 1$, and time lags are non-negative. We present two ways of separating the introduced constraints: a MIP formulation and an enumeration algorithm. We also show how to take these constraints into account in the pricing problem, which turns into the knapsack problem with hard and soft conflicts (KPHSC), a generalization of the KPC. We formulate the KPHSC as a mixed integer program (MIP). Finally, for the two special cases of the BPPTL mentioned, we propose an exact branch-cut-and-price (BCP) algorithm which uses our cut separation routines, a branch-and-bound algorithm for solving the pricing problem, a strong diving primal heuristic, and the Ryan&Foster strong branching. We generate new instances for the BPPTL with $L = \infty$ inspired by the phytosanitary treatments planning application described above. We show that our BCP algorithm significantly outperforms a commercial MIP solver applied to the compact formulation for the problem. We also test our BCP algorithm on literature instances of the BPP-P, the SALBP-1, and the BPP-GP. We are able improve the best known lower and upper bounds for numerous instances. Thus, optimality is reached for the majority of open instances.

The article is organized as follows. In Section 2 we present our formulations of BPPTL. A BCP algorithm for the case $L = \infty$ is presented in Section 3. A modification of this algorithm for the case with $L = 1$ and non-negative time lags is given in Section 4. Computational results are presented in Section 5. We draw conclusions and discuss future work in Section 6.

2 Integer programming formulations

2.1 A compact formulation

Recall that L is the maximum number of bins per time period. Let $T \in \mathbb{Z}_+$ be an upper bound to the number of time periods required to assign all the bins in an optimal solution, $\mathcal{L} = \{1, \dots, L\}$, and $\mathcal{T} = \{1, \dots, T\}$. Let binary variable $x_{i,b,p}$ take value one if and only if item $i \in V$ is assigned to bin (b, p) with $b \in \mathcal{L}$ and $p \in \mathcal{T}$. Also let binary variable $u_{b,p}$ take value one if only if bin (b, p) with $b \in \mathcal{L}$ and $p \in \mathcal{T}$ has at least one item assigned to it. The BPPTL can then be formulated as the following IP.

$$\min \sum_{b\in\mathcal{L}}\sum_{p\in\mathcal{T}} u_{b,p} \tag{2a}$$

$$\text{s.t.} \sum_{b\in\mathcal{L}}\sum_{p\in\mathcal{T}} x_{i,b,p} = 1 \qquad \forall i \in V, \tag{2b}$$

$$\sum_{i\in V} w_i x_{i,b,p} \le W u_{b,p} \qquad \forall b \in \mathcal{L}, p \in \mathcal{T}, \tag{2c}$$

$$l_{i,j} + \sum_{p\in\mathcal{T}} p \cdot \sum_{b\in\mathcal{L}} x_{i,b,p} \le \sum_{p\in\mathcal{T}} p \cdot \sum_{b\in\mathcal{L}} x_{j,b,p} \qquad \forall (i,j) \in A, \tag{2d}$$

$$x_{i,b,p} \in \{0,1\} \qquad \forall i \in V, b \in \mathcal{L}, p \in \mathcal{T}, \tag{2e}$$

$$u_{b,p} \in \{0,1\} \qquad \forall b \in \mathcal{J}, p \in \mathcal{T}. \tag{2f}$$

In this formulation, constraints (2b) require each item to be assigned to exactly one bin. Constraints (2c) impose two conditions: i) if an item is assigned to a bin (b,p) then the corresponding variable $u_{b,p}$ is equal to one; ii) the items assigned to a bin must have a cumulative weight smaller than or equal to the capacity of the bin. Constraints (2d) guarantee that the time lags are satisfied. This formulation can be improved by disaggregating constraints (2c), or breaking some symmetries. We discuss these variants in Appendix 6.

Given a directed path in graph G from node i to node j, its length is the sum of the lags for the arcs in the path. Let $d(i,j)$ represent the length of the longest path from node i to node j in G. We convene $d(i,j) = -\infty$ if there is no path from i to j.

For each $i \in V$, let $es(i)$ and $ls(i)$ respectively represent the earliest and latest time periods in which i can be assigned to a bin. A natural pre-processing consists in fixing to zero all variables $x_{i,b,p}$ such that $i \in V$, $b \in \mathcal{L}$, and $p < es(i)$ or $p > ls(i)$. An approach to compute the values $es(i)$ and $ls(i)$ consists in adding two extra nodes to the graph G: the source node s fixed to artificial time period 0 and the sink node f fixed to artificial time period $T+1$. Then for each item $i \in V$ we add arcs (s,i) and (i,f) with time lag one. Then $es(i) = d(s,i)$ and $ls(i) = T + 1 - d(i,f)$.

2.2 A Dantzig-Wolfe reformulation

A natural Dantzig-Wolfe reformulation of model (2) consists in convexifying constraints (2c). Given $p \in \mathcal{T}$, let $\mathcal{B}_p$ be the collection of sets of items which may be assigned to a bin in time period p: $B \in \mathcal{B}_p$ if and only if $\sum_{i\in B} w_i \le W$, and $es(i) \le p \le ls(i)$, $\forall i \in B$. For each $B \in \mathcal{B}_p$, we define a binary variable λ_B taking value one if and only if set B of items occupies one bin in period p in the solution. We denote by $\mathbb{1}_B$ the indicator function of B: $\mathbb{1}_B(i) = 1$ if $i \in B$, and $\mathbb{1}_B(i) = 0$ otherwise, for all $i \in V$. The reformulation is then the following.

$$\min \sum_{p\in\mathcal{T}}\sum_{B\in\mathcal{B}_p} \lambda_B \tag{3a}$$

$$\text{s.t.} \sum_{p\in\mathcal{T}}\sum_{B\in\mathcal{B}_p} \mathbb{1}_B(i)\lambda_B = 1 \qquad \forall i \in V, \tag{3b}$$

$$l_{i,j} + \sum_{p\in\mathcal{T}} p \cdot \sum_{B\in\mathcal{B}_p} \mathbb{1}_B(i)\lambda_B \le \sum_{p\in\mathcal{T}} p \cdot \sum_{B\in\mathcal{B}_p} \mathbb{1}_B(j)\lambda_B \qquad \forall (i,j) \in A, \tag{3c}$$

$$\sum_{B\in\mathcal{B}_p} \lambda_B \le |\mathcal{L}| \qquad \forall p \in \mathcal{T}, \tag{3d}$$

$$\lambda_B \in \{0,1\} \qquad \forall p \in \mathcal{T}, B \in \mathcal{B}_p. \tag{3e}$$

The linear relaxation of model (3) can be solved by the column generation approach in which the pricing problem is the standard $0-1$ knapsack problem. It is known however (Pereira, 2016) that precedence constraints (3c) do not affect the value of the lower bound obtained by such relaxation for the BPP-P and

the SALBP-1. We also show below in our computational experiments that the branch-and-price algorithm that solves formulation (3) does not have a good performance because of a bad quality of the lower bound given by the linear relaxation.

Thus in the following section, we propose an alternative extended formulation for the BPPTL, which provides a better relaxation. Moreover, it allows us to exploit the symmetry related to time periods. Then, in the following two sections we will show how to turn this relaxation into an exact formulation for two important special cases of the problem. This is done by defining an exponential family of constraints that cut off infeasible solutions of the relaxation.

2.3 An alternative relaxation

In any feasible solution of the BPPTL, a set B of items assigned to the same bin must satisfy the following conditions:

$$\sum_{i \in B} w_i \leq W \tag{4a}$$

$$d(i,j) \leq 0 \text{ and } d(j,i) \leq 0 \qquad \forall i,j \in B. \tag{4b}$$

Constraint (4a) follows from (1a) and constraints (4b) follow from (1b). These two conditions state that the bin capacity constraint has to be satisfied, and that (transitive) lag constraints forbid some items from being packed in the same bin.

Let $\mathcal{B}$ be the collection of all possible non-empty sets B of items satisfying constraints (4a) and (4b). For each $B \in \mathcal{B}$, we define a binary variable λ_B taking value one if and only if set B of items occupies one bin in the solution. A relaxation of the BPPTL then can be formulated as follows.

$$\min \sum_{B \in \mathcal{B}} \lambda_B \tag{5a}$$

$$\text{s.t.} \sum_{B \in \mathcal{B}} \mathbb{1}_B(i) \lambda_B = 1 \qquad \forall i \in V, \tag{5b}$$

$$\lambda_B \in \{0,1\} \qquad \forall B \in \mathcal{B}. \tag{5c}$$

In this formulation the objective function is to minimize the number of bins selected in the solution. This relaxation has a similar structure to the one used for the simple assembly line balancing problem (Pereira, 2015) and the bin packing problem with precedences (Pereira, 2016), albeit with a different definition of $\mathcal{B}$. Moreover, integrality constraints (5c) are relaxed to the linear constraints $0 \leq \lambda_B \leq 1 \; \forall B \in \mathcal{B}$ in (Pereira, 2015, 2016).

A feasible solution $\bar{\lambda}$ for model (5) does not necessarily define a feasible solution for the BPPTL, since it is not always possible to assign bins in set $\{B \in \mathcal{B} : \; \bar{\lambda}_B = 1\}$ to time periods while satisfying the time lag constraints. This can be seen in the example illustrated in Figure 3. In this example, a feasible solution to (5) has items $\{A, B\}$ assigned to one bin and items $\{C, D\}$ to another. However, it is not possible to assign those bins to time periods satisfying the time lag constraints. In fact, item A must be assigned before D, and item C must be assigned before B. One can see that the issue stems from the fact that when items are assigned to bins, lags between items become lags between bins, and cycles of positive length may appear. This concept of cycles between bins is formalized in Section 3, and is used to characterize infeasible solutions.

3 The case with unlimited number of available bins

In this section we assume that $L = \infty$, i.e. there is no restriction on the number of bins that can be assigned to a time period. We denote this variant of the problem as the BPPTL$^\infty$. We propose in this section a branch-cut-and-price algorithm based on relaxation (5) to solve this variant to optimality. In Section 3.1,

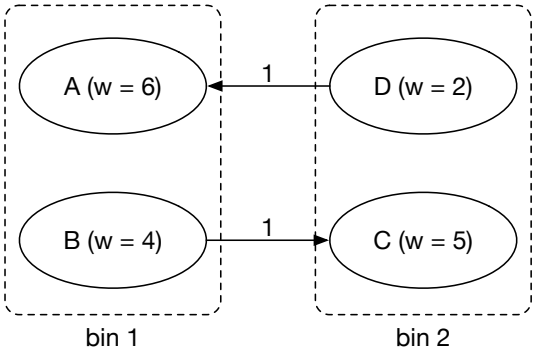


Figure 3: Example of instance in which a feasible solution to relaxation (5) does not define a feasible solution for the BPPTL. Here $W = 10$.

we give a characterization of feasible solutions of the BPPTL$^\infty$ expressed by constraints involving only variables λ_B. Later, in Section 3.2, we describe the components of the branch-cut-and-price algorithm, i.e. separation algorithms for the cuts, an IP formulation and a dedicated branch-and-bound for the pricing problem, primal heuristic and branching.

3.1 Characterization of valid assignments

Let $\mathcal{P}$ be a partition of V. We denote by $\mathcal{P}(i)$ the element $B \in \mathcal{P}$ such that $i \in B$. For each $B, B' \in \mathcal{P}$, we define $\bar{d}(B, B') = \max\{d(i,j) : i \in B, j \in B'\}$ as the maximum distance (in terms of time lags) between an item in B and an item in B'. We introduce the notion of *aggregated graph*, which is useful for defining the necessary and sufficient conditions for a feasible solution to (5) to be feasible for the BPPTL.

Definition 1. Let $G = (V, A, l)$ be a directed valued graph, and $\mathcal{P}$ be a partition of V. The *aggregated graph* of G induced by $\mathcal{P}$ is the valued digraph $G^{\mathcal{P}} = (\mathcal{P}, A^{\mathcal{P}}, \bar{d})$, where $\mathcal{P}$ is the set of nodes in the graph, $A^{\mathcal{P}} \coloneqq \left\{(B, B') \subseteq \mathcal{P} \times \mathcal{P} : \bar{d}(B, B') > -\infty\right\}$ is the set of arcs in the graph, and $\bar{d}(B, B')$ is the length of arc $(B, B') \in A^{\mathcal{P}}$.

We now show that whenever the aggregated graph induced by a partition is acyclic, the elements of the partition can be assigned to time periods satisfying the time lag constraints.

Proposition 1. *Let $G = (V, A, l)$ be a directed valued graph and $\mathcal{P}$ be a partition of V. Also let $G^{\mathcal{P}} = (\mathcal{P}, A^{\mathcal{P}}, \bar{d})$ be the aggregated graph of G induced by $\mathcal{P}$. There is a mapping $\tau : \mathcal{P} \longrightarrow \mathbb{Z}_+$, such that for each $(i,j) \in A$, $\tau(\mathcal{P}(i)) + l_{i,j} \leq \tau(\mathcal{P}(j))$ if and only if there is no cycle of positive length in $G^{\mathcal{P}}$.*

Proof. Proof. The proof relies on the fact that graph G has no cycle of positive length if and only if a mapping $\tau : V \longrightarrow \mathbb{Z}_+$ exists, such that for each $(i,h) \in A$, $\tau(i) + l_{i,h} \leq \tau(j)$ (see Lemma 1 in Appendix). Consider this mapping τ for the graph $G^{\mathcal{P}}$. Then for each $(B, B') \in A^{\mathcal{P}}$, it holds that $\tau(B) + \bar{d}(B, B') \leq \tau(B')$. But for each $i, j \in A$, we have $l_{i,j} \leq d(i,j)$, and since $\bar{d}(B, B') = \max\{d(i,j) : i \in B, j \in B'\}$, then $d(i,j) \leq \bar{d}(\mathcal{P}(i), \mathcal{P}(j))$. This implies that for each $(i,j) \in A$, $\tau(\mathcal{P}(i)) + l_{i,j} \leq \tau(\mathcal{P}(i)) + \bar{d}(\mathcal{P}(i), \mathcal{P}(j)) \leq \tau(\mathcal{P}(j))$. □

We call a partition $\mathcal{P} \subseteq \mathcal{B}$ *suitable* if graph $G^{\mathcal{P}}$ does not have a directed cycle of positive length, and we say that $\mathcal{P}$ is *unsuitable* otherwise. We denote by $\mathcal{N}$ the family of *unsuitable* partitions of V.

The partition of items induced by a feasible solution of an instance of the BPPTL$^\infty$ must be *suitable*, because the assignment of bins to time periods implies that there are no positive cycles in the aggregated graph. But Proposition 1 also shows that any *suitable* partition $\mathcal{P} \subseteq \mathcal{B}$ induces a feasible solution for the instance, and the proof also gives a procedure to recover the actual solution (the time of each bin) from the partition by computing the longest distance from an artificial source node to each node in the aggregated graph, which can be done in time $\mathcal{O}\left(|\mathcal{P}|\left|A^{\mathcal{P}}\right|\right)$.

A classical way to ensure feasibility is to add the following set of constraints (known as *no-good cuts*) to formulation (5):

$$\sum_{B \in \mathcal{P}} \lambda_B \leq |\mathcal{P}| - 1 \qquad \forall \mathcal{P} \in \mathcal{N}. \tag{6}$$

One of our contributions is to avoid using these constraints by finding a better characterization of feasible solutions.

Formulation (5) together with constraints (6) defines an exact formulation of the BPPTL$^\infty$. However, each constraint in (6) forbids only one *unsuitable* partition, and the coefficient of a variable λ_B in these constraints depends on set B. This makes dynamic generation of variables λ very difficult. We now introduce an alternative to constraints (6), which defines a computationally tractable equivalent formulation.

The alternative constraints rely on the fact that cycles in the aggregated graphs are related to lags between pairs of items. Consider a partition $\mathcal{P}$ such that the induced aggregated graph $G^{\mathcal{P}}$ has a cycle of positive length $(B_1, B_2, \ldots, B_R)$, and denote $B_{R+1} = B_1$. For each arc (B_r, B_{r+1}) in the cycle there are elements $t_r \in B_r$ and $h_{r+1} \in B_{r+1}$ such that $\bar{d}(B_r, B_{r+1}) = d(t_r, h_{r+1})$. If we consider the set $\{(h_1, t_1), \ldots, (h_R, t_R)\}$ of pairs of items, any partition where each of these pairs is in the same bin induces an aggregated graph with a positive length cycle. Here notations t and h stand for the "tail" and "head" of the corresponding arc.

Figure 4 presents an example of a partition that induces a positive cycle in the aggregated graph. Any partition in which pairs of nodes (h_1, t_1), (h_2, t_2) and (h_3, t_3) each belong to the same block of the partition, induces the aggregated graph with a positive length cycle.

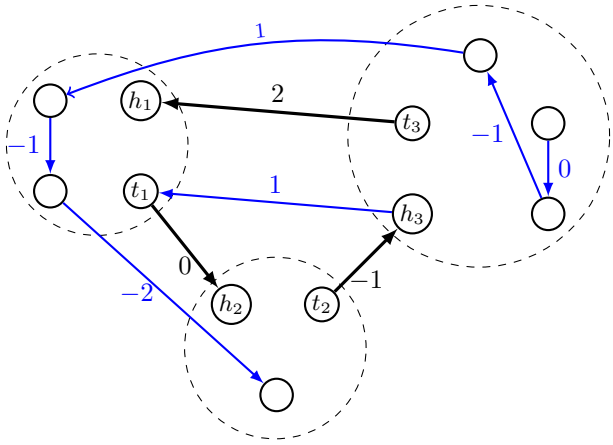


Figure 4: Example of partition that induces a positive cycle. Here $(h_1, h_2, h_3, t_1, t_2, t_3) \in \mathcal{C}$.

We now define the collection of all sets of pairs which induce a positive length cycle in the aggregated graph, and thus an infeasible solution.

Definition 2. For each $R \in \mathbb{N}$, $R \geq 2$, we define $\mathcal{C}_R \subseteq V^{2R}$ as the collection of all tuples $(h_1, \ldots, h_R, t_1, \ldots, t_R)$ that satisfy the following properties.

- All elements $h_1, \ldots, h_R, t_1, \ldots t_R$ are different.
- For each $r \in \{1, \ldots, R\}$, $d(h_r, t_r) \leq 0$ and $d(t_r, h_r) \leq 0$.
- $\sum_{r=1}^{R} d(t_r, h_{r+1}) \geq 1$, where $h_{R+1} = h_1$.

We also denote $\mathcal{C} = \bigcup_{R \geq 2} \mathcal{C}_R$, and for each tuple $C = (h_1, \ldots, h_R, t_1, \ldots, t_R) \in \mathcal{C}$ we define as $\mathcal{F}_C$ the set of pairs of items which constitute C: $\mathcal{F}_C = \big\{\{h_1, t_1\}, \{h_2, t_2\}, \ldots, \{h_R, t_R\}\big\}$.

We can now characterize an *unsuitable* partition as a partition that contains a forbidden set of pairs.

Proposition 2. *A partition $\mathcal{P} \subseteq \mathcal{B}$ is* unsuitable *if and only if $C \in \mathcal{C}$ exists, such that for each pair $\{h_r, t_r\} \in \mathcal{F}_C$, $\mathcal{P}$ contains an element B_r containing both h_r and t_r: $\{h_r, t_r\} \subseteq B_r \in \mathcal{P}$.*

Proof. Proof. Let $\mathcal{P} \subseteq \mathcal{B}$ be a partition of V, and let $C \in \mathcal{C}_R$, $R \geq 2$, be such that each $\{h_r, t_r\} \in \mathcal{F}_C$ is contained in $B_r \in \mathcal{P}$ for each $r \in \{1, \ldots, R\}$. Then, by definition of $\bar{d}$, for each $r \in \{1, \ldots, R\}$ we have $\bar{d}(B_r, B_{r+1}) \geq d(t_r, h_{r+1})$, where $B_{R+1} = B_1$. This implies that $\sum_{r=1}^{R} \bar{d}(B_r, B_{r+1}) \geq 1$, and then there is a circuit in $(G^{\mathcal{P}}, \bar{d})$ of positive length. Since the circuit can be decomposed into multiple cycles, and the length of the circuit is equal to the sum of the length of those cycles, one of those cycles must be of positive length, and then $\mathcal{P}$ must be *unsuitable*.

Assume now that $\mathcal{P}$ is *unsuitable*, and consider a cycle of positive length in graph $G^{\mathcal{P}}$ with the minimum possible number of nodes. Let R be the number of nodes of that cycle, and let the nodes of the cycle be $B_1, B_2, \ldots, B_R$.

For each arc (B_r, B_{r+1}) in this cycle, let $t_r \in B_r$ and $h_{r+1} \in B_{r+1}$ be such that $d(t_r, h_{r+1}) = \bar{d}(B_r, B_{r+1})$, where $B_{R+1} = B_1$. Thus we have $\{h_r, t_r\} \subseteq B_r$, and since the cycle in the aggregated graph has positive length, we have $\sum_{r=1}^{R} d(t_r, h_{r+1}) = \sum_{r=1}^{R} \bar{d}(B_r, B_{r+1}) \geq 1$.

We now show that all these elements $(h_1, \ldots, t_R)$ are different. Since all B_r are different elements of a partition of V, for each $r, r' \in \{1, \ldots, R\}$ with $r \neq r'$ it holds $h_r \neq h_{r'}$, $t_r \neq t_{r'}$, and $h_r \neq t_{r'}$. If $h_r = t_r$ then $\bar{d}(B_{r-1}, B_{r+1}) \geq d(t_{r-1}, h_r) + d(t_r, h_{r+1}) = \bar{d}(B_{r-1}, B_r) + \bar{d}(B_r, B_{r+1})$, and then removing B_r from the cycle would still give a cycle of positive length, which contradicts the minimality of the cycle considered. Then, all the elements $h_1, \ldots, t_R$ are different.

Finally, since $B_r \in \mathcal{B}$, then $d(h_r, t_r) \leq 0$ and $d(t_r, h_r) \leq 0$. □

The following reformulation is valid for the BPPTL$^{\infty}$ due to Proposition 2.

$$\min \sum_{B \in \mathcal{B}} \lambda_B \tag{7a}$$

$$\text{s.t.} \sum_{B \in \mathcal{B}} \mathbb{1}_B(i) \lambda_B = 1 \qquad \forall i \in V, \tag{7b}$$

$$\sum_{r=1}^{R} \sum_{\substack{B \in \mathcal{B}, \\ \{h_r, t_r\} \subseteq B}} \lambda_B \leq R - 1 \qquad \forall R \geq 2, \forall (h_1, \ldots, t_R) \in \mathcal{C}_R, \tag{7c}$$

$$\lambda_B \in \{0, 1\} \qquad \forall B \in \mathcal{B}. \tag{7d}$$

3.2 The branch-cut-and-price algorithm

We develop a branch-cut-and-price approach to solve formulation (7). At each node of the branch-and-bound tree the linear relaxation is solved by the following column and cut generation procedure.

- *Step 0:* Start with an initial set of variables λ and constraints (7c).
- *Step 1:* Solve the restricted master problem (RMP). The RMP is the linear relaxation of (7) with the current restricted subset of columns and constraints.
- *Step 2:* Solve the pricing problem which looks for a variable λ_B with $B \in \mathcal{B}$ with negative reduced cost. If such a column is found, add λ_B to the RMP and go back to *Step 1*.
- *Step 3:* Solve the separation problem. The separation problem consists in finding a constraint (7c) violated by the current solution of the RMP.

Approaches for solving the separation problem are presented in Section 3.2.1. The MIP formulation for the pricing problem is proposed in Section 3.2.2. Other components of the BCP algorithm are described in Section 3.2.4.

3.2.1 Separation.

Given a solution $\bar{\lambda} = \left(\bar{\lambda}_B\right)_{B\in\mathcal{B}}$ that satisfies (7b), we are looking for a constraint in (7c) that is violated by $\bar{\lambda}$.

If $\bar{\lambda}$ satisfies (7d), i.e. it is an integer solution, it induces a partition $\mathcal{P}$ of the items, and finding a violated constraint in (7c) is equivalent to finding a cycle of positive length in the aggregated graph $G^{\mathcal{P}}$, according to Proposition 2. We first construct the aggregated graph $G^{\mathcal{P}}$. Then we compute the longest distance between each pair of nodes in this graph, which can be done in time $\mathcal{O}\left(|\mathcal{P}|^3\right)$ with the Floyd-Warshall algorithm. Finally, we look for a cycle of a positive length by checking if the distance from any node to itself is positive. In the event that a cycle of positive length is found, the proof of Proposition 2 describes the procedure to find the tuple $(h_1, \ldots, t_R) \in \mathcal{C}$ and to generate the associated violated constraint.

Separating integer solutions is enough for the algorithm to be valid. However, in order to strengthen the relaxation, we also propose procedures to separate fractional solutions. To do so, we define an auxiliary directed multigraph in which finding a cycle with some specific properties is equivalent to finding a violated constraint in (7c).

Let $\left(\bar{\lambda}_B\right)_{B\in\mathcal{B}}$ be a solution satisfying (7b) but not necessarily (7d). For each $i, j \in V$ with $i \neq j$ we define the value $\xi_{i,j} = \sum_{\substack{B\in\mathcal{B},\\ \{i,j\}\subseteq B}} \bar{\lambda}_B$. Then we construct multigraph $\mathcal{G} = (V, \mathcal{A})$ with two labels in each arc. Each arc $a \in \mathcal{A}$ is represented by a tuple $(h(a), t(a), l(a), v(a)) \in V \times V \times \mathbb{Z} \times \mathbb{R}$, where $h(a)$ is the head of the arc, and $t(a)$ is the tail of the arc. The set of arcs is composed of two subsets, $\mathcal{A} = \mathcal{A}_G \cup \mathcal{A}_\lambda$, where these subsets are defined as follows:

- For each $(i, j) \in A$, arc a in which $h(a) = i$, $t(a) = j$, $l(a) = d(i, j)$ and $v(a) = 0$ belongs to $\mathcal{A}_G$.
- For each pair $\{i, j\} \subseteq V$ such that $i \neq j$ and $\xi_{i,j} > 0$, arc a in which $h(a) = i$, $t(a) = j$, $l(a) = 0$, and $v(a) = 1 - \xi_{i,j}$ belongs to $\mathcal{A}_\lambda$.

There is one arc in $\mathcal{A}_G$ for each arc in the original graph G, and two arcs in $\mathcal{A}_\lambda$, one in each direction, for each pair of nodes that are contained in any $B \in \mathcal{B}$ such that $\bar{\lambda}_B > 0$. Note that in this graph there might be two arcs with the same head and tail, one of them in $\mathcal{A}_G$ and the other in $\mathcal{A}_\lambda$.

Proposition 3. *Let $\left(\bar{\lambda}_B\right)_{B\in\mathcal{B}}$ be a solution satisfying* (7b)*. Then it satisfies all constraints in* (7c) *if and only if multigraph $\mathcal{G} = (V, \mathcal{A})$ constructed from $\bar{\lambda}$ has no cycle $(a_1, a_2, \ldots, a_K)$ such that $\sum_{k=1}^{K} l(a_k) \geq 1$ and $\sum_{k=1}^{K} v(a_k) < 1$.*

Proof. Proof. ($\Rightarrow$) Assume first that $\bar{\lambda}$ violates a constraint in (7c), then there is $R \geq 2$ and $(h_1, \ldots, t_R) \in \mathcal{C}_R$ such that

$$\sum_{r=1}^{R} \sum_{\substack{B\in\mathcal{B},\\ \{h_r,t_r\}\subseteq B}} \bar{\lambda}_B > R - 1.$$

This is equivalent to

$$1 > \sum_{r=1}^{R} \left(1 - \sum_{\substack{B\in\mathcal{B},\\ \{h_r,t_r\}\subseteq B}} \bar{\lambda}_B\right) = \sum_{r=1}^{R} \left(1 - \xi_{h_r,t_r}\right).$$

With this we define the following cycle with $K = 2R$ arcs in graph $\mathcal{G}$. For each $r \in \{1, \ldots, R\}$, we take arc $a_{2r-1} \in \mathcal{A}_\lambda$ such that $t(a_{2r-1}) = h_r$ and $h(a_{2r-1}) = t_r$, and we take arc $a_{2r} \in \mathcal{A}_G$ such that $t(a_{2r}) = t_r$ and $h(a_{2r}) = h_{r+1}$. We have $v(a_{2r-1}) = 1 - \xi_{h_r,t_r}$ and $v(a_{2r}) = 0$. Summing over all arcs in the cycle, we obtain $\sum_{k=1}^{2R} v(a_k) < 1$. We also have $l(a_{2r-1}) = 0$ and $l(a_{2r}) = d(t_r, h_{r+1})$. Then by definition of $\mathcal{C}_R$ we have $\sum_{k=1}^{2R} l(a_k) = \sum_{r=1}^{R} d(t_r, h_{r+1}) \geq 1$.

($\Leftarrow$) Assume now that there are cycles $(a_1, a_2, \ldots, a_K)$ in graph $\mathcal{G}$ satisfying both $\sum_{k=1}^{K} l(a_k) \geq 1$ and $\sum_{k=1}^{K} v(a_k) < 1$. Among these cycles we take one with the minimum number K of arcs. We now show that the arcs in this cycle are alternating between arcs of $\mathcal{A}_G$ and $\mathcal{A}_\lambda$. Let a and a' be two consecutive arcs in the cycle, with $h(a) = i$, $t(a) = j$, $h(a') = j$ and $t(a') = k$. Now consider two cases.

i) If both arcs belong to $\mathcal{A}_G$, then we can replace both arcs by arc (i, k) in $\mathcal{A}_G$ since $d(i, k) \geq d(i, j) + d(j, k)$.

ii) If both arcs belong to $\mathcal{A}_\lambda$, then we have

$$\begin{aligned}
\xi_{i,j} + \xi_{j,k} &= \sum_{\substack{B \in \mathcal{B}, \\ \{i,j\} \subseteq B}} \bar{\lambda}_B + \sum_{\substack{B \in \mathcal{B}, \\ \{j,k\} \subseteq B}} \bar{\lambda}_B \\
&= \underbrace{\sum_{\substack{B \in \mathcal{B}, \\ \{i,j\} \subseteq B, k \notin B}} \bar{\lambda}_B + \sum_{\substack{B \in \mathcal{B}, \\ \{i,j\} \subseteq B, k \in B}} \bar{\lambda}_B + \sum_{\substack{B \in \mathcal{B}, \\ \{j,k\} \subseteq B, i \notin B}} \bar{\lambda}_B}_{\leq \sum_{\substack{B \in \mathcal{B}, \\ j \in B}} \bar{\lambda}_B} + \underbrace{\sum_{\substack{B \in \mathcal{B}, \\ \{j,k\} \subseteq B, i \in B}} \bar{\lambda}_B}_{\leq \sum_{\substack{B \in \mathcal{B}, \\ \{i,k\} \subseteq B}} \bar{\lambda}_B} \\
&\leq 1 + \xi_{i,k}.
\end{aligned}$$

The first consequence of this inequality is that $\xi_{i,k} > 0$, and arc (i, k) belongs to $\mathcal{A}_\lambda$, as otherwise we would have $v(a) + v(a') \geq 1$. The second consequence is that $v(a) + v(a') = 2 - \xi_{i,j} - \xi_{j,k} \geq 1 - \xi_{i,k}$, and value v for arc (i, k) is not larger than $v(a) + v(a')$. Then we can replace arcs a and a' by arc (i, k).

In both cases, there is a contradiction with the minimality of the number of arcs in the cycle. Thus we can conclude that the arcs are alternating between $\mathcal{A}_G$ and $\mathcal{A}_\lambda$. Thus K must be even, and we can denote $K = 2R$. Without loss of generality we assume $a_1 \in \mathcal{A}_\lambda$. We define $h_r = t(a_{2r-1})$ and $t_r = h(a_{2r-1})$ for each $r \in \{1, \ldots, R\}$. Now we are going to show that $(h_1, \ldots, t_R) \in \mathcal{C}_R$.

The minimality of the cycle implies that all nodes $(h_1, \ldots, t_R)$ are different. Since arc (h_r, t_r) belongs to $\mathcal{A}_\lambda$, there must be some $B \in \mathcal{B}$ with $\bar{\lambda}_B > 0$ such that $\{h_r, t_r\} \subseteq B$. By definition of $\mathcal{B}$, this implies that $d(h_r, t_r) \leq 0$ and $d(t_r, h_r) \leq 0$. Finally, since $a_{2r-1} \in \mathcal{A}_\lambda$, $l(a_{2r-1}) = 0$, then $\sum_{r=1}^{R} l(a_{2r}) \geq 1$, which implies that $\sum_{r=1}^{R} d(t_r, h_{r+1}) \geq 1$. □

The first approach to find such a cycle in multigraph $\mathcal{G} = (V, \mathcal{A})$ is based on an IP. Let y_a be a binary variable which determines whether arc $a \in \mathcal{A}$ participates in the cycle or not. Consider the following IP formulation.

$$\min \sum_{a \in \mathcal{A}} v(a) y_a \tag{8a}$$

$$\text{s.t.} \sum_{a \in \mathcal{A}} l(a) y_a \geq 1, \tag{8b}$$

$$\sum_{a \in \mathcal{A} : h(a) = i} y_a = \sum_{a \in \mathcal{A} : t(a) = i} y_a \qquad \forall i \in V, \tag{8c}$$

$$y_a \in \{0, 1\} \qquad \forall a \in \mathcal{A}. \tag{8d}$$

It can be seen that there is a cycle $(a_1, a_2, \ldots, a_K)$ in $\mathcal{G}$ such that $\sum_{k=1}^{K} l(a_k) \geq 1$ and $\sum_{k=1}^{K} v(a_k) < 1$ if and only if the optimal value of (8) is strictly smaller than one. Constraints (8c) force the solution to be a collection of cycles. Due to constraints (8b), at least one cycle in this collection must have a positive sum of values l. If the objective function is strictly less than one, then any cycle in the collection has the sum of values v strictly less than one.

Generating several constraints in each separation round helps to improve convergence of the cut generation procedure. This can be achieved by solving formulation (8) multiple times. After each solving, a constraint is added to avoid selecting the same cut again. Suppose that cycle $(a_1, a_2, \ldots, a_K)$ is found after solving the IP. Without loss of generality, let $a_1 \in \mathcal{A}_G$ and $R = 2K$. Then the constraint $\sum_{r=1}^{R} y_{a_{2r}} \leq R - 1$ is added to (8), which is resolved again.

We now describe the second approach to find a desired cycle in multigraph $\mathcal{G} = (V, \mathcal{A})$. Given a set A' of arcs, we denote $v(A') = \sum_{a \in A'} v(a)$ and $l(A') = \sum_{a \in A'} l(a)$, and given two nodes $i, j \in V$ we denote $v_{i,j}$ the minimum possible sum of v values of a path from i to j. The approach consists in a partial enumeration of cycles C in $\mathcal{G}$ satisfying $v(C) < 1$ and $l(C) \geq 1$.

We perform one iteration for each node $i \in V$. In this iteration, we try to find cycles C containing node i such that $v(C) < 1$ and $l(C) \geq 1$. Then we mark node i to exclude cycles containing i from the search in subsequent iterations. In each iteration, we use a recursive procedure which takes the current partial path from node i to current node j, containing set C of arcs and set $V_C \cup \{j\}$ of nodes. This procedure iterates over arcs $a \in \delta^+(j)$. If head $h(a)$ of arc a has been marked, arc a is ignored. Otherwise, if node $h(a)$ is contained in V_C then a subset C' of the arcs in $C \cup \{a\}$ forms a cycle. In this case, if conditions $v(C') < 1$ and $l(C') \geq 1$ held, then we store cycle C' inducing a violated inequality. If node $h(a)$ is neither blocked nor contained in V_C we check conditions $v(C) + v(a) + v(a') < 1$ and $l(C) + l(a) + l(a') \geq 1$, where $a' = (h(a), i)$. If none of these conditions is true, it is unlikely that arcs in $C \cup \{a\}$ are contained in a cycle inducing a valid inequality, and arc a is ignored. If both conditions are true, we augment the partial path with arc a and recursively call the same procedure. The separation algorithm by partial enumeration is formally presented in Algorithm 1.

Function `DFSRecursion`(i, j, V_C, C)**:**
  for $a \in \delta^+(j)$ **do**
    if $h(a)$ *is marked* **then continue**
    if $h(a) \in V_C$ **then**
      Find $C' \subseteq C \cup \{a\}$ that forms a cycle.
      if $v(C') < 1$ **and** $l(C') \geq 1$ **then** store cycle C'
      continue
    end
    $a' \longleftarrow (h(a), i)$
    if $v(C) + v(a) + v(a') \geq 1$ **or** $l(C) + l(a) + l(a') \leq 0$ **then continue**
    `DFSRecursion`(i, $h(a)$, $V_C \cup \{j\}$, $C \cup \{a\}$)
  end
end

Function `Main`($\mathcal{G}$, v, l)**:**
  for $i \in V$ **do**
    `DFSRecursion`(i, i, $\emptyset$, $\emptyset$)
    Mark node i
  end
end

Algorithm 1: Fractional separation by partial enumeration

3.2.2 Pricing problem.

Consider an optimal dual solution $(\bar{\pi}, \bar{\mu})$ of the restricted master problem, where $(\pi_i)_{i \in V}$ are the dual variables of constraints (7b) and $(\mu_C)_{C \in \mathcal{C}}$ are the dual variables of constraints (7c).

The pricing problem consists in finding a set of items $B \in \mathcal{B}$ such that the reduced cost of variable λ_B is minimized. The coefficient of λ_B in the constraint (7b) corresponding to $i \in V$ is one if $i \in B$, and zero otherwise. The coefficient of λ_B in the constraint (7c) corresponding to tuple $C \in \mathcal{C}$ is equal to the number of item pairs $\{i, j\} \in \mathcal{F}_C$ such that both i and j are contained in B. We define $\theta(B, C) = |\{\{i, j\} \in \mathcal{F}_C : i \in B, j \in B\}|$. Then the reduced cost of variable λ_B is $1 - \sum_{i \in B} \bar{\pi}_i - \sum_{C \in \mathcal{C}} \theta(B, C) \bar{\mu}_C$.

We now describe a MIP to find a set B with minimum reduced cost. Let $\bar{\mathcal{C}}$ be the set of active constraints (7c): $\bar{\mathcal{C}} = \{C \in \mathcal{C} : \bar{\mu}_C < 0\}$. Also let $\bar{\mathcal{F}} = \bigcup_{C \in \bar{\mathcal{C}}} \mathcal{F}_C$. Let binary variable x_i be equal to one if item $i \in V$ belongs to B, and zero otherwise. To simplify the notation, we introduce $\psi_{i,j} =$

$\sum_{C\in\bar{\mathcal{C}}:\{i,j\}\in\mathcal{F}_C} \bar{\mu}_C$. Also let binary variable $y_{i,j}$ be equal to one if both items from pair $\{i,j\} \in \bar{\mathcal{F}}$ belong to B, and zero otherwise. Then the pricing problem can be formulated as the following MIP.

$$\max \sum_{i\in V} \bar{\pi}_i x_i + \sum_{\{i,j\}\in\bar{\mathcal{F}}} \psi_{i,j} y_{i,j} \tag{9a}$$

$$\text{s.t.} \sum_{i\in V} w_i x_i \leq W, \tag{9b}$$

$$x_i + x_j \leq 1 \qquad \forall \{i,j\} \subseteq V, d(i,j) \geq 1 \text{ or } d(j,i) \geq 1, \tag{9c}$$

$$x_i + x_j \leq 1 + y_{i,j} \qquad \forall \{i,j\} \in \bar{\mathcal{F}}, \tag{9d}$$

$$x_i \in \{0,1\} \qquad \forall i \in V, \tag{9e}$$

$$y_{i,j} \geq 0 \qquad \forall \{i,j\} \in \bar{\mathcal{F}}. \tag{9f}$$

Given an optimal solution $(\bar{x}, \bar{y})$ to (9), we add to the restricted master problem variable $\lambda_{\bar{B}}$, where $\bar{B} = \{i \in V : \bar{x}_i = 1\}$. Constraints (9b) and (9c) require that $\bar{B} \in \mathcal{B}$. Since $\psi_{i,j} < 0$ for $\{i,j\} \in \bar{\mathcal{F}}$ and we are maximizing, each variable $\bar{y}_{i,j}$ takes the minimum possible value. Therefore, $\bar{y}_{i,j}$ is equal to one if and only if both $i \in \bar{B}$ and $j \in \bar{B}$. Therefore,

$$\sum_{\{i,j\}\in\bar{\mathcal{F}}} \sum_{\substack{C\in\bar{\mathcal{C}}:\\ \{i,j\}\in\mathcal{F}_C}} \mu_C \bar{y}_{i,j} = \sum_{C\in\mathcal{C}} \theta(B,C)\bar{\mu}_C,$$

and the objective value of solution $(\bar{x}, \bar{y})$ is equal to the reduced cost of $\lambda_{\bar{B}}$ with the opposite sign.

The pricing problem can be defined as the knapsack problem with hard and soft conflicts (KPHSC). Hard conflicts correspond to pairs $\{i,j\}$ such that either $d(i,j) \geq 1$ or $d(j,i) \geq 1$. Soft conflicts are pairs that are penalized for being both in the set, and correspond to pairs $\{i,j\} \in \bar{\mathcal{F}}$. The KPHSC generalizes the NP-hard knapsack problem with conflicts. The latter is NP-hard, and it is studied for example in (Sadykov and Vanderbeck, 2013; Bettinelli et al., 2017; Coniglio et al., 2020). To our knowledge, the KPHSC has not yet been studied in the literature.

3.2.3 A branch-and-bound algorithm for the pricing problem.

Solving pricing subproblem directly using a general purpose MIP solver happens to be a bottleneck of the BCP algorithm, especially for large instances. Therefore, we propose a dedicated branch-and-bound algorithm to solve (9) which is inspired from the one by (Sadykov and Vanderbeck, 2013) for the knapsack problem with conflicts.

Each node of the tree search has a list of selected items V^+, and a list of forbidden items V^-. At each node, the branching scheme consists in choosing an item in $V \setminus (V^+ \cup V^-)$ and adding it to V^+ in the left branch and to V^- in the right branch. A lower (primal) bound is computed at each node by taking into account the profits of the selected items, and the penalties incurred by pairs of selected items. If the value of this bound is better than the value of the best known solution so far, the latter is updated. An upper (dual) bound is obtained as the sum of the lower bound and the overestimation of the maximum profit which can be obtained from items in $V \setminus (V^+ \cup V^-)$. This overestimation is calculated by solving the fractional binary knapsack problem formulated in the following way.

- The knapsack size is equal to the residual bin capacity after packing all items in V^+, i.e. $W - \sum_{i\in V^+} w_i$.
- The profit of an item $i \in V \setminus (V^+ \cup V^-)$ is equal to $\hat{p}_i(V^+) = \bar{\pi}_i + \sum_{j\in V^+} \psi_{i,j}$. This profit takes into account the penalties incurred by packing i with items in V^+. Penalties between pairs of items which are both not in V^+ are disregarded. This is valid since penalties always reduce the value of the solution.

The solution of such fractional binary knapsack problem can be computed in linear time if items are ordered according to a non-decreasing profit/weight ratio $\hat{p}_i(V^+)/w_i$. Thus, the list of items $V \setminus (V^+ \cup V^-)$ is

always kept ordered according to this ratio. Each time set V^+ is modified, residual bin capacity, values $\hat{\rho}_i(V^+)$, the ordering of items, and set V^- (due to hard conflicts) are updated.

3.2.4 Other components of the algorithm.

It is well known that the column generation approach may have convergence issues. To improve convergence, we use the automatic dual price smoothing stabilization technique proposed by (Pessoa et al., 2018).

Having a good feasible solution is important for the efficiency of the BCP algorithm. Such a solution provides an upper bound for the optimal objective value, and thus many nodes in the branch-and-bound tree may be pruned by bound. To obtain feasible solutions, we use the strong diving heuristic proposed by (Sadykov et al., 2019). We now briefly describe how the standard diving heuristic (Joncour et al., 2010) can be applied for the BPPTL and present its strong diving variant.

After solving a node in the branch-and-bound tree, we have a solution $\bar{\lambda}$ to the linear relaxation of formulation (7). If $\bar{\lambda}$ is integer, the node is pruned, otherwise we apply the diving heuristic. In this algorithm, we select a column λ_B such that its value $\bar{\lambda}_B$ in the fractional solution is the closest to 1. We then add this column to the partial solution and change the right-hand-side values of constraints (7b) corresponding to items in B to zero. Afterwards, the linear relaxation of (7) is resolved by column generation (without cut separation). This iterative process continues until the solution to the linear relaxation becomes integer or until the relaxation becomes infeasible.

In order to increase the efficiency of the diving heuristic, we apply the following problem-specific enhancement to it. Each time a column λ_B is fixed to one, we update lag $l_{i,j}$ to $\max\{l_{i,j}, 0\}$ for every pair $\{i, j\} \subseteq B$. Following this update, we recalculate values $d(i, j)$ and $d(j, i)$, for all pairs $\{i, j\} \subseteq V$, i.e. the lengths of the longest paths between each pair of nodes in the graph. If a value $d(i, j)$ becomes positive, the set of hard conflicts in the pricing problem is updated, i.e. constraint $x_i + x_j \leq 1$ is added to formulation (9), and columns λ_B such that $\{i, j\} \subseteq B$ are removed from the master problem.

We use the strong diving variant (Sadykov et al., 2019) of this diving heuristic. In this variant, we select up to 10 candidate columns λ_B, and we fix them temporarily to one, one-by-one, and resolve the master problem. Then we select the candidate column which resulted in the smallest increase in the lower bound obtained by solving the linear relaxation of formulation (7).

After applying the strong diving heuristic, if the primal-dual gap is positive, we perform branching. Here we use the Ryan and Foster branching (Ryan and Foster, 1981). To do so, we find a pair of items $\{i, j\} \subseteq V$ such that value $\sum_{B \in \mathcal{B}:\, \{i,j\} \subseteq B} \bar{\lambda}_B$ is fractional. Then we create two child nodes: in the first we impose the requirement that items i and j are put into the same bin, and in the second that items i and j are assigned to different bins. We respectively add the corresponding constraints to the pricing problem: $x_i = x_j$ in the first branch, and $x_i + x_j \leq 1$ in the second. In the branch-and-bound algorithm, this results in respectively merging items i and j, or adding hard conflict between i and j. We also remove from the master problem columns which do not satisfy newly imposed constraints.

Choosing a pair of items to branch on is an important decision, which has a large impact on the efficiency of the BCP algorithm. To find better branching candidates, we use two-phase strong branching. In the first phase, we take up to 30 branching candidates (i.e. pairs of items), create child nodes for them, and solve only the restricted master problem for them without column and cut generation. Up to three best candidates are then chosen according to the product rule (Achterberg, 2007). For child nodes of these candidates, in the second phase, we perform full column and cut generation. The size of the branch-and-bound tree is estimated for them according to the approach suggested in (Kullmann, 2009). Then the branching candidate with the smallest estimated tree size is chosen.

4 The case with one available bin per time period and non-negative lags

If we consider the case in which only one bin can be used in each time period ($L = 1$), then the BCP algorithm proposed in Section 3 cannot be used. In this case, given a partition of the item set in bins, the problem of determining if those bins can result in a feasible solution is NP-complete, as shown by (Finta and Liu, 1996). For this reason, we limit ourselves to the case in which all lags are non-negative numbers. We denote this problem by BPPTL$^1_+$.

In this context, we can assume that the graph is acyclic. Otherwise it either has a cycle of positive length and then it is trivially infeasible, or it has a cycle with all the arcs having lag zero. In the latter case we can contract all the items in the cycle into one item and have an equivalent problem because these items must be assigned to the same time period, and thus to the same bin.

A problem related to the BPPTL$^1_+$ has been studied by (Kramer et al., 2017), who consider the makespan objective function, i.e. minimizing the last time period in which any bin is used. If all time lags are either zero or one, it is equivalent to minimizing the makespan and the number of bins used. This case generalizes both the bin packing problem with precedence constraints, and the simple assembly line balancing problem, as shown in (Kramer et al., 2017). However, if some time lags are strictly larger than one, minimizing the makespan is not equivalent to the BPPTL$^1_+$, since a solution that is optimal for one objective function may be sub-optimal for the other objective function. An example of this can be seen in Figure 5. In this example, if we optimize the use of one of the two objective functions we get a solution that is sub-optimal for the other objective function. Minimizing the makespan gives a solution with makespan 5 that uses 4 bins, but minimizing for the number of bins used gives a solution that uses 3 bins with makespan 6.

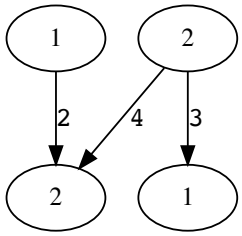


Figure 5: Example of instance with different optimal solutions for different objective functions. Here $W = 2$.

In this section, we show how to modify the BCP algorithm developed in Section 3 to solve to optimality the BPPTL$^1_+$.

4.1 Valid assignments

Any feasible solution for an instance of the BPPTL$^1_+$ is feasible for the BPPTL$^\infty$, and then the assignment of items to bins (which are also time periods in the BPPTL$^1_+$) must be composed of sets in $\mathcal{B}$, and must also satisfy constraints (7c). However, satisfying these constraints is not sufficient to induce a feasible solution for BPPTL$^1_+$, as can be seen in the example in Figure 6. In this example the time lags force each node to be assigned to the same time period, but the bin capacity does not allow one to assign all items to the same bin. Thus, there is a feasible solution to this instance of the BPPTL$^\infty$, but there are no feasible solutions for the corresponding instance of the BPPTL$^1_+$.

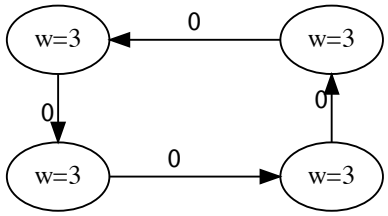


Figure 6: Example of instance with feasible solutions for $BPPTL^{\infty}$ but not for $BPPTL^{1}_{+}$. Capacity $W = 10$

In the following proposition, we characterize feasible solutions for the BPPTL$^1_+$ with a simple condition.

Proposition 4. *A partition $\mathcal{P} \subseteq \mathcal{B}$ induces a feasible solution to the $BPPTL^{1}_{+}$ if and only if the aggregated graph $G^{\mathcal{P}}$ is acyclic.*

Proof. Proof. Recall that any item set in $\mathcal{P}$ satisfies the capacity constraint. Therefore, only lag constraints have to be checked. If there is a feasible solution, then each set $B \in \mathcal{P}$ can be assigned to a time period p, and then the aggregated graph must be acyclic because otherwise the arc in the cycle with a tail assigned to the largest time period would be violated since all lags are non-negative. In the other direction, if $\mathcal{G}^{\mathcal{P}}$ is acyclic there is a topological order of the aggregated nodes. A feasible solution can be obtained by assigning bins to time periods following this topological order. □

In the proof, time lag values are not relevant. Then, any partition that induces a feasible solution also induces a feasible solution for any other graph with the same set of arcs, even with different (non-negative) lags. This implies that the time lag values are relevant just to define the conflicts for set $\mathcal{B}$. Thus it only matters if the value of each time lag is zero or a positive number. Therefore, all positive time lags can be replaced by unitary time lags without changing the set of optimal solutions.

In Section 3.1, we define set $\mathcal{C}_R$ to characterize partitions that induce an aggregated graph with a cycle of positive length, and thus an infeasible solution to the BPPTL$^\infty$. For the BPPTL$^1_+$, partitions that induce an aggregated graph with a zero length cycle are also infeasible. Analogously to Definition 2 for cycles of positive length, we now define the set of tuples which induce cycles of length 0 in the aggregated graph.

Definition 3. For each $R \in \mathbb{N}$, $R \geq 2$, let $\mathcal{C}^0_R \subseteq V^{2R}$ be the set of all tuples $(h_1, h_2, \ldots, h_R, t_1, \ldots, t_R)$ that satisfy the following properties.

- All the elements $h_1, \ldots, h_R, t_1, \ldots t_R$ are different.
- For each $r \in \{1, \ldots, R\}$, $d(h_r, t_r) \leq 0$ and $d(t_r, h_r) \leq 0$.
- $\sum_{r=1}^{R} d(t_r, h_{r+1}) = 0$, where $h_{R+1} = h_1$.

Recall the definition of value $\theta(B, C)$:

$$\theta(B, C) = \left|\left\{r \in \{1, \ldots, R\} \,:\, \{h_r, t_r\} \subseteq B\right\}\right|.$$

With this definition, constraints (7c) can be rewritten as:

$$\sum_{B \in \mathcal{B}} \theta(B, C)\lambda_B \leq R - 1 \qquad \forall R \geq 2, \forall C \in \mathcal{C}_R. \tag{10}$$

An intuitive extension of these constraints is the following.

$$\sum_{B\in\mathcal{B}} \theta(B,C)\lambda_B \leq R-1 \qquad \forall R\geq 2, \forall C\in\mathcal{C}_R^0, \tag{11}$$

However, constraints (11) forbid valid solutions. Indeed, if all pairs (h_r,t_r) in a given $C\in\mathcal{C}_R^0$ belong to the same element B in a partition $\mathcal{P}$, this partition does not induce a cycle of length 0 in the aggregated graph.

To overcome this issue, we define for $C\in V^{2R}$ and $B\in\mathcal{B}$ the value $\phi(B,C)=\min\{1,\theta(B,C)\}$. That is, $\phi(B,C)$ is equal to one if any pair in $\mathcal{F}_C$ is contained in B, and is equal to zero otherwise. This definition allows us to characterize partitions that induce an aggregated graph with cycles of length 0.

Proposition 5. *Given a partition $\mathcal{P}\subseteq\mathcal{B}$, the aggregated graph $G^{\mathcal{P}}$ does not have a cycle of zero length if and only if solution λ induced by $\mathcal{P}$ satisfies the following set of constraints:*

$$\sum_{B\in\mathcal{P}} \phi(B,C)\lambda_B \leq R-1 \qquad \forall R\geq 2, \forall C\in\mathcal{C}_R^0. \tag{12}$$

Proof. Proof. Assume that there is a cycle of zero length in $G^{\mathcal{P}}$, and take one with the minimum number of nodes, $B_1,\ldots,B_R$. We take for each $r\in\{1,\ldots,R\}$ nodes $h_r,t_r\in B_r$ such that $d(t_r,h_{r+1})=0$, where $h_{R+1}=h_1$. Then, for $C=(h_1,\ldots,t_R)$ we have $C\in\mathcal{C}_R^0$ due to the minimality of R. For each $r\in\{1,\ldots,R\}$, we have $\phi(B_r,C)=1$. Then constraint (12) for this tuple C is violated.

In the other direction, let $C\in\mathcal{C}_R^0$ be a tuple such that the respective constraint in (12) is violated. For each $r\in\{1,\ldots,R\}$ there is at most one set $B\in\mathcal{B}$ such that $\lambda_B=1$ and $\{h_r,t_r\}\subseteq B$. Then there are R different sets B_r, $r=\{1,\ldots,R\}$, such that $\phi(B_r,C)=1$ and $\lambda_{B_r}=1$. By definition of $\mathcal{C}_R^0$, there is a cycle of length 0 which contains nodes $G^{\mathcal{P}}$. □

By Proposition 5, the following IP formulation is valid for the BPPTL$_+^1$.

$$\min \sum_{B\in\mathcal{B}} \lambda_B \tag{13a}$$

$$\text{s.t.} \sum_{B\in\mathcal{B}} \mathbb{1}_B(i)\lambda_B = 1 \qquad \forall i\in V, \tag{13b}$$

$$\sum_{B\in\mathcal{B}} \theta(B,C)\lambda_B \leq R-1 \qquad \forall R\geq 2, \forall C\in\mathcal{C}_R, \tag{13c}$$

$$\sum_{B\in\mathcal{B}} \phi(B,C)\lambda_B \leq R-1 \qquad \forall R\geq 2, \forall C\in\mathcal{C}_R^0, \tag{13d}$$

$$\lambda_B\in\{0,1\} \qquad \forall B\in\mathcal{B}. \tag{13e}$$

Here, constraints (13c) forbid cycles of positive length in the aggregated graph, and constraints (13d) forbid cycles of zero length in the aggregated graph.

4.2 Separation

Given a partition $\mathcal{P}\subseteq\mathcal{B}$ induced by an integer solution, a violated constraint can be separated by finding any cycle in the aggregated graph $G^{\mathcal{P}}$. If the length of the cycle is positive, the corresponding constraint (13c) is violated. If the length of the cycle is zero, the corresponding constraint (13d) is violated. The depth-first-search algorithm can be used to find a cycle in the graph $G^{\mathcal{P}}$ or determine that the graph has no cycle. This can be done in time $\mathcal{O}(|\mathcal{P}|+|A^{\mathcal{P}}|)$.

To separate constraints (13d), we adapt the partial enumeration approach developed in Section 3.2.1. The following proposition is useful for this purpose.

Proposition 6. *Let* $\left(\bar{\lambda}_B\right)_{B\in\mathcal{B}}$ *be a solution satisfying* (13b). *If multigraph* $\mathcal{G}=(V,\mathcal{A})$ *constructed from* $\bar{\lambda}$ *has a cycle* $C=(a_1,a_2,\ldots,a_K)$ *such that* $\sum_{k=1}^{K} l(a_k)=0$ *and* $\sum_{k=1}^{K} v(a_k)<1$, *and for every* $B\in\mathcal{B}$ *with* $\bar{\lambda}_B>0$ *there is at most one pair* $\{i_B,j_B\}\in\mathcal{F}_C$ *with* $\{i_B,j_B\}\subseteq B$, *then there is a constraint* (13d) *violated by* $\bar{\lambda}$.

Proof. Proof. Similarly to Proposition 3, we can prove that there is a cycle $C'\subseteq C$ such that constraint $\sum_{B\in\mathcal{B}}\theta(B,C')\lambda_B\leq R-1$ is violated by $\bar{\lambda}$. For every $B\in\mathcal{B}$ with $\bar{\lambda}_B>0$ we have $\phi(B,C')=\theta(B,C')$, since at most one of the pairs in $\mathcal{F}_{C'}$ is contained in B. Therefore, constraint (13d) for C' violated by $\bar{\lambda}$. □

The algorithm to search for a violated constraint (13c) or (13d) is similar to Algorithm 1, with two differences. First, when looping over the arcs which leave current node j of current partial path C, we skip arcs $a=(i,j)\in\mathcal{A}_\lambda$ such that there is $B\in\mathcal{B}$ with $\bar{\lambda}_B>0$ and both $\{i,j\}\subseteq B$ and $\{i',j'\}\subseteq B$ for some $a'=(i',j')\in C\cap\mathcal{A}_\lambda$. Second, in addition to storing cycles with a positive sum of time lags, we also store cycles with the sum of time lags equal to zero.

4.3 Pricing problem

Consider an optimal dual solution $(\bar{\pi},\bar{\mu},\bar{\mu}^0)$, with $(\pi_i)_{i\in V}$ of the restricted master problem, where $(\pi_i)_{i\in V}$ are the dual variables of constraints (13b), $(\mu_C)_{C\in\mathcal{C}}$ are the dual variables of constraints (13c), and $(\mu^0_C)_{C\in\mathcal{C}^0}$ are the dual variables of constraints (13d). The binary coefficient of λ_B in the constraint (13d) corresponding to tuple $C\in\mathcal{C}^0$ is equal to $\phi(B,C)$. Recall that $\phi(B,C)$ is equal to one if and only if there is an item pair $\{i,j\}\in\mathcal{F}_C$ such that both i and j are contained in B. Then the reduced cost of variable λ_B is $1-\sum_{i\in B}\bar{\pi}_i-\sum_{C\in\mathcal{C}}\theta(B,C)\bar{\mu}_C-\sum_{C\in\mathcal{C}^0}\phi(B,C)\bar{\mu}^0_C$.

We now describe a MIP to find a set B of items with the minimum reduced cost. Let $\bar{\mathcal{C}}$ and $\bar{\mathcal{C}^0}$ be the sets of active constraints (13c) and (13d). Variables x and y are defined in the same ways as for formulation (9). Additionally, a binary variable z_C for each cycle $C\in\bar{\mathcal{C}^0}$ determines whether any of the pairs in $\mathcal{F}_C$ is contained in B. Then the pricing problem can be formulated as the following MIP.

$$\max \sum_{i\in V}\bar{\pi}_i x_i+\sum_{\{i,j\}\in\bar{\mathcal{F}}}\left(\sum_{\substack{C\in\bar{\mathcal{C}}:\\ \{i,j\}\in\mathcal{F}_C}}\bar{\mu}_C\right)y_{i,j}+\sum_{C\in\bar{\mathcal{C}^0}}\bar{\mu}^0_C z_C-1 \tag{14a}$$

$$\text{s.t.} \sum_{i\in V} w_i x_i\leq W, \tag{14b}$$

$$x_i+x_j\leq 1 \qquad \forall\{i,j\}\subseteq V,\ d(i,j)\geq 1 \text{ or } d(j,i)\geq 1, \tag{14c}$$

$$x_i+x_j\leq 1+y_{i,j} \qquad \forall\{i,j\}\in\bar{\mathcal{F}}, \tag{14d}$$

$$x_i+x_j\leq 1+z_C \qquad \forall C\in\bar{\mathcal{C}^0},\forall\{i,j\}\in\mathcal{F}_C, \tag{14e}$$

$$x_i\in\{0,1\} \qquad \forall i\in V, \tag{14f}$$

$$y_{i,j}\geq 0 \qquad \forall\{i,j\}\in\bar{\mathcal{F}}, \tag{14g}$$

$$z_C\geq 0 \qquad \forall C\in\bar{\mathcal{C}^0}. \tag{14h}$$

Given an optimal solution $(\bar{x},\bar{y},\bar{z})$ to (14), we add to the restricted master problem variable $\lambda_{\bar{B}}$, where $\bar{B}=\{i\in V:\bar{x}_i=1\}$. Since $\mu^0_C<0$ for $C\in\bar{\mathcal{C}^0}$ and we are maximizing, each variable $\bar{z}_C$ takes the minimum possible value. Therefore, $\bar{z}_C$ is equal to one if and only if $\{i,j\}\subseteq B$ for at least one pair $\{i,j\}\in\mathcal{F}_C$. This implies that z_C is equal to $\phi(B,C)$, and the objective value of solution $(\bar{x},\bar{y},\bar{z})$ is equal to the reduced cost of $\lambda_{\bar{B}}$ with the opposite sign.

The proposed formulation for the BPPTL^1_+ can be strengthened by reducing collection $\mathcal{B}$ of possible sets of items. Given a triple of items (i,k,j) such that $d(i,k)=0$ and $d(k,j)=0$, it is known (Peeters and Degraeve, 2006) that sets B such that $i,j\in B$ and $k\notin B$ can be removed from $\mathcal{B}$. Thus, the following

constraints can be added to the pricing problem (14):

$$x_i + x_j \leq 1 + x_k, \quad \forall \{i,j,k\} \subseteq V,\ d(i,k) = 0 \text{ and } d(k,j) = 0. \tag{15}$$

Our branch-and-bound algorithm for the KPHSC presented in Section 3.2.3 can be adapted to solve the formulation (14)-(15). For every $C \in \bar{\mathcal{C}}^0$, the value of the lower (primal) bound is adjusted by $\bar{\mu}^0_C$ if at least one of pairs $\{i,j\} \in \mathcal{F}_C$ is contained in V^+. Otherwise, penalty $\bar{\mu}^0_C$ is disregarded in the calculation of the upper (dual) bound as it can only reduce the solution value. Constraints (15) are verified when adding items to set V^-: if $i,j \in V^+$, $d(i,k) = 0$, and $d(k,j) = 0$ then the branch in which item k is added to V^- is not created. Moreover, the best known solution is not updated by the solution formed by items in V^+ if $i,j \in V^+$, $d(i,k) = 0$, $d(k,j) = 0$, and $k \notin V^+$.

5 Computational experiments

We implemented the proposed BCP algorithms in C++ language using a generic branch-cut-and-price library BaPCod (Vanderbeck et al., 2020). BapCod uses Cplex 12.8 for solving master and pricing subproblems. We also use Cplex 12.8 to solve compact formulation (2) described in Section 2. The experiments were run on a 2 Deca-core Ivy-Bridge Haswell Intel Xeon E5-2680 v3 server running at 2.50 GHz with 128 GB of RAM. Each instance is solved using a single thread.

5.1 Variant with an unlimited number of available bins

In this section we benchmark different approaches proposed in the paper for the variant of the problem with an unlimited number of available bins. In Section 5.1.1 we describe the test instances which are inspired from the application for the planning of phytosanitary treatments. In Section 5.1.2 we describe a procedure to generate random instances which are based on academic instances of the standard bin-packing problem. All generated instances are freely available at the address `math.u-bordeaux.fr/~rsadykov/#instances`. In Section 5.1.3 we test different cut separation algorithms. Computational comparison of the best variant of the compact formulation and the BCP algorithm is performed in Section 5.1.4. In Section 5.1.5, we compare our BCP algorithm with the standard branch-and-price algorithm based on formulation (3).

5.1.1 Generation of application instances.

The first set of instances we generated are inspired from the application of phytosanitary treatment planning in a vineyard, described in Section 1. Consider a set $\mathcal{Q} = \{1,\dots,Q\}$ of periodic meta-requests for treatment and a planning time horizon $\mathcal{T} = \{1,\dots,T\}$. For each request $q \in \mathcal{Q}$, the ideal elapsed time between two consecutive treatments is denoted by e_q. For additional flexibility, the minimum and maximum elapsed times between two consecutive treatments of request $q \in \mathcal{Q}$ are set to $e^-_q \leq e_q$ and to $e^+_q \geq e_q$. Assuming that the planning is embedded in a rolling horizon approach, the last treatment before the planning time horizon and the first treatment after the planning time horizon are fixed. Let $f_q \in \{-e_q + 1,\dots,0\}$ be the time period of the last treatment of request $q \in \mathcal{Q}$ before the planning time horizon. Let n_q be the number of treatments of request $q \in \mathcal{Q}$ during the time horizon: it is equal to the maximum integer such that $f_q + n_q e_q \leq T$. Then the time period of the first treatment of request $q \in \mathcal{Q}$ after the planning time horizon is equal to $f'_q = f_q + (n_q + 1) \cdot e_q$. The duration of each treatment i^q_j of request $q \in \mathcal{Q}$ is equal to $w_{i^q_j}$. Treatments are performed by an unlimited fleet of identical vehicles. During a time period, each vehicle is capable of performing treatments of a total duration not exceeding W. The usage cost of a vehicle during one time period is unitary. The objective is to perform all necessary treatments during the planning time horizon while respecting total durations of vehicles and the minimum and maximum elapsed times between any two consecutive treatments of the same request, and to minimize the total vehicle cost.

This problem can be modeled as the BPPTL as follows. Define the bin capacity equal to W and define $L = \infty$. Define two artificial items i^s and i^f with weights $w_{i^s} = w_{i^f} = W$. For each request $q \in \mathcal{Q}$ define n_q items $i^q_1, i^q_2, \dots, i^q_{n_q}$ with weights equal to $w_{i^q_1}, w_{i^q_2}, \dots, w_{i^q_{n_q}}$ respectively. In the time-lag graph,

we define arc (i^s, i^f) with lag $T+1$, and arc (i^f, i^s) with lag $-(T+1)$. For each request $q \in \mathcal{Q}$ and each treatment $k \in \{1, \ldots, n_q - 1\}$ we define arc (i^q_k, i^q_{k+1}) with lag e^-_q, and arc (i^q_{k+1}, i^q_k) with lag $-e^+_q$. For each $q \in \mathcal{Q}$ we define arc (i^s, i^q_1) with lag $\max\{1, e^-_q + f_q\}$, arc (i^q_1, i^s) with lag $-(e^+_q + f_q)$, arc $(i^q_{n_q}, i^f)$ with lag $\max\{1, e^-_a - (f'_q - T - 1)\}$, and arc $(i^f, i^q_{n_q})$ with lag $-(e^+_a - (f'_q - T - 1))$.

The items created for each request $q \in \mathcal{Q}$ form a double chain in the graph. Each node is connected to the next node in the chain by two arcs, one in each direction. The lags impose the minimum and maximum elapsed time between two consecutive treatments of a request. Items i^s and i^f must be scheduled exactly $T+1$ time periods apart. Then all other items are scheduled inside T time periods strictly between items i^s and i^f. Thus all treatments are performed within the planning time horizon.

Instances are randomly generated using four integer parameters: the number Q of requests, the number T of time periods, the average number N of request treatments and U the average number of treatments one vehicle can perform in a day. Given tuple (Q, T, N, U), an instance is generated as follows. For each request $q \in \mathcal{Q}$, let e_q be a random integer in the interval $\left[\left\lfloor 0.7 \cdot \frac{T}{N} \right\rfloor, \left\lceil 1.3 \cdot \frac{T}{N} \right\rceil\right]$, and let $e^-_q = \lfloor 0.8 \cdot e_q \rfloor$ and $e^+_q = \lceil 1.2 \cdot e_q \rceil$. For each request $q \in \mathcal{Q}$, also let f_q be a random integer in interval $[-e_q + 1, 0]$. The bin capacity W is set to 60, and we define the weight $w_{i^q_j}$ for each treatment of request $q \in A$ as a random integer in interval $\left[\left\lfloor 0.4 \cdot \frac{60}{U} \right\rfloor, \left\lceil 1.6 \cdot \frac{60}{U} \right\rceil\right]$. The dataset we generated consists of 252 instances, one instance for each combination of parameters $Q \in \{3, 5, 7, 9\}$, $T \in \{20, 40, 60, 80, 100, 120\}$, $N \in \{4, 7, 10\}$, and $U \in \{2, 3, 4\}$.

In Figure 1 shown in Section 1, we present the time-lag graph for an instance generated with parameters $Q = 7$, $T = 60$, $N = 4$. Parameter U does not have an impact on the time-lag graph. In this instance, random values generated for the ideal elapsed time of a request are $e_1 = 13$, $e_2 = 11$, $e_3 = 20$, $e_4 = 13$, $e_5 = 14$, $e_6 = 20$, and $e_7 = 14$. Random values generated for the last treatment of a request before the planning time horizon are $f_1 = 0$, $f_2 = -8$, $f_3 = -19$, $f_4 = -12$, $f_5 = -5$, $f_6 = -7$, and $f_7 = -12$.

5.1.2 Generation of random instances.

The instances in the second set we generate are derived from classic bin packing instances by (Falkenauer, 1996). This set contains eight classes of instances : four classes of "uniform" instances with 120, 250, 500, and 1000 items, and four classes of "triplet" instances with 60, 120, 149, and 501 items. We randomly generated time lags between items according to the following procedure. In order to ensure that all instances are feasible, we randomly generate tentative time period $f_i \in [1, 100]$ for every item $i \in V$. We consider "freedom" parameter F and "density" parameter D. We randomly generate an undirected graph $G' = (V, E)$: edge (i, j) such that $i, j \in V$ belongs to E with probability D. For every edge $(i, j) \in E$, we add lags $l_{i,j} = f_j - f_i - F$ and $l_{j,i} = f_i - f_j - F$. To ensure that all items are scheduled within the time horizon of 100 time periods, we add two artificial items: "source" item 0 and "sink" item $n+1$. Finally we add time lags $l_{0,n+1} = 101$, $l_{n+1,0} = -101$, $l_{0,i} = 1$ and $l_{i,n+1} = 1$ for all $i \in V$.

We took five BPP instances for each class, and for each of these instances, we generated randomly three BPPTL instances which correspond to parameters $(F, D) \in \{(1, 30\%), (3, 10\%), (10, 3\%)\}$. Thus, we generated 15 instances for each of the eight classes, or 120 instances in total.

5.1.3 Comparison of separation approaches.

In Section 3.2.1 two approaches to separate fractional solutions in the BCP algorithm are proposed: solving a MIP and a partial enumeration algorithm. In this section we compare these approaches, as well as the third alternative, which consists in only separating integer solutions. The results for the first set of instances are given in Table 1. Here, column "Gap" presents the average optimality gap, "Root Gap" gives the average gap at the root node of the branch-and-bound tree, "# Opt" the number of instances solved to optimality, "# Nodes" the average number of explored nodes in the branch-and-bound tree, "% Mast" is the time spent to solve the restricted master problem, "% Pric" is the time spent to solve the pricing problem, and "% Cuts" is the time spent to separate and add cuts. The values in the last three columns are in percentage of the total time. The sum of these three values is less than 100%, as the remaining time is due to auxiliary components of BaPCod. The results in Table 1 show that the separation of both integer

and fractional solutions by enumeration is the approach that allows us to solve to optimality the largest number of instances, and also to get the smallest average optimality gap over all instances. We can also underline that separating fractional solutions is very important for the efficiency of our BCP algorithm. The pricing problem solution is fast and does not constitute a bottleneck of the BCP algorithm. The majority of time is spent for cut separation, but this time clearly pays off. If cut separation is only used for integer solutions, the number of nodes becomes very large. In this case BaPCod generates a significant overhead, spending more that 50% of the time to manage the search tree. The time spent in the strong diving heuristic is around 30% in general. This time overlaps with the time to solve the restricted master problem and the pricing problem, as well as with the cut separation time, as all these components are used by the heuristic.

Separation approach	Gap	Root Gap	# Opt	# Nodes	% Mast	% Pric	% Cuts
Only integer solutions	4.1%	4.6%	159	13359.6	36.8%	3.5%	0.0%
Fractional solutions by MIP	3.0%	4.4%	180	660.4	27.4%	4.5%	60.9%
Fractional solutions by enum.	2.8%	4.2%	181	420.8	34.5%	7.1%	53.9%

Table 1: Comparison of different separation methods.

5.1.4 Comparison of the BCP algorithm with the Cplex solver.

In this section, we computationally compare two proposed approaches to solve the first set generated instances with an unlimited number of available bins. The first approach is the best configuration for the compact formulation, which is solved by Cplex (see Appendix B for the results obtained by the different variants). The second approach is our BCP algorithm with the enumeration separation algorithm applied to fractional solutions due to the results we obtained in Section 5.1.3. Both approaches are run with the time limit of one hour.

Table 2 shows the results of this computational comparison. We give results for different subsets of instances grouped by the generation parameters used, as well as the overall results. In the table, column "# Inst." shows the number of instances in the respective subset. Columns under "Opt" show the percentage of the instances that were solved to optimality within the time limit. Columns under "Gap" show the average optimality gap. Column "Root Gap" shows the average optimality gap at the root node. This gap is computed as $\frac{UB-LB}{UB}$, where LB in this case is the lower bound obtained in the root node, and UB is the value of the best known solution. As "Gap" and "Root Gap" are computed differently, the latter can be smaller than the former. Columns under "Time" show the average time in seconds. Finally, column "# Nodes" shows the average number of nodes explored in the branch-and-bound tree of the BCP algorithm.

The results, presented in Table 2 show that our BCP algorithm is substantially more efficient than the Cplex solver applied to the compact formulation of the problem. When considering the results for different subsets of instances, we note that the BCP algorithm is not very "sensitive" to the length of the time horizon, whereas the efficiency of Cplex decreases with the increase of parameter T. This is not surprising, as the size of the compact formulation depends heavily on T, whereas the theoretical and practical computational complexity of the components of the BCP algorithm never depend on T. On the other hand, when parameters Q or N increase, both the Cplex and the BCP algorithm become less efficient. The impact of parameter U is however very different. Smaller values of U are better for the BCP algorithm, and larger values are better for Cplex. This behaviour is similar to that observed when solving the classical BPP. When the average number of items which can fit into one bin increases, the quality of lower bounds obtained by the linear relaxation of compact formulations converge to the quality of the column generation lower bound. As the quality of lower bounds become similar, MIP solvers start to be more efficient than column generation approaches due to the fact the linear relaxation of compact formulations can be solved much faster.

120

	# Inst.	Opt		Gap		Root Gap		Time		# Nodes
		BCP	Cplex	BCP	Cplex	BCP	Cplex	BCP	Cplex	BCP
$T = 20$	36	80.6%	63.9%	1.0%	2.6%	0.9%	8.9%	867.3	1582.3	136.6
$T = 40$	36	75.0%	50.0%	2.0%	6.9%	2.4%	11.6%	920.9	2022.7	253.2
$T = 60$	36	72.2%	44.4%	3.1%	8.5%	3.9%	11.5%	1059.5	2304.0	401.6
$T = 80$	36	69.4%	30.6%	2.9%	10.0%	4.7%	12.7%	1143.8	2851.9	452.9
$T = 100$	36	63.9%	33.3%	4.1%	12.9%	5.3%	13.4%	1418.9	2853.9	686.1
$T = 120$	36	72.2%	16.7%	2.8%	15.8%	6.1%	13.0%	1167.0	3159.3	494.4
$T = 140$	36	69.4%	16.7%	3.5%	16.8%	6.3%	13.3%	1278.8	3097.5	521.1
$Q = 3$	63	100.0%	65.1%	0.0%	6.5%	1.6%	17.8%	72.7	1623.2	21.9
$Q = 5$	63	79.4%	31.7%	2.9%	9.8%	6.1%	11.1%	925.7	2595.7	542.0
$Q = 7$	63	55.6%	19.0%	3.8%	11.1%	5.3%	11.0%	1718.2	3145.2	583.8
$Q = 9$	63	52.4%	30.2%	4.4%	14.5%	4.0%	8.4%	1772.6	2848.3	535.6
$N = 4$	84	100.0%	73.8%	0.0%	2.3%	3.6%	9.2%	71.3	1298.4	62.2
$N = 7$	84	65.5%	31.0%	3.2%	9.2%	4.2%	11.9%	1383.4	2845.4	595.3
$N = 10$	84	50.0%	4.8%	5.1%	20.0%	4.9%	15.0%	1912.2	3515.4	605.0
$U = 2$	84	92.9%	32.1%	0.2%	9.3%	1.4%	9.3%	345.9	2631.8	276.1
$U = 3$	84	66.7%	39.3%	2.6%	10.4%	3.2%	10.8%	1280.8	2368.7	514.7
$U = 4$	84	56.0%	38.1%	5.6%	11.9%	8.2%	16.1%	1740.1	2658.7	471.6
All set 1	252	71.8%	36.5%	2.8%	10.5%	4.2%	12.1%	1122.3	2553.1	420.8

Table 2: Results of the computational comparison between the best variant of the compact formulation solved by Cplex and the best variant of the BCP algorithm

5.1.5 Comparison of the BCP algorithm with the standard branch-and-price algorithm.

In the standard branch-and-price algorithm based on formulation (3), we solve the binary knapsack pricing problem with the algorithm by Pisinger (1997). The branching strategy by Vanderbeck (2011) is used. We employ the same stabilization technique and the same diving heuristics as for our BCP algorithm. In Table 3, we show the results of the computational comparison on the second set of instances. The columns are the same as in Table 2. Here, "Gap" and "Root Gap" are calculated only for instances for which both a lower bound and a feasible solution are found. We put "-" if no such instances exist for a certain class.

The second set of instances is more difficult to solve than the first set. One can see in Table 3 that BCP algorithm is much more efficient than the branch-and-price algorithm. The main reason is the quality of column generation lower bounds. Another reason is that the restricted master problem based on formulation (3) is harder to solve due to numerous time lag constraints (3c) in the master. For large instances, the column generation procedure cannot even terminate in one hour. In the last row of Table 3, we also compare the BCP and BP algorithms on the first set of application instances. This comparison also shows a clear superiority of the BCP algorithm. We see the quality of column generation lower bounds as the main bottleneck of the BCP algorithm. To improve its efficiency, one should search for new families of strong "non-robust" cutting planes, i.e. cuts which change the structure of the pricing problem.

We do not give detailed results for the Cplex solver applied to the second set of random instances, as the solver performed significantly worse that the BCP algorithm: only one instance from 120 has been solved to optimality, and the run of an absolute majority of remaining instances could not terminate due to insufficient memory.

5.2 Variant with one available bin per time period

In this section we computationally estimate the efficiency of our BCP algorithm for the variant of the BPPTL with one available bin per time period, i.e. for the BPPTL^1_+. We also compare the BCP algorithm

	# Inst.	Opt		Gap		Root Gap		Time		Nodes	
		BCP	BP	BCP	BP	BCP	BP	BCP	BP	BCP	BP
$t60$	15	46.7%	26.7%	6.5%	15.1%	7.8%	14.5%	2010.8	2937.8	1463.3	2669.5
$t120$	15	33.3%	0.0%	9.0%	20.7%	8.6%	10.9%	2595.6	>3600	1380.2	3.0
$t249$	15	0.0%	0.0%	6.3%	-	6.3%	7.1%	>3600	>3600	9.1	1.0
$t501$	15	0.0%	0.0%	5.6%	-	5.6%	-	>3600	>3600	1.0	1.0
$u120$	15	20.0%	0.0%	6.7%	17.5%	6.6%	9.9%	2932.6	>3600	1420.1	3.0
$u250$	15	6.7%	0.0%	5.8%	-	5.9%	6.8%	3431.4	>3600	24.6	1.0
$u500$	15	0.0%	0.0%	6.4%	-	6.4%	-	>3600	>3600	1.0	1.0
$u1000$	15	0.0%	0.0%	-	-	-	-	>3600	>3600	1.0	1.0
All set 2	120	13.3%	3.3%	6.7%	16.8%	6.9%	9.8%	3180.7	3517.1	537.5	335.1
All set 1	252	71.8%	39.7%	2.8%	11.3%	4.2%	9.2%	1122.3	2315.7	420.8	5254.6

Table 3: Results of the computational comparison between the best variant of the compact formulation solved by Cplex and the best variant of the BCP algorithm

to other approaches available in the literature for special cases of the BPPTL^1_+.

5.2.1 Instances.

We tested the branch-cut-and-price algorithm on the BPP-GP instances considered by (Kramer et al., 2017). They created a set of instances for the BPP-GP by extending a benchmark set for the SALBP-1 proposed in (Otto et al., 2013). For each instance of the SALBP-1, they created two instances of the BPP-GP by defining random time lags on the arcs of the precedence graph, one of them with random time lag values in $\{0, 1\}$ and the other with random time lag values in $\{0, 1, 2, 3\}$. We will refer to these two special cases of the BPP-GP as BPP-GP01 and BPP-GP03. The literature instances for the BPP-P and the SALBP-1 were also considered by (Kramer et al., 2017), as these two problems are special cases of BPP-GP01.

The authors of (Kramer et al., 2017) kindly provided us with the results of their algorithm for the instances of the SALBP-1, the BPP-P, the BPP-GP01, and the BPP-GP03. As mentioned in the introduction, the algorithm in (Kramer et al., 2017) is designed for the makespan objective function. Thus, the problem considered in (Kramer et al., 2017) is a special case of the BPPTL^1_+ only if all time lag values are in $\{0, 1\}$. Thus, we tested our BCP algorithm only on the instances of the SALBP-1, the BPP-P, and the BPP-GP01. Instances containing 20, 50, and 100 items are used. We skip instances with 1000 items as they are out of reach for our algorithm. For each problem variant and each value n equal to the number of items, there are 525 instances in the test set. The structure of precedence graphs is described in (Otto et al., 2013).

(Kramer et al., 2017) propose a heuristic procedure to find feasible solutions and thus upper bounds. They also use known techniques from the literature to compute lower bounds. The best upper and lower bounds reported in (Kramer et al., 2017) are usually very good. Most of the instances are solved to optimality. A summary of known results can be seen in Table 4. This table combines the bounds reported in (Kramer et al., 2017), as well as the bounds we obtained using the code from (Pereira, 2016) provided by the author. Column "# Not Opt" shows the number of instances for which the best known lower bound is strictly smaller than the best known upper bound. Column "# Opt" gives the number of instances for which the best known bounds are equal, i.e. the number of instances with known optimum solutions. In column "Gap Unsolved", the average optimality gap is shown for the open instances.

# Items	type	# Not Opt	# Opt	Gap Unsolved
$n = 20$	BPP-GP01	0	525	-
	BPP-P	0	525	-
	SALBP-1	0	525	-
$n = 50$	BPP-GP01	15	510	3.9%
	BPP-P	0	525	-
	SALBP-1	0	525	-
$n = 100$	BPP-GP01	67	458	2.8%
	BPP-P	12	513	2.0%
	SALBP-1	9	516	2.2%

Table 4: Number of instances solved to optimality in (Kramer et al., 2017) and average optimality gap for unsolved instances

5.2.2 Experimental results of the BCP algorithm.

In this section, we test our BCP algorithm on the instances described in Section 5.2.1. We use two variants of the BCP algorithm. First variant BCP-K is initialized with the best known upper bounds obtained in (Kramer et al., 2017). Second variant BCP-∞ does not use any initial upper bounds. We impose the time limit of one hour for each instance. Given an instance, we denote by LB(ALG) and UB(ALG) the best lower and upper bounds obtained by algorithm ALG, which can be either BCP-K, BCP-∞, or KDI, where the latter is the approach proposed in (Kramer et al., 2017). In the BCP algorithm, the pricing problem is solved by the custom branch-and-bound algorithm when treating the BPP-GP01 and BPP-P instances. For the SALBP-1 instances, the MIP solver is used to solve the pricing problem. This choice is explained by the fact that some SALBP-1 instances have a large bin capacity and very few hard conflicts, and our branch-and-bound algorithm is not efficient for such instances of the knapsack problem with hard and soft conflicts.

In Table 5 we compare upper bounds UB(BCP-∞) and UB(KDI). Instances are divided into three groups: "Better", "Equal" and "Worse", depending on whether UB(KDI)> UB(BCP-∞), UB(KDI)= UB(BCP-∞), and UB(KDI)< UB(BCP-∞). Columns under "Avg. Diff." give the average absolute difference between values UB(KDI) and UB(BCP-∞) for the corresponding group of instances. For some groups of instances, this average difference is marked with "∞", which means that for at least one instance in this group no feasible solution was found by algorithm BCP-∞.

		Better		Equal	Worse	
# Items	type	#	Avg. Diff.	#	#	Avg. Diff.
$n = 20$	BPP-GP01	0	-	525	0	-
	BPP-P	0	-	525	0	-
	SALBP-1	0	-	525	0	-
$n = 50$	BPP-GP01	1	1.0	520	4	1.0
	BPP-P	0	-	501	24	1.0
	SALBP-1	0	-	515	32	1.0
$n = 100$	BPP-GP01	13	1.0	480	32	∞
	BPP-P	2	1.0	372	151	1.5
	SALBP-1	0	-	339	186	∞

Table 5: Comparison of upper bounds UB(BCP-∞) and UB(KDI)

The results in Table 5 show that the heuristic proposed in (Kramer et al., 2017) is usually much better to obtain good feasible solutions than the algorithm BCP-∞. Nevertheless, our algorithm is able to improve the best known solutions for 16 instances. Using the variant BCP-K of our algorithm, no

more best known solutions were improved.

In Table 6, we compare lower bounds LB(BCP-K) and LB(KDI). In the same vein as above, the instances are divided into the groups "Better", "Equal" and "Worse", depending on whether LB(KDI)< LB(BCP-K), LB(KDI)= LB(BCP-K), and LB(KDI)> LB(BCP-K). The results show that the lower bounds obtained by our algorithm BCP-K are on average worse than the best known ones for instances of the BPP-P and the SALBP-1. However, our lower bounds are on average better than the best known ones for instances of the BPP-GP01. This is not surprising as the BPP-GP has been less studied in the literature. Overall, our algorithm improved 63 best known lower bounds among 103 open instances, i.e. for the majority of open instances.

		Better		Equal	Worse	
# Items	type	#	Avg. Diff.	#	#	Avg. Diff.
$n = 20$	BPP-GP01	0	-	525	0	-
	BPP-P	0	-	525	0	-
	SALBP-1	0	-	525	0	-
$n = 50$	BPP-GP01	13	1.0	511	1	1.0
	BPP-P	0	-	501	24	1.0
	SALBP-1	0	-	522	3	1.0
$n = 100$	BPP-GP01	36	1.4	472	17	1.1
	BPP-P	8	1.0	448	69	1.2
	SALBP-1	6	1.0	487	32	1.0

Table 6: Comparison of lower bounds LB(BCP-K) and LB(KDI)

In Table 7, we show the optimality status of instances before and after applying our algorithm. Column "# Opt KDI" shows the number of instances with known optimum solutions in the literature. Column "# Opt. +BCP" gives the number of instances with known optimality based on the literature results and the results obtained by our BCP algorithm. Finally, "# Not Opt." shows the number of instances that still remain open. This table shows that for 79 instances the optimality status is obtained for the first time among 103 open instances. 24 instances still remain open.

# Items	type	# Opt KDI	# Opt. +BCP	# Not Opt.
$n = 20$	BPP-GP01	525	0	0
	BPP-P	525	0	0
	SALBP-1	525	0	0
$n = 50$	BPP-GP01	510	14	1
	BPP-P	525	0	0
	SALBP-1	525	0	0
$n = 100$	BPP-GP01	458	49	18
	BPP-P	513	10	2
	SALBP-1	516	6	3

Table 7: Optimality status of instances

Finally, in Table 8, we give statistics for our algorithm BCP-K applied to instances described in Section 5.2.1. Columns under "# Nodes" show the average number of explored nodes in the branch-and-bound tree for both unsolved instances and instances solved to optimality. Column "Not Solved" gives the number of instances not solved to optimality by the algorithm BCP-K. Column "Solved" shows the number of instances solved to optimality. Column "Time" gives the average time in seconds our algorithm took when it could solve instances to optimality. It can be seen from the results in Table 8 that algorithm BCP-K is less efficient than the approach by (Kramer et al., 2017) for the BPP-P and the SALBP-1,

but more efficient for the BPP-GP01. We should mention, however, that our algorithm is initialized by the best known solutions obtained in (Kramer et al., 2017). Thus, it makes sense to combine our BCP algorithm with the approach proposed in (Kramer et al., 2017) to solve instances of the BPP-GP01. These two methods seem to have a complementary strength, according to the results presented in Table 7.

		# Nodes		# Instances		Time
# Items	type	Not Solved	Solved	Not Solved	Solved	Solved
$n = 20$	BPP-GP01	-	1.5	0	525	0.19
	BPP-P	-	1.7	0	525	0.18
	SALBP-1	-	1.2	0	525	0.84
$n = 50$	BPP-GP01	8049.0	15.6	2	523	6.59
	BPP-P	3479.4	40.3	24	501	36.92
	SALBP-1	621.7	4.5	3	522	25.64
$n = 100$	BPP-GP01	5969.1	26.7	34	491	34.71
	BPP-P	2491.7	36.4	71	454	57.70
	SALBP-1	52.2	2.2	35	490	182.20

Table 8: Statistics of the BCP-K algorithm

6 Conclusions

In this work, we introduce the bin packing problem with time lags (BPPTL). As a motivation, we describe an application of the BPPTL in which one needs to decide on the planning of phytosanitary treatments in a vineyard. We first present an IP formulation for the problem. Then for two important special cases of the BPPTL with an unlimited number of available bins and with one available bin per time period, we propose an exact branch-cut-and-price (BCP) algorithm. This algorithm relies on a known IP formulation with an exponential number of variables which define a relaxation for the problem, and a new family of constraints which serve to cut infeasible solutions produced by the relaxation. We present two approaches to separate the new family of constraints. The BCP algorithm also incorporates an automatic dual price smoothing technique to improve the convergence of column generation, a strong diving heuristic for finding feasible solutions, and a strong multi-phase Ryan&Foster branching.

The computational experiments show that our BCP algorithm substantially outperforms a commercial MIP solver on newly generated instances inspired from the phytosanitary treatment planning application. The experiments on literature instances for special cases of the BPPTL show that the BCP algorithm can complement known approaches, as it allows us to obtain the optimality status for 70% of previously open instances.

Nevertheless, we believe that the efficiency of our BCP algorithm can still be improved. The experiments on literature instances show that the primal heuristic should be improved. The BCP algorithm currently relies on the generic strong diving heuristic, which can be slow and relatively inefficient for larger instances. Therefore, developing specialized heuristics for the BPPTL is an important research direction. Known heuristics for the related problem BPP-GP, such as the one proposed in (Kramer et al., 2017), may serve as a basis for this work.

The BCP algorithm is currently limited to two special cases of the BPPTL. Extending it to the general case is an interesting research perspective. This seems not to be easy to do, as in general it is an NP-complete problem (Finta and Liu, 1996) to determine whether a partition of items into bins is feasible or not with respect to the time lags. Thus, any family of cutting planes to cut off infeasible assignments of items to bins will be NP-hard to separate even if only integer assignments are considered. An extension to the general case is relevant in practice, as sometimes we cannot assume that we may use any number of resources as we want in any time period.

Another important generalization concerns related scheduling problems. Release dates and deadlines

can be modeled with time lags, and thus addressed easily. However, considering objective functions like makespan or total tardiness, and therefore time period-dependent penalization of item placement, would require a substantial modification of the algorithm.

In the phytosanitary treatment planning application, we assume that the time needed for a vehicle to go from one sector to another is negligible in comparison to the duration of treatments. Thus transition times of vehicles are ignored. In practice, this is not always true. If transition times are not negligible, the problem shifts from the class of precedence-constrained bin packing problems to the class of periodic vehicle routing problems (Campbell and Wilson, 2014).

In the standard periodic vehicle routing problem (PVRP), there are only a few alternatives for visiting each client. For example, a client may be visited on Monday and Thursday, or on Tuesday and Friday. If the planning time horizon is sufficiently long, having only a few alternatives dramatically limits the flexibility of visits. Flexible variants of the PVRP exist, such as one proposed in (Archetti et al., 2017). However, flexibility in that case concerns the amount of goods delivered to a client during each visit. An extension of the BPPTL to a flexible periodic vehicle routing problem in which one imposes minimum and maximum time lags on consecutive visits to the same client has a large number of applications in the context of provision of services such as periodic maintenance visits. A BCP algorithm similar to the one proposed in this paper may be efficient for such a flexible periodic vehicle routing problem, provided that the pricing problem takes into account the vehicle transition times.

Finally, the BPPTL structure is sufficiently rich that there may be other applications which result in time lag graphs in a class different from the class of double-linked chain graphs. Finding such applications and suggesting test instances for them is useful for further progress in solution approaches for the BPPTL.

Acknowledgments

We thank Raphael Kramer for kindly providing us with results for individual instances and other details on their work.

We thank Jordi Pereira for kindly providing us with results for individual instances as well as his code for the BPP-P.

Experiments presented in this paper were carried out using the PlaFRIM (Federative Platform for Research in Computer Science and Mathematics), created under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the "Programme d'Investissements d'Avenir" (see `www.plafrim.fr/en/home`).

This work was supported by the Government of Chile through CONICYT-PFCHA/Doctorado Nacional/2017-211713157 (ORL).

References

Tobias Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universitat Berlin, 2007.

Claudia Archetti, Elena Fernández, and Diana L. Huerta-Muñoz. The flexible periodic vehicle routing problem. *Computers & Operations Research*, 85:58 – 70, 2017.

C. Artigues. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154–159, 2017.

M Bartusch, R H Möhring, and F J Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16(1):199–240, dec 1988. ISSN 1572-9338. doi: 10.1007/BF02283745. URL `https://doi.org/10.1007/BF02283745`.

Andrea Bettinelli, Valentina Cacchiani, and Enrico Malaguti. A branch-and-bound algorithm for the knapsack problem with conflict graph. *INFORMS Journal on Computing*, 29(3):457–473, 2017.

Andrew Bilyk, Lars Mönch, and Christian Almeder. Scheduling jobs with ready times and precedence constraints on parallel batch machines using metaheuristics. *Computers & Industrial Engineering*, 78: 175–185, 2014.

Ann Melissa Campbell and Jill Hardin Wilson. Forty years of periodic vehicle routing. *Networks*, 63(1): 2–15, 2014.

N. Christofides, R. Alvarez-Valdés, and J.M. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, 1987.

Stefano Coniglio, Fabio Furini, and Pablo San Segundo. A new combinatorial branch-and-bound algorithm for the knapsack problem with conflicts. *European Journal of Operational Research*, online, 2020.

Mauro Dell'Amico, José Carlos Díaz Díaz, and Manuel Iori. The Bin Packing Problem with Precedence Constraints. *Operations Research*, 60(6):1491–1504, 2012. doi: 10.1287/opre.1120.1109. URL `https://doi.org/10.1287/opre.1120.1109`.

Maxence Delorme and Manuel Iori. Enhanced pseudo-polynomial formulations for bin packing and cutting stock problems. *INFORMS Journal on Computing*, 32(1):101–119, 2020.

Maxence Delorme, Manuel Iori, and Silvano Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255(1):1–20, 2016.

Lionel Dupont and Clarisse Dhaenens-Flipo. Minimizing the makespan on a batch machine with non-identical job sizes: an exact procedure. *Computers & Operations Research*, 29(7):807–819, 2002.

Samir Elhedhli, Lingzi Li, Mariem Gzara, and Joe Naoum-Sawaya. A Branch-and-Price Algorithm for the Bin Packing Problem with Conflicts. *INFORMS Journal on Computing*, 23(3):404–415, 2011.

Simon Emde, Lukas Polten, and Michel Gendreau. Logic-based benders decomposition for scheduling a batching machine. *Computers & Operations Research*, 113:104777, 2020.

Emanuel Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

Albert E. Fernandes Muritiba, Manuel Iori, Enrico Malaguti, and Paolo Toth. Algorithms for the Bin Packing Problem with Conflicts. *INFORMS Journal on Computing*, 22:401–415, 2010.

Lucian Finta and Zhen Liu. Single machine scheduling subject to precedence delays. *Discrete Applied Mathematics*, 70(3):247–266, 1996. ISSN 0166-218X. doi: https://doi.org/10.1016/0166-218X(96)00110-2. URL `http://www.sciencedirect.com/science/article/pii/0166218X96001102`.

Michel Gendreau, Gilbert Laporte, and Frédéric Semet. Heuristics and lower bounds for the bin packing problem with conflicts. *Computers and Operations Research*, 31(3):347 – 358, 2004.

C Joncour, S Michel, R Sadykov, D Sverdlov, and F Vanderbeck. Column Generation based Primal Heuristics. *Electronic Notes in Discrete Mathematics*, 36:695–702, 2010. ISSN 1571-0653. doi: https://doi.org/10.1016/j.endm.2010.05.088. URL `http://www.sciencedirect.com/science/article/pii/S1571065310000892`.

V. Kaibel and M. Pfetsch. Packing and partitioning orbitopes. *Mathematical Programming.*, 114(1):1–36, 2008.

Leonid V Kantorovich. Mathematical methods of organizing and planning production. *Management science*, 6(4):366–422, 1960. Translation of a 1939 paper in Russian.

Ali Khanafer, François Clautiaux, and El-Ghazali Talbi. New lower bounds for bin packing problems with conflicts. *European Journal of Operational Research*, 206(2):281 – 288, 2010.

Ali Khanafer, François Clautiaux, and El-Ghazali Talbi. Tree-decomposition based heuristics for the two-dimensional bin packing problem with conflicts. *Computers & Operations Research*, 39(1):54 – 63, 2012.

Raphael Kramer, Mauro Dell'Amico, and Manuel Iori. A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints. *International Journal of Production Research*, 55(21):6288–6304, 2017. doi: 10.1080/00207543.2017.1341065. URL https://doi.org/10.1080/00207543.2017.1341065.

O Kullmann. *Handbook of Satisfiability*, chapter Fundaments of branching heuristics, pages 205–244. IOS Press, Amsterdam, 2009.

David R Morrison, Edward C Sewell, and Sheldon H Jacobson. An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset. *European Journal of Operational Research*, 236(2):403–409, 2014. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2013.11.033. URL http://www.sciencedirect.com/science/article/pii/S0377221713009508.

Alena Otto, Christian Otto, and Armin Scholl. Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research*, 228(1):33–45, 2013. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2012.12.029. URL http://www.sciencedirect.com/science/article/pii/S0377221713000039.

Marc Peeters and Zeger Degraeve. An linear programming based lower bound for the simple assembly line balancing problem. *European Journal of Operational Research*, 168(3):716–731, 2006.

Jordi Pereira. Empirical evaluation of lower bounding methods for the simple assembly line balancing problem. *International Journal of Production Research*, 53(11):3327–3340, 2015. doi: 10.1080/00207543.2014.980014. URL https://doi.org/10.1080/00207543.2014.980014.

Jordi Pereira. Procedures for the bin packing problem with precedence constraints. *European Journal of Operational Research*, 250(3):794–806, 2016. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2015.10.048. URL http://www.sciencedirect.com/science/article/pii/S0377221715009741.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360, 2018.

Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. A generic exact solver for vehicle routing and related problems. *Mathematical Programming B*, 183:483–523, 2020.

David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):758–767, 1997.

N. Rafiee Parsa, B. Karimi, and A. Husseinzadeh Kashan. A branch and price algorithm to minimize makespan on a single batch processing machine with non-identical job sizes. *Computers & Operations Research*, 37(10):1720–1730, 2010.

D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport Urban Passenger Vehicle and Crew Scheduling*, pages 269 – 280. North-Holland, Amsterdam, 1981.

Ruslan Sadykov and François Vanderbeck. Bin Packing with Conflicts: A Generic Branch-and-Price Algorithm. *INFORMS Journal on Computing*, 25(2):244–255, 2013. doi: 10.1287/ijoc.1120.0499. URL http://dx.doi.org/10.1287/ijoc.1120.0499.

Ruslan Sadykov, François Vanderbeck, Artur Pessoa, Issam Tahiri, and Eduardo Uchoa. Primal heuristics for branch-and-price: the assets of diving methods. *INFORMS Journal on Computing*, 31(2):251–267, 2019.

Armin Scholl and Christian Becker. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693, 2006. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2004.07.022. URL http://www.sciencedirect.com/science/article/pii/S0377221704004795.

R. Uzsoy. Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 32(7):1615–1635, 1994.

R. Uzsoy. Scheduling batch processing machines with incompatible job families. *International Journal of Production Research*, 33(10):2685–2708, 1995.

François Vanderbeck. Branching in branch-and-price: a generic scheme. *Mathematical Programming*, 130 (2):249–294, 2011.

François Vanderbeck, Ruslan Sadykov, and Issam Tahiri. BaPCod — a generic Branch-And-Price Code. `https://realopt.bordeaux.inria.fr/?page_id=2`, 2020.

Laguna Wei, Zhixing Luo, Roberto Baldacci, and Andrew Lim. A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems. *INFORMS Journal on Computing*, 32(2):428–443, 2020.

Appendix A. Proof of Lemma 1

Lemma 1. *Let $G = (V, A, l)$ be a directed valued graph. Graph G has no cycle of positive length if and only if a mapping $\tau : V \longrightarrow \mathbb{Z}_+$ exists, such that for each $(i, h) \in A$, $\tau(i) + l_{i,h} \leq \tau(j)$.*

Proof. Proof. This proof is an adaptation of a similar result taken from (Bartusch et al., 1988). If such τ exists, consider any cycle with nodes $i_1, \ldots, i_K, i_{K+1} = i_1$. Then, for each $k \in \{1, \ldots, K\}$ we have $\tau(i_k) + l_{i_k,i_{k+1}} \leq \tau(i_{k+1})$. If we sum all these inequalities we get $\sum_{k=1}^{K} l_{i_k,i_{k+1}} \leq 0$, and then any cycle must have non-positive length.

Assume now that there is no cycle of positive length, and consider an extra *source* node $s \notin V$, and the graph $\bar{G} = (V \cup \{s\}, \bar{A}, \bar{l})$ such that $\bar{A} = A \cup \{(s, i) : i \in V\}$, and $\bar{l}_{i,j} = 1$ if $i = s$, and $\bar{l}_{i,j} = l_{i,j}$ otherwise.

For each $i \in V$, define $\tau(i)$ as the length of the longest path from s to i in $\bar{G}$. Since (G, A, l) has no cycle of positive length, then $(\bar{G}, \bar{A}, \bar{l})$ has no cycle of positive length, and then the longest path from s to i is a well defined finite number, taking a value of at least 1.

Finally, for each $(i, j) \in A$ it holds that $\tau(i) + l_{i,j} \leq \tau(j)$, since the length of the longest path from s to j cannot be smaller that the length of the longest path from s to i plus $l_{i,j}$. □

Appendix B. Results for the variants of the compact model

Some modifications to formulation (2) can be made. Constraints (2c) can be replaced by constraints giving a stronger linear relaxation:

$$\sum_{i \in V} w_i x_{i,b,p} \leq W \qquad \forall b \in \mathcal{L}, p \in \mathcal{T}, \tag{16a}$$

$$x_{i,b,p} \leq u_{b,p} \qquad \forall i \in V, b \in \mathcal{L}, p \in \mathcal{T}. \tag{16b}$$

These constraints should be used if some items have a weight equal to 0, since constraints (2c) would not count a bin in the objective value if all items assigned to it have weight 0.

Constraints (2d) can also be replaced by constraints which give a stronger linear relaxation:

$$\sum_{p'=1}^{p-l_{i,j}} \sum_{b \in \mathcal{L}} x_{i,b,p'} \geq \sum_{p'=1}^{p} \sum_{b \in \mathcal{L}} x_{j,b,p'} \qquad \forall (i, j) \in A, \forall p \in \mathcal{T} : p - l_{i,j} \leq T. \tag{17}$$

The number of constraints (17) is $\mathcal{O}(T|A|)$, while the number of constraints in (2d) is $\mathcal{O}(|A|)$. Thus, the stronger relaxation comes at the price of a larger formulation. The first use of constraints (17) in the context of a scheduling problem was made by (Christofides et al., 1987). See (Artigues, 2017) for a more detailed discussion of these constraints.

Additionally, redundant constraints can be added to break symmetry. The bins used at each time period are equivalent: bins can be replaced without an impact on the feasibility and cost of the solution. Thus, without loss of generality we can impose that the bins with lower indices are chosen:

$$u_{b-1,p} \geq u_{b,p} \qquad \forall p \in \mathcal{T}, \forall b \in \mathcal{L} \setminus \{1\}. \tag{18}$$

These cuts are classical in the cutting and packing community, and are subsets of those proposed in (Kaibel and Pfetsch, 2008).

We now analyze the performance of some variants of the compact formulation presented in Section 2. Specifically, we seek to determine how different combinations of constraints affect the performance of the Cplex solver on our set of instances.

We consider four different variants of formulation (2).

- Capacity constraints. We denote `Cap = Coupled` if we use constraints (2c), and `Cap = Decoupled` if we use constraints (16).
- Time lag constraints. We denote `Pre = Strong` if we use constraints (17) and `Pre = Weak` if we use constraints (2d).
- Breaking symmetries. We denote `Sim1` if we add constraints (18) to the formulation, and `Sim0` otherwise.
- Early and late starts. We denote `Es1` if we apply the early and late start pre-processing, and `Es0` otherwise.

To compare these variants, we used Cplex to solve each of the $2^4 = 16$ combinations, running each instance with a time limit of one hour. The results are presented in Table 9. Columns under "Gap" contain the average optimality gap over all instances, with the gap computed as $\frac{UB-LB}{UB}$. Columns under "# Opt" contain the number of instances solved to optimality. Table 9 shows that the most efficient variant is (`Sim1`, `Es0`, `Cap = Coupled`, `Pre = Weak`). Using this variant of the compact formulation, Cplex solves 98 out of 252 instances.

		Gap				# Opt			
		Sim0		Sim1		Sim0		Sim1	
		Es0	Es1	Es0	Es1	Es0	Es1	Es0	Es1
Cap = Coupled	Pre = Strong	10.4%	10.6%	10.7%	10.5%	72	74	90	92
	Pre = Weak	11.7%	10.8%	11.9%	10.5%	59	60	91	86
Cap = Decoupled	Pre = Strong	32.8%	33.0%	32.0%	31.7%	72	71	85	87
	Pre = Weak	31.8%	23.7%	34.5%	28.4%	68	69	86	87

Table 9: Comparison of different compact formulations.

Chapter 5

Conclusions

In this thesis we develop integer programming methods to two classes of problems related to scheduling: the general production scheduling problem (GPSP) and the bin packing with time lags (BPPTL). We motivate our study of the GPSP by considering two important strategic mine planning applications, and the BPPTL by considering s phytosanitary treatment problem in wine production scheduling.

We begin by studying the Bienstock-Zuckerberg (BZ) algorithm, which is well suited to solve the linear relaxation of GPSP. We propose several speed-ups for this algorithm, improving its performance in many different ways. We show that the BZ algorithm significantly outperforms alternative approaches to solve the LP relaxation of GPSP, and that this algorithm can effectively be used to compute the LP relaxation solution of real size instances of different mine planning problems and classic RCPSP problems.

We continue by incorporating the use of integer-variables in our study of GPSP. We focus on two specific open pit mine planning problems: the open pit phase design problem (OPPDP) and the open pit production scheduling problem (OPPSP). The methodology that we propose uses the BZ algorithm to solve the LP relaxation, and additionally introduces rounding heuristics, pre-processing techniques, cutting planes, and a branch-and-bound scheme. Experiments performed in real open pit mining instances show that the methodology can be used to get integer feasible solution within an average 1.25% gap of optimality for OPPDP instances, and within 0.71% gap of optimality for OPPSP instances.

Finally, motivated by the planning of phytosanitary treatments in a vineyard, we study the BPPTL. We propose both a compact formulation and an extended formulation. We develop a branch-cut-and-price (BCP) methodology based on the extended formulation that can be used to solve two specific cases of the BPPTL. This methodology combines a separation algorithms (for both integer and fractional solutions) of the relaxation for the two studied problem classes, a strong diving heuristic to find feasible solutions, automatic dual price smoothing to improve the column generation convergence, and a Ryan and Foster branching scheme. Computational experiments show that the BCP algorithm significantly outperforms commercial MIP solvers using the compact formulation on newly generated instances. The algorithm also allows us to close the optimality gap for 70% of previously open instances in a public benchmarking set.

We plan to pursue the following future research directions that follow up on this research:

- Dan Bienstock and Mark Zuckerberg observed that the BZ algorithm can be generalized to problems where the pricing problem is unimodular. However, this is not a strict requirement to prove convergence. There are

a number of scheduling problems, very similar to the GPSP, that currently cannot be solved with the BZ algorithm, and that are largely unsolvable due to the challenge of solving the LP relaxation. Extending and tuning such a generalized BZ algorithm could be very useful in practical problems.

- In our strategic open pit mining problems we are ignoring an important practical constraint concerning the shape of extracted pits. Such constraints should impose that blocks be extracted in contiguous groups of a minimum-width. It is not clear how to formulate this problem. Much less while preserving the problem structure that we exploit.

- A number of commercial software systems for solving open pit mine planning problems rely on the so-called receding time window heuristics. An interesting research direction would consist in studying this class of heuristics, and integrating them with our proposed approach. The methodologies we have developed could serve to improve the performance of each iteration in a receding time window algorithm, or, they could serve as a means to polish or improve a solution generated by such.

- We believe that the hourglass cuts could be used for the multi-modal resource constrained project scheduling problem formulations, which are known to exhibit very poor LP bounds. Moreover, it is possible that an extension of the BZ algorithm might be suitable for solving the LP relaxation of this problem. Overall, multi-modal RCPSPs have received little attention in the academic literature and would be very interesting to study. A major challenge of the multi-modal problem is that relaxing capacity constraints leads to a scheduling problem that is still NP-Complete. Multi-modal scheduling problems are very important in the context of project scheduling in general (Critical Path Method, crashing), and in particular, in underground mine planning.

- For the BPPTL it would be very interesting to study the cases with $1 < L < \infty$, which so far are ignored by our approach. We believe that it may be possible to apply for these problems the techniques developed for GPSP, by directly considering the compact formulation instead of a column generation approach. Using column generation seems difficult, as in general it is an NP-complete problem [FL96] to determine whether a partition of items into bins is feasible or not with respect to the time lags.

- While the algorithm that we propose for BPPTL significantly outperforms other approaches in the academic literature, we believe that it is possible to introduce several speed-ups. Two directions seem most promising. First, the bottleneck of our approach is the pricing algorithm, which is the knapsack problem with hard and soft conflicts (KPHSC). Approaching the KPHSC directly by applying a MIP solver is probably not the best way to solve it, especially when a bin can hold few items on average. It is likely that a constraint programming approach with effective propagation and domain-reduction techniques would be more efficient. Second, the primal

heuristic that we use is a generic strong diving heuristic, which can be slow and relatively inefficient for larger instances. Extending the work proposed by [KDI17], may be a promising first direction.

In addition, we expect to continue working with mining companies interested in these types of methodologies. Mining is a very important problem in Chile, and small methodological improvements could mean big savings for these companies.

References

[FL96] Finta, L. and Liu, Z. "Single machine scheduling subject to precedence delays". In: *Discrete Applied Mathematics* vol. 70, no. 3 (1996), pp. 247–266.

[KDI17] Kramer, R., Dell'Amico, M., and Iori, M. "A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints". In: *International Journal of Production Research* vol. 55, no. 21 (2017), pp. 6288–6304.