UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

MODELOS DE OPTIMIZACIÓN LINEAL ENTERA Y APLICACIONES A LA MINERÍA

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

GONZALO IGNACIO MUÑOZ MARTÍNEZ

PROFESOR GUÍA:
MARTÍN MATAMALA VÁSQUEZ

MIEMBROS DE LA COMISIÓN:
MARCOS GOYCOOLEA GUZMÁN
EDUARDO MORENO ARAYA
DANIEL ESPINOZA GONZÁLEZ
MAURICE NOEL QUEYRANNE
HÉCTOR RAMÍREZ CABRERA

SANTIAGO DE CHILE
JUNIO 2012

RESUMEN DE LA MEMORIA
PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MATEMÁTICO
POR: GONZALO MUÑOZ MARTÍNEZ
FECHA: 25/06/2012
PROF. MARTÍN MATAMALA V.

MODELOS DE OPTIMIZACIÓN LINEAL ENTERA Y APLICACIONES A LA MINERÍA

El tema principal de esta memoria es el estudio del problema de planificación minera, junto con el análisis detallado de un nuevo algoritmo propuesto en la literatura para resolverlo.

El problema de planificación minera consiste en determinar la secuencia de extracción de una mina a tajo abierto. Para esto, típicamente se modela la mina como un conjunto de bloques y se diseña un calendario tentativo de su extracción. En la industria minera se resuelve este problema siguiendo una serie de pasos consecutivos que terminan por entregar una planificación. Por otro lado, existen distintos modelos de optimización que han sido propuestos para resolver este problema, pero debido a que las instancias son de gran tamaño, obtener soluciones requiere de técnicas y algoritmos más elaborados. Una instancia típica posee millones de variables y millones de restricciones.

Por estas razones resulta ser un problema desafiante y con una aplicación muy importante, para el cual distintos tipos de descomposiciones han sido propuestas para resolverlo, junto con pre-procesos, heurísticas o versiones simplificadas de manera de hacer el problema más manejable. Todas estas técnicas pueden resolver problemas del orden de 200.000 bloques, lo cual es bueno, pero está lejos de poder resolver problemas reales. Recientemente, Chicoisne et. al. (Operations Research, 2012) desarrollaron un algoritmo eficiente capaz de resolver instancias con millones de bloques, pero en una versión simplificada del problema. Y por otro lado, Bienstock y Zuckerberg (Optimization Online, 2009) propusieron otro algoritmo, el cual es capaz de resolver eficientemente instancias de millones de bloques y bajo ningún supuesto importante. Este último resulta ser un algoritmo muy ingenioso y será el foco principal de este trabajo estudiarlo.

En el desarrollo de este trabajo se estudió principalmente el problema de planificación minera, junto con la implementación del algoritmo de Bienstock y Zuckerberg. Además, usando algunas técnicas originales y otras técnicas clásicas, se diseñaron una serie de mejoras al algoritmo que lo hacen más eficiente, aprovechando la estructura del problema de planificación minera. Se verá que estas modificaciones producen mejoras significativas en el tiempo necesario para resolver las instancias disponibles.

Y por último, y como un inicio para trabajo futuro, se propone una generalización del algoritmo a un contexto más amplio. Esta generalización se implementó para un nuevo modelo de Optimización Robusta propuesto en este trabajo para el mismo problema de minería, de esta forma dando un primer paso a una nueva manera de considerar incertidumbre en este problema.

AGRADECIMIENTOS

A mi Madre, simplemente por ser mi Madre. Una mujer excepcional cuyas enseñanzas no caben en estas páginas. A mi hermana por haber sido un punto fijo en mi vida Universitaria, y con quien armamos un segundo hogar. Y a mi Padre por hacerse presente cada vez que se necesitara algo.

A la Fernanda por haberse arriesgado en elegir acompañar a un Matemático en los años mas duros de la carrera, y mantenerse ahí, inmóvil sobre todo lo que venga.

A mi familia completa por haber estado siempre presente en este duro viaje que es la Universidad, apoyando y celebrando los logros obtenidos, a pesar de nunca poder explicarles de manera clara qué hace un Ingeniero Matemático.

A mis amigos Renguinos por siempre estar ahí. A pesar de la cantidad de años que han pasado, seguimos provocándonos carcajadas en exceso.

A "los cabros" por haber sido una tremenda compañía durante los años universitarios. Empezamos como una mezcla de personas con distintos orígenes y armamos tremenda historia juntos.

A todos mis compañeros de la oficina 436. Juntos hicimos un ambiente de estudio muy agradable, compartiendo muchos de nuestros intereses científicos más profundos. Se extrañarán las horas mirando la pizarra tratando de resolver algo en grupo.

A Marcos y Eduardo por habérsela jugado absolutamente conmigo. Gracias a ellos desarrollé mi interés por la investigación y probablemente no habría logrado ni la mitad de los logros académicos que he tenido sin el gran apoyo de ambos.

En general se me vienen muchas personas a la cabeza. Gente que hizo que este paso por Beauchef haya sido sumamente entretenido. Un montón "personajes" que han jugado un rol súper interesante en esta etapa que ya se acaba.

- Gonzalo, ¿haz pensado en estudiar Ingeniería Matemática?.
- No... estoy entre varias Ingenierías, pero Matemática no.

Índice general

Capítulo 1

Introducción

1.1. Problemas a gran escala

En esta sección se hará un pequeño repaso de los métodos clásicos para resolver problemas de optimización lineal a gran escala. Se denominará "problemas a gran escala" a problemas con un gran número de variables y/o restricciones, cantidad que hace al problema poco manejable computacionalmente.

En el caso del problema de planificación minera, debido al gran tamaño de las minas a tajo abierto, al formularlo como un problema de optimización lineal entera se debe lidiar con ambos problemas: muchas variables y restricciones. Típicamente se resuelven estos problemas mediante una secuencia de problemas de optimización pequeños.

Probablemente el método más conocido para resolver problemas a gran escala cuando se tiene un gran número de variables es lo que se conoce como *Generación de Columnas*. Si se considera el algoritmo Simplex, en la práctica, para problemas con muchas variables, se observa que la mayoría de las columnas del problema no entran a la base nunca, por lo que se podrían olvidar dichas columnas para hacer el problema más pequeño. De esta manera, se puede comenzar resolviendo un problema considerando sólo un subconjunto de columnas, y luego, con algún método adecuado, se buscan columnas con costo reducido positivo (en el caso de maximización) para ser agregadas. Por ejemplo, para el problema cutting-stock [19], se pueden encontrar dichas columnas usando un problema de optimización discreto "fácil" (un problema knapsack). Y como se verá para el método de Dantzig-Wolfe [10], en ciertos problemas con estructura especial, se pueden encontrar dichas columnas con un problema de programación lineal simple.

Análogamente, si se aplica la idea de generación de columnas, en el dual se están generando restricciones. En general, los métodos que caben dentro de esta descripción se conocen como métodos de *planos cortantes* y sirven para resolver problemas con muchas restricciones. En este caso, al igual que en Generación de Columnas, se considera sólo un subconjunto de restricciones y mediante un problema de optimización auxiliar se van agregando otras.

Por otro lado, si se tiene un problema de optimización del tipo máx$\{cx \mid Ax \leq b, Dx \leq d\}$

donde se supone que optimizar sobre el poliedro $P_A = \{x \mid Ax \leq b\}$ es "fácil", el método de descomposición de Dantzig-Wolfe [10] re-formula el problema de manera de que pueda ser resuelto usando Generación de Columnas, donde cada columna es generada resolviendo un problema del tipo

$$\begin{aligned} \text{máx} \quad & c'x - \mu' Dx \\ \text{s.a} \quad & Ax \leq b \end{aligned}$$

que se supone fácil de resolver. Para una explicación más detallada de Generación de Columnas y del método de Dantzig-Wolfe ver el Apéndice A o [28]. En el caso que se estudiará se tiene una estructura de este tipo, caso que será desarrollado en la Sección 2.2.

Otros métodos usados incluyen relajación Lagrangeana, donde se penalizan restricciones para resolver problemas más fáciles e iterar hasta alcanzar el óptimo; o también la descomposición de Bender, usado principalmente en programación estocástica. Estos métodos también han sido utilizados para abordar el problema de planificación minera.

1.2. El problema de planificación minera

La minería a tajo abierto consiste en la extracción de minerales excavando desde la superficie hacia el fondo. Hustrulid y Kuchta [23] describen el proceso completo de extracción dividiéndolo en tres fases: planificación, implementación y producción. Este trabajo se enfocará en la fase de planificación.

En la fase de planificación se desarrolla un cronograma tentativo para la excavación de la mina en base a la información que se haya recopilado sobre ésta (mediante sondeos, por ejemplo), pero también considerando las limitaciones que posea la planta (capacidad de extracción, de procesamiento, etc.). En palabras simples, se estima qué extraer, cuándo, y cómo procesarlo. Para esto, se discretiza la mina en bloques (normalmente todos del mismo tamaño), donde para cada bloque se tienen múltiples opciones de procesamiento (destinos), y se cuenta con un horizonte de tiempo (periodos) para extraerlo.

Para poder extraer un bloque se debe extraer primero lo que se encuentra "encima" de éste, más aún, se debe extraer un cono sobre el bloque en cuestión que respete un ángulo predeterminado para no comprometer la estabilidad del terreno. Ver Hustrulid et. al. [24]. Al "cono" que se debe extraer antes de un determinado bloque se le llama conjunto de *precedentes* de dicho bloque. La Figura 1.1 muestra un ejemplo de un conjunto de precedentes de un bloque en una mina en 2 dimensiones. Este conjunto de precedentes de cada bloque define restricciones al momento de realizar la planificación; a estas restricciones se les llamará *restricciones de precedencia*.

Cabe mencionar que para describir el conjunto de precedentes por completo no es necesario imponer restricciones sobre todos ellos, sino que basta considerar un conjunto minimal en el conjunto de precedentes, el cual se denomina *precedencia inmediata*, tal que al imponer las restricciones sobre la precedencia inmediata, las restricciones se propaguen a toda la precedencia. Formalmente, si se denota por $P(b)$ el conjunto de precedentes de

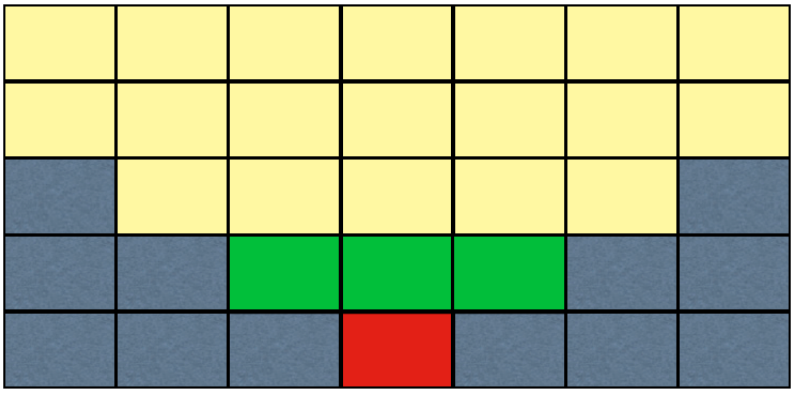


Figura 1.1: Ejemplo de un conjunto de precedentes (bloques amarillos y verdes) de un bloque determinado (bloque rojo) en una mina en 2 dimensiones. Los bloques verdes representan la precedencia inmediata del bloque rojo.

un bloque b, entonces la precedencia inmediata $PI(b)$ está dada por:

$$PI(b) = P(b) \setminus \bigcup_{a \in P(b)} P(a)$$

Por ejemplo, en la Figura 1.1, si bien se necesita extraer los bloques verdes y amarillos antes del bloque rojo, basta imponer que antes del bloque rojo se deban extraer los bloques verdes solamente, para obtener las restricciones que se quieren. Como no hace diferencia teórica qué restricciones se consideran, simplemente se les llamarán *restricciones de precedencia*, sin importar si se está considerando precedencia inmediata o no, pero en la práctica se trabaja con precedencia inmediata para ser más eficientes al momento de formular el problema.

En la industria minera típicamente se divide la etapa de planificación en una serie de pasos para obtener el calendario de extracción tentativo. Para un resumen sobre los pasos de este proceso ver [7]. Este esquema es muy usado por los softwares comerciales diseñados para la fase de planificación (por ejemplo, Whittle de Gemcom Software [33]) pero cuentan con limitaciones importantes, por ejemplo, se decide cómo procesar un determinado bloque sólo tomando en cuenta su composición. Esto es lo que se conoce como *ley de corte*, y básicamente define un umbral bajo el cual se decide qué hacer con un bloque: si es "valioso" se procesa y si no, se bota.

Johnson [25, 26] fue el primero en proponer resolver el problema de planificación minera utilizando un modelo de optimización. El modelo de Johnson define variables que determinan el método de procesamiento (destino) y el periodo de extracción de cada bloque. Además, junto con las restricciones de precedencia, considera una serie de restricciones de recursos como capacidad de los equipos, capacidad de plantas, mano de obra disponible, entre otras. En su formulación, Johnson permite que un bloque sea subdividido en fracciones, donde cada fracción puede ser enviada a distintos destinos, además de permitir que los bloques sean extraídos parcialmente.

$$
\begin{aligned}
\text{máx} \quad & \sum_{b\in\mathcal{B}}\sum_{d=0}^{D-1}\sum_{t=0}^{T-1} p_{b,d,t}(x_{b,d,t} - x_{b,d,t-1}) & \\
\text{s.a} \quad & \sum_{b\in\mathcal{B}}\sum_{d=0}^{D-1} a_{b,d,t,r}(x_{b,d,t} - x_{b,d,t-1}) \leq c_{r,t} & \forall r \in R, \forall t = 0,\ldots,T-1 \\
& \sum_{t=0}^{T-1}\sum_{d=0}^{D-1}(x_{b,d,t} - x_{b,d,t-1}) \leq 1 & \forall b \in \mathcal{B} \\
& x_{a,d,t} \leq x_{b,d,t} & \forall (a,b) \in \mathcal{A}, \forall d \in D, \forall t = 0,\ldots,T-1 \\
& x_{b,d,t} \leq x_{b,d,t-1} & \forall b \in \mathcal{B}, \forall d \in D, \forall t = 0,\ldots,T-1 \\
& x_{b,d,-1} = 0 & \forall b \in \mathcal{B}, \forall d \in D \\
& x_{b,d,t} \in [0,1] & \forall b \in \mathcal{B}, \forall d \in D, \forall t = 0,\ldots,T-1
\end{aligned}
$$

Figura 1.2: Formulación de Johnson para el problema de planificación minera

La Figura 1.2 muestra el modelo considerado por Johnson. En este modelo se considera un conjunto de bloques $\mathcal{B}$, las precedencias entre bloques están representadas como pares ordenados en un conjunto $\mathcal{A}$, se tiene un conjunto de destinos D, un conjunto de recursos R y T periodos de tiempo para extracción. Con esto, $x_{b,d,t}$ representa la fracción del bloque b que ha sido extraído y enviado al destino d en el tiempo t o antes. $c_{r,t}$ representa la cantidad de recurso r para el periodo t, y $a_{b,d,t,r}$ representa el consumo de recurso r cuando el bloque b es enviado al destino d en el tiempo t.

Claramente el problema propuesto por Johnson es de gran tamaño, dado que una mina real cuenta con cientos de miles de bloques y millones de restricciones de precedencia. Formulaciones alternativas también han sido propuestas donde, por ejemplo, no se permite extraer un bloque parcialmente en un periodo, entre otras variaciones y consideraciones. Esto da paso a diversos modelos de programación lineal (LP) como el de Johnson, programación entera mixta (MIP) o programación entera pura (IP), todos ellos de gran tamaño y dificultad. A pesar de ser modelos de distintos tipos, la mayoría de éstos son considerablemente similares.

Junto con las distintas formulaciones y variaciones del modelo, distintas técnicas han sido desarrolladas para resolverlo. El mismo Johnson propuso un algoritmo de tipo Dantzig-Wolfe para resolver su formulación. Dagdelen y Johnson [9] fueron los primeros en proponer relajación Lagrangeana para abordar el problema, usando multiplicadores para descomponer el problema multi-periodo en varios problemas que sólo cuentan con las restricciones de precedencia. Estos últimos pueden ser resueltos eficientemente usando el algoritmo propuesto por Lerchs y Grossman [27].

Diversos métodos de pre-procesamiento también han sido propuestos para abordar este problema. Caccetta y Hill [6] proponen un modelo de programación entera mixta, y luego pre-procesan usando lo que se conoce como Ultimate Pit (ver Sección 4.3.4), para

dar paso a un esquema de Branch and Cut para resolver el problema. Boland et. al. [5] propusieron un método para reducir el número de variables en el modelo de Caccetta y Hill, además de derivar desigualdades para mejorar la formulación del problema en base a las restricciones de producción y las precedencias. Esta última estrategia también es utilizada por Fricke [15] para encontrar desigualdades que mejoran distintas formulaciones. Gaupp [16] también reduce la cantidad de variables utilizadas identificando, en base a las cantidades de recurso, el primer y el último periodo donde es posible extraer cada bloque, para luego generar cortes como en [5] y resolver el problema usando relajación Lagrangeana.

Por otro lado, Gershon [17,18] propuso heurísticas para encontrar una solución factible al problema directamente, utilizando una serie de sub-problemas de optimización lineal de 1 periodo, pero modificando la función objetivo de manera de tomar en cuenta el horizonte de tiempo en el cual se puede explotar una mina. Lamentablemente para estas heurísticas no se presenta un análisis de la calidad de las soluciones obtenidas. Fricke [15] propuso 3 heurísticas para encontrar soluciones factibles, las cuales están basadas en programación entera, relajación Lagrangeana y otras heurísticas (donde simplemente se estima un orden "adecuado" para la extracción de los bloques y se extraen en dicho orden). Estas últimas producen soluciones a un 2 % de optimalidad en algunos casos, pero en general no tienen un buen rendimiento, además de considerar instancias de hasta 420 bloques solamente.

A pesar de todos los esfuerzos, las instancias que se han podido abordar hasta acá con algún método son de menos de 250.000 bloques. Para un resumen completo de las distintas formulaciones y métodos propuestos hasta el 2008 ver Osanloo et. al. [29].

Recientemente, Cullenbine et. al. [8] propusieron una heurística donde se considera un modelo "parcialmente exacto", es decir, un modelo que consiste básicamente en: un sub-modelo exacto (IP) dentro de una "ventana de tiempo" y un sub-modelo relajado fuera de dicha ventana. Esta ventana de tiempo es desplazada a través de los periodos disponibles para el procesamiento de la mina de manera de ir generando una solución factible. Esta heurística mostró funcionar muy bien (obteniendo soluciones dentro de un 2 % de optimalidad), pero sólo pudo resolver instancias de hasta 25.000 bloques.

Por otro lado, Chicoisne et. al. [7] estudiaron una versión simplificada del modelo, donde se considera un destino elegido a-priori para cada bloque, y una restricción de capacidad por cada periodo. En la Figura 1.3 se presenta este modelo simplificado, que se le denomina formulación C-PIT. En este caso se propone un algoritmo muy eficiente que es capaz de resolver la relajación lineal de este problema en instancias de gran tamaño (millones de bloques), pero bajo el supuesto recién descrito. Una vez resuelta la relajación lineal se proponen heurísticas que, a partir del óptimo de la relajación lineal, construyen una solución factible para el problema de programación entera, obteniendo soluciones muy buenas en la práctica (a menos de un 6 % del óptimo).

Y por último, Bienstock y Zuckerberg [3] presentaron un formulación similar a la de Johnson, pero en un contexto más general, permitiendo cualquier tipo de restricción adicional, junto con un algoritmo eficiente que es capaz de resolver la relajación lineal del

$$
\begin{aligned}
\text{máx} \quad & \sum_{b\in\mathcal{B}}\sum_{t=0}^{T-1} p_{b,t}(x_{b,t} - x_{b,t-1}) & \\
\text{s.a} \quad & \sum_{b\in\mathcal{B}} a_b(x_{b,t} - x_{b,t-1}) \leq c_t & \forall t = 0,\dots,T-1 \\
& x_{a,t} \leq x_{b,t} & \forall (a,b) \in \mathcal{A}, \forall t = 0,\dots,T-1 \\
& x_{b,t} \leq x_{b,t-1} & \forall b \in \mathcal{B}, \forall t = 0,\dots,T-1 \\
& x_{b,-1} = 0 & \forall b \in \mathcal{B} \\
& x_{b,t} \in \{0,1\} & \forall b \in \mathcal{B}, \forall t = 0,\dots,T-1
\end{aligned}
$$

Figura 1.3: Formulación C-PIT

problema. Este algoritmo, aparte de su robustez en cuanto al tipo de restricciones adicionales que se pueden considerar, ha mostrado ser capaz de resolver grandes instancias en poco tiempo (considerando millones de bloques), lo que motivó su estudio cuidadoso.

Siguiendo con estas ideas, el objetivo principal de este trabajo será encontrar eficientemente la solución de la relajación lineal de la formulación que se presentará, utilizando el algoritmo propuesto por Bienstock y Zuckerberg junto con ciertas mejoras propuestas, para luego utilizar heurísticas, principalmente inspiradas en lo desarrollado por Chicoisne et. al. y Cullenbine et. al., que lleven a una buena solución factible.

En el Capítulo 2 se dará una mirada general al algoritmo de Bienstock y Zuckerberg, comparándolo con el método de Dantzig-Wolfe, con el cual comparte ciertos aspectos. Luego, en los Capítulos 3 y 4 se profundizará en el caso del problema de planificación minera y se discutirá la implementación del algoritmo para este caso, mostrando las mejoras desarrolladas para éste. Los resultados computacionales se detallarán en el Capítulo 5 y en el Capítulo 6 se mostrarán experimentos realizados con algunas heurísticas para obtener soluciones factibles.

1.3. Considerando incertidumbre

Uno de los problemas de los modelos y algoritmos de optimización tradicionales desarrollados para resolver el problema de planificación minera es que no consideran incertidumbre sobre los parámetros del problema.

Las fuentes de incertidumbre a las que se ve expuesta este problema se pueden agrupar principalmente en 3 tipos:

- Incertidumbre geológica.
- Incertidumbre en las especificaciones técnicas (capacidades, precedencias, etc.).
- Incertidumbre económica, como capital, costos, etc.

El primer tipo de incertidumbre ha sido altamente estudiado en el último tiempo, problema para el cual se han propuesto diversas técnicas para ser abordado. Principalmente

se han usado técnicas de simulación para obtener buenas soluciones al problema. Por ejemplo, Dimitrakopoulos [11] usa simulación condicional para generar representaciones equi-probables de la mina y así evaluar una planificación y/o generar sugerencias. Dimitrakopoulos et. al. [12] usaron simulación condicional para mostrar la importancia de tomar en cuenta la incertidumbre geológica de una mina y el riesgo al momento de resolver el problema. Otros trabajos donde también se ha usado simulación incluyen Dowd [14], Dimitrakopoulos et. al. [13], entre otros.

Otras maneras de lidiar con este tipo de incertidumbre también han sido propuestas. Godoy y Dimitrakopoulos [20] desarrollaron un algoritmo basado en Optimización Lineal que usa simulación para generar una serie de planificaciones y luego las combina de manera de minimizar la probabilidad de desviarse de un objetivo de producción. Ramazan y Dimitrakopoulos [31] proponen considerar la incertidumbre geológica usando simulación como una manera de estimar la probabilidad de que un bloque sea extraído en un determinado periodo, para luego usar un modelo de programación entera mixta que use estas probabilidades para maximizar el VPN (valor presente neto) de la extracción de la mina, pero a la vez tratando de que la solución sea similar a lo predicho por las simulaciones.

También se ha abordado este problema usando Optimización Estocástica. Entre ellos se encuentran Ramazan y Dimitrakopoulos [32] quienes usaron simulación condicional incorporada a un modelo de Optimización Estocástica que maximiza el VPN considerando incertidumbre, y además penalizando el que no se cumpla cierto objetivo predeterminado para la producción. Boland et. al. [4] también usaron Optimización Estocástica, desarrollando un modelo que permite alterar fácilmente las decisiones tomadas a medida que se obtiene mayor información geológica de la mina (por ejemplo, a medida que se excava).

En este trabajo, y como paso final al desarrollo de esta memoria, se decidió abordar el problema de incertidumbre económica, más específicamente, incertidumbre en el precio de los minerales que se extraen en la mina. Esto surgió de la idea de desarrollar una generalización del algoritmo de Bienstock y Zuckerberg, de manera de poder aplicar un esquema de algoritmo similar en un contexto distinto al de programación lineal.

En el Capítulo 7 se propondrá un modelo de Optimización Robusta para considerar incertidumbre en los precios de los minerales. También se verá que este modelo puede ser planteado como un problema de Optimización Cónica, por lo cual se propondrá una generalización del algoritmo de Bienstock y Zuckerberg a este contexto.

Hasta donde es conocido, este problema no ha sido abordado anteriormente, al menos desde un punto de vista de Optimización Robusta. Todos los trabajos han sido apuntados a incertidumbre geológica, por lo que es un gran aporte poder desarrollar un algoritmo que sea capaz de resolver un modelo que considere variaciones en los precios de los minerales. Además, el modelo que se propondrá puede ser fácilmente extendido a incertidumbre geológica, por lo que el mismo algoritmo puede ser utilizado.

Capítulo 2

Algoritmo BZ General

En el siguiente capítulo se detallará el algoritmo general propuesto por Bienstock y Zuckerberg (ver [3]). Es este esquema de algoritmo el que se estudiará a lo largo de este documento, mostrando su implementación para el caso del problema de planificación minera. Para comenzar, se dará una visión global del algoritmo.

2.1. El algoritmo

Se considera el siguiente problema de programación lineal:

$$\begin{array}{lll} (P1) & \text{máx} & c'x \\ & \text{s.a} & Ax \leq b \\ & & Dx \leq d \end{array}$$

Con esto, se usará el siguiente esquema de algoritmo:

1. Comenzar con z^0 factible.
2. Definir $k = 1$ y $\mu^0 = \vec{0}$.
3. Sea w^k solución de $L(P1, \mu^{k-1})$, donde

$$\begin{array}{lll} L(P1, \mu^{k-1}) & \text{máx} & c'x - (\mu^{k-1})'(Dx - d) \\ & \text{s.a} & Ax \leq b \end{array}$$

4. Si $k \geq 2$ y $H^{k-1}w^k = h^{k-1}$ PARAR.
5. Buscar H^k y h^k tales que $H^k w^k = h^k$.
6. Sea z^k una solución óptima de $P2^k$, donde

$$\begin{array}{lll} (P2^k) & \text{máx} & c'x \\ & \text{s.a} & Ax \leq b \\ & & Dx \leq d \\ & & H^k x = h^k \end{array}$$

7. Obtener los duales μ^k de las restricciones $Dx \leq d$.

8. Si $\mu^k = \mu^{k-1}$ PARAR.

9. $k \leftarrow k+1$ e ir al paso 3.

Se denotará este algoritmo como BZ. La idea es que los problemas $L(P1, \mu^{k-1})$ y $P2^k$ sean fáciles de resolver. Se llamará a $L(P1, \mu^{k-1})$ el *sub-problema* y a $P2^k$ el *problema master*.

Observación 2.1.1. Como se verá más adelante, este algoritmo es correcto independiente de las elecciones de H^k y h^k, pero la idea de agregar este sistema es hacer que cada problema $P2^k$ sea considerablemente más simple que el problema original $P1$. Considerar un sistema trivial, como por ejemplo $H^k = 0$ y $h^k = 0$, tendría como consecuencia que el problema $P2^1$ sea lo mismo que resolver el problema original, lo cual no es el objetivo del algoritmo.

Es fácil ver que $X^k = \{x \mid H^k x = h^k\}$ define un espacio lineal afín, y si se considera X^k tal que X^{k-1} sea un sub-espacio del primero, se obtiene un algoritmo finito. En efecto, como $w^k \notin X^{k-1}$, entonces

$$\dim(X^k) > \dim(X^{k-1}).$$

Antes de probar la correctitud del algoritmo se requiere el siguiente resultado:

Proposición 2.1.2. *Sea x^* una solución óptima del problema* $\max\{c'x \mid Ax \leq b, Dx \leq d\}$ *y μ^* duales óptimos asociados a las restricciones $Dx \leq d$, entonces x^* es óptimo para el problema*

$$\begin{array}{ll} \max & c'x - (\mu^*)'(Dx - d) \\ s.a & Ax \leq b \end{array} \tag{2.1}$$

La prueba de esta Proposición sigue directamente de dualidad fuerte (ver la demostración en [2] por ejemplo, o la demostración hecha en la Proposición 7.3.1 para un caso más general).

Proposición 2.1.3. *Si el algoritmo alcanza una condición de parada, entonces el último z^k encontrado (óptimo para $P2^k$) es óptimo.*

Demo :
Sea k la iteración donde se paró. Si el algoritmo paró pues $H^{k-1}w^k = h^{k-1}$, entonces claramente w^k es $L(P2^{k-1}, \mu^{k-1})$-factible, problema donde las restricciones $Dx \leq d$ de $P2^{k-1}$ son penalizadas, i.e

$$\begin{array}{lll} L(P2^{k-1}, \mu^{k-1}) & \max & c'x - (\mu^{k-1})'(Dx - d) \\ & \text{s.a} & Ax \leq b \\ & & H^{k-1}x = h^{k-1} \end{array}$$

Sea v el valor de la función objetivo de $L(P2^{k-1}, \mu^{k-1})$ en w^k, v^{k-1} el valor óptimo de $P2^{k-1}$ (que es alcanzado en z^{k-1}) y v^* el valor óptimo de $P1$. Claramente $v^{k-1} \leq v^*$. Por otro lado, v también es el valor de la función objetivo de $L(P1, \mu^{k-1})$ en w^k, entonces $v^* \leq v$.

Finalmente, dado que μ^{k-1} son óptimos para el dual de $P2^{k-1}$, por la Proposición 2.1.2, z^{k-1} es óptimo para $L(P2^{k-1}, \mu^{k-1})$, entonces $v \leq v^{k-1}$. Juntando todo:

$$v^* \leq v \leq v^{k-1} \leq v^*$$

lo que prueba la optimalidad de z^{k-1}.

Si el algoritmo paró porque $\mu^k = \mu^{k-1}$ entonces se puede elegir $w^{k+1} = w^k$ y se tiene que es óptimo para $L(P1, \mu^k)$. Además se tendría que $H^k w^{k+1} = h^k$, con lo que se obtiene la otra condición de parada, de donde se deduce el resultado.

■

2.2. Comparación con el método de Dantzig-Wolfe

Al estudiar este algoritmo, se notaron ciertas similaridades con el método de descomposición de Dantzig-Wolfe [10], mencionado brevemente en la Sección 1.1 y explicado en el Apéndice A, por lo que se decidió explorar este aspecto para determinar las diferencias de estos 2 métodos. Se le llamará a este método DW.

2.2.1. El método de descomposición de Dantzig-Wolfe

Se considera el siguiente algoritmo para resolver el problema $P1$:

1. Comenzar con un punto extremo x^0 del poliedro definido por $Ax \leq b$.
2. Definir $k = 1$ y $\mu^0 = \vec{0}$.
3. Sea λ una solución óptima de

$$\begin{aligned} \text{máx} \quad & \sum_{i=0}^{k-1} \lambda_i c' x^i \\ \text{s.a} \quad & \sum_{i=0}^{k-1} \lambda_i D x^i \leq d \\ & \sum_{i=0}^{k-1} \lambda_i = 1 \\ & \lambda_i \geq 0 \end{aligned} \tag{2.2}$$

   Por simplicidad se asumirá que x^0 es factible para $P1$, si no, se debe considerar un conjunto de puntos extremos iniciales tales que este problema no sea infactible.

4. Obtener los duales μ^k de las restricciones $\sum_{i=0}^{k-1} \lambda_i D x^i \leq d$ y α^k dual asociado a la restricción $\sum_{i=0}^{k-1} \lambda_i = 1$.

5. Sea x^k una solución de
$$\begin{array}{ll} \text{máx} & c'x - (\mu^{k-1})'Dx \\ \text{s.a} & Ax \leq b \end{array} \tag{2.3}$$

6. Si $c'x^k - (\mu^{k-1})'Dx^k > \alpha^k$ PARAR. La solución óptima es $x^* = \sum_{i=0}^{k-1} \lambda_i x^i$.

7. $k \leftarrow k+1$ e ir al paso 3.

La idea es reemplazar el poliedro $\{x \mid Ax \leq b\}$ por la combinación convexa de sus puntos extremos, y en vez de considerarlos todos de una vez, se van generando uno a uno. Esto es un caso particular del conocido método de Generación de Columnas.

Sea $\mathcal{X} = \{x^1, \ldots, x^{k-1}\}$ un conjunto de puntos extremos del poliedro $\{x \mid Ax \leq b\}$. Entonces el dual del problema (2.2) es

$$\begin{array}{ll} \text{mín} & \mu'd + \alpha \\ \text{s.a.} & \\ & \alpha + \mu' D x^i \geq c' x^i \qquad \forall i \\ & \mu \geq 0 \end{array} \tag{2.4}$$

Por lo tanto, para saber si se han generado suficientes puntos extremos x^i, para los duales actuales μ^k y α^k se verifica si la desigualdad $\alpha^k + (\mu^k)'Dx^i \geq c'x^i$ es satisfecha por todos los puntos extremos del poliedro, lo cual es equivalente a resolver (2.3) y verificar si la solución es mayor a α^k. Esto es equivalente a encontrar columnas con costo reducido positivo que deban ser incluidas en (2.2).

2.2.2. El método BZ desde otro punto de vista

Como se mencionó anteriormente, en el algoritmo BZ, en cada problema $P2^k$ se está optimizando sobre un poliedro intersectado con $X^k = \{x \mid H^k x = h^k\}$, el cual define un espacio lineal afín. Para simplificar notación se asumirá que este espacio es lineal (la única diferencia la hace una constante).

Por lo tanto, como (2.3) es igual al problema $L(P1, \mu^{k-1})$, se puede ver que la principal diferencia entre DW y BZ es que en vez de resolver (2.2), se está considerando un espacio lineal generado por cierta colección vectores $\{x^i\}_i$ (abusando de notación) y se resuelve el problema master

$$\begin{array}{ll} \text{máx} & \sum_i \lambda_i c' x^i \\ \text{s.a.} & \\ & \sum_i \lambda_i A x^i \leq b \\ & \sum_i \lambda_i D x^i \leq d \end{array} \tag{2.5}$$

Claramente este problema es una relajación de (2.2), dado que $\sum_i \lambda_i = 1$ implica $\sum_i \lambda_i A x^i \leq b$. Si se supone que se resuelve (2.5), se tiene que el dual de este problema es:

$$\begin{array}{ll} \text{mín} & \rho' b + \mu' d \\ \text{s.a.} & \\ & \rho' A x^i + \mu' D x^i = c' x^i \qquad \forall i \\ & \rho \geq 0 \\ & \mu \geq 0 \end{array} \tag{2.6}$$

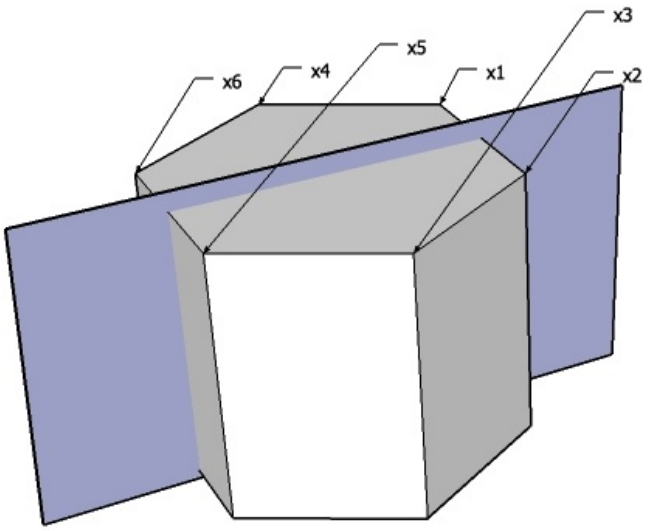


Figura 2.1: Ejemplo donde Dantzig-Wolfe puede tener un bajo desempeño

Entonces, ¿cuáles serían las ventajas de BZ sobre DW? La principal razón es que en (2.5) se está explorando un espacio lineal, en vez de la envoltura convexa de los puntos generados. Si se toma el ejemplo ilustrado en la Figura 2.1, donde el poliedro octagonal representa la región definida por $\{x \mid Ax \leq b\}$ y el hiper-plano representa la región definida por $\{x \mid Dx \leq d\}$, entonces el algoritmo DW podría necesitar generar todos los puntos $x_1, \ldots, x_6$ antes de salir de la tapa superior del octágono. En cambio, si se considera (2.5), basta generar 3 puntos para describir toda la cara superior. Además se tiene la siguiente proposición:

Proposición 2.2.1. *Si μ es factible para el problema (2.6), entonces $\exists\ \alpha$ tal que μ, α es factible para (2.4).*

Demo :
Como μ es factible, entonces existe $\rho \geq 0$ tal que

$$c'x^i = \rho' A x^i + \mu' D x^i$$

Ahora, sea $\alpha = \rho' b$. Como $Ax^i \leq b$ para todo i y $\rho \geq 0$ implica $\rho' A x^i \leq \rho' b$. Entonces,

$$\begin{aligned} c'x^i &= \rho' A x^i + \mu' D x^i \\ &\leq \rho' b + \mu' D x^i \\ &= \alpha + \mu' D x^i \end{aligned}$$

Por lo tanto, las variables α, μ son factibles para (2.4).

■

Esto significa que considerando el algoritmo BZ, se están tomando en cuenta menos duales (o la misma cantidad) que con el método de Dantzig-Wolfe, lo que refuerza la idea de moverse *más rápido* al considerar un problema master aparentemente un poco más difícil.

Nombre	Variables	Rest. 1	Rest. 2	CPX	DW	BZ
mine_n	12720	23544	12	8.44	0.11	0.1
mine_k	339672	2637360	24	95631.97	47.55	10.89
mine_z_small	376000	2912840	40	327359.74	72.5	13.05
mine_a	651680	1485240	40	29785.60	38.96	4.57
mine_z_medium	878310	19068135	30	Mem	439.33	101.17
mine_m	2130840	13128260	40	Mem	73.39	14.21
mine_m_simple	2146720	13128260	40	Mem	292.9	23.2
mine_mc_limit	3380610	45532275	30	Mem	862.5	207.17
mine_g	3651354	24920028	63	Mem	3954.4	244.86
mine_z_large	5809260	31593210	60	Mem	6590.58	472.33
mine_s	5940840	2899320	60	5700.51	2306.42	28.5
mine_d	17129574	303762273	102	Mem	20133.2	1566.93
mine_t_full	39583080	423991200	120	Mem	Mem	12918.34
mine_r	49406274	1268887356	162	Mem	Mem	Mem
mine_t	52228480	121466240	160	Mem	Mem	Mem
mine_mc	85613680	60709700	40	Mem	1376.06	662.32
mine_e	110109800	3259280450	100	Mem	Mem	6609
mine_l	149714910	2773201200	90	Mem	Mem	Mem

Cuadro 2.1: Tiempo (en segundos) necesarios para resolver las instancias disponibles. La columna *Rest. 1* posee el número de restricciones "fáciles" del problema, es decir las denotadas anteriormente como $Ax \leq b$. La columna *Rest. 2* posee la cantidad restricciones $Dx \leq d$. Y las columnas CPX, DW y BZ poseen los tiempos de ejecución de CPLEX, DW y BZ al resolver cada instancia. *Mem* indica que no se pudo resolver la instancia por problemas de memoria.

2.3. Comparación computacional

Para comparar el rendimiento de estos métodos se aprovecharon las instancias del problema de planificación minera disponibles, las cuales fueron probadas en una CPU modelo Intel(R) Xeon(R) con 2.4 GHz y con 24 GB de memoria RAM. El algoritmo BZ utilizado fue el que se describirá en el Capítulo 4, y que posee todos los speed-ups que se describirán ahí. Si bien no se han descrito cuáles serán las mejoras que se harán, basta mencionar que fueron añadidas tanto a BZ como a DW para que la comparación fuera válida.

El Cuadro 2.1 muestra los resultados obtenidos al ejecutar el algoritmo BZ y DW en las instancias disponibles. Además se decidió agregar los resultados obtenidos al tratar de resolver dichas instancias con la librería de optimización CPLEX [34].

Lo primero que se puede observar es la efectividad de DW para resolver problemas a gran escala. CPLEX sólo resolvió 5 instancias, mientras DW resolvió la mayoría en buenos tiempos. Sin embargo, existe una superioridad clara de BZ sobre DW: BZ puede resolver más instancias y las que se pueden resolver con ambos algoritmos requieren mu-

cho menos tiempo si se usa BZ en vez de DW. La comparación quizás no sea del todo justa pues DW es un algoritmo más general y la versión implementada del algoritmo BZ es específica para el problema de planificación minera, pero se usaron todos los speed-ups diseñados para el problema de planificación minera en ambos algoritmos para hacer una mejor comparación.

Capítulo 3

Problema de planificación en minería

En este capítulo se definirá el modelo a utilizar para resolver el problema de planificación en minería a tajo abierto. Este problema sirvió como motivación para el estudio del algoritmo BZ y a continuación será descrito. En el Capítulo 4 se discutirán los detalles de la implementación del algoritmo para este problema específico.

3.1. El problema Max-Closure

Como paso previo a la descripción del problema y del modelo, se definirá el problema Max-Closure, que es de gran importancia para el problema de minería.

Definición 3.1.1. Dado un grafo dirigido $G = (V, A)$ un conjunto $S \subseteq V$ se dice *cerrado* si

$$u \in S \wedge (u, v) \in A \Rightarrow v \in S$$

A un conjunto cerrado S también se le llama *closure*. Con esta definición, dados pesos w_i para $i \in V$, se define el problema de Max-Closure como:

$$\begin{array}{ll} \text{máx} & \sum_{i \in V} w_i x_i \\ \text{s.t.} & x_i \leq x_j \quad \forall (i,j) \in A \\ & x_i \in \{0, 1\} \end{array} \tag{3.1}$$

es decir, encontrar el conjunto cerrado de peso máximo. A las restricciones $x_i \leq x_j$ se les llamará restricciones de *precedencia*, y son las que aseguran que la solución entregada por el problema, definida por $S^* = \{i \in V \mid x_i^* = 1\}$ con x^* óptimo, sea efectivamente un closure.

Es fácil ver que la relajación lineal de (3.1) entrega soluciones enteras, en efecto, la matriz definida por las restricciones de precedencia junto con las cotas $0 \leq x_i \leq 1$ definen una matriz totalmente unimodular. Por lo tanto, basta resolver la relajación lineal de este problema para encontrar el óptimo.

Picard [30] mostró que este problema se puede formular como un problema Min-Cut, y por lo tanto se puede resolver con cualquier algoritmo adecuado para resolver este último. La construcción desarrollada por Picard es bastante simple: se define un nuevo grafo $G' = (V', A')$ donde $V' = V \cup \{s, t\}$ con s y t nuevos nodos. Si el peso w_i es positivo, entonces se agrega el arco (s, i) a A' con capacidad w_i. Si el peso w_i es negativo se agrega

un arco (i, t) a A' con capacidad $-w_i$. Y por último, a cada arco $(i, j) \in A$ se le asocia el mismo arco $(i, j) \in A'$ con capacidad infinita. Claramente cualquier corte de capacidad mínima en G' no incluirá arcos de capacidad infinita, más aún, si el corte de capacidad mínima en G' queda definido por $(S, \bar{S})$ con $s \in S$ y $\bar{S} = V' \setminus S$ entonces $S \setminus \{s\}$ corresponde al closure de peso máximo en G.

Como se verá a continuación, este problema es de gran importancia al momento de formular el problema de planificación minera en una mina a tajo abierto.

3.2. El modelo

En esta sección se detallará el modelo ha utilizar, y que es el usado por Bienstock y Zuckerberg [3]. Para el modelo se considerará lo siguiente:

- El conjunto de bloques de la mina se denotará como $\mathcal{B} = \{0, \ldots, n-1\}$.
- Se considerará un grafo dirigido $G = (\mathcal{B}, \mathcal{A})$ representando las restricciones de precedencia, es decir, $(a, b) \in \mathcal{A}$ si el bloque b debe ser extraído antes que el bloque a.
- El número de periodos será denotado T (empezando desde el periodo 0) y el número de destinos R (también empezando desde el 0).
- Las variables $y_{b,d,t}$ indicarán si el bloque b es extraído en el periodo t y enviado al destino d. Cabe mencionar que y es en realidad un vector (se usará notación matricial), pero por simplicidad se usarán los 3 subíndices.
- Las restricciones adicionales, o *side-constraints* serán denotadas como $\hat{D}y \leq d$ y a priori serán *cualquier* tipo de restricción. El número de side-constraints será notado como q. No confundir el lado derecho de las side-constraints con los subíndices de las variables que identifican los destinos, se usa la misma letra, pero el contexto dejará de lado toda ambigüedad.
- Todos los bloques tendrán un valor asociado al ser extraídos y enviados a un determinado destino $\tilde{c}_{b,d}$ en el primer periodo. Y para los otros periodos se considerará una tasa de descuento para cada bloque (esto pues típicamente se maximiza el Valor Presente Neto de la extracción de la mina). Por lo tanto, el beneficio obtenido al extraer un bloque b en el periodo t y enviarlo al destino d será:

  $$\hat{c}_{b,d,t} = \alpha^t \tilde{c}_{b,d}$$

  con $\alpha \in (0, 1)$.

Con estas consideraciones, se puede formular el problema directamente como un problema de programación entera:

$$\begin{aligned}
&\text{máx} \quad \hat{c}'y \\
&\text{s.t.} \quad \sum_{t=0}^{\tau-1}\sum_{d=0}^{R-1} y_{a,d,t} \leq \sum_{t=0}^{\tau-1}\sum_{d=0}^{R-1} y_{b,d,t} \quad \forall (a,b) \in \mathcal{A}, \tau = 0,\ldots,T-1 \\
&\quad \sum_{t=0}^{T-1}\sum_{d=0}^{R-1} y_{b,d,t} \leq 1 \quad \forall b \in \mathcal{B} \\
&\quad \hat{D}y \leq d \\
&\quad y_{b,d,t} \in \{0,1\}
\end{aligned} \tag{3.2}$$

Una versión un poco más refinada de esta formulación permite sub-dividir cada bloque en fracciones y enviar cada fracción a distintos destinos si fuera conveniente, como en el modelo de Johnson. Pero como el enfoque será la relajación lineal de este problema, en ese caso no hace diferencia si se permite sub-dividir los bloques.

Como comentario final sobre este problema, los tamaños aproximados de las instancias que se tienen disponibles son los siguientes:

- Periodos : 6 - 80.
- Destinos : 2 o 3.
- Side-constraints : 12 - 200.
- Bloques : 1,000 - 2,000,000.
- Precedencias : 3,000 - 90,000,000.

Por lo que claramente es un problema muy difícil de resolver, inclusive su relajación lineal.

3.3. Relajación lineal del problema

Como fue mencionado anteriormente, se trabajará con la relajación lineal del problema (3.2). La idea es resolverla para obtener una cota superior del problema y/o utilizar su solución en la construcción de una solución factible. Se le llamará a esta relajación PCP^{at} (Precedence Constrained Problem):

$$\begin{aligned}
(PCP^{at}) \quad &\text{máx} \quad \hat{c}'y \\
&\text{s.t.} \quad \sum_{t=0}^{\tau-1}\sum_{d=0}^{R-1} y_{a,d,t} \leq \sum_{t=0}^{\tau-1}\sum_{d=0}^{R-1} y_{b,d,t} \quad \forall (a,b) \in \mathcal{A}, \tau = 0,\ldots,T-1 \\
&\quad \sum_{t=0}^{T-1}\sum_{d=0}^{R-1} y_{b,d,t} \leq 1 \quad \forall b \in \mathcal{B} \\
&\quad \hat{D}y \leq d \\
&\quad y \geq 0
\end{aligned}$$

El *at* es debido a que la formulación decide si un bloque es extraído *en* un determinado destino y periodo, al contrario de la siguiente formulación que será presentada. Se puede transformar esta formulación a una mejor usando el siguiente Teorema, probado en [3].

Teorema 3.3.1. *El problema PCP^{at} puede ser reformulado como la relajación lineal de un problema Max-Closure con side-constraints.*

Demo :
Se define el siguiente sistema de inecuaciones sobre nuevas variables $x_{b,d,t}$:

$$\begin{aligned} x_{b,0,0} &\geq 0 \\ x_{b,R-1,t} &\leq x_{b,0,t+1} \quad t < T-1 \\ x_{b,d,t} &\leq x_{b,d+1,t} \quad d < R-1 \\ x_{b,R-1,T-1} &\leq 1 \\ x_{a,R-1,t} &\leq x_{b,R-1,t} \quad \forall (a,b) \in \mathcal{A}, \forall t \end{aligned}$$

Se define la función $atbyKey : \mathcal{B} \times [0, D-1] \times [0, T-1] \to \mathcal{B} \times [0, D-1] \times [0, T-1]$ como

$$\text{atbyKey}(b,d,t) = \begin{cases} (b, R-1, t-1) & \text{si } d = 0, t > 0 \\ (b, d-1, t) & \text{si } d > 0 \\ \text{no definida} & \text{si } d = 0, t = 0 \end{cases}$$

Con esta función se tiene una correspondencia uno-a-uno con el espacio definido por

$$\begin{aligned} \sum_{t=0}^{\tau-1} \sum_{d=0}^{R-1} y_{a,d,t} &\leq \sum_{t=0}^{\tau-1} \sum_{d=0}^{R-1} y_{b,d,t} \quad \forall (a,b) \in \mathcal{A}, \tau = 0, \ldots, T-1 \\ \sum_{t=0}^{T-1} \sum_{d=0}^{R-1} y_{b,d,t} &\leq 1 \quad \forall b \in \mathcal{B} \\ y &\geq 0 \end{aligned}$$

de hecho, se puede ir de un sistema a otro usando las transformaciones

$$\begin{aligned} x_{b,d,t} &= \sum_{t'=0}^{t-1} \sum_{d'=0}^{R-1} y_{b,d',t'} + \sum_{d'=0}^{d} y_{b,d',t} \\ y_{b,d,t} &= \begin{cases} x_{b,d,t} - x_{\text{atbyKey(b,d,t)}} & \text{si } d \neq 0 \vee t \neq 0 \\ x_{b,d,t} & \text{si } d = 0, t = 0 \end{cases} \end{aligned} \tag{3.3}$$

Usando esto, y definiendo c y D a partir de $\hat{c}$ y $\hat{D}$ apropiadamente, se obtiene la siguiente formulación equivalente

$$\begin{aligned} (PCP^{by}) \quad \text{máx} \quad & c'x \\ x_{b,0,0} &\geq 0 \\ x_{b,R-1,t} &\leq x_{b,0,t+1} \quad t < T-1 \\ x_{b,d,t} &\leq x_{b,d+1,t} \quad d < R-1 \\ x_{b,R-1,T-1} &\leq 1 \\ x_{a,R-1,t} &\leq x_{b,R-1,t} \quad \forall (a,b) \in \mathcal{A}, \forall t \\ Dx &\leq d \end{aligned}$$

A esta formulación se le denomina la formulación *by* y resulta ser la relajación lineal de un problema Max-Closure con side-constraints. En esta nueva formulación aparecen nuevas restricciones de precedencia.

■

Observación 3.3.2. Claramente esta equivalencia entre formulaciones *at* y *by* no depende de las side-constraints, por lo que se usarán ambas formulaciones en distintos contextos: con y sin side-constraints.

Observación 3.3.3. De la demostración del Teorema 3.3.1 se puede ver que la mina se puede representar como un nuevo grafo $G' = (\mathcal{B}', \mathcal{A}')$, al cual se le desea calcular su Max-Closure con side-constraints. En este nuevo grafo se tiene un nodo por cada tupla (b, d, t) y tiene arcos de acuerdo a las restricciones de precedencia descritas en la demostración ya mencionada.

La importancia de contar con esta formulación alternativa es que, cuando se utilice el algoritmo BZ, se pueden penalizar las side-constraints y obtener un problema Max-Closure en el sub-problema, el cual se puede resolver eficientemente. Esto será desarrollado en el siguiente capítulo.

Capítulo 4

Implementación del algoritmo BZ

En el siguiente capítulo se discutirá la implementación desarrollada para el caso del problema de planificación minera. Para este caso, el algoritmo BZ se puede utilizar de manera que el problema master y el sub-problema sean considerablemente más fáciles de resolver.

4.1. El algoritmo BZ para el caso de minería

De acuerdo al Teorema 3.3.1, el problema PCP^{by} puede formularse como

$$
\begin{array}{lrrll}
(PCP^{by}) & \text{máx} & c'x & & \\
 & \text{s.t.} & x_a & \leq & x_b \quad \forall (a,b) \in \mathcal{A}' \\
 & & Dx & \leq & d \\
 & & x & \in & [0,1]^N
\end{array}
$$

donde $G' = (\mathcal{B}', \mathcal{A}')$ es el nuevo grafo que representa a la mina considerando todos los destinos y periodos de extracción (discutido en la Observación 3.3.3). Como es usual, $Dx \leq d$ son las side-constraints. Se representarán las restricciones de precedencia junto con las cotas para las variables como $Ax \leq b$, obteniendo un problema como $P1$, donde optimizar sobre $\{x \mid Ax \leq b\}$ es "fácil".

Cabe mencionar que al aplicar el algoritmo BZ en este contexto se deberá resolver el sub-problema $L(PCP^{by}, \mu^{k-1})$, problema PCP^{by} donde se penalizan las side-constraints con el multiplicador μ^{k-1}. Este problema tiene la forma de un problema Ultimate Pit (el cual se discutirá en la Sección 4.3) y es equivalente a resolver un problema Max-Closure. Esto fue observado por primera vez por Lerchs y Grossman [27] quienes propusieron un algoritmo basado en teoría de grafos para resolver este problema.

4.1.1. Caracterizaciones

En este caso, el problema posee una fuerte estructura que permite determinar explícitamente qué sistema $H^k x = h^k$ es conveniente de imponer iteración tras iteración. Para esto se revisarán los siguientes resultados detallados en [3]:

Lema 4.1.1. *Sea* $P = \{x \in \mathbb{R}^N \mid Ax \leq b, Dx \leq d\}$ *la región factible de* PCP^{by} *y* x *un punto extremo de* P*. Se denotan por* $\bar{A}x = \bar{b}$*,* $\bar{D}x = \bar{d}$ *las restricciones activas en* x *y sea* $\bar{q}$ *el número de filas linealmente independientes de* $\bar{D}$*. Si se denota por* $\ker(\bar{A})$ *al núcleo de* $\bar{A}$*, entonces se tiene que* $\ker(\bar{A})$ *puede ser generado por* $\bar{q}$ *vectores.*

Demo :
Como x es punto extremo, y $\bar{D}$ tiene $\bar{q}$ filas linealmente independientes, entonces $\bar{A}$ debe tener al menos $N - \bar{q}$ filas linealmente independientes. Luego, por el teorema núcleo-imagen, se tiene que $\dim(\ker(\bar{A})) \leq \bar{q}$

■

También se necesitará el siguiente resultado conocido:

Lema 4.1.2. *Sea* $Q = \{x \in \mathbb{R}^N \mid Rx \leq r\}$ *un poliedro cualquiera. Si los coeficientes de la matriz* R *y del vector* r *son números racionales, entonces todo punto extremo de* Q *solamente tiene coeficientes racionales.*

Demo :
Sea $\bar{x}$ un punto extremo de Q. Por la equivalencia entre punto extremo y solución básica factible se tiene que existe una sub-matriz de R, que se denotará $\bar{R}$, de rango N tal que

$$\bar{R}\bar{x} = \bar{r}$$

donde $\bar{r}$ son los coeficientes correspondientes a las filas de $\bar{R}$. Como $\bar{R}$ es de rango completo, entonces

$$\bar{x} = (\bar{R})^{-1}\bar{r}.$$

Es fácil ver que $(\bar{R})^{-1}$ tiene sólo coeficientes racionales, en efecto, si se computa la inversa de $\bar{R}$ mediante la eliminación de Gauss-Jordan, las matrices elementales sólo tendrán coeficientes racionales, por lo que $(\bar{R})^{-1}$ también lo hará.

Por lo tanto, como $(\bar{R})^{-1}$ y $\bar{r}$ sólo poseen coeficientes racionales, $\bar{x}$ también lo hará, lo que concluye la demostración.

■

Ahora, sea P el espacio factible del problema PCP^{by}, donde se tienen q side-constraints, y sea x un punto extremo de P. Desde ahora en adelante se asumirá que todos los datos del problema son números racionales, lo cual es razonable dado que sólo se pueden manejar este tipo de números computacionalmente.

Gracias al Lema 4.1.2, x sólo posee valores racionales. Se denominarán *valores fraccionarios* a aquellos valores que no sean enteros. Sea r el número de valores fraccionarios distintos que posee x, donde los valores están dados por $\{\alpha_1, \ldots, \alpha_r\}$ y se denota el vector indicador de cada α_j como $\theta^j \in \{0,1\}^N$ (es decir, $\theta^j_i = 1$ ssi $x_i = \alpha_j$), se puede descomponer x como

$$x = \tilde{x} + \sum_{j=1}^{r} \alpha_j \theta^j$$

donde $\tilde{x}$ es la parte entera de x. Con esta descomposición en mente se prueba el siguiente lema:

Lema 4.1.3. *Sea x un punto extremo de P, y se considera la descomposición de x recién mencionada. Entonces se tiene que los vectores θ^j son linealmente independientes (más aún, son ortogonales) y pertenecen a $\ker(\bar{A})$, donde $\bar{A}$ es como en el Lema 4.1.1.*

Demo :
La ortogonalidad (e independencia lineal) se tiene directo de que los soportes de los θ^j son disjuntos, en efecto, si $\theta_i^j = 1$, entonces $x_i = \alpha_j$ y luego $\theta_i^l = 0 \ \forall l \neq j$.

Ahora se probará que $\bar{A}\theta^j = 0$. Para las restricciones de precedencias, dada una restricción $x_a \leq x_b$, si es activa entonces $x_a = x_b$. Por lo tanto, si $x_a = \alpha_j$ entonces $x_b = \alpha_j$, pero esto implica que $\theta_a^j = 1$ y que $\theta_b^j = 1$ respectivamente, de donde se tiene que $\theta_a^j - \theta_b^j = 0$. Análogamente se tiene el mismo resultado si $x_a \neq \alpha_j$.

Por último, si una restricción $x_i \geq 0$ o $x_i \leq 1$ está activa, entonces como los α_j son fraccionarios se tiene que $\theta_i^j = 0 \ \forall j$. Lo que concluye la demostración.

■

Corolario 4.1.4. *Sea P y x como en el Lema 4.1.1, q el número de side-constraints y r el número de valores fraccionarios distintos de x. Entonces $r \leq q$.*

Demo :
Directo, pues $\dim(\ker(\bar{A})) \leq \bar{q}$, con $\bar{q}$ como en el Lema 4.1.1, y los vectores θ^j son ortogonales y pertenecen a $\ker(\bar{A})$. Como $\bar{q} \leq q$ se tiene el resultado.

■

4.1.2. Particionamiento

El último resultado permite afirmar que el número de valores distintos que toma una solución óptima no es más que el número de side-constraints, y con esto se puede tener una idea de qué tipo de restricciones $H^k x = h^k$ conviene imponer en el algoritmo para que cada problema $P2^k$ sea "fácil" de resolver, pero intente "adivinar" la estructura de la solución óptima. De aquí nace la idea de particionar las variables, de manera de imponer que los conjuntos de variables contenidas en un elemento de la partición posean el mismo valor en la solución. Formalmente, en cada iteración del algoritmo se definirá

$$C^k = \{C_1^k, \ldots, C_l^k\}$$

partición de $\mathcal{B}' = \{0, \ldots, N-1\}$ (con $N = n \cdot R \cdot T$), y cada $H^k x = h^k$ consistirá en las restricciones $x_i = x_j,\ \forall i, j \in C_h^k,\ 1 \leq h \leq l$, de manera de tratar de encontrar la descomposición correcta para la solución de PCP^{by}.

De esta manera, el sistema agregado en $P2^k$ es de la forma $H^k x = 0$ (es decir $h^k = 0$), consistente en las restricciones antes mencionadas. Desde acá en adelante se considerará este sistema.

Al introducir esta idea, el algoritmo BZ queda como:

1. Definir $\mu^0 = 0$, $C_1^0 = \mathcal{B}'$, $C^0 = \{C_1^0\}$ y $k = 1$.
2. Sea w^k la solución de $L(PCP^{by}, \mu^{k-1})$ (problema PCP^{by} donde se penalizan las side-constraints con el multiplicador μ^{k-1}).

   Si $k \geq 2$ e w^k satisface $H^{k-1}x = 0$ PARAR.
3. Se define C^k partición de $\mathcal{B}'$ y se imponen las restricciones $H^k x = 0$ tales que son satisfechas por w^k y consistentes en $x_i = x_j,\ \forall i, j \in C_h^k$.
4. Resolver $P2^k$, obteniendo como solución z^k y con variables duales μ^k correspondientes a las side-constraints. Si $\mu^k = \mu^{k-1}$ PARAR.

La pregunta ahora es, ¿cómo definir las particiones?. Se debe tomar en cuenta que w^k debe satisfacer el sistema $H^k x = 0$, luego para esto Bienstock y Zuckerberg [3] proponen dos ideas:

1. Si se define como
$$I^k = \{j \in \mathcal{B} \mid w_j^k = 1\}$$
$$O^k = \{j \in \mathcal{B} \mid w_j^k = 0\}$$
entonces se puede tomar C^k como todos los conjuntos no-vacíos en la colección
$$\{I^k \cap C_h^{k-1}\} \cup \{O^k \cap C_h^{k-1}\}.$$
2. O tomar la descomposición de z^{k-1} dada por
$$z^{k-1} = \tilde{z}^{k-1} + \sum_{j=1}^{r} \alpha_j \theta^j$$
y luego definir como $\tilde{C}_j^k$ al soporte de θ^j y $\tilde{C}_0^k$ al soporte de $\tilde{z}^{k-1}$. Luego, si se considera $\tilde{C}'^k = \{\tilde{C}_0^k, \ldots, \tilde{C}_r^k\}$ (con $r \leq q$) se puede definir C^k como la colección de los conjuntos no vacíos de la colección
$$\{I^k \cap \tilde{C}_h^k\} \cup \{O^k \cap \tilde{C}_h^k\}$$

En ambas ideas se utiliza I^k y O^k para intersectar los elementos de la partición. A esto se le denominará *refinamiento de la partición*. Es importante mencionar que w^k es un vector entero (pues es la solución de un Max-Closure), por lo que $I^k \cup O^k = \mathcal{B}'$.

En el primer caso, el número de elementos en la partición a lo más se va duplicando en cada iteración (lo que podría ser malo), pero da un poco más de "libertad" en cada problema $P2^k$. La gracia de la segunda idea es que en cada iteración se tienen a lo más $2(q+2)$ elementos en la partición, lo cual es muy bueno cuando se tienen pocas side-constraints, y además se sabe que la solución de PCP^{by} no puede tener más de $q+2$ valores distintos en sus componentes (a lo más q fraccionarios, más 1 o 0). A esta segunda idea se le denominará "Reset", porque de cierta forma se está haciendo un "reset" sobre la partición.

Cabe mencionar que el problema $P2^k$ también puede ser representado como un problema del tipo PCP^{by}, por lo que también son válidos los resultados de descomposición y por ende es válida la segunda idea presentada.

El problema de la idea de Reset es que podría causar que el algoritmo quede atascado en un ciclo infinito, por lo que Bienstock y Zuckerberg sugieren utilizar ambas ideas, i.e, usar la primera idea que asegura que el algoritmo terminará (pues en el peor caso se refina la partición hasta obtener el problema original), y, como paso opcional, si es que la función objetivo aumenta estrictamente o si el problema $P2^k$ comienza a crecer mucho, se puede utilizar la segunda idea para reducir el tamaño de la partición, y así reducir el número de variables en el problema $P2^k$.

4.2. Una primera implementación

Usando la descripción del algoritmo, es directo implementar una versión básica de éste. El algoritmo fue implementado con las siguientes características:

- Como $L(PCP^{by}, \mu^{k-1})$ es un problema Max-Closure, será resuelto usando el algoritmo pseudo-flow de Hochbaum. Ver [21, 22].
- El grafo utilizado para resolver $L(PCP^{by}, \mu^{k-1})$ será guardado completamente en memoria utilizando listas de adyacencia.
- La primera idea de particionamiento será utilizada. Esta idea asegura un algoritmo finito y es muy simple de implementar.
- Dado que las particiones reducen considerablemente el número de variables en $P2^k$ (y por lo tanto, el número de restricciones de precedencia también), cada problema $P2^k$ se asumirá lo suficientemente pequeño como para ser resuelto utilizando CPLEX [34].
- Se añadió otro criterio de parada: alcanzar un Gap predeterminado entre la cota superior entregada por $L(PCP^{by}, \mu^k)$ (pues es una relajación del problema) y la cota inferior dada por $P2^k$.
- El algoritmo será ejecutado en una CPU modelo Intel(R) Xeon(R) con 2.4 GHz y con 24 GB de memoria RAM.

Nombre	Bloques	Prec.	Dest.	Periodos	CPX	BZ-0
mine_n	1060	3922	2	6	8.44	0.19
mine_z_small	9400	145640	2	20	327359.74	26.5
mine_k	14153	219778	2	12	95631.97	27.78
mine_a	16292	74260	2	20	29785.60	26.6
mine_z_medium	29277	1271207	2	15	Mem	190.37
mine_m	53271	656411	2	20	Mem	97.2
mine_m_simple	53668	656411	2	20	Mem	124.04
mine_g	57958	1186665	3	21	Mem	731.62
mine_z_large	96821	1053105	2	30	Mem	1147.21
mine_s	99014	96642	2	30	5700.51	519.83
mine_mc_limit	112687	3035483	2	15	Mem	440.81
mine_d	167937	5956121	2	51	Mem	5392.3
mine_r	304977	15665274	2	81	Mem	Mem
mine_t	326428	1518326	2	80	Mem	Mem
mine_t_full	329859	7066518	2	60	Mem	Mem
mine_e	1101098	65185607	2	50	Mem	Mem
mine_l	1663499	92440037	3	30	Mem	Mem
mine_mc	2140342	3035483	2	20	Mem	Mem

Cuadro 4.1: Tiempos (en segundos) de ejecución de CPLEX y del algoritmo BZ para resolver cada instancia. Las primeras 5 columnas consisten en los datos de las instancias (nombre, número de bloques, número de precedencias entre bloques, número de destinos y número de periodos respectivamente). La columna CPX muestra el tiempo que ocupa CPLEX en resolver el problema, y la columna BZ-0 el tiempo que requiere la primera implementación del algoritmo BZ. *Mem* indica que no se pudo resolver la instancia por problemas de memoria.

Con estas consideraciones se obtuvieron muy buenos resultados en comparación con CPLEX, pero todavía existen varias instancias que no pueden ser resueltas debido a problemas de memoria. El Cuadro 4.1 muestra el tiempo necesario para resolver las instancias disponibles con el algoritmo BZ utilizando un Gap objetivo de 10^{-6}. Además se agregó el tiempo necesitado por CPLEX para resolver las mismas instancias. En esta tabla se puede observar que el algoritmo es capaz de resolver varias instancias en poco tiempo (con un par de excepciones por supuesto), pero como se verá más adelante, una serie de speed-ups pueden ser utilizados para obtener un mejor rendimiento del algoritmo, y también se mostrarán maneras de lidiar con el problema de memoria.

4.3. Speed-Ups

Al implementar el algoritmo BZ para el problema de planificación minera se observó que en la mayoría de las instancias la parte que utiliza tiempo y memoria es resolver $L(PCP^{by}, \mu^{k-1})$, por lo que se intentó hacer algo mejor en esta parte del algoritmo para acelerar el rendimiento general de éste, por lo tanto, muchos speed-ups están diseñados

para este paso. Sin embargo, también se desarrollaron otros speed-ups para otras partes del algoritmo.

Un hecho importante que será utilizado es el siguiente:

Proposición 4.3.1. *Dados multiplicadores μ, si se llama $L(PCP^{by}, \mu)^{at}$ a la formulación* at *del problema $L(PCP^{by}, \mu)$ entonces se tiene*

$$L(PCP^{by}, \mu)^{at} = L(PCP^{at}, \mu)$$

donde $L(PCP^{at}, \mu)$ es el problema obtenido al penalizar las side-constraints de PCP^{at} con los multiplicadores μ

Demo :
La demostración es directa y sigue de la demostración del Teorema 3.3.1.

■

Esta proposición será muy útil pues algunos speed-ups son más intuitivos en $L(PCP^{at}, \mu^{k-1})$ (dado que acá las variables indican exactamente cuándo extraer un bloque y a qué destino enviarlo), por lo que se presentarán en este contexto, y en la implementación simplemente se adaptaron a $L(PCP^{by}, \mu^{k-1})$ usado la transformación *at-by* presentada en el Teorema 3.3.1.

4.3.1. Representación del Grafo

Primero, una mejora muy importante diseñada para este algoritmo fue la representación implícita del grafo que representa al problema $L(PCP^{by}, \mu^{k-1})$.

Como se mencionó anteriormente, este problema es un Max-Closure que es resuelto usando el algoritmo pseudo-flow de Hochbaum [21, 22]. En este caso se está buscando un Max-Closure en un grafo definido por las restricciones de precedencias obtenidas en la transformación utilizada en el Teorema 3.3.1. Por lo tanto se tienen 3 tipos de restricciones de precedencia en $L(PCP^{by}, \mu^{k-1})$:

$$x_{b,R-1,t} \leq x_{b,0,t+1} \quad t < T-1. \tag{4.1}$$
$$x_{b,d,t} \leq x_{b,d+1,t} \quad d < R-1. \tag{4.2}$$
$$x_{a,R-1,t} \leq x_{b,R-1,t} \quad \forall (a,b) \in \mathcal{A}. \tag{4.3}$$

De esta manera, el grafo dirigido G' usado para encontrar el Max-Closure de acuerdo a $L(PCP^{by}, \mu^{k-1})$ queda definido por:

- Un nodo por cada variable. En este caso, por cada tupla (b, d, t).
- Un arco por cada restricción de precedencia definida por (4.1), (4.2) y (4.3).

Y anteriormente se definió como G al grafo que contiene las precedencias entre bloques. Con estos dos grafos en mente se puede pensar en dos opciones para trabajar con G':

- Trabajar con el grafo completo, como se hizo en la primera implementación. En otras palabras, guardar todos los arcos del grafo G' en memoria (usando cualquier representación), por lo tanto, cuando se necesite encontrar los vecinos de un nodo se pueden obtener directamente de la representación usada. A ésta se le denominó "Representación explícita".
- Sólo guardar el grafo G en memoria. Y cuando se necesiten los vecinos de un determinado nodo de G', definido por (b, d, t), se pueden determinar estos vecinos fácilmente:
  - Si a es vecino de b en G, y $d = R - 1$, entonces de acuerdo a (4.3) (a, d, t) es vecino de (b, d, t) en G'.
  - Si $d < R - 1$, entonces de acuerdo a (4.2) $(b, d + 1, t)$ es vecino de (b, d, t) en G'.
  - Y finalmente, si $t < T - 1$ y $d = R - 1$, entonces de acuerdo a (4.1) $(b, 0, t + 1)$ es vecino de (b, d, t) en G'.

  Se le denominó a ésta "Representación implícita", abreviado IMP, y se puede ver fácilmente que utiliza menos memoria que la Explícita, pero no tiene acceso directo a los vecinos de un nodo, lo que podría tener como consecuencia un algoritmo más lento.

4.3.2. Reducciones en el Sub-problema

Como se mencionó en la Proposición 4.3.1, $L(PCP^{by}, \mu^{k-1})$ es un problema en formato *by*, pero sin side-constraints, por lo que tiene una formulación equivalente en formato *at*.

Otro hecho importante es que las side-constraints son generalmente restricciones de capacidad, más aún, cada side-constraint generalmente afecta sólo a un periodo de tiempo (por ejemplo, capacidad de producción por cada año), por lo que se considerará este caso en el siguiente speed-up (que es el caso de todas las instancias que se utilizarán). Usando esto, se pueden re-escribir las side-constraints de PCP^{at} como:

$$\hat{D}_t y_{\cdot,\cdot,t} \leq d_t \quad t = 0, \ldots, T-1$$

Donde $y_{\cdot,\cdot,t}$ representa las componentes de y que corresponden al periodo t. Y como la componente "temporal" de atbyKey(b, d, t) (definido en el Teorema 3.3.1) es siempre menor o igual a t, reemplazando las variables *by* x en las side-constraints se obtiene una estructura similar a la anterior:

$$D_t x_{\cdot,\cdot,t} \leq d_t \quad t = 0, \ldots, T-1.$$

Pero ahora, abusando de notación, $x_{\cdot,\cdot,t}$ representa componentes de x que correspondan a un periodo menor o igual a t.

Además, usando el mismo razonamiento, se puede descomponer μ^{k-1} como

$$\mu^{k-1} = (\mu^{k-1,0}, \ldots, \mu^{k-1,T-1})$$

donde cada $\mu^{k-1,t}$ son las variables duales de las side-constraints que corresponden al periodo t.

Proposición 4.3.2. *Sea $\tau \geq 0$. Si $\mu^{k-1,t} = 0 \ \forall t \geq \tau$ entonces existe una solución óptima y de $L(PCP^{at}, \mu^{k-1})$ tal que $\forall b, d$*

$$y_{b,d,t} = 0 \quad \forall t \geq \tau + 1$$

Demo :
Sea y una solución óptima. Como $\mu^{k-1,t} = 0 \ \forall t \geq \tau$, entonces los coeficientes en la función objetivo para $y_{b,d,t}$ con $t \geq \tau$ están dados por:

$$\alpha^t \tilde{c}_{b,d}$$

Ahora, por contradicción se supone la afirmación no es verdad y se llama P_t al conjunto de bloques extraídos en el periodo t, i.e

$$P_t = \{b \in \mathcal{B} \mid y_{b,d,t} > 0 \text{ para algún } d\}$$

y se denota el beneficio total de P_t como

$$\hat{c}(P_t) = \sum_{b \in P_t} \sum_{d=0}^{R-1} \hat{c}_{b,d,t} y_{b,d,t}$$

Sea $t^* \geq \tau + 1$ tal que $P_{t^*} \neq \phi$ (el cual se está suponiendo que existe). Claramente se tiene que

$$\hat{c}(P_{t^*}) = \sum_{b \in P_{t^*}} \sum_{d=0}^{R-1} \alpha^{t^*} \tilde{c}_{b,d} y_{b,d,t^*}$$

- Si $\hat{c}(P_{t^*}) < 0$ se puede extraer cada bloque de P_{t^*} un periodo después, si $t^* < T - 1$, y todavía satisfacer las restricciones de precedencia, pero incrementando la función objetivo estrictamente (debido a la tasa de descuento). Si $t^* = T - 1$ simplemente se puede no extraer este conjunto de bloques.
- Si $\hat{c}(P_{t^*}) \geq 0$ se puede extraer cada bloque de P_{t^*} un periodo antes y todavía satisfacer las restricciones de precedencia, incrementando la función objetivo (debido a la tasa de descuento y a que $t^* \geq \tau + 1$). Este incremento no necesariamente es estricto.

En el primer caso hay una contradicción con la optimalidad de y, y en el segundo caso se está construyendo otra solución óptima y^* tal que $y^*_{b,d,t} = 0 \quad \forall t \geq \tau + 1$, que es lo que se quería.

■

Lo que esta proposición está diciendo es que, si estas condiciones se cumplen, se puede resolver un problema más pequeño, esto es, sólo considerar los primeros $\tau + 1$ periodos en $L(PCP^{at}, \mu^{k-1})$. Este "problema reducido" es un problema en formato *at*, por lo que se puede resolver usando su formulación *by* (para resolverlo como Max-Closure) y luego reconstruir la solución completa fácilmente.

Corolario 4.3.3. *La solución óptima x de $L(PCP^{by}, \mu^{k-1})$ correspondiente a la solución óptima y de $L(PCP^{at}, \mu^{k-1})$, donde y es la discutida en la Proposición 4.3.2 satisface*

$$x_{b,d,t} = x_{atbyKey(b,d,t)} \quad \forall t \geq \tau + 1$$

Demo :
Como $y_{b,d,t} = 0 \ \forall t \geq \tau + 1$ y $y_{b,d,t} = x_{b,d,t} - x_{\text{atbyKey}(b,d,t)}$ implica que $x_{b,d,t} = x_{\text{atbyKey}(b,d,t)}$ si $t \geq \tau + 1$ (en este caso $t > 0$, por lo que atbyKey(b, d, t) está bien definido).

■

Ahora se probará que estas condiciones se cumplen en las primeras T iteraciones del algoritmo BZ, por lo que al menos se tiene un speed-up para esas iteraciones.

Proposición 4.3.4. *Se supondrá que $d_t \neq 0$. Si siempre se utiliza la solución del subproblema discutida en el Corolario 4.3.3, entonces en la k-ésima iteración del algoritmo BZ se tiene $\mu^{k-1,t} = 0 \ \forall t \geq k - 1$. Más aún, si $t \geq k$ entonces (b, d, t) y atbyKey(b, d, t) pertenecen al mismo elemento de la partición obtenida al refinar con x^k (recordar que como $t > 0$ en este caso, entonces atbyKey(b, d, t) existe).*

Demo :
Se utilizará inducción.

Para $k = 1$, por definición $\mu^0 = 0$, y por el Corolario 4.3.3 se puede escoger x^0 tal que

$$x^0_{b,d,t} = x^0_{\text{atbyKey}(b,d,t)} \quad \text{si } t \geq 1$$

por lo tanto la primera partición (que está completamente definida por x^0) satisface lo que se necesita.

Ahora se supone que la condición se tiene para $k > 1$ y se probará para $k + 1$. En la k-ésima iteración se tiene que (b, d, t) y atbyKey(b, d, t) pertenece al mismo elemento de la partición actual si $t \geq k$, entonces para cualquier vector z^k factible para $P2^k$ se tiene que

$$z^k_{b,d,t} = z^k_{\text{atbyKey}(b,d,t)} \quad \text{si } t \geq k$$

entonces, por la igualdad $y_{b,d,t} = x_{b,d,t} - x_{\text{atbyKey}(b,d,t)}$, se tiene que las side-constraints satisfacen

$$D_t z^k_{\cdot,\cdot,t} = 0 \quad \forall t \geq k$$

y como se está asumiendo que $d_t \neq 0$, por holgura complementaria, los duales óptimos μ^k deben satisfacer

$$\mu^{k,t} = 0 \quad \forall t \geq k$$

por lo tanto, en la iteración $k + 1$-ésima se tendrá que $\mu^{k,t} = 0 \quad \forall t \geq k$ que es lo que se quiere.

Ahora, usando el Corolario 4.3.3 nuevamente, se escoge x^{k+1} óptimo para $L(PCP^{by}, \mu^k)$ tal que

$$x^{k+1}_{b,d,t} = x^{k+1}_{\text{atbyKey}(b,d,t)} \quad \forall t \geq k + 1$$

y por hipótesis de inducción, (b, d, t) y atbyKey(b, d, t) pertenecen al mismo elemento en la partición previa para $t \geq k$, por lo que al refinar usando x^{k+1} se tiene que, al menos, (b, d, t) y atbyKey(b, d, t) pertenecerán al mismo elemento de la nueva partición para $t \geq k + 1$. Que es la segunda condición que se necesitaba.

■

Con este resultado se puede asegurar que en cada iteración los duales se van haciendo no nulos a lo más para un periodo extra, en otras palabras, los únicos duales no nulos obtenidos en $P2^k$ pueden ser $\mu^{k,t}$ con $t \leq k$, bajo la condición que $d_t \neq 0$ que es típicamente el caso. Por lo tanto, en las primeras T iteraciones se puede resolver el sub-problema considerando k periodos, lo que reduce el tiempo y la memoria utilizada para resolver $L(PCP^{by}, \mu^{k-1})$. A este speed-up se le denominó Time Shrink, abreviado TMSH.

Otra reducción importante es que se puede convertir cada sub-problema $L(PCP^{at}, \mu^{k-1})$ en un problema de 1 destino, lo que reducirá considerablemente el tamaño del problema haciendo más fácil de resolver el problema $L(PCP^{by}, \mu^{k-1})$.

En efecto, este resultado es fácil de ver y no necesita ninguna hipótesis como el speed-up anterior. Cada problema $L(PCP^{at}, \mu^{k-1})$ es simplemente el mismo problema de planificación minera, pero sin side-constraints. De esta manera, si un bloque es extraído, éste será enviado a su mejor destino (el destino con mayor valor objetivo después de penalizar los coeficientes), por lo tanto, para cada bloque basta considerar dicho destino, y de esta manera se reducirá el número de variables en un factor $1/D$. Para usar este speed-up se debe "elegir el mejor destino" en $L(PCP^{at}, \mu^{k-1})$ para reducir el problema, y después encontrar la formulación *by* del problema reducido para resolverlo como un Max-Closure. A este speed-up se le llamó One Destination Shrink, abreviado 1DES.

4.3.3. Warm Starts Iterativos

El problema $L(PCP^{by}, \mu^{k-1})$ es un problema Max-Closure, y como ya se mencionó antes, se resuelve usando el algoritmo pseudo-flow de Hochbaum. Y como se está resolviendo un Max-Closure en cada iteración del algoritmo BZ, podría ser una buena idea hacer warm start sobre el algoritmo pseudo-flow usando la solución anterior.

Es fácil ver que en las distintas iteraciones el grafo construido no cambia su estructura, sólo las capacidades de sus arcos. Pero al cambiar las capacidades el pseudo-flow anterior podría no ser ni siquiera factible. Por suerte, Hochbaum describe un algoritmo que arregla este problema, permitiendo empezar el algoritmo usando la solución previa. Ver [22] para los detalles.

El único problema es que si también se usa el speed-up Time Shrink se está resolviendo un problema de k periodos en la iteración k-ésima (para $k \leq T$), y en la siguiente iteración se resolverá un problema con $k + 1$ periodos, por lo que la información previa no puede ser utilizada directamente. Por esto, si Time Shrink se encuentra activado, entonces este warm start sólo será utilizado después de la iteración T. A este speed-up se le llamó Pseudo-Flow Warm Start, abreviado PSWS.

Observación 4.3.5. Se puede probar que incluso si se usa Time Shrink en las primeras T iteraciones, se puede construir un warm start cuidadosamente, de esta manera extendiendo este speed-up. Pero debido a lo engorroso de la implementación (y la demostración) se decidió no incluir esta extensión.

Por otra parte, se está resolviendo cada problema $P2^k$ utilizando el algoritmo Simplex implementado en la librería de CPLEX. Por lo tanto, se hará warm start de Simplex utilizando la solución anterior como punto de partida en cada iteración. Esto resultó ser muy útil, especialmente cuando el problema master crece mucho. A éste se le llamó Simplex Warm Start, abreviado SPWS.

4.3.4. Preproceso y Warm Start Globales

En la práctica, para diseñar la planificación de excavación de una mina, se llevan a cabo una serie de pasos para obtener un calendario tentativo. En uno de estos pasos, llamado *Delineación de Contorno del Pit Final*, se delimita la mina, identificando en qué parte de la mina completa tendrá lugar la excavación (ver [7]). Esta delimitación es conocida como "Ultimate Pit Limit".

Se define el problema $UPIT$ como:

$$\begin{array}{lllll} (UPIT) & \text{máx} & c'x & & \\ & & x_a & \leq & x_b \quad \forall (a,b) \in \mathcal{A} \\ & & x_b & \in & [0,1] \quad \forall b \in \mathcal{B} \end{array}$$

Donde c es un vector con el beneficio del *mejor destino* de cada bloque. Éste es un problema de un periodo. Claramente es equivalente a resolver el problema PCP^{by} removiendo todas las side-constraints. Esto pues debido a la tasa de descuento es siempre mejor extraer todo en el primer periodo, y al no considerar ninguna side-constraint no tiene sentido considerar más de 1 destino por bloque.

Se puede probar que, si los coeficientes de la matriz $\hat{D}$ son no negativos, entonces la solución entregada por $UPIT$ contiene el conjunto de los bloques que deberían ser incluidos en la solución óptima del problema original (ver [6]). Y éste es típicamente el caso: en la mayoría de las instancias que se considerarán las side-constraints son capacidades (de procesamiento y extracción), por lo que la matriz $\hat{D}$ posee "pesos" de cada bloque.

Por este motivo, y dado que $UPIT$ es un problema Max-Closure, una buena idea sería utilizar este problema (que, como ya se ha mencionado, puede ser resuelto rápidamente) para pre-procesar la mina antes de utilizar el algoritmo BZ. En otras palabras, si un bloque no es extraído en $UPIT$, entonces no será extraído en PCP^{at} (ni siquiera parcialmente, dado que se está trabajando en la relajación), por lo que se puede eliminar y así reducir el número de variables. A este pre-proceso se le denominó UPIT.

Y como warm start global, se decidió utilizar el algoritmo Critical Multiplier [7] (abreviado CMA). Este es un algoritmo muy rápido, diseñado para resolver el mismo problema pero bajo 2 condiciones: sólo 1 destino por cada bloque, y sólo una side-constraint por periodo tal que su lado izquierdo (en la formulación at) sea no-negativo. Por estos motivos no

puede ser utilizado directamente en un contexto general, pero ya que tiene un rendimiento muy bueno (incluso mucho mejor que BZ en instancias que son válidas para ambos algoritmos) se consideró como una buena idea usarlo para hacer warm start en el algoritmo BZ.

Lo que se hizo fue utilizar la siguiente heurística para empezar con una partición no trivial en el algoritmo BZ, en el caso que $\hat{D}$ sólo tenga coeficientes no negativos:

1. Elegir el mejor destino para cada bloque. De esta manera se obtiene un problema con un destino.

2. Por cada side-constraint por separado, usar el algoritmo CMA. Es decir, relajar todas las side-constraints, excepto una, y resolver el problema. Hacer esto por cada side-constraint.

3. Por cada solución entregada por CMA, obtener la partición inducida por esta solución. Esto eso, particionar el espacio de variables completo de acuerdo a los valores obtenidos para cada variable en cada solución. De esta manera se obtienen q particiones (donde q es el número de side-constraints).

4. Obtener un refinamiento de las q particiones intersectando todos los elementos de todas las particiones.

5. Comenzar el algoritmo BZ con esta partición.

A este warm start se le llamó "CMA Warm Start" abreviado simplemente CMA. Es importante mencionar que dado que este último warm start cambia la partición inicial, no puede ser utilizado junto con el speed-up Time Shrink, por lo que los experimentos deberán considerar este problema. Dado que son excluyentes, en este caso sería ideal determinar cuál es mejor empíricamente.

Capítulo 5

Experimentos Computacionales y Análisis

En el siguiente capítulo se mostrarán los resultados computacionales obtenidos con la implementación del algoritmo BZ junto con los speed-ups discutidos en el capítulo anterior.

5.1. Análisis de Sensibilidad

Como primer paso, se mostrará un análisis hecho para estimar el aporte de cada speed-up por separado. Para esto, se considerará la implementación básica del algoritmo, la cual será comparada con el mismo algoritmo con uno de los speed-ups activados. Esto dará una noción del efecto de cada speed-up por si solo, argumentando así si es que vale la pena considerarlo en general.

Por cada speed-up los resultados obtenidos se detallan en el Cuadro 5.1, donde se incluye una tabla con todos los tiempos obtenidos para resolver las distintas instancias del problema de planificación minera. En general, los resultados indican lo siguiente:

- **Representación del Grafo:** A pesar de que la representación explícita del grafo tiene como consecuencia un algoritmo más rápido (debido al rápido acceso a los vecinos de un nodo), el usar la representación implícita no tiene grandes consecuencias negativas en el tiempo que toma el algoritmo. En general se obtiene un algoritmo levemente más lento (en promedio, un 20 % más lento) pero permite resolver más instancias, lo que compensa la ralentización del algoritmo. Por esto se determinó que, si bien no es un speed-up propiamente tal, debe ser utilizado al momento de usar el algoritmo BZ.
- **Time Shrink:** Como era de esperar, se pudo observar un efecto positivo al utilizar esta mejora. A pesar de que este speed-up sólo se pueda utilizar en las primeras T iteraciones se observó un aumento en la velocidad del algoritmo en todas las instancias, obteniendo una reducción de un 30 % en promedio en los tiempos del algoritmo.

Nombre	IMP	$\frac{\text{IMP}}{\text{BZ-0}}$	TMSH	$\frac{\text{TMSH}}{\text{BZ-0}}$	1DES	$\frac{\text{1DES}}{\text{BZ-0}}$	PSWS	$\frac{\text{PSWS}}{\text{BZ-0}}$	SPWS	$\frac{\text{SPWS}}{\text{BZ-0}}$	UPIT	$\frac{\text{UPIT}}{\text{BZ-0}}$	CMA	$\frac{\text{CMA}}{\text{BZ-0}}$
mine_n	0.19	1.00	0.16	0.84	0.14	0.74	0.15	0.79	0.19	1.00	0.2	1.05	0.12	0.63
mine_z_small	33.44	1.26	20.16	0.76	16.64	0.63	20.49	0.77	28.04	1.06	24.72	0.93	17.38	0.66
mine_k	31	1.12	19.46	0.70	17.84	0.64	22.57	0.81	30.84	1.11	26.64	0.96	15.61	0.56
mine_a	24.38	0.92	17.48	0.66	16.39	0.62	19.25	0.72	26.34	0.99	14.95	0.56	2.21	0.08
mine_z_medium	297.34	1.56	144.94	0.76	141.99	0.75	121	0.64	204.14	1.07	191.38	1.01	99.08	0.52
mine_m	92.84	0.96	61.46	0.63	69.18	0.71	70.2	0.72	97.52	1.00	37.85	0.39	47.66	0.49
mine_m_simple	119.33	0.96	91.63	0.74	85.26	0.69	91.44	0.74	130.77	1.05	54.67	0.44	120.37	0.97
mine_g	986.37	1.35	337.23	0.46	369.13	0.50	644.93	0.88	701.92	0.96	674.5	0.92	305.72	0.42
mine_z_large	1334.52	1.16	689.51	0.60	772.57	0.67	837.27	0.73	1157.46	1.01	1140.59	0.99	726.52	0.63
mine_s	529.64	1.02	460.4	0.89	487.8	0.94	574.57	1.11	215.75	0.42	329.17	0.63	127.63	0.25
mine_mc_limit	535	1.21	307.36	0.70	312.45	0.71	386	0.88	433.44	0.98	444.95	1.01	289.3	0.66
mine_d	10033.77	1.86	1340.45	0.25	3322.37	0.62	4071.67	0.76	5554.02	1.03	5229.3	0.97	3420.87	0.63
mine_r	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-
mine_t	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-	N.A	-	N.A	-
mine_t_full	118284.76	-	Mem	-	Mem	-	Mem	-	Mem	-	N.A	-	N.A	-
mine_e	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-
mine_l	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-	Mem	-
mine_mc	4170.59	-	Mem	-	Mem	-	Mem	-	Mem	-	1325.27	-	Mem	-

Cuadro 5.1: Análisis de Sensibilidad para todos los speed-ups diseñados. Todas las columnas pares poseen el tiempo de ejecución (en segundos) del algoritmo BZ con sólo el speed-up activado correspondiente al título de la columna, y desde la tercera columna, todas las columnas impares poseen la razón entre el tiempo de ejecución con un speed-up activado y el tiempo de ejecución del algoritmo BZ sin ningún speed-up, de acuerdo a la columna etiquetada como BZ-0 en el Cuadro 4.1. *N.A* indica que no se pudo aplicar el speed-up a la instancia en cuestión.

- **One Destination Shrink:** Este simple speed-up también tuvo un efecto positivo en los tiempos de ejecución del algoritmo sobre todas las instancias. En promedio se obtuvo una reducción de un 30 % aproximadamente en estos tiempos.
- **Pseudo-Flow Warm Start:** Si bien este warm start no tuvo un efecto positivo en todas las instancias, sí lo tuvo en la mayoría; en efecto, sólo una instancia se vio perjudicada por este warm start, y no fue un aumento drástico en el tiempo de ejecución. En general se observó un efecto positivo, con una disminución de un 20 % aproximadamente del tiempo de ejecución en promedio.
- **Simplex Warm Start:** Este warm start aparentemente no fue de tanta ayuda como los otros. En general no se tuvo un cambio notorio en los tiempos de ejecución (menos de un 10 % de variación en casi todos los tiempos), muchas veces incluso obteniendo un peor rendimiento del algoritmo. A pesar de esto, y dado que la variación no fue considerablemente alta, se cree que vale la pena usar este warm start en el algoritmo BZ. Además, en una instancia especial y de mayor tamaño que las que se presentan en este informe (que por razones de confidencialidad no se pudo incluir) este warm start mostró tener una gran importancia.
- **UPIT:** Este preproceso también mostró ser de gran utilidad, no sólo logrando reducir el tiempo de ejecución en un 20 % en promedio, sino que también permitiendo resolver una instancia extra (debido a que con este preproceso se reduce el tamaño del problema). En algunos casos el algoritmo se vio levemente perjudicado, y en la mayoría de estos casos se debe a una razón muy simple: dado que es un preproceso común en la minería, muchas instancias que se encuentran disponibles ya han sido preprocesadas de esta manera, por lo que se está perdiendo tiempo y no se reduce el tamaño del problema. Sin embargo, en los pocos casos donde el algoritmo empeoró, el aumento en el tiempo de ejecución fue menor.
- **CMA Warm Start:** Y por último, la heurística descrita para comenzar el algoritmo BZ con una partición no trivial también tuvo un resultado positivo en todas las instancias donde se puede aplicar. En promedio se obtuvo la mayor reducción del tiempo de ejecución entre todos los speed-ups, siendo ésta de un 45 % aproximadamente.

Observación 5.1.1. Se debe recordar que CMA Warm Start requiere que las side-constraints $\hat{D}y \leq d$ tengan lado izquierdo no negativo, al igual que UPIT, por lo que no pudo ser aplicado en todas las instancias.

Observación 5.1.2. Es tentador determinar, a partir de este análisis, que CMA Warm Start es un mejor speed-up que Time Shrink en promedio, por lo que se debería elegir el primero sobre el segundo (pues ambos son excluyentes). Pero se debe tomar en cuenta que la idea principal es combinar todos estos speed-ups y ver cómo se complementan entre ellos, y recién ahí se podrá tener una noción de cual podría ser mejor que el otro.

5.2. Efecto de todos las mejoras

A continuación se mostrarán los experimentos realizados usando todos los speed-ups disponibles. Dado que Time Shrink y CMA Warm Start son excluyentes, se consideraron

Nombre	ALL-C	$\frac{\text{ALL-C}}{\text{BZ-0}}$	ALL-T	$\frac{\text{ALL-T}}{\text{BZ-0}}$
mine_n	0.06	0.32	0.1	0.53
mine_z_small	10.54	0.40	13.05	0.49
mine_k	8.13	0.29	10.89	0.39
mine_a	0.85	0.03	4.57	0.17
mine_z_medium	86.87	0.46	101.17	0.53
mine_m	11.31	0.12	14.21	0.15
mine_m_simple	26.63	0.21	23.2	0.19
mine_g	281.66	0.38	244.86	0.33
mine_z_large	534.48	0.47	472.33	0.41
mine_s	17.93	0.03	28.5	0.05
mine_mc_limit	207.17	0.47	245.61	0.56
mine_d	2830.23	0.52	1566.93	0.29
mine_r	66136.93	-	Mem	-
mine_t	N.A	-	Mem	-
mine_t_full	N.A	-	12918.34	-
mine_e	24232.73	-	6609	-
mine_l	Mem	-	Mem	-
mine_mc	790.5	-	662.32	-

Cuadro 5.2: Resultados obtenidos para la ejecución del algoritmo BZ con todos los speed-ups incluidos. Las columnas 2 y 4 muestran los tiempos (en segundos) requeridos para cada instancia usando los conjuntos de speed-ups ALL-C y ALL-T respectivamente. La columna 3 muestra el cuociente entre los tiempos obtenidos considerando ALL-C y los tiempos obtenidos en la primera implementación del algoritmo BZ (BZ-0) detallados en el Cuadro 4.1. Lo mismo para la columna 5, considerando ALL-T. *N.A* indica que no se pudo aplicar ALL-C debido a que las instancias no cumplen las hipótesis requeridas.

2 conjuntos de speed-ups:

- Todos los speed-ups, a excepción de CMA Warm Start, que será denotado por **ALL-T** debido a que incluye a Time Shrink.
- Todos, a excepción de Time Shrink, que será denotado **ALL-C** ya que incluye a CMA Warm Start.

Considerando estos conjuntos, se ejecutó el algoritmo sobre todas las instancias. Los resultados se muestran en el Cuadro 5.2 donde se detallan los tiempos obtenidos para cada instancia usando los conjuntos ALL-T y ALL-C. También se agregaron columnas que comparan estos tiempos con los obtenidos para la implementación básica del algoritmo BZ (de acuerdo al Cuadro 4.1).

Para el conjunto ALL-C se aprecia una reducción drástica del tiempo de ejecución del algoritmo en comparación a este mismo sin ningún speed-up. En promedio se redujo el tiempo de ejecución en un 68 %, reduciendo en más de 50 % este tiempo en la mayoría de las instancias. Además, con este conjunto se pudo resolver una instancia que antes no se

podía resolver, y que tampoco se pudo resolver con ALL-T. El problema que posee ALL-C es que no puede ser utilizado en todas las instancias, lo que perjudica su utilidad.

Para el conjunto ALL-T también se puede observar una importante reducción en el tiempo de ejecución del algoritmo BZ. En promedio se obtuvo una reducción de un 65 %, y también se redujo en más de un 50 % el tiempo de ejecución de la mayoría de las instancias. Otro hecho positivo es que con ALL-T se pudo resolver una instancia que con ALL-C no se pudo resolver debido a que no se podía aplicar CMA Warm Start.

Al observar los resultados, no es claro cuál es mejor: ALL-C o ALL-T. Se observa una leve tendencia que insinúa que ALL-T es mejor para instancias más grandes (en cantidad de bloques), pero no existe una superioridad absoluta de un conjunto de speed-ups sobre el otro. Más aún, ambos poseen instancias que el otro no puede resolver.

5.3. Un último speed-up

A pesar de haber reducido drásticamente los tiempos obtenidos para los distintos conjuntos de speed-ups, todavía existen instancias que no se pueden resolver debido a problemas de memoria.

Como se mencionó antes, en general el "cuello de botella" del algoritmo estaba en resolver cada problema $L(PCP^{by}, \mu^k)$, pero usando los speed-ups discutidos se obtuvo una buena reducción en el tiempo necesario para resolver este paso del algoritmo, además de reducir la memoria necesaria para resolver dicho problema.

Pero como se comentó brevemente en la Sección 4.2, se usó la primera idea de particionamiento, y en algunas minas (generalmente minas muy grandes) el tamaño de la partición crece lo suficiente como para hacer que cada problema $P2^k$ sea imposible de resolver usando CPLEX. Por este motivo se decidió revisar la segunda idea de particionamiento explicada en la Sección 4.1.2, es decir, lo que se denominó Reset y que tiene la ventaja de controlar el tamaño de la partición.

El problema de Reset es que si simplemente se usa cada vez que la función objetivo aumenta estrictamente, entonces el algoritmo tiene un muy mal rendimiento. Por esto se decidió usar un umbral para el tiempo que toma resolver el problema master, de manera que si se supera ese umbral se aplica Reset. Por supuesto que esta idea hará que el algoritmo haga más iteraciones, pero se puede esperar que en algunos casos las iteraciones tomen menos tiempo, o al menos que permita resolver instancias que antes no se podían resolver.

El Cuadro 5.3 muestra los tiempos obtenidos al aplicar Reset con un umbral de 100 segundos, es decir, si el tiempo para resolver el problema master toma más de 100 segundos, se aplica Reset. Sólo se agregaron las instancias donde este umbral se alcanzó y se tuvo que hacer Reset, el resto de las instancias no se vieron afectadas por este cambio. Este umbral es arbitrario y se fijó basado en distintos experimentos. En general otros umbrales razonables tienen un rendimiento similar. Además, se decidió mostrar los resultados en

Nombre	ALL-C	RST-C	$\frac{\text{ALL-C}}{\text{RST-C}}$	ALL-T	RST-T	$\frac{\text{ALL-T}}{\text{RST-T}}$
mine_r	66136.93	101649.39	1.54	Mem	107198.05	-
mine_t	N.A	N.A	-	Mem	13412.17	-
mine_t_full	N.A	N.A	-	12918.34	6599.01	0.51

Cuadro 5.3: Tiempo (en segundos) que toma el algoritmo BZ en resolver las instancias indicadas. Las columnas ALL-T y ALL-C poseen los tiempos (en caso que se pudo resolver la instancia) requeridos usando cada conjunto. La columna RST-C posee los tiempos necesitados para resolver las instancias usando Reset con umbral de 100 segundos junto con ALL-C, y lo análogo para la columna RST-T.

comparación con ALL-T y ALL-C, pues, aunque 2 de las instancias que se vieron afectadas no cumplen los requerimientos para ALL-C, la otra instancia solo se pudo resolver con este último.

En el Cuadro 5.3 se observan 3 cosas importantes. Primero, se pudo resolver una instancia extra, que antes no se podía resolver por problemas de memoria. Segundo, la instancia que sólo se podía resolver con ALL-C, ahora se pudo resolver con ALL-T usando Reset. Y tercero, la instancia *mine_t_full* redujo su tiempo de ejecución.

Lamentablemente la reducción de tiempo observada para la instancia antes mencionada es un comportamiento inusual, pues en general Reset aplicado a instancias que se pueden resolver sin problemas (al probar otros umbrales) no reduce el tiempo de ejecución del algoritmo. Además, se puede observar que en la instancia *mine_r* tampoco se redujo el tiempo de ejecución al considerar el conjunto ALL-C. Sin embargo, esto no le resta importancia, dado que el principal aporte de Reset es reducir el tamaño del problema master para poder resolver instancias más complicadas, y ese objetivo se cumplió.

Capítulo 6

Experimentos con Heurísticas

Como originalmente se trata de buscar soluciones factibles para el problema (no de la relajación lineal), sería ideal encontrar la manera de construir soluciones factibles cercanas al óptimo de la relajación. La idea no es sólo utilizar la relajación lineal para obtener una cota superior, sino que usarla como guía para construir una buena solución factible.

Si bien no fue el foco principal del presente trabajo, se realizaron diversos experimentos con algunas heurísticas simples y/o conocidas para lograr obtener soluciones factibles, obteniendo resultados bastante buenos.

6.1. Adaptación de TopoSort

En la siguiente sección se detallará la heurística principal utilizada para obtener soluciones factibles al problema de planificación minera. Para esto se necesita la siguiente definición usada en [7]:

Definición 6.1.1. Dado un vector y factible para PCP^{at}, se define la función $ET : \mathcal{B} \times [0, D-1] \times [0, T-1] \to \mathbb{R}^{\mathcal{B}}$ (Expected Time) como:

$$(ET(y))_b = \sum_{d=0}^{R-1} \sum_{t=0}^{T-1} t \cdot y_{b,d,t} + T \cdot \left(1 - \sum_{d=0}^{R-1} \sum_{t=0}^{T-1} y_{b,d,t}\right)$$

La idea de esta definición es interpretar cada coordenada $y_{b,d,t}$ como una "probabilidad" de extraer el bloque b en el destino d y en el periodo t, agregando un periodo extra (T) de manera que las probabilidades sumen 1. De esta forma la función ET se interpretaría como el periodo esperado donde cada bloque es extraído.

Observación 6.1.2. Dada una solución factible y, es claro ver que $\forall (a, b) \in \mathcal{A}$ se tiene que $(ET(y))_a \leq (ET(y))_b$.

Con esta información disponible, dada una solución óptima y^* de PCP^{at} se usará la siguiente adaptación de **TopoSort**, la cual es una heurística desarrollada en [7] que mostró un buen rendimiento en las instancias estudiadas en dicho documento. Se denotan los coeficientes de la matriz $\hat{D}$ como $\hat{D}_{b,d,t}$ y se construye una solución factible y de la siguiente manera:

1. Definir $\mathcal{B}^* = \mathcal{B}$ e $y_{b,d,t} = 0 \quad \forall b, d, t$.
2. Sea d^* un vector de $\mathbb{R}^{\mathcal{B}}$ que contiene un destino por cada bloque.
3. Si $\mathcal{B}^* = \phi$ PARAR.
4. Sea $b \in \arg\min\{(ET(y^*))_a \mid a \in \mathcal{B}^*\}$.
5. Si existe $\tau \leq T - 1$ tal que:

$$\sum_{t=0}^{\tau} y_{a,d_a^*,t} = 1 \ \forall a \text{ tal que } (b,a) \in \mathcal{A} \wedge \hat{D}y + \hat{D}_{b,d_b^*,\tau} \leq d \tag{6.1}$$

   entonces definir t como el menor τ que cumple (6.1).

   Si no existe tal τ, entonces $\mathcal{B}^* \leftarrow \mathcal{B}^* \setminus \{b\}$ e ir al paso 3.
6. Hacer $y_{b,d_b^*,t} = 1$, $\mathcal{B}^* \leftarrow \mathcal{B}^* \setminus \{b\}$ e ir al paso 3.

Observación 6.1.3. La heurística TopoSort requiere que los coeficientes de la matriz $\hat{D}$ sean no negativos, por lo que sólo se pudo aplicar a aquellas instancias que cumplen con esto. Por estos motivos, en este capítulo no habrán resultados asociados a las instancias *mine_t* y *mine_t_full*.

La idea de esta heurística es tomar los bloques ordenados según $(ET(y))_b$ y extraerlos "lo antes posible" en algún destino, esto es, en el tiempo más pronto donde su precedencia ya ha sido extraída y donde las side-constraints lo permitan. Esto tiene sentido gracias a la Observación 6.1.2 y a que los coeficientes de $\hat{D}$ son no negativos, por lo que siempre se mantiene factibilidad de la solución que se construye.

Para la elección del vector d^* simplemente se escoge el destino que cumple lo siguiente:

$$d_b^* = d \Leftrightarrow \sum_{t=0}^{T-1} y_{b,d,t}^* \geq \sum_{t=0}^{T-1} y_{b,d',t}^* \quad \forall d'$$

Es decir, se escoge el destino donde se envían más fracciones del bloque. El problema que tiene esta heurística es que no toma en cuenta que un bloque pueda ser dividido en fracciones y ser enviado a varios destinos: siempre se envían los bloques a solo un destino, lo que puede estar lejos de la solución óptima del modelo que considera esto.

En el Cuadro 6.1 se muestran los resultados obtenidos al utilizar TopoSort a partir de la solución óptima de la relajación lineal.

Lo primero que se puede observar es la rapidez del algoritmo para resolver todas las instancias, esto es muy útil pues permite, por ejemplo, probar variantes de la misma heurística y quedarse con la mejor. Por otro lado, los Gaps obtenidos en varias instancias resultan ser bastante buenos, pero de la misma forma existen muchas instancias donde los Gaps son demasiado altos (más del 10 %), por lo que se necesitan mejoras. Incluso una instancia posee un Gap mayor a 1, lo que indica que TopoSort generó una instancia con valor negativo, lo cual no es deseable.

Nombre	Gap	Tiempo
mine_n	0.034	0
mine_z_small	0.103	0.02
mine_k	0.141	0.01
mine_a	0.064	0
mine_z_medium	0.205	0.06
mine_m	0.286	0.01
mine_m_simple	0.130	0.01
mine_g	0.039	0.09
mine_z_large	0.103	0.12
mine_s	0.034	0.01
mine_mc_limit	0.019	0.18
mine_d	0.015	0.42
mine_r	0.048	0.97
mine_e	0.411	2.78
mine_mc	1.113	4.72

Cuadro 6.1: Resultados obtenidos para la heurística TopoSort en las instancias donde se pudo resolver la relajación lineal. Se muestra el Gap obtenido por la heurística y el tiempo (en segundos) que tomó encontrar la solución factible. Dado v_{LP} óptimo para la relajación LP y v_{IP} el valor obtenido para la solución factible, el Gap mostrado está dado por $1 - v_{IP}/v_{LP}$.

6.2. Primeras mejoras a TopoSort

De los resultados anteriores es claro que se necesitan diversas mejoras para TopoSort. Al analizar un poco más detalladamente la heurística utilizada, además de los resultados obtenidos (con especial atención en el caso donde se obtuvo una solución con valor objetivo negativo) se pueden observar 2 potenciales problemas de TopoSort:

1. Al ir llenando las capacidades, TopoSort no identifica si es que los últimos bloques extraídos proveen algún beneficio. Esto podría suceder si es que existen bloques "malos" que son extraídos para llegar a bloques beneficiosos en el LP, pero donde los recursos no permiten extraer estos últimos en el problema exacto.

2. Como se mencionó anteriormente, TopoSort envía los bloques a 1 destino solamente, lo que podría perjudicar la solución entregada.

Para corregir estos problemas, se desarrollaron las siguientes mejoras para TopoSort:

1. Para resolver el primer problema, se formuló un problema $UPIT$, pero con los bloques que considera la solución entregada por TopoSort. De esta manera, al resolver este nuevo problema $UPIT$, se "limpia" la solución factible actual al eliminar bloques perjudiciales para la función objetivo que se encuentren en el fondo de la mina. A esto se le denominó UPIT-Post.

2. Para abordar el segundo problema, para cada bloque se fijó el periodo donde es extraído, de acuerdo a la solución entregada por TopoSort. Luego, usando un problema

Nombre	TS	UP	ND
mine_n	0.034	0.034	0.034
mine_z_small	0.103	0.103	0.096
mine_k	0.141	0.082	0.082
mine_a	0.064	0.062	0.062
mine_z_medium	0.205	0.173	0.159
mine_m	0.286	0.086	0.086
mine_m_simple	0.130	0.102	0.085
mine_g	0.039	0.039	0.039
mine_z_large	0.103	0.103	0.103
mine_s	0.034	0.011	0.011
mine_mc_limit	0.019	0.017	0.017
mine_d	0.015	0.015	0.015
mine_r	0.048	0.048	0.037
mine_e	0.411	0.249	0.238
mine_mc	1.113	0.032	0.031

Cuadro 6.2: Gaps obtenidos al agregar las mejoras a TopoSort. Se muestran los Gaps obtenidos por TopoSort (TS), luego los Gaps al aplicar UPIT-Post (UP) a las soluciones, y luego los Gaps obtenidos al aplicar NewDest (ND) después de UPIT-Post. Los tiempos de ejecución no se muestran pues todos son del orden del tiempo que tomó TopoSort originalmente.

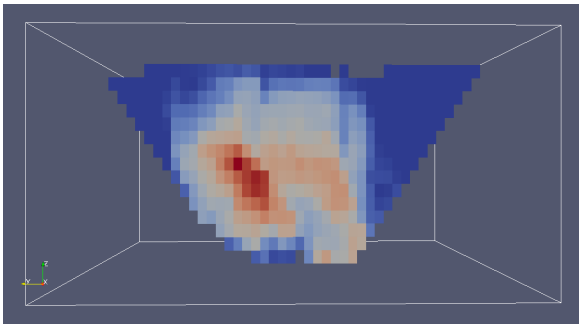
de programación lineal por cada periodo, se decide qué porción de cada bloque es enviada a cada destino. A esto se le llamó NewDest, y se aplicó después de UPIT-Post.

Es claro ver que usando estas técnicas el valor de la función objetivo sólo puede crecer (aunque quizás no estrictamente). Los resultados obtenidos se detallan en el Cuadro 6.2.

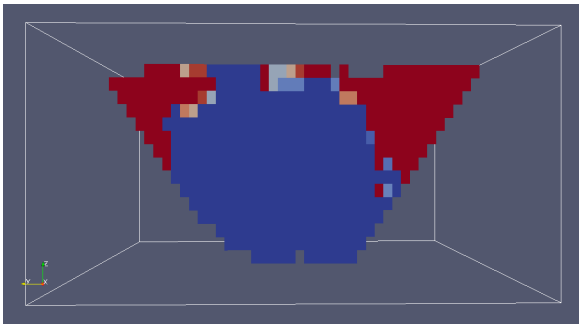
De los resultados se puede observar una disminución considerable en el Gap de varias instancias. Primero que todo, se logró "corregir" la solución que poseía valor objetivo negativo, obteniendo un Gap pequeño en ésta. El resto de las instancias mostró una disminución menos drástica, pero aún así considerable, y dada la rapidez de estas mejoras no está de más usarlas siempre.

Con estas mejoras se obtuvieron mejores soluciones, pero todavía existen instancias donde el Gap es considerable. Para poder determinar la razón por la que estos Gaps siguen siendo altos se decidió mirar más detalladamente las soluciones de las instancias que presentan problemas.

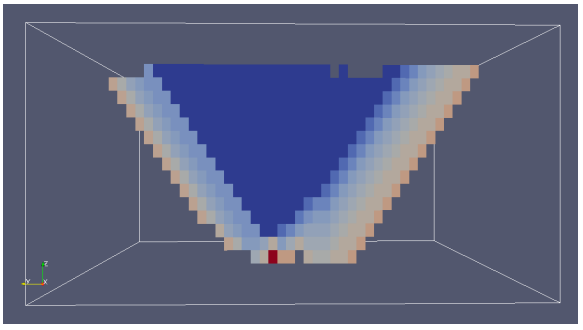
Si se analiza, por ejemplo, la solución obtenida para *mine_z_small* el problema se vuelve más claro. Esta es una mina que tiene simplemente 2 destinos (extraer-procesar y extraer-desechar). La Figura 6.1 detalla diversa información de esta mina: se muestra la distribución del mineral en 6.1a, la distribución de los bloques en los distintos destinos (que en este caso es simplemente procesar o no un bloque) en 6.1b, el Expected Time de



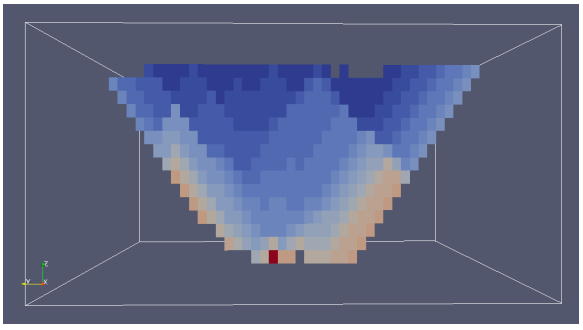
(a) Distribución del mineral en la mina. La escala de colores va desde Azul (para bloque sin mineral) hasta Rojo (para bloques con alto porcentaje de mineral).



(b) Distribución de los destinos donde se envían los bloques en la solución óptima del LP. Los bloques de color Azul son aquellos que bloques procesados, y los de color Rojo son aquellos que son desechados. El resto de los bloques son aquellos donde sólo se procesa una fracción de ellos.



(c) Expected Time de los bloques para la solución óptima del LP. Los colores van desde Azul hasta Rojo de acuerdo al periodo esperado donde se extrae cada bloque.



(d) Tiempos de Extracción para la solución obtenida con TopoSort. Los colores van desde Azul hasta Rojo de acuerdo al periodo donde se extrae cada bloque.

Figura 6.1: Información de la mina *mine_z_small*. Se muestra un corte de ésta.

cada bloque en 6.1c y los tiempo de extracción para la solución entregada por TopoSort (con las mejoras) en 6.1d. Todo esto para la solución óptima del LP.

De la Figura 6.1b se observan claramente las regiones con bloques "beneficiosos" y bloques "malos". Esto, junto con las Figuras 6.1c y 6.1d, indican que la principal diferencia entre la solución del LP y la obtenida por TopoSort es que esta última intenta extraer todos los bloques lo antes posible, sin importar el procesamiento que se le hace al bloque, lo que tiene por consecuencia que las regiones "malas" sean extraídas muy tempranamente, lo que no pasa en la solución del LP.

Para corregir esto, se decidió seguir lo indicado por el LP más estrictamente. Esto se logró definiendo, para una solución y factible para PCP^{at}:

$$(Tmin(y))_b = \text{mín}\{\ t \mid \sum_{d=0}^{R-1} y_{b,d,t} > 0\}$$

Nombre	TS2	UP	ND
mine_n	0.034	0.034	0.034
mine_z_small	0.045	0.045	0.037
mine_k	0.010	0.010	0.010
mine_a	0.064	0.063	0.063
mine_z_medium	0.113	0.113	0.099
mine_m	0.029	0.029	0.028
mine_m_simple	0.101	0.101	0.084
mine_g	0.013	0.013	0.013
mine_z_large	0.011	0.011	0.010
mine_s	0.002	0.002	0.001
mine_mc_limit	0.003	0.003	0.002
mine_d	0.004	0.004	0.004
mine_r	0.034	0.034	0.022
mine_e	0.143	0.143	0.131
mine_mc	0.002	0.002	0.002

Cuadro 6.3: Gaps obtenidos al usar TopoSort2 y al agregar las mejoras a ésta. Se muestran los Gaps obtenidos por TopoSort2 (TS2), luego los Gaps al aplicar UPIT-Post (UP) a las soluciones, y por último los Gaps obtenidos al aplicar NewDest (ND) después de UPIT-Post. Nuevamente no se muestran los tiempos por las mismas razones anteriores.

el tiempo mínimo donde se extrae un bloque. Y usando esto se modificó el algoritmo TopoSort cambiando la condición $\tau \leq T - 1$ por

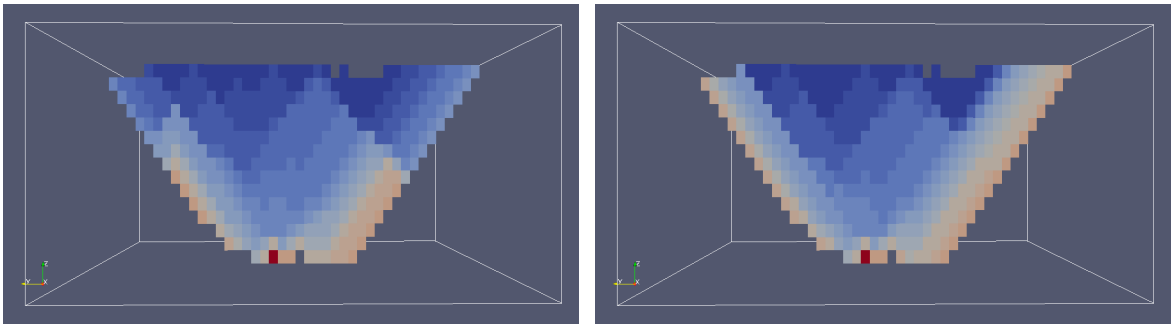
$$(Tmin(y^*))_b \leq \tau \leq T - 1$$

en el paso 5. De esta manera se fuerza a que los bloques no sean extraídos antes de lo indicado por el LP. A esta heurística se le denominó TopoSort2, y usando este cambio se obtuvieron los Gaps detallados en el Cuadro 6.3.

En los resultados se puede apreciar una mejora drástica en la efectividad de la heurística. Se obtuvieron muchos Gaps menores al 1 %, obteniendo solo una instancia con un Gap mayor al 10 %. Además se puede observar que al considerar TopoSort2 el efecto mejoras UPIT-Post y NewDest se vuelve menos notorio.

Para observar la principal diferencia en las soluciones se volverá a utilizar *mine_z_small*. En la Figura 6.2 se muestra la diferencia entre la nueva solución entregada por TopoSort2 y la antigua solución.

En esta figura se puede ver el efecto que tuvo agregar $Tmin(y^*)$ a TopoSort, el cual era el deseado: forzar a que los bloques sean extraídos no antes que lo indicado por la solución de la relajación lineal. En la solución entregada para la mina *mine_z_small* por TopoSort2 se puede ver que las regiones con bloques "malos" son extraídas después que lo considerado por TopoSort, lo cual logra que dicha solución se asemeje más a la solución del LP, obteniendo así un menor Gap.



(a) Tiempos de Extracción para la solución obtenida con TopoSort.

(b) Tiempos de Extracción para la solución obtenida con TopoSort2.

Figura 6.2: Soluciones obtenidas por TopoSort y TopoSort2 para la mina *mine_z_small*. Se muestra un corte de ésta, donde los colores van desde Azul hasta Rojo de acuerdo al periodo donde se extrae cada bloque.

6.3. Una nueva heurística como mejora a TopoSort

A pesar de obtener buenos Gaps en la mayoría de las instancias, todavía existen problemas en algunas de ellas. El principal problema observado es la distribución de los bloques "beneficiosos", es decir, el orden en el que se extraen los bloques que incrementan el valor de la función objetivo.

Para intentar remediar esto, se diseñó la heurística **RollingTime**, basada en la heurística *Sliding Time Window* propuesta en [8]. Esta heurística, a partir de una solución factible y^* para PCP^{at} (que puede ser la solución óptima del LP) construye una solución y de la siguiente forma:

1. Sea τ un entero arbitrario, $t^* = 0$ e $y_{b,d,t} = 0 \quad \forall b, d, t$.
2. Definir
$$\mathcal{B}^* = \left\{ b \in \mathcal{B} \mid \left(ET(y^*)\right)_b \leq t^* + \tau \wedge \sum_{d=0}^{R-1} \sum_{t=0}^{T-1} y_{b,d,t} = 0 \right\}.$$
3. Resolver el problema de planificación minera (con variables enteras) para 1 periodo usando CPLEX, pero considerando sólo los bloques en $\mathcal{B}^*$. Obteniendo una solución $\hat{y}$.
4. Hacer $y_{b,d,t^*} = \hat{y}_{b,d}$ $\forall b, d$.
5. $t^* \leftarrow t^* + 1$. Si $t^* = T$ PARAR, si no, ir al paso 2.

Esta heurística permite que un bloque pueda ser dividido en distintas fracciones y que éstas sean enviadas a distintos destinos, pero debe ser extraído por completo en solo 1 periodo. Esta elección de destinos será realizada por un problema de optimización de 1 periodo. Además, debido a la Observación 6.1.2, no hay problema en ir fijando variables periodo a periodo como se hace en esta heurística.

Nombre	Gap TS2	Gap RT	Tiempo RT
mine_n	0.034	0.014	3.65
mine_z_small	0.037	0.026	158.89
mine_k	0.010	0.044	138.48
mine_a	0.063	0.020	227.66
mine_z_medium	0.099	0.044	379.14
mine_m	0.028	0.031	24.66
mine_m_simple	0.084	0.022	224.46
mine_g	0.013	0.083	663.26
mine_z_large	0.010	0.013	1216.03
mine_s	0.001	0.035	208.05
mine_mc_limit	0.002	0.056	1705.07
mine_d	0.004	0.066	5834.94
mine_r	0.022	0.009	13594.47
mine_e	0.131	0.047	5438.45
mine_mc	0.002	0.054	2365.19

Cuadro 6.4: Resultados obtenidos al aplicar RollingTime (RT) a las soluciones entregadas por TopoSort2 junto con las mejoras (denotado simplemente TS2). Además se agregó el tiempo (en segundos) que tomó RollingTime en cada instancia.

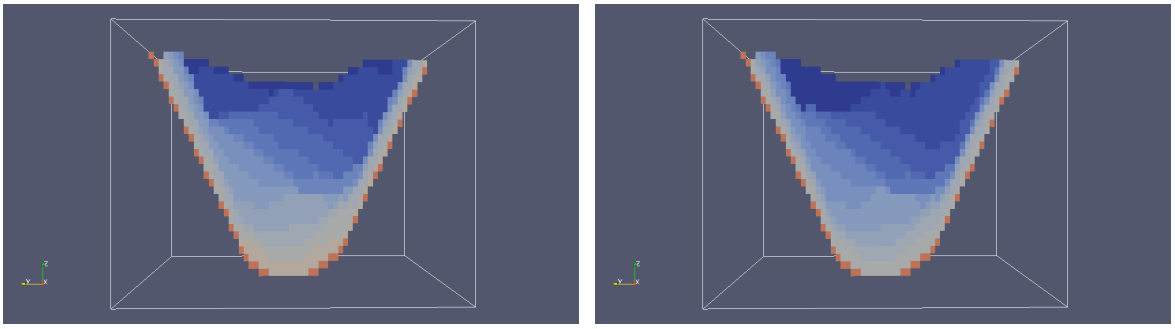
El nombre proviene del hecho que se está considerando una "ventana de tiempo de tamaño τ" sobre los bloques que se consideran para resolver el problema de 1 periodo, y en cada iteración se mueve esta ventana un periodo más adelante. Esto permite cierta flexibilidad sobre el orden en el que se extraen los bloques, no tomando tan estrictamente el orden entregado por los Expected Time.

Se observó que $\tau = 2$ es un buen tamaño de la ventana, pues mantiene el tamaño del problema a resolver con CPLEX acotado, pero no es tan miope como al considerar $\tau = 1$. Es por esto que se usó dicho τ en los experimentos. Además, usando esta heurística sobre la solución óptima del LP se observaron 2 problemas:

- La heurística toma demasiado tiempo, pues los problemas son de tamaño considerable.
- Los Gaps que se obtienen no son tan buenos como para justificar el tiempo que toma encontrar las soluciones.

Por estos motivos, se decidió aplicar esta heurística a las soluciones obtenidas con TopoSort2, es decir, usar y^* como la solución que se entrega TopoSort2, lo que mantiene el tamaño de los problemas resueltos con CPLEX más acotado.

Los resultados obtenidos se presentan en el Cuadro 6.4, donde se muestran los Gaps obtenidos al aplicar RollingTime a las soluciones obtenidas por TopoSort2 (junto con las mejoras).



(a) Tiempos de Extracción para la solución obtenida con TopoSort2.

(b) Tiempos de Extracción para la solución obtenida con RollingTime.

Figura 6.3: Soluciones obtenidas por TopoSort2 y RollingTime para la mina *mine_z_medium*. Se muestra un corte de ésta, donde los colores van desde Azul hasta Rojo de acuerdo al periodo donde se extrae cada bloque.

Del Cuadro 6.4 se ve la importancia de mantener los problemas que se resuelven con CPLEX acotados. Las instancias disponibles se pudieron resolver en tiempos razonables, en efecto, la mayoría tomó menos de 1 hora, pero una de las instancias tomó más de 3 horas en encontrar una solución, lo cual no es un tiempo considerablemente alto, pero indica claramente el cuidado que se debe tener con los problemas, pues los tiempos crecen rápidamente.

Por otro lado, no siempre se mejoró el Gap con respecto al de la solución entregada por TopoSort2, sin embargo, las soluciones que tenían Gaps mayores a 5 % bajaron considerablemente, obteniendo para todas las instancias un Gap menor al 5 % con TopoSort2 o con RollingTime. Esto es un resultado completamente satisfactorio, no solo por la buena calidad de las soluciones, sino que también por el poco tiempo que toma encontrarlas, sin necesidad de utilizar métodos más sofisticados como Branch and Bound u otro método para resolver problemas de programación entera.

En la Figura 6.3 se muestran las soluciones entregadas por TopoSort2 y por RollingTime sobre la instancia *mine_z_medium*, la cual fue una de las instancias donde se redujo el Gap considerablemente. En esta figura se ve que ambas soluciones son bastante similares; sólo se observa un cambio sutil en la distribución entre los distintos periodos de los bloques "beneficiosos", logrado así una mejor solución. El cambio no es drástico en cuanto a la estructura de la solución, pero aumenta la calidad de la solución de manera considerable.

Capítulo 7

Generalización del Algoritmo

Debido a la eficiencia del algoritmo BZ fue natural buscar alguna generalización de éste. Como el algoritmo que fue propuesto en [3] ya se encuentra planteado de manera general para modelos de Optimización Lineal, se decidió explorar otras áreas de Optimización donde un esquema de estas características pueda ser aplicado.

En este trabajo, debido a la amplia gama de aplicaciones que tienen los modelos Cónicos, se decidió explorar esta área.

7.1. Optimización Cónica

Primero se recordarán las siguientes definiciones básicas:

Definición 7.1.1. Sea E un espacio vectorial Euclideano de dimensión finita. Un conjunto $K \subseteq E$ es un *cono* (puntiagudo) si cumple las siguientes propiedades:

1. K es no vacío y cerrado para la suma, es decir:

$$v, w \in K \Rightarrow v + w \in K$$

2. K es un conjunto cónico:

$$v \in K, \lambda \geq 0 \Rightarrow \lambda v \in K$$

3. K es puntiagudo:

$$v \in K, -v \in K \Rightarrow v = 0$$

Se denota por $v \succeq_K w$ si $v - w \in K$. Claramente esto define un orden parcial. Además se denota por $v \succ_K w$ si $v - w \in \text{int}(K)$. Análogamente se denota como $v \preceq_K w$ si $w - v \in K$ y lo mismo para $\prec_K$.

Con esta definición, dado un cono K (convexo, puntiagudo, cerrado y con interior no vacío), $c \in \mathbb{R}^n$, una función lineal $A : \mathbb{R}^n \to E$ y $b \in E$, se define un problema *cónico* como

$$\min_x \{c'x \mid Ax \succeq_K b\}. \tag{7.1}$$

Claramente si se considera $E = \mathbb{R}^m$ y $K = \mathbb{R}^m_+$ se obtiene un problema de Optimización Lineal.

Definición 7.1.2. Se dice que un problema cónico es *estrictamente factible* si existe x tal que $Ax \succ_K b$.

Si se supone que un problema es *estrictamente factible*, entonces se tienen todas las nociones de dualidad que existen en los problemas lineales. Se puede definir el problema dual de (7.1) como:

$$\max_{\lambda} \{\langle b, y\rangle \mid A^* y = c, \lambda \succeq_{K_*} 0\} \tag{7.2}$$

donde $\langle \cdot, \cdot \rangle$ denota el producto interno de E, A^* es el conjugado de A y K_* es el *cono dual* de K y está definido como

$$K_* = \{\lambda \in E \mid \langle \lambda, v\rangle \geq 0 \ \forall v \in K\}.$$

Con esto, se pueden generalizar los Teoremas de *Dualidad Fuerte, Dualidad Débil y Holgura Complementaria*. Para detalles ver [1].

Un ejemplo que se utilizará en lo siguiente es el conocido *Cono de Lorentz* de dimensión m, conocido también como el "Cono de Helado". Éste se define como

$$L^m = \{x \in \mathbb{R}^m \mid \|(x_1, \ldots, x_{m-1})\| \leq x_m\} \tag{7.3}$$

donde $\|\cdot\|$ denota la norma euclidiana, es decir

$$\|x\| = \sqrt{\sum_{i=1}^{m} x_i^2}$$

Es fácil ver que el cono dual de L^m es él mismo, es decir $L^m_* = L^m$.

7.2. Formulando robustez como un problema cónico

En el mundo real los datos de un problema no son siempre certeros. En general se pueden presentar errores de medición, valores que pueden cambiar aleatoriamente, etc. Por estos motivos, se vuelve importante encontrar soluciones "robustas" a los problemas que tengan incertidumbre en sus datos, es decir, soluciones que aseguren un buen comportamiento a pesar de la variabilidad que pueden poseer los datos del problema.

En lo siguiente se supondrá que se tiene un problema de optimización lineal

$$\begin{array}{ll} \text{máx} & c'x \\ \text{s.t.} & Ax \leq b \end{array} \tag{7.4}$$

donde hay incertidumbre sobre los datos c. Matemáticamente hablando, se considerará que

$$c = c^* + Pu, \quad u'u \leq 1$$

donde c^* son los valores nominales y Pu son las perturbaciones que se permitirán en los valores. Al considerar que $u'u \leq 1$ se está considerando que los datos varían dentro de un elipsoide.

Como se busca que la solución sea robusta, se abordó este problema desde el punto de vista del *peor caso*. Es decir, se desea resolver

$$\max_x f(x)$$

donde

$$f(x) = \text{ínf}\{c'x \mid \exists u, u'u \leq 1, c = c^* + Pu, Ax \leq b\}.$$

En este modelo también se podría considerar incertidumbre sobre los datos A o b, pero como primer paso se decidió sólo considerar variaciones en la función objetivo. Para un análisis más general y detallado ver [1].

Este problema se puede reformular como

$$\begin{array}{lll} \max & r & \\ \text{s.t.} & r \leq c'x & \forall c \in \mathcal{U} \\ & Ax \leq b & \end{array} \tag{7.5}$$

donde $\mathcal{U} = \{c \mid \exists u, u'u \leq 1,\ c = c^* + Pu\}$. Ahora se reescribirá este problema como un problema cónico, de acuerdo a [1].

Claramente, la restricción $r \leq c'x, \forall c \in \mathcal{U}$ se satisface si y sólo si

$$\begin{aligned} 0 &\leq \min_{u,u'u\leq 1} \{c'x - r \mid c = c^* + Pu\} \\ &= \min_{u,u'u\leq 1} \{(c^* + Pu)' x - r\} \\ &= (c^*)'x - r + \min_{u,u'u\leq 1} \{u'P'x\} \\ &= (c^*)'x - r - \|P'x\| \end{aligned}$$

por lo tanto, si se define

$$C = \begin{bmatrix} P' & 0 \\ (c^*)' & -1 \end{bmatrix}$$

y se denota por $L = L^{n+1}$ el cono de Lorentz de dimensión $n+1$, se tendrá que

$$r \leq c'x,\ \forall c \in \mathcal{U} \Leftrightarrow C \begin{bmatrix} x \\ r \end{bmatrix} \succeq_L 0$$

de donde se obtiene el problema cónico

$$\begin{array}{lll} \max & r & \\ \text{s.t.} & Ax \leq b & \\ & C\,[x, r]' \succeq_L 0 & \end{array} \tag{7.6}$$

7.3. El algoritmo BZ Cónico

En esta sección se trabajará con un problema como

$$\begin{array}{rll} z^* = & \text{máx} & c'x \\ & \text{s.t.} & Ax \preceq_{K^1} b \\ & & Dx \preceq_{K^2} d \end{array} \tag{7.7}$$

el cual se supondrá estrictamente factible, con K^1 y K^2 conos. Sea $\pi^0 \in K^2_*$, cono dual de K^2 y $k = 1$. Con esto se propone la siguiente generalización del algoritmo BZ:

1. Sea $\Omega_k \subseteq R^n$ una relajación de $P = \{x \mid Ax \preceq_{K^1} b,\ Dx \preceq_{K^2} d\}$. Es decir, $P \subseteq \Omega_k$.
2. Resolver

$$\begin{array}{rll} z^U_k = & \text{máx} & c'x - \langle \pi^{k-1}, Dx - d\rangle \\ & \text{s.t.} & Ax \preceq_{K^1} b \\ & & x \in \Omega^k. \end{array} \tag{7.8}$$

   obteniendo una solución óptima w^k.
3. Si $k \geq 2$ y $w^k \in \langle \mathcal{X}^{k-1}\rangle$, entonces PARAR.
4. Sea $\mathcal{X}^k = \{v^1, \ldots, v^{n_k}\}$ un conjunto finito de puntos en $\mathbb{R}^n$ tales que $w^k \in \langle \mathcal{X}^k\rangle$.
5. Resolver

$$\begin{array}{rll} z^L_k = & \text{máx} & c'x \\ & \text{s.t.} & Ax \preceq_{K^1} b \\ & & Dx \preceq_{K^2} d \\ & & x = \sum_{i=1}^{n_k} \lambda^k_i v^i \end{array} \tag{7.9}$$

   Sean $x^k \in \mathbb{R}^n$, $\lambda^k \in R^{n_k}$ y $\pi^k \in K^2_*$ los óptimos primales y los duales correspondientes a las restricciones $Dx \preceq_{K^2} d$ del problema (7.9), respectivamente.
6. Si $\pi^k = \pi^{k-1}$ PARAR.
7. $k \leftarrow k + 1$. Ir al paso 1.

Con este algoritmo se tienen los mismos resultados vistos en el Capítulo 2.

Es fácil ver que si se considera $\mathcal{X}^k$ tal que $\langle \mathcal{X}^k\rangle \supseteq \langle \mathcal{X}^{k-1}\rangle$ se obtiene un algoritmo finito. En efecto, como $w^k \notin \langle \mathcal{X}^{k-1}\rangle$, entonces

$$\dim(\langle \mathcal{X}^k\rangle) > \dim(\langle \mathcal{X}^{k-1}\rangle).$$

Además se tiene el siguiente resultado análogo a la Proposición 2.1.2:

Proposición 7.3.1. *Considerar el problema* $\text{máx}\{c'x \mid Ax \preceq_{K^1} b, Dx \preceq_{K^2} d\}$. *Sea* $\hat{x}$ *una solución óptima y* $\hat{\pi} \in K_*^2$ *duales óptimos asociados a las restricciones* $Dx \preceq_{K^2} d$, *entonces* $\hat{x}$ *es óptimo para el problema*

$$\begin{array}{ll} \text{máx} & c'x - \langle \hat{\pi}, Dx - d \rangle \\ s.a & Ax \preceq_{K^1} b \end{array} \tag{7.10}$$

La prueba de esta Proposición también sigue directamente de dualidad fuerte para el caso cónico, y se presenta a continuación.

Demo :
Se considera el problema (7.7). El dual de este problema está dado por

$$\begin{array}{ll} \text{mín} & \langle b, \rho \rangle + \langle \pi, d \rangle \\ \text{s.t.} & \\ & A^*\rho + D^*\pi = c \\ & \rho \succeq_{K_*^1} 0 \\ & \pi \succeq_{K_*^2} 0 \end{array} \tag{7.11}$$

Sea $\hat{x}$ el óptimo primal, y sus duales correspondientes $\hat{\rho}$, $\hat{\pi}$. Claramente el problema (7.11) es equivalente a resolver,

$$\text{mín}\{\langle d, \pi \rangle + f(\pi) : \pi \succeq_{K_*^2} 0\}$$

donde,

$$f(\pi) = \text{mín}\{\langle b, \rho \rangle : A^*\rho = c - D^*\pi, \rho \succeq_{K_*^1} 0\}.$$

Si se aplica dualidad a este último problema se ve que

$$f(\pi) = \text{máx}\{\langle c - D^*\pi, x \rangle : Ax \preceq_{K^1} b\}$$

de donde se obtiene que el problema (7.11) es equivalente a

$$\langle d, \hat{\pi} \rangle + f(\hat{\pi}) = \langle d, \hat{\pi} \rangle + \text{máx}\{\langle c - D^*\hat{\pi}, x \rangle : Ax \preceq_{K^1} b\}.$$

Por lo que el problema (7.7) es equivalente a

$$\text{máx}\{c'x + \langle d - Dx, \hat{\pi} \rangle : Ax \preceq_{K^1} d\}. \tag{7.12}$$

donde se usó que $\langle D^*\pi, x \rangle = \langle \pi, Dx \rangle$. Con esto se concluye que el óptimo de (7.12) es igual al óptimo de (7.7). Más aún, se concluye que $\hat{x}$ es dicho óptimo, pues es factible y por holgura complementaria se tiene que $\langle d - D\hat{x}, \hat{\pi} \rangle = 0$, por lo que $\hat{x}$ tiene el mismo valor en la función objetivo de (7.12) y (7.7).

■

Corolario 7.3.2. *Bajo el mismo contexto de la Proposición (7.3.1),* $\hat{x}$ *es óptimo para el problema*

$$\begin{array}{ll} \text{máx} & c'x - \langle \hat{\pi}, Dx - d \rangle \\ s.a & Ax \preceq_{K^1} b \\ & x \in \Omega \end{array} \tag{7.13}$$

donde Ω *es una relajación de* $P = \{x \mid Ax \preceq_{K^1} b,\ Dx \preceq_{K^2} d\}$.

Demo :
Se sabe que $\hat{x}$ es óptimo de (7.10). Además, como (7.10) es una relajación de (7.13) y $\hat{x} \in P$, (entonces es factible para (7.13)) se concluye.

■

Proposición 7.3.3. *Sea z^* óptimo del problema (7.7), entonces en cada iteración se tiene que $z_k^L \leq z^* \leq z_k^U$. Además, si el algoritmo alcanza una condición de parada, la última solución encontrada para (7.9) es óptima.*

Demo :
Claramente $z_k^L \leq z^*$. Además, para cualquier x factible para (7.7), si se tiene que $\pi \succeq_{K_*^2} 0$, por definición de K_*^2 se tiene que

$$\langle \pi, d - Dx \rangle \geq 0$$

por lo que $c'x - \langle \pi, Dx - d \rangle \geq c'x$ y entonces $z^* \leq z_k^U$. El resto de la demostración es idéntica a la demostración de la Proposición 2.1.3, reemplazando la Proposición 2.1.2 por el Corolario 7.3.2.

■

7.4. Un modelo de Optimización Robusta para el problema de minería

El problema de planificación minera que se ha estudiado posee diversas fuentes de incertidumbre, lo que es un problema al momento de considerar un modelo determinista. Entre estas distintas fuentes de incertidumbre, la incertidumbre geológica es la que ha sido más estudiada. En esta sección se propondrá un nuevo modelo que considere otro tipo de incertidumbre en los datos.

7.4.1. Reformulación del problema

Con el objetivo de aplicar la generalización propuesta del algoritmo BZ se decidió estudiar un nuevo modelo, que considere variabilidad en los beneficios entregados por cada bloque. Esto se realizó modificando el modelo de la siguiente manera:

- Se asumirá que cada mina posee M tipos de mineral, los cuales tendrán un precio $p_0^m \in \mathbb{R}$, $m = 0, \ldots, M-1$.
- Se considerará que cada bloque posee un costo (negativo) asociado al destino al cual es enviado, el cual será denotado por $\tilde{c}_{b,d} \in \mathbb{R}$. Típicamente son costos de extracción y extracción con procesamiento.
- Cada bloque poseerá una cantidad del mineral m que es extraído de él dependiendo del destino donde es enviado. Esta cantidad de mineral será denotada $\hat{q}_{b,d}^m \in \mathbb{R}$.

- El beneficio de cada par bloque-destino está dado por la cantidad de minerales que se extraen del bloque al ser enviado al destino d según el precio de cada mineral, es decir $\sum_m \hat{q}^m_{b,d} \cdot p^m_0$.
- Al igual que antes, se definen las variables $y_{b,d,t}$ que indican si un bloque b es enviado al destino d en el periodo t.
- Se agregan las variables w^m_t, $t = 0, \ldots, T-1, m = 0, \ldots, M-1$ que indicarán la cantidad de mineral m extraído en el periodo t.

Con estas definiciones, si se denota por $p^m_t = p^m_0 \cdot \alpha^t$ y $\hat{c}_{b,d,t} = \alpha^t \tilde{c}_{b,d}$, donde α es la tasa de descuento, se puede reescribir el modelo determinista como

$$
\begin{aligned}
(PCP^{at}) \quad \text{máx} \quad & \sum_{m=0}^{M-1} \sum_{t=0}^{T-1} p^m_t w^m_t + \hat{c}' y \\
\text{s.t.} \quad w^m_t &= \sum_{b \in \mathcal{B}} \sum_{d=0}^{R-1} \hat{q}^m_{b,d} y_{b,d,t} \quad \forall m = 0, \ldots, M-1 \\
\sum_{t=0}^{\tau-1} \sum_{d=0}^{R-1} y_{a,d,t} &\leq \sum_{t=0}^{\tau-1} \sum_{d=0}^{R-1} y_{b,d,t} \quad \forall (a,b) \in \mathcal{A}, \tau = 0, \ldots, T-1 \\
\sum_{t=0}^{T-1} \sum_{d=0}^{R-1} y_{b,d,t} &\leq 1 \quad \forall b \in \mathcal{B} \\
\hat{D} y &\leq d \\
y &\geq 0
\end{aligned}
$$

y al igual que antes, usando la transformación vista en el Teorema 3.3.1 se pueden transformar las variables y para llegar a un problema de la forma

$$
\begin{aligned}
\text{máx} \quad & \sum_{m=0}^{M-1} (p^m)' w^m + c' x \\
\text{s.t.} \quad & w^m = Q^m x \quad \forall m = 0, \ldots, M-1 \\
& Ax \leq b \\
& Dx \leq d
\end{aligned}
\tag{7.14}
$$

donde $Ax \leq b$ tiene las restricciones de precedencia (en el grafo expandido) y las cotas de las variables, $Dx \leq d$ son las side-constraints, y c y Q^m se obtienen al hacer el cambio de variables de y a x.

7.4.2. Agregando incertidumbre al modelo

En la práctica, para obtener el beneficio de cada par bloque-destino, típicamente se procede como se explicó en la sección anterior con el objetivo de maximizar el valor presente neto de la extracción del mineral.

Este modelo considera un precio de cada mineral fijo p^m_0 el cual es descontado a lo largo del tiempo para obtener un vector p^m de precios del mineral m en cada periodo. El problema

de esto es que en la realidad hay incertidumbre sobre el precio de los minerales, sobre todo si se tiene un horizonte de tiempo significativo, lo que limita el modelo determinista antes considerado. Por estas razones se decidió incorporar esta incertidumbre al modelo, de manera de obtener soluciones que sean robustas con respecto a las variaciones de p^m.

Como fue discutido en la Sección 7.2, se considerará que

$$p^m = \bar{p}^m + P^m u^m, \quad (u^m)' u^m \leq 1$$

donde $\bar{p}^m$ son los valores nominales por cada mineral y $P^m u^m$ son las perturbaciones que se permitirán en los precios de cada mineral.

En este caso, para maximizar el peor caso posible se debe resolver el problema

$$\max_{w^0,\ldots,w^{M-1},x} f(w^0,\ldots,w^{M-1},x)$$

donde

$$f(w^0,\ldots,w^{M-1},x) = \inf\{\textstyle\sum_{m=0}^{M-1}(p^m)' w^m + c'x \mid \begin{array}{l} \exists u^m, (u^m)' u^m \leq 1, p^m = \bar{p}^m + P^m u^m, \\ w^m = Q^m x, Ax \leq b, Dx \leq d\} \end{array}$$

lo que lleva al problema

$$\begin{array}{ll} \max & \displaystyle\sum_{m=0}^{M-1} r^m + c'x \\ \text{s.t.} & r^m \leq (p^m)' w^m \quad \forall m = 0,\ldots,M-1,\ p^m \in \mathcal{U}^m \\ & w^m = Q^m x \quad \forall m = 0,\ldots,M-1 \\ & Ax \leq b \\ & Dx \leq d \end{array} \tag{7.15}$$

donde $\mathcal{U}^m = \{p^m \mid \exists u^m, (u^m)' u^m \leq 1, p^m = \bar{p}^m + P^m u^m\}$. El que a su vez, siguiendo el procedimiento considerado en la Sección 7.2, lleva al problema cónico

$$\begin{array}{ll} \max & \displaystyle\sum_{m=0}^{M-1} r^m + c'x \\ \text{s.t.} & w^m = Q^m x \quad \forall m = 0,\ldots,M-1 \\ & Ax \leq b \\ & Dx \leq d \\ & C^m [w^m, r^m]' \succeq_L 0 \quad \forall m = 0,\ldots,M-1 \end{array} \tag{7.16}$$

donde

$$C^m = \begin{bmatrix} (P^m)' & 0 \\ (\bar{p}^m)' & -1 \end{bmatrix}$$

y $L = L^{T+1}$ es el cono de Lorentz de dimensión $T+1$, con T el número de periodos.

Observación 7.4.1. Claramente el modelo puede ser modificado para que considere incertidumbre geológica. Si por ejemplo, se consideran variaciones en la cantidad de material estimado para cada bloque, se puede hacer un desarrollo similar de manera de permitir variaciones sobre las matrices Q^m para llegar a un modelo similar a (7.16).

El problema (7.16) es del mismo tipo que (7.7), por lo que se puede aplicar el algoritmo BZ propuesto en este capítulo. Esto se hizo considerando lo siguiente:

- Para definir $\mathcal{X}$ se siguió con la misma estrategia usada en el caso Lineal, es decir, utilizando particionamiento de las variables x. Las variables w^m y r^m se considerarán aparte, lo cual no afectará mucho al algoritmo pues son solamente $M(T+1)$ variables extra.
- Al definir el sub-problema se penalizarán las side-constraints y las restricciones cónicas.
- Cada problema master será resuelto usando la librería de Optimización No Lineal de CPLEX.
- Se usará $\Omega_k = \mathbb{R}^n_+$, con lo que se obtiene un problema similar al caso Lineal. Más aún, el sub-problema resulta ser equivalente a un Max-Closure como se demostrará a continuación.

Proposición 7.4.2. *Considerar el problema penalizado*

$$\begin{array}{ll} \text{máx} & \sum_{m=0}^{M-1} r^m + c'x - \mu'(Dx-d) + \sum_{m=0}^{M-1} (\pi^m)' C^m \left[w^m, r^m\right]' \\ s.t. & w^m = Q^m x, \quad \forall m = 0, \ldots, M-1 \\ & Ax \leq b \end{array} \tag{7.17}$$

donde $\pi^m \succeq_L 0$ *y* $\mu \geq 0$ *son duales factibles del problema* (7.16)*. Entonces el problema* (7.17) *es equivalente a un problema Max-Closure.*

Demo :
Primero se reescribe el problema como

$$\begin{array}{ll} \text{máx} & \sum_{m=0}^{M-1} (1-\pi_T^m) r^m + (c' - \mu' D)x + \mu' d + \sum_{m=0}^{M-1} (\pi^m)' \begin{bmatrix} (P^m)' \\ (\bar{p}^m)' \end{bmatrix} w^m \\ \text{s.t.} & w^m = Q^m x, \quad \forall m = 0, \ldots, M-1 \\ & Ax \leq b \end{array} \tag{7.18}$$

donde $\pi^m = (\pi_0^m, \ldots, \pi_T^m)$.

Por otro lado, el dual del problema (7.16) está dado por

$$\begin{array}{ll} \text{mín} & \rho' b + \mu' d \\ \text{s.t.} & \sum_{m=0}^{M-1} (\gamma^m)' Q^m + \rho' A + \mu' D = c \\ & -\gamma^m - (\pi^m)' \begin{bmatrix} (P^m)' \\ (\bar{p}^m)' \end{bmatrix} = 0 \\ & \pi_T^m = 1 \\ & \rho \geq 0 \\ & \mu \geq 0 \\ & \gamma^m \in \mathbb{R} \\ & \pi^m \succeq_L 0 \end{array} \tag{7.19}$$

Por lo tanto, como $\pi_T^m = 1$, la función objetivo de (7.18) no depende de ningún r^m, más aún, se puede reemplazar $w^m = Q^m x$ y se obtiene un problema Max-Closure.

■

Con este resultado se puede utilizar nuevamente el algoritmo pseudo-flow de Hochbaum para resolver el sub-problema, lo cual es bueno considerando que, a pesar del gran tamaño de las instancias, el sub-problema se puede resolver eficientemente.

7.5. Resultados Computacionales

Con lo discutido anteriormente se pudo implementar la nueva versión del algoritmo BZ que considera variaciones en los precios de los minerales, reutilizando gran parte del algoritmo original. Además, debido a la dificultad del problema, el tamaño de la partición crece rápidamente, por lo que se decidió agregar experimentos con Reset, el cual fue discutido en la Sección 5.3.

Por disponibilidad de los datos sólo se consideraron minas que poseen un tipo de mineral. Las instancias que se utilizaron poseen una matriz P de variación del precio del mineral y un precio nominal p^* como datos de entrada.

Además, aparte de determinar cuánto tiempo toma el algoritmo en resolver cada instancia, se decidió comparar la calidad de las soluciones obtenidas para el caso robusto con las soluciones obtenidas al no considerar incertidumbre. Como se considera solo un mineral, se supone que el precio del mineral es p, donde se agrega incertidumbre de la forma

$$p = p^* + Pu, \quad u'u \leq 1.$$

Se sabe que el problema robusto asociado es

$$\begin{array}{lll} \text{máx} & r + c'x & \\ \text{s.t.} & r \leq p'w & \forall p \in \mathcal{U} \\ & w = Qx & \\ & Ax \leq b & \\ & Dx \leq d & \end{array} \tag{7.20}$$

donde $\mathcal{U} = \{p \mid \exists u, u'u \leq 1, p = p^* + Pu\}$. Se optó por medir la calidad de una solución (w, x) mediante su valor *nominal*, su valor en el *peor* caso y en el *mejor* caso. El nominal claramente es $(p^*)'w + c'x$.

Como (w, x) están fijos, para calcular el peor caso basta resolver

$$\begin{aligned} \min_u\{p'w + c'x \mid p = p^* + Pu, u'u \leq 1\} &= c'x + \min_u\{(p^* + Pu)'w \mid u'u \leq 1\} \\ &= (p^*)'w + c'x + \min_u\{u'P'w \mid u'u \leq 1\} \\ &= (p^*)'w + c'x - \|P'w\| \end{aligned}$$

Nombre	Peor Valor		Valor Nominal		Mejor Valor	
	Determ.	Robusta	Determ.	Robusta	Determ.	Robusta
mine_n	1.7381	1.9245	2.4486	2.3422	3.1591	2.7599
mine_mc_limit	81.4058	90.5321	132.4818	126.5713	183.5578	162.6104
mine_mc	105.6240	114.5729	151.2943	146.4474	196.9646	178.3220

Cuadro 7.1: Resultados obtenidos (todos en base 10^7) para el problema de optimización robusta. Se muestran 2 soluciones: la solución determinista y la solución robusta, las cuales se obtuvieron del problema original (sin incertidumbre) y del problema que considera incertidumbre, respectivamente. Para ambas soluciones se muestra el Peor Valor de solución al considerar la incertidumbre, el Valor Nominal y el Mejor Valor.

Análogamente, para calcular el mejor caso:

$$\begin{aligned} \max_u\{p'w + c'x \mid p = p^* + Pu, u'u \leq 1\} &= (p^*)'w + c'x + \max_u\{u'P'w \mid u'u \leq 1\} \\ &= (p^*)'w + c'x + \|P'w\| \end{aligned}$$

Por lo tanto, dada una solución (w, x), su mejor valor, valor nominal y peor valor son respectivamente

- $(p^*)'w + c'x + \|P'w\|$
- $(p^*)'w + c'x$
- $(p^*)'w + c'x - \|P'w\|$

En el Cuadro 7.1 se presentan los resultados obtenidos para el problema robusto en comparación a los resultados obtenidos para el problema sin considerar variación en los precios, para las instancias donde estaban disponibles los datos. Se llamarán *soluciones deterministas* a las soluciones obtenidas sin considerar variación en los precios (modelo determinista) y *soluciones robustas* a las soluciones que consideran dicha variación.

Con respecto a los valores obtenidos, se puede observar la importancia de usar un modelo que considere incertidumbre. En el peor caso, las soluciones deterministas tienen un valor aproximadamente de un 10 % peor que las soluciones robustas, lo cual puede ser malo, dependiendo del riesgo en el que se desea incurrir. Si a esto se le agrega que el valor nominal de las soluciones robustas son aproximadamente un 4 % peores que el óptimo (que es la solución determinista), vale la pena considerar la solución robusta, sobre todo si la compañía es adversa al riesgo. El Mejor Valor obtenido por las soluciones tiene este mismo tipo de relación (lo cual se ve claramente en las fórmulas respectivas), es decir, la solución robusta es aproximadamente un 10 % peor que la solución determinista. Esto ilustra un hecho simple y intuitivo: al considerar incertidumbre y tratar de mejorar el *peor caso*, se obtiene una solución un poco peor en cuanto a su valor nominal, pero con un riesgo considerablemente menor a las obtenidas usando enfoques deterministas tradicionales.

En el Cuadro 7.2 se muestran los tiempos que tomó resolver cada instancia. En esta

Nombre	BZ Original	BZ Cónico	BZ Cónico - RST
mine_n	0.1	1.08	-
mine_mc_limit	201.1	11101.52	4910.37
mine_mc	333.62	366411.38	12418.52

Cuadro 7.2: Tiempos (en segundos) de ejecución del algoritmo BZ en su versión original y cónica, donde esta última fue implementada para el caso de optimización robusta discutido. Además se agregan los tiempos obtenidos al usar Reset en el algoritmo BZ Cónico con un umbral de 1000 segundos; la primera instancia no posee un valor pues no alcanzó a aplicar Reset.

tabla también se muestran los tiempos utilizados por la versión original del algoritmo para tener una referencia. Además se agregó una columna que indica los resultados obtenidos al agregar Reset, ya que se observó que el problema master se torna difícil de resolver muy rápidamente.

De los tiempos obtenidos se puede notar claramente el aumento drástico en la dificultad del problema. El tiempo necesario para resolver el problema cónico es considerablemente mayor al tiempo que toma el algoritmo original, lo cual era esperable. A pesar de esto, los tiempos obtenidos son bastante buenos para la dificultad del problema.

Por otra parte Reset mostró ser fundamental al momento de reducir el tamaño del problema master. En instancias donde anteriormente no había sido necesario aplicar Reset ahora se vuelve de suma importancia considerarlo. Con este speed-up se pudo reducir drásticamente el tiempo de ejecución del algoritmo para ambas instancias donde se aplicó, lo cual es una buena señal para otras instancias que se consideren a futuro.

Capítulo 8

Conclusiones y Trabajo Futuro

Con este trabajo se estudió e implementó satisfactoriamente un algoritmo capaz de resolver la relajación lineal del problema de planificación minera. Este algoritmo, desarrollado por D. Bienstock y M. Zuckerberg, es capaz de explotar la estructura de las soluciones óptimas del problema, tratando de "adivinarla" y actualizando esta adivinanza iteración tras iteración. Esto lo hace un algoritmo muy interesante y, además de ya haber mostrado una gran eficiencia en este problema específico, vale la pena estudiarlo desde un punto de vista más general para así intentar aplicarlo a otro problema.

Por otra parte, el problema de planificación minera presenta mucha estructura, y no sólo sus soluciones óptimas, sino que también el grafo que representa el problema PCP^{by}. Esto permitió desarrollar speed-ups al algoritmo BZ que aprovecharan esta estructura para hacer que el algoritmo resuelva problemas más pequeños en cada paso (Implicit Representation, Time Shrink, One Destination Shrink) y junto con otros speed-ups más generales (como todos los Warm Starts y el Preproceso) resultaron aumentar drásticamente la eficiencia de este algoritmo. Si bien no se pudo determinar en general cuál conjunto de speed-ups es mejor, se optó por elegir ALL-T como el conjunto que será utilizado por defecto en cada llamada al algoritmo, debido a su robustez sobre las instancias (no necesita tantos supuestos como ALL-C) y su tendencia a ser más rápido en instancias más grandes. Esto último a pesar de no poder resolver una instancia que si pudo resolver ALL-C, pero afortunadamente al considerar Reset esto se pudo reparar.

Desde un punto de vista aplicado, esto último es uno de los grandes aportes hechos por este trabajo: aprovechar la estructura y propiedades de las instancias que definen el problema de planificación minera para desarrollar un algoritmo eficiente que pueda resolver la relajación lineal de dicho problema, ya sea para obtener una cota superior del problema de programación entera, o para guiar la construcción de una solución factible como se hizo en el Capítulo 6. El algoritmo propuesto originalmente funciona muy bien, pero aprovechando dichas propiedades se puede obtener un mejor resultado.

A pesar de todo, hubo una instancia que no pudo ser resuelta por problemas de memoria. Esto resulta ser muy complicado de resolver, pues la instancia ni siquiera logra realizar una iteración. Quizás una implementación aún más rigurosa del algoritmo, o usando estructuras de datos más elaboradas podrían ayudar resolver este problema.

Otra contribución hecha por este trabajo es la obtención de muy buenas soluciones factibles para el problema de programación entera. Si bien resolver la relajación lineal ya es un problema desafiante y de mucha importancia, obtener soluciones factibles es crucial para la planificación minera, y en este trabajo, adaptando y mejorando una heurística conocida, se pudo obtener resultados satisfactorios en muy poco tiempo: Gaps menores a un 5 % en segundos para instancias de hasta cientos de millones de variables y miles de millones de restricciones está lejos de ser trivial.

Y por último, el otro gran aporte de este trabajo ha sido la aplicación de la generalización algoritmo BZ a un contexto más general, en este caso, proponiendo un algoritmo generalizado a problemas cónicos, y aplicándolo a un nuevo modelo que considere robustez en las soluciones frente a variaciones en los datos. Este es un problema real muy importante debido a las fluctuaciones de los precios de los minerales, por lo cual es necesario contar con soluciones que aseguren un nivel de ingreso mínimo. A pesar de contar con una fuente limitada de instancias, se pudo observar que el algoritmo propuesto funciona muy bien, pudiendo resolver eficientemente un problema de alta complejidad. Cabe mencionar que en este caso Reset tuvo un rol mucho más notorio que antes debido a la rapidez con la crece el tamaño de la partición, lo que lo hizo fundamental al momento de resolver rápidamente las instancias disponibles.

Como continuación a este trabajo, sería interesante tomar este mismo esquema de Optimización Robusta y aplicarlo a un problema tan estudiado como es considerar incertidumbre geológica de la mina, para comparar cómo funcionaría este algoritmo frente a los algoritmos ya propuestos para este problema. Otra continuación importante sería usar esta generalización para abordar otros problemas de optimización a gran escala y/o problemas de gran complejidad como son los problemas cónicos, que posean buenas relajaciones simples de resolver, lo que nos permitiría usar la generalización propuesta.

Apéndice A

El Método de Dantzig-Wolfe y Generación de Columnas

En muchas aplicaciones de Optimización a gran escala las restricciones de los modelos poseen estructuras especiales que pueden ser explotadas. Por ejemplo, sub-sistemas de variables y restricciones independientes que son unidos por otro conjunto de restricciones o variables, o simplemente conjuntos de restricciones "complicadas" que si pudieran ser eliminadas reducirían considerablemente la dificultad del problema.

En este Apéndice se detallarán los métodos de Generación de Columnas y la Descomposición de Dantzig-Wolfe para problemas de programación lineal, técnicas que son usadas para abordar problemas como los mencionados anteriormente. Para una revisión más detallada de estos métodos ver [28].

A.1. Generación de Columnas

Generación de Columnas es uno de los métodos más usados hoy en día para lidiar con problemas que poseen gran cantidad de variables. Este método fue usado por primera vez por Gilmore y Gomory (1961) como parte de un algoritmo para resolver el problema cutting-stock [19].

Sea J un conjunto de variables (columnas). Si se considera el siguiente problema de optimización

$$\begin{aligned} \text{mín} \quad & \sum_{j \in J} c_j x_j \\ \text{s.t.} \quad & \sum_{j \in J} a_j x_j \geq b \\ & x_j \geq 0, \quad j \in J \end{aligned} \tag{A.1}$$

entonces en cada iteración del algoritmo Simplex se busca una variable no-básica con costos reducidos negativos para entrar a la base. En otras palabras, dado el vector de variables duales $u \geq 0$, se desea buscar

$$\arg\min\{\bar{c}_j = c_j - u' a_j \mid j \in J\}. \tag{A.2}$$

Si se tiene que J es muy grande, entonces encontrar una columna que deba entrar a la base se vuelve un problema no-trivial. Además, por los mismos motivos, típicamente se

trabaja con un subconjunto de variables $J' \subseteq J$ en el problema (A.1), de manera resolver un problema *restringido*, y obtener una solución factible $\bar{\lambda}$ con dual $\bar{u}$. Con este último se resuelve el problema (A.2) para determinar si existe una nueva columna que agregar al conjunto J'. Esto es lo que se conoce como *Generación de Columnas*.

Muchas veces las columnas a_j están determinadas implícitamente como elementos de un conjunto $\mathcal{A}$ y los costos c_j pueden ser determinados a partir de a_j, por lo que se puede reemplazar el problema (A.2) por

$$\text{mín}\{c(a) - \bar{u}'a \mid a \in \mathcal{A}\}. \tag{A.3}$$

Este último problema se conoce como el *sub-problema* o el *generador de columnas*, y tiene la ventaja de resolver un problema de Optimización en vez de revisar todas las columnas como en (A.2), lo cual puede ser de gran utilidad si es que el conjunto $\mathcal{A}$ representa elementos con mucha estructura, como objetos combinatoriales por ejemplo.

A.2. Descomposición para problemas de Programación Lineal

El método de Dantzig-Wolfe [10] ha sido ampliamente utilizado en problemas de programación lineal a gran escala que poseen cierta estructura especial. Si se considera un problema de programación lineal

$$\begin{array}{ll} \text{mín} & c'x \\ \text{s.t.} & Ax \geq b \\ & Dx \geq d \end{array} \tag{A.4}$$

y se define $P_A = \{x \in \mathbb{R}^n \mid Ax \geq b\}$ (que por simplicidad se asumirá acotado), entonces se puede definir $\{p_q\}_{q \in Q}$ como el conjunto de puntos extremos de P_A y reescribir cada $x \in P_A$ como

$$x = \sum_{q \in Q} p_q \lambda_q$$

para ciertos multiplicadores λ_q tales que $\sum_{q \in Q} \lambda_q = 1$. Sustituyendo esto en (A.4), y definiendo $D_q = Dp_q$ y $c_q = c'p_q$ se obtiene el problema equivalente

$$\begin{array}{ll} \text{mín} & \sum\limits_{q \in Q} c_q \lambda_q \\ \text{s.t.} & \sum\limits_{q \in Q} D_q \lambda_q \geq d \\ & \sum\limits_{q \in Q} \lambda_q = 1 \end{array} \tag{A.5}$$

Típicamente Q es un conjunto grande, pero la ventaja que tiene el problema (A.5) es que posee menos restricciones que el problema (A.4), donde las restricciones $Ax \geq b$ fueron reemplazadas por $\sum_{q \in Q} \lambda_q = 1$, la que se conoce como *restricción de convexidad*. Además, para lidiar con el tamaño de Q, se puede usar Generación de Columnas, donde cada columna está asociada a punto extremo p_q.

En efecto, si se considera el mismo procedimiento que en la sección anterior, y se obtienen duales $\bar{u}$ y $\bar{v}$ óptimos para el problema (A.5) restringido a un conjunto $Q' \subseteq Q$ (donde $\bar{v}$ es el dual correspondiente a la restricción de convexidad), entonces es fácil ver que para encontrar una columna con costo reducido negativo se debe resolver

$$\text{mín}\{(c' - \bar{u}'D)x - \bar{v} \mid Ax \geq b\}. \tag{A.6}$$

lo cual entregará un nuevo punto extremo de P_A para ser agregado al conjunto de columnas. Este último es un problema de programación lineal, el cual es fácil de resolver si es que se supone que el poliedro P_A es escogido adecuadamente.

En resumen, el método de Dantzig-Wolfe permite resolver un problema de tipo (A.4) usando Generación de Columnas mediante la resolución del problema (A.5) restringido a un conjunto pequeño de columnas Q', para luego generar más columnas con un problema de programación lineal como (A.6) que idealmente será "fácil" de resolver.

Para más interpretaciones y aplicaciones de este método de descomposición ver [10] o [28].

Bibliografía

[1] A. Ben-Tal and A. Nemirovski. *Lectures on Modern Optimization*. 2012.

[2] D. Bertsimas and J. N. Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific and Dynamic Ideas, 1997.

[3] D. Bienstock and M. Zuckerberg. A new algorithm for precedence constrained production scheduling. *Optimization Online*, 2009.

[4] N. Boland, I. Dumitrescu, and G. Froyland. A multistage stochastic programming approach to open pit mine production scheduling with uncertain geology. *Optimization Online*, 2008.

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

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

[7] R. Chicoisne, D. Espinoza, M. Goycoolea, E. Moreno, and E. Rubio. A new algorithm for the open-pit mine scheduling problem. *Operations Research*, 2012.

[8] C. Cullenbine, K. Wood, and A. Newman. Improving the tractability of the open pit mining block sequencing problem using a sliding time window heuristic with lagrangian relaxation. *Optimization Letters*, 88(3):365–377, 2011.

[9] K. Dagdelen and T.B. Johnson. Optimum open pit mine production scheduling by lagrangian parameterization. *19th APCOM Symposium of the society of mining engineers (AIME)*, 1986.

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

[11] R. Dimitrakopoulos. Conditional simulation algorithms for modelling orebody uncertainty in open-pit optimization. *International Journal of Mining, Reclamation and Environment*, 12:173–179, 1998.

[12] R. Dimitrakopoulos, C. Farrelly, and M.C. Godoy. Moving forward from traditional optimisation: Grade uncertainty and risk effects in open pit mine design. *Transactions of the IMM, Section A Mining Industry*, 111:A82–A89, 2002.

[13] R. Dimitrakopoulos, L. Martinez, and S Ramazan. A maximum upside / minimum downside approach to the traditional optimization of open pit mine design. *Journal of Mining Science*, 43:73–82, 2007.

[14] P. Dowd. Risk in minerals projects: analysis, perception and management. *Transactions of the IMM, Section A Mining Industry*, pages A9–A18, 1997.

[15] C. Fricke. *Applications of Integer Programming in Open Pit Mining.* PhD thesis, Department of Mathematics and Statistics, The University of Melbourne, April 2006.

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

[17] M.E. Gershon. Heuristic approaches for mine planning and production scheduling. *Journal of Mining and Geological Engineering*, 5:1–13, 1987.

[18] M.E. Gershon. An open-pit production scheduler: algorithm and implementation. *AIME Transactions*, 282:793–796, 1987.

[19] C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859, 1961.

[20] M.C. Godoy and R. Dimitrakopoulos. Managing risk and waste mining in long-term production scheduling of open-pit mines. *SME Transactions*, 316:43–50, 2004.

[21] D.S. Hochbaum. A new-old algorithm for minimum cut in closure graphs. *Networks*, 37(4):171–193, 2001.

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

[23] W. Hustrulid and K. Kuchta. *Open Pit Mine Planning and Design.* Taylor and Francis, 2006.

[24] W. A. Hustrulid, M.K. Carter, and D.J.A. Van Zyl. *Slope stability in surface mining.* Society for Mining Metallurgy and Exploration, 2000.

[25] T.B. Johnson. *Optimum open pit mine production scheduling.* PhD thesis, Operations Research Department, University of California, Berkeley, May 1968.

[26] T.B. Johnson. Optimum open-pit mine production scheduling. In A. Weiss, editor, *A decade of digital computing in the mining industry*, chapter 4. AIME, New York, 1969.

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

[28] M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

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

[30] J. C. Picard. Maximal closure of a graph and applications to combinatorial problems. *Management Science*, 22:1268–1272, 1976.

[31] S. Ramazan and R. Dimitrakopoulos. Traditional and new mip models for production scheduling with in-situ grade variability. *International Journal of Surface Mining, Reclamation and Environment*, 18:85–98, 2004.

[32] S. Ramazan and R. Dimitrakopoulos. Stochastic optimisation of long-term production scheduling for open pit mines with a new integer programming formulation. *Orebody Modelling and Strategic Mine Planning, The Australasian Institute of Mining and Metallurgy, Spectrum Series*, 14:385–391, 2007.

[33] Gemcom Software. http://www.gemcomsoftware.com.

[34] IBM ILOG CPLEX Optimization Studio. http://www.ibm.com.