PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

UNA GENERALIZACIÓN DE LAS DESIGUALDADES DE *KNAPSACK COVER*

IGNACIO MORALES

Tesis para optar al grado de
Magíster en Ciencias de la Ingeniería

Profesor Supervisor:
JOSÉ VERSCHAE

Santiago de Chile, Junio 2018

PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA

UNA GENERALIZACIÓN DE LAS DESIGUALDADES DE *KNAPSACK COVER*

IGNACIO MORALES

Miembros del Comité:
JOSÉ VERSCHAE
ÁLVARO LORCA
ANDREAS WIESE
GLORIA ARANCIBIA

Tesis para optar al grado de
Magíster en Ciencias de la Ingeniería

Santiago de Chile, Junio 2018

Agradecimientos a Benjamin,
Florencia, mis padres y hermanos
por el apoyo incondicional.

AGRADECIMIENTOS

Quisiera agradecer primero que todo a mis padres Andrés Morales y Rosalía Muñoz, por apoyarme incondicionalmente a través de estos años, mis hermanos Felipe, Camila e Ignacia por ayudarme y escucharme en los momentos en que los necesité y a mi pareja María Florencia Ramos, con la cuál iniciamos nuestra propia familia y me ha apoyado e incentivado a seguir siempre adelante durante esta larga trayectoria.

Además, quiero agradecer a mi profesor supervisor José Verschae por las incontables reuniones y charlas que tuvimos tanto para el apoyo académico como en lo personal. Le agradezco la paciencia y perseverancia durante todo este periodo de investigación y cuando fui su alumno y ayudante.

Por otro lado, quisiera agradecer a la Pontificia Universidad Católica de Chile y todos los profesores que tuve durante mi pre y postgrado, por todos los conocimientos que me han otorgado y las experiencias que quedarán siempre en mi memoria.

No puedo dejar de mencionar a mis compañeros de clases y funcionarios de la universidad que me ayudaron durante el camino que realice. Un agradecimiento especial a Natalia Ahumada, Constanza Toro e Ise de Smet.

Finalmente, quiero agradecer por el financiamiento recibido por la Dirección de Postgrado UC, el Proyecto FONDECYT Nr. 11140579, el Proyecto PCI PII 20150140 de Conicyt, y el Ministerio de Educación gracias a la Beca Bicentenario del 2011.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ABSTRACT

Covering problems appear naturally in discrete optimization. In its most basic version, we must choose a set of items of minimum cost that covers some given demand. This is the classic *Minimum Knapsack* or *Knapsack Covering* problem, and it is NP-hard. Unlike the packing version, the *Knapsack* problem, its natural LP-relaxation has an unbounded integrality gap. Carr et al. (2000) devised a set of inequalities, the *knapsack-cover inequalities*, to strenghten this relaxation and reduce the integrality gap to 2.

In this thesis we consider a natural setting where items can be taken fractionally but the cost is non-linear; which is a generalization of the usual *Minimum Knapsack*. We extend the *knapsack-cover inequalities* to the case with piecewise linear cost functions. We show that the relaxation has an integrality gap of 2 and a primal-dual algorithm with the same guarantee. By approximating general non-decreasing and left-continuous cost functions by piecewise linear functions, we can obtain a prima-dual $(2+\varepsilon)$-approximation algorithm for this general setting. Finally, we test our algorithm computationally using data based on the power generation problem with a single demand. In this setting we are given a power demand and a set of power plants with non-linear and discontinous cost functions which we modeled after thermoelectrical power plants. We show that our primal-dual algorithm give average errors of at most 4 % when compared with the obtained dual solution. Moreover, we observe that for larger demand values the gap further decreases. We leave as an open problem whether our inequalities can be used for more general problems, as the Unsplittable Flow-cover Problem on a path with non-linear costs which, in some sense, adds a temporal dimension to the problem. This might open up the possibility to consider dynamic power generation problems.

Keywords: Minimum Knapsack, Knapsack Cover, Knapsack Cover Inequalities, Non-linear cost, Cover Problem, Primal-dual Algorithm.

RESUMEN

Los problemas de cubrimiento aparecen naturalmente en optimización discreta. En su versión más básica, debemos elegir un conjunto de artículos de costo mínimo que cubra una determinada demanda. Este es el clásico problema de la mochila en su versión de cubrimiento o *Minimum Knapsack*, que es NP-hard. A diferencia de la versión original, la relajación lineal natural de esta versión tiene un gap de integralidad no acotado. Carr et al. (2000) idearon un conjunto de desigualdades, las *knapsack-cover inequalities*, para fortalecer esta relajación y reducir el gap a 2.

En esta tesis, consideramos un contexto natural donde cada elemento se puede tomar de manera fraccionada, pero con costo no lineal, lo que es una generalización del problema anterior. Extendemos las *knapsack-cover inequalities* para el caso de funciones de costos lineales por partes, mostramos que su relajación y un algoritmo primal-dual mantienen el gap de integralidad en 2. Al aproximar funciones de costos generales no decrecientes y continua por la izquierda por funciones lineales por partes, podemos obtener un algoritmo primal-dual $(2+\varepsilon)$-aproximado para estas funciones generales. Finalmente, probamos nuestro algoritmo computacionalmente usando datos basados en el problema de generación de energía con demanda simple, donde se tiene una demanda eléctrica y un conjunto de centrales de producción con funciones de costo no lineales y discontinuas. Mostramos que nuestro algoritmo primal-dual alcanza errores promedio por debajo del $4\,\%$ al compararse con la solución dual obtenida. Además, observamos que disminuye el error al aumentar la demanda. Una pregunta abierta es si nuestras desigualdades se pueden utilizar para problemas más generales, como el *Unsplittable Flow-cover Problem on a path* con costos no lineales que, en cierto sentido, agrega una dimensión temporal al problema. Esto abriría la posibilidad de considerar problemas de generación de energía dinámicos.

Palabras Claves: Problema de cubrimiento de la mochila, *Minimum Knapsack*, costos no lineales, problema de cubrimiento, algoritmo primal-dual.

CAPÍTULO 1. INTRODUCCIÓN

Comencemos pensando que somos una tienda que vende harina, que sabemos que para mañana tenemos un pedido de 100 kilogramos [Kg] y que para cubrir el pedido, tenemos tres productores de harina. El primero entrega sacos de 5 Kg a \$110, el segundo sacos de 10 Kg a \$200 y el tercero sacos de 8 Kg a \$150. Además, cada productor nos dice que puede entregar a lo más 18, 9 y 11 sacos respectivamente. Como dueños de la tienda, debemos decidir a quién o quiénes comprarles y cuantos sacos a cada uno. Podríamos partir pensando en ordenarlos de tal manera que compremos primero a la persona donde el Kg sea el más barato y continuar. Como el kilogramo nos cuesta \$21, \$20 y \$18,75 respectivamente, entonces usando esta lógica compraríamos los 11 sacos de 8 Kg y 2 sacos de 10 Kg, lo que nos daría un costo de $\$150 \times 11 + \$200 \times 2 = \$2050$, pero obviamente, podemos cambiar un saco de 10 kg por uno de 5 kg y el costo nos quedaría de \$1960. En este ejemplo pudimos calcular fácilmente el resultado óptimo, pero si escalamos esto a una demanda mayor y una cantidad más grande de productores, el problema deja de ser tan sencillo. Este problema general es llamado *Minimum Knapsack* (MK), que consiste en una demanda fija y conocida D, n posibles elementos en un conjunto E, donde cada elemento $i \in E$ cubre una cantidad de u_i unidades de demanda pagando un monto c_i, y el objetivo es encontrar un subconjunto de elementos S que cubra la demanda minimizando los costos dados por $\sum_{i \in S} c_i$.

Si ahora en vez de comprar el harina, decidimos producirla nosotros, pero para eso tenemos que ver qué tecnología utilizamos para generarla, tendríamos el siguiente problema. Al preguntar en las fábricas de harina, descubrimos que podemos fabricar un molino que cuesta \$1000 y que produce harina a un costo de \$10 el Kg, comprar un molino usado a \$700, donde nos regalan 50 Kg de harina lista y produce a un costo de \$12 el Kg o crear una fábrica de harina con una nueva tecnología que cuesta \$2000 y produce harina a un costo de \$5 pero que al día solo puede producir 30 Kg. Con esta información, el problema es mucho más complejo, lo que escapa del modelo entregado por MK. Es más, el resolver este problema, o un caso más genérico de éste, con más opciones para decidir como

producir, presenta problemas computacionales, ya que cuenta con una cantidad muy grande de posibles soluciones. De hecho en la actualidad aún no se conoce ningún algoritmo eficiente para resolverlo encontrando la mejor solución.

El problema recién descrito cae en la clase de problemas NP-difíciles, los cuales si admiten un algoritmo eficiente (i.e., con tiempo de ejecución polinomial en el tamaño de la entrada), entonces la famosa conjetura de P$\neq$NP sería falsa. Como muchos problemas de optimización NP-difíciles son utilizados diariamente en la academia, gobiernos e industrias de todo el mundo, fue necesario idear nuevas técnicas para poder abordarlos. Es así que nació la idea de resolver los problemas de manera aproximada, pero a su vez teniendo una certeza de qué tan lejos estamos de la solución óptima. De esta manera se crea el concepto de algoritmos de aproximación, en donde se define que una solución es α-aproximada si su valor es a lo más α veces el valor de la solución óptima (para problemas de minimización). Una técnica importante para desarrollar estos algoritmos son los algoritmos primales-duales. Éstos consisten en definir una relajación lineal del problema y usar información del dual de la relajación para poder construir una buena solución del problema primal, sin necesidad de resolverlo.

Los algoritmos de tipo primal-dual fueron desarrollados en primera instancia por Dantzig et al. (1956) como un método para resolver problemas lineales. Luego, Kuhn se basó en este método para crear el "Método Húngaro" primal-dual para resolver el problema de asignación (Kuhn, 1955), problema fundamental en optimización combinatorial que consta en encontrar el emparejamiento perfecto de peso mínimo en un grafo con costo en sus arcos. Aunque el método creado originalmente no sobrevivió como un método para programación lineal en el tiempo, ideas similares se continuan utilizando para encontrar algoritmos aproximados para problemas en optimización combinatorial. La idea principal desarrollada por estos métodos se basa en ir construyendo paralelamente una solución primal y una dual al mismo tiempo. Para problemas NP-completos, asumiendo que P no es igual a NP, los algoritmos primal-dual que corren en tiempo polinomial no lograrían encontrar siempre soluciones que cumplan todas las ecuaciones de holgura complementaria,

ya que en el caso de cumplirlas, podríamos asegurar que estamos en presencia del óptimo en un tiempo polinomial. La idea principal de los algoritmos de aproximación que utilizan este método, es el relajar las ecuaciones de holgura complementaria del primal, ya que solo nos aseguramos de que cuando la restricción del dual se cumple con igualdad, entonces le podremos dar un valor a la variable primal asociada. La idea de relajar las ecuaciones es que no se exige que cuando una variable dual tenga valor distinto de 0, entonces la restricción primal se cumpla con igualdad.

Es muy dificil decidir cual fue el primer algoritmo primal-dual que efectivamante entra dentro de este marco (Goemans y Williamson, 1997), pero es ampliamente aceptado que uno de los primeros algoritmos fueron los de Bar-Yehuda y Even (1981) y Chvatal (1979) para los problemas de cubrimiento de vértice con pesos y el problema de cubrimiento de conjuntos, respectivamente. Luego, otros investigadores han utilizado estas técnicas para crear algoritmos de aproximación para otros problemas en optimización combinatorial. Por ejemplo, Agrawal et al. (1995) desarrollan un algoritmo $2\lceil \log_2(r+1) \rceil$-aproximado para *Generalized Steiner Problem on Networks*, donde r es el requerimiento más alto de los arcos del grafo, Goemans y Williamson (1995) generan algoritmos de aproximación que corren en tiempo $O(n^2 \log n)$ y que alcanzan en su mayoría una 2-aproximación para una gran clase de problemas sobre grafos; Bertsimas y Teo (1998) dan un algoritmo primal-dual para un conjunto de problemas de cubrimiento general, introducen nociones de formulaciones reducibles y fuerza de las desigualdades válidas que ayudan en la creación de algoritmos primal-dual y su análisis; y Levi, Roundy, y Shmoys (2006) desarrollan un tipo de algoritmo primal-dual para problemas determinísticos de inventarios que alcanza una 2-aproximación para el *Joint Replenishment Problem* y para el *Assembly Problem*, y logra resolver el *Single-Item Lot-Sizing Problem* a optimalidad.

Hoy en día es muy grande la cantidad de problemas a los cuales se les ha encontrado mejores soluciones y mejores tiempos de resolución utilizando las técnicas de algoritmos primales-duales. El avance de estas técnicas se debe en gran parte al trabajo que se ha invertido en mejorar las relajaciones lineales de problemas enteros o enteros mixtos de

optimización combinatorial al encontrar nuevas restricciones que reducen los poliedros que aparecen al relajar los problemas con variables integrales.

Mas aún, gran parte de las mejoras en relajaciones lineales viene dado por el descubrimiento de nuevas desigualdades válidas, es decir, restricciones que son cumplidas por los puntos integrales o soluciones del problema original y que reducen el poliedro de la relajación. Un ejemplo de esto son las desigualdades llamadas *knapsack cover inequalities* (KCI), las cuales fueron creadas por Carr et al. (2000) para poder alcanzar una mejor relajación en problemas de cubrimiento con capacidad simple, como lo es el problema de *Minimum Knapsack* que mencionamos en un principio.

Las KCI han sido utilizadas en otros problemas para mejorar las relajaciones de sus modelos integrales. La mayoría de estos problemas utilizan restricciones de cubrimiento de la forma $\sum_i a_i x_i \geq D$ en sus modelos, donde a es un vector de constantes y x un vector de decisiones binarias. También han sido utilizadas o extendidas para poder cubrir problemas de cubrimiento más generales. En esta línea, Carnes y Shmoys (2015) dan un generalizaciones de las KCI para el problema de *Facility Location* y *Single-Item Lot-Sizing with Linear Holding Costs*, los cuales son generalizaciones del problema *Minimum Knapsack* y veremos en detalle los dos primeros en el Capítulo 3 de Casos Particulares. En el mismo artículo, los autores presentan algoritmos de tipo primal-dual que alcanzan una 2-aproximación para los tres problemas mencionados anteriormente. En el 2017, McCormick et al. desarrollan otra generalización de las KCI para una clase de problemas que llaman *Precedence Constrained Covering Problems* (PCCP), el cual también generaliza el problema de MK al agregarle restricciones de precedencia. El principal resultado es que entregan un algoritmo primal-dual fuertemente-polinomial para PCCP con factor de aproximación igual al *ancho* de las restricciones de precedencia (i.e., el largo de la anticadena más larga).

Otra generalización de MK fue desarrollada por Takazawa y Mizuno la cual llaman *Minimum Knapsack Problem with a Forcing Graph* (Takazawa y Mizuno, 2017), ésta agrega

restricciones que dan la flexibilidad de que, para algunos pares de elementos, se deba agregar al menos uno de ellos. El problema es modelado con un grafo, donde los items son vertices y un conjunto de arcos E donde para cada arco $(i, j) \in E \subseteq \{1, .., n\} \times \{1, .., n\}$ se tiene la restricción de que la suma de las variables relacionada al elemento i y j debe ser mayor o igual a 1. Los autores extienden el algoritmo primal-dual de Carnes y Shmoys (2015) manteniendo la constante de aproximación. Además, extienden nuevamente su algoritmo para alcanzar una Δ_2-aproximación, donde Δ_2 es el segundo número más grande de coeficientes distintos de cero en las restricciones. Las KCI han sido utilizadas también para mejorar las aproximaciones de problemas de calendarización como el *General Scheduling problem* con (Bansal y Pruhs, 2014) y sin (Cheung et al., 2017) tiempos de iniciación para cada trabajo y el *unsplittable flow-cover problem on a path* (Bar-Noy et al., 2001; Höhn et al., 2014). El *General Scheduling problem* contemplado en ambos resultados es anotado como $1||\Sigma f_j$ consta de n trabajos a ser calendarizados en una sola máquina, donde cada uno consta de un tiempo de procesamiento, un tiempo de liberación o iniciación (tiempo en el que puede partir el procesamiento de un trabajo, que puede ser 0) y una función de costos no decreciente, que indica el costo incurrido de terminar el trabajo i en un tiempo determinado. El objetivo es minimizar los costos de procesar todos los trabajos en la máquina que puede procesar un trabajo a la vez.

Por otro lado, a finales del 2016 Li (2017) considera otra generalización de las KCI para abordar el problema *Non-Uniform Capacitated Multi-Item Lot Sizing*, el cual consta de un horizonte de tiempo finito T y discreto, un conjunto de n tipos de elementos, donde cada uno tiene una demanda de $d_i > 0$ y que deben ser cubiertas en el tiempo r_i. En cada momento s se puede generar una cantidad de elementos que debe ser menor a una constante C_s incurriendo en un costo de K_s, esta orden puede ser del tamaño deseado y de cualquier tipo. Además, llevar inventario entre periodos tiene un costo de mantenimiento distinto por cada tipo de elemento, es decir, para cada elemento existe una función que nos dice cuanto cuesta llevar una unidad de un periodo s a r_i. Estas funciones de costos de mantención son no crecientes en s. El objetivo de este problema es de minimizar la suma de los costos de producción y de mantención cubriendo las demandas de cada elemento.

Cabe destacar que un tipo de elemento puede tener demandas en varios periodos, ya que esto se puede ver como dos elementos distintos que tienen una sola demanda. Para este problema, Li (2017) utiliza su generalización de las KCI y utiliza técnicas de redondeo para obtener un algoritmo 10-aproximado.

También existen aplicaciones a problemas más concretos, donde por ejemplo se puede modelar mecanismos para que los autos eléctricos se conecten a una red inteligente que les permita vender electricidad a los autos cuando tengan energía de sobra. En este caso, la red tendría entes llamados "agregadores" que deben decidir cómo cubrir la demanda de la red teniendo distintas fuentes de energía. Para este problema que debe resolver cada ente, se desarrolló un algoritmo primal-dual que alcanza una 2-aproximación (Zhong et al., 2017) basándose en las desigualdades de *knapsack cover* (Carr et al., 2000) y en el algoritmo de Carnes y Shmoys (2015).

Otra aplicación que tiene restricciones de cubrimiento y que es de interés para esta investigación es el problema de generación de energía eléctrica, donde un planificador central debe elegir qué planta utilizar para generar energía según la demanda que tendrá en el futuro. Como cada planta puede extraer energía utilizando una tecnología o una combinación de estas, las funciones de costo (de producción) de las plantas son muy distintas entre sí. Es por esto que la decisión de cuánto producir por cada planta no es sencilla. Una de las principales dificultades es que estas funciones de costo generalmente tienen una naturaleza no lineal y con discontinuidades. Por ejemplo, el costo inicial de encender una planta, que generalmente es elevado, se ve reflejado en su función de costos como una discontinuidad en el punto $x = 0$.

Para poder abordar este tipo de problema, partiremos mostrando problemas similares, pero más sencillos, que ya han sido resueltos en la literatura. Luego, seguiremos trabajando en la misma línea, para llegar a una simplificación del problema expuesto en el párrafo anterior. En ésta se consideran funciones de costo lineales por tramos y continuas, pero no necesariamente convexas. Luego mostraremos como aproximar funciones más generales

que mantengan las dificultades del problema antes mencionado, como las discontinuidades.

En resumen, la hipótesis de esta tesis es que existe un conjunto de desigualdades válidas para el problema *Minimum Knapsack* con funciones de costos no lineales que generaliza las *knapsack cover inequalities* manteniendo el gap de integralidad. Con esta hipótesis en mente, nuestros objetivos son:

- Encontrar una generalización de las KCI que mantengan el mismo gap de integralidad.
- Encontrar un algoritmo eficiente que nos entregue una solución aproximada del problema principal.

A continuación explicaremos en detalle la organización de la presente tesis de magister.

1.1. Organización de la Tesis

Comenzamos en el Capítulo 2 explicando algunas definiciones y la nomenclatura utilizada a los largo de toda la tesis, de manera de poder comprender mejor el resto de los capítulos. Luego, en el Capítulo 3 vemos en detalle como se han abordado algunas generalizaciones de MK en la literatura, las técnicas utilizadas y como mantienen el gap de integralidad.

Para lograr nuestro objetivo, en el Capítulo 4 abordamos un problema intermedio, donde las funciones de costos son continuas y lineales por tramo. Para este caso particular, proponemos un conjunto de desigualdades validas que generalizan las KCI y que mantienen el gap de integralidad de 2. Además, mostramos un algoritmo primal-dual que entrega siempre una solución 2-aproximada. En el Capítulo 5, utilizamos las desigualdades

y el algoritmo del Capítulo 4 para construir un algoritmo que entrega una $(2+\varepsilon)$ aproximación para una generalización de MK donde cada elemento tiene una función de costo no-decreciente, continua por la izquierda y con una cantidad finita de discontinuidades.

Finalmente, concluiremos con resultados experimentales que el algoritmo tiene un desempeño promedio mucho mejor que la cota teórica del peor caso. Veremos en el Capítulo 6 que en promedio, nuestro algoritmo, no supera un error de un $9{,}2\,\%$ dado por un error de un $4\,\%$ en el gap de integralidad y un $5\,\%$ de error al aproximar las funciones de costo cuadráticas que modelan plantas térmicas de producción eléctrica. Para simular estas plantas se tomó una instancia del Sistema Interconectado del Norte Grande que consta de plantas grandes, medianas y pequeñas. Además, para los resultados, nos pondremos en distintos escenarios de costos fijos para ver como cambia el rendimiento del algoritmo y ver en que escenarios tiene un potencial de uso más alto.

CAPÍTULO 2. DEFINICIONES PREVIAS

En esta sección explicaremos algunas definiciones utilizadas en esta tesis, así como la nomenclatura usada. Este capítulo tiene como propósito el dar un apoyo para comprender de mejor manera el resto de los capítulos.

a) **Problema de decisión:** Sea $\mathcal{L}$ un lenguaje y ϕ una frase. Un problema de decisión consiste en determinar si la frase ϕ pertenece al lenguaje $\mathcal{L}$. Por ejemplo, para el problema de decisión "¿es ϕ un numero primo?", el lenguaje serían los números primos $\mathcal{L} = \{1, 2, 3, 5, 7, \dots\}$ y la pregunta es ¿$\phi \in \mathcal{L}$?

b) **Tamaño de la entrada:** Se define el tamaño de la entrada de un algoritmo como la cantidad de bits necesaria para poder representar la entrada en un computador para un modelo en particular. Cabe destacar que una entrada puede tener distintos tamaños dependiendo de la representación en binario elegida.

c) **Cotas asintóticas:** En análisis de algoritmos existen tres notaciones para comparar funciones cuando el argumento tiende a infinito. Dada funciones f y g, se denota la relación de g con los conjuntos $O(f(n))$, $\omega(f(n))$ y $\Theta(f(n))$ de la siguiente manera:
   - $g(n)$ está en $O(f(n))$ si y solo si existen constantes $c > 0$ y $n_0 > 0$ tal que $g(n) \leq cf(n)$ para todo $n > n_0$
   - $g(n)$ está en $\omega(f(n))$ si y solo si existen constantes $c > 0$ y $n_0 > 0$ tal que $g(n) \geq cf(n)$ para todo $n > n_0$
   - $g(n)$ está en $\Theta(f(n))$ si y solo si $g(n) \in O(f(n))$ y $g(n) \in \omega(f(n))$

   Por ejemplo, la función $\frac{24n^3}{2n}$ está en $\Theta(n^2)$. las constantes no son relevantes en las cotas asintóticas.

   Es importante destacar que se pueden establecer una jerarquía en la cota superior asintótica. Por ejemplo, n está en $O(n)$ y en $O(n^2)$, pero n^2 no está en $O(n)$.

d) **Tiempo de ejecución:** Para una entrada, el tiempo de ejecución es la cantidad de pasos u operaciones básicas que debe hacer el algoritmo para entregar una solución. Hablaremos de tiempo de ejecución del algoritmo como el mayor tiempo

de ejecución de las posibles entradas que pueden ingresar en él. Normalmente, el tiempo de ejecución exacto es muy difícil de calcular, es por esto que se habla en términos de cotas asintóticas.

e) **Conjunto NP:** Se define NP como el conjunto de problemas de decisión para los cuales existe un certificado de tamaño polinomial en la entrada del problema para cada instancia donde la respuesta es "si", con el cual se puede verificar que la respuesta es "si" en tiempo polinomial. Podemos ver que los problemas de decisión que tienen un algoritmo polinomial para responder si o no, pertenecen a NP, ya que el mismo algoritmo sirve como certificado.

f) **Reducción:** En análisis de algoritmos una reducción es la transformación de un problema de decisión a otro de manera consistente. Por ejemplo, Si tengo un problema P y una instancia cualquiera de P φ, el problema de decisión es ver si $\varphi \in P$. El problema H puede reducirse a otro problema P' si y solo si existe una función f tal que para toda instancia φ se cumple que $\varphi \in P$ si y solo si $f(\varphi) \in P'$.

g) **Problemas NP-difíciles:** Se define como NP-*difícil* o NP-*hard* al conjunto de todos los problemas P tal que todo problema P' en NP puede ser reducido en tiempo polinomial a P. Es importante destacar que existen problemas que no están en NP por que tienen una dificultad mayor y que son NP-duros. -

h) **Problemas NP-completo:** Se definen como los problemas NP-duros que se encuentran en NP. Es decir, son los problemas más difíciles de resolver dentro en NP y si es posible resolver uno de estos en tiempo polinomial, entonces se podrían resolver todos los problemas en NP en tiempo polinomial.

i) **Problema de optimización:** Dada un función f y un conjunto X, un problema de optimización es escoger un $x \in X$ tal que minimice (o maximice) $f(x)$, a este x en particular se le llama punto óptimo. Cabe destacar que un problema de optimización se relaciona con múltiples problemas de decisión de la forma, ¿Existe un $x \in X$ tal que $f(x) \leq k$? Por un lado, si se resuelven los problemas de decisión para todos los k, entonces se puede encontrar el óptimo del problema de

optimización. Por otro lado, si se resuelve el problema de optimización, entonces se conocerán las respuestas para todos los problemas de decisión asociados.

Por esta relación entre los algoritmos de optimización y de decisión, podemos extender las definiciones mencionadas anteriormente sobre los algoritmos de optimización.

j) **Algoritmo de aproximación:** Es un algoritmo usado para encontrar soluciones aproximadas de un problema de optimización. Están a menudo asociados con problemas NP-*hard*. Como se cree que es poco probable que alguna vez se descubran algoritmos eficientes, de tiempo polinomial, que resuelvan exactamente problemas NP-*hard*, se opta por encontrar soluciones no-óptimas en tiempo polinomial. Lo que se busca con estos algoritmos es encontrar soluciones para las que está demostrado que son de calidad y cuyos tiempos de ejecución sea polinomial. Una forma de medir la calidad de un algoritmo es midiendo que tan alejado está el costo que entrega en el peor de los casos de la solución óptima, esta noción de alejamiento se define por el factor de aproximación.

k) **Factor de aproximación:** Dado un problema de optimización (minimización) P, la solución óptima x^*, su valor óptimo $f(x^*)$ y $\alpha > 1$. Se dice que una solución $\overline{x}$ es una α-aproximación de P si $f(\overline{x}) \leq \alpha f(x^*)$. Se dice que un algoritmo es α-aproximado para P si para cualquier instancia de P, el algoritmo entrega una salida $\overline{x}$ que es una α-aproximación. La definición es análoga para problemas de maximización con $f(\overline{x}) \geq \alpha f(x^*)$ y $\alpha < 1$.

l) **PTAS:** Un *Polynomial-Time Approximation Scheme* (PTAS) es un conjunto de algoritmos $\{A_\varepsilon\}_{\varepsilon>0}$ tales que para todo $\varepsilon > 0$ el algoritmo A_ε es un algoritmo $(1+\varepsilon)$-aproximado que corre en tiempo polinomial según el tamaño de la entrada del problema original. Esto quiere decir que podría ser no polinomial en ε.

m) **Relajación de un problema de optimización:** Sea P un problema de optimización y X el conjunto factible. Un problema P' con conjunto factible X' es una relajación de P si tiene la misma función objetivo que P y sus espacios factibles

cumplen que $X' \supseteq X$. Por ejemplo, en el caso de tener un problema con variables integrales, el mismo problema sin la restricción de integralidad se le llama relajación lineal.

n) **Gap de integralidad:** Considera un problema de optimización entera (minimización) P y sea P' el problema obtenido al relajar la integralidad de sus variables. Se define Gap de integralidad como la razón entre el valor del óptimo del problema relajado P' y el valor de la solución óptima del problema integral P. El Gap se calcula como

$$\text{Gap} = \frac{\text{OPT}_{P'}}{\text{OPT}_P},$$

donde OPT_P es el valor óptimo del problema P.

CAPÍTULO 3. ANÁLISIS DE CASOS PARTICULARES

En esta sección revisaremos en detalle las técnicas utilizadas por Tim Carnes y David Shmoys en su artículo *Primal-dual schema for capacitated covering problems* (Carnes y Shmoys, 2015), en donde abordan tres problemas de cubrimiento fundamentales en el área de optimización. Nosotros explicaremos las técnicas utilizadas para crear un algoritmo primal-dual de dos de ellos, el problema de *Knapsack Cover* y *Single-Demand Facility Location*, ya que esto nos ayudará a comprender la construcción de nuestro algoritmo en el Capítulo siguiente, en donde abordaremos una generalización de los problemas recién mencionados.

3.1. Problema de Minimum Knapsack (MK)

El problema de *Minimum Knapsack* o problema de cubrimiento de la mochila, también llamado en ingles *Knapsack Cover* o *Dual Knapsack Problem*, consiste en una capacidad (o demanda) fija $D \geq 0$ y un set de n elementos E, donde cada elemento $i \in E$ tiene un valor de cubrimiento u_i y un costo fijo c_i. El objetivo es encontrar un subconjunto $S \subseteq E$ tal que cubra la demanda, es decir, que $\sum_{i \in S} u_i \geq D$ y que minimice el costo total, siendo este definido como $\sum_{i \in S} c_i$.

La forma natural de modelar este problema con programación entera es

$$
\begin{aligned}
[\text{IP}_1] \, \text{mín} \sum_{i \in E} y_i c_i \\
s.a. \sum_{i \in E} u_i y_i \geq D, \\
y \in \{0,1\}^n,
\end{aligned}
$$

siendo y_i una variable de decisión binaria que indica si se usa o no el elemento i.

Como se sabe desde 1972, el problema MK es NP-duro (Karp, 1972) y es por esto que el problema se ha abordado de diferentes maneras para intentar solucionarlo o dar

soluciones razonables. Para efectos de esta tesis, nos enfocaremos en las *Knapsack Covers inequalities* (KCI) creado por Carr. et al (Carr et al., 2000).

3.1.1. *Knapsack Cover Inequalities*

Las KCI nacen de la necesidad de mejorar el poliedro que se genera al relajar $[\text{IP}_1]$. Este poliedro tiene un gap de integralidad mayor o igual a la constante D, es decir, podría ser arbitrariamente alto. Para observar esto, basta con ver el siguiente ejemplo, donde $E = \{1, 2\}$, $u_1 = D - 1$, $c_1 = 0$,$u_2 = D$ y $c_2 = 1$. En cualquier solución integral factible, el elemento 2 tendrá que ser utilizado, por lo tanto el óptimo de $[\text{IP}_1]$ es 1, y al ver la relajación lineal, es posible tomar $y_1 = 1$ e $y_2 = 1/D$ lo que nos da un costo de $1/D$, por lo tanto, el gap es al menos $\frac{1}{1/D} = D$.

Para corregir esto, Carr et al. (2000) crearon un conjunto de desigualdades válidas que mejoran el poliedro generado al relajar el problema. Las KCI se basan en la idea de preguntarse ¿Cuánto más se debe cubrir si es que se fija un subconjunto de elementos $A \subseteq E$ en la solución final? En este caso, lo que queda por cubrir es lo que se denomina **demanda residual** definida por $D(A) = \text{máx}\{0, D - \sum_{i \in A} u_i\}$ o solamente $D(A) = D - \sum_{i \in A} u_i$ si nos restringimos a tomar conjuntos A tales que $\sum_{i \in A} u_i < D$.

Además, esta demanda debe ser cubierta por los elementos restantes, es decir con los elementos en $E \setminus A$. Es fácil ver que lo que queda no es más que otro problema de MK con demanda $D' = D(A)$ y con un conjunto de elementos $E' = E \setminus A$. Mas aún, como en cualquier instancia de MK, si es que existe algún elemento que tenga un valor de cubrimiento mayor a la demanda, este puede ser truncado sin afectar ninguna solución entera factible. Es por esto que se define $u_i(A) = \text{mín}\{u_i, D(A)\}$ como el **cubrimiento truncado** de cada elemento. Con todo esto en mente, podemos hacer la siguiente definición:

Definición 3.1. *Se definen las* ***Knapsack Cover inequalities*** *(Carr et al., 2000) como el conjunto de desigualdades válidas*

$$\sum_{i \in E \setminus A} u_i(A) y_i \geq D(A), \qquad \forall A \subseteq E.$$

Observación 3.1. *Podemos ver que en el ejemplo anterior, este set de desigualdades nos da un gap de integralidad de 1, ya que la solución relajada debe cumplir la siguiente restricción*

$$u_2(\{1\}) y_2 = min\{D-1, 1\} y_2 \geq D(\{1\}) = 1,$$

lo que para $D > 1$ significa que $y_2 \geq 1$, lo que nos da $y_2 = 1$ para la relajación lineal.

Lema 3.1. *Las KC-inequalities son desigualdades válidas.*

DEMOSTRACIÓN. Podemos ver que las KCI son desigualdades válidas con los siguientes dos pasos:

1. 1^o Toda solución factible y de $[\text{IP}_1]$, cumple la desigualdad $\sum_{i \in E \setminus A} y_i u_i \geq D(A)$ para todo $A \subseteq E$.
2. 2^o Podemos ver la desigualdad anterior para cada conjunto A como un problema de *Minimum Knapsack* con los objetos en $E \setminus A$ y como en cualquier problema de estos, podemos truncar el valor de cubrimiento sin afectar las soluciones factibles.

Para poder ver el primer punto, tomemos y como una solución factible de $[\text{IP}_1]$, S como el conjunto de elementos escogidos en esa solución, es decir $\sum_{i \in E} u_i y_i = \sum_{i \in S} u_i$ y un conjunto $A \subseteq E$ con $\sum_{i \in A} u_i < D$, como sabemos por la factibilidad de y que

$$\sum_{i \in E} u_i y_i \geq D,$$

entonces podemos reescribir esto como

$$\sum_{i \in E} u_i y_i = \sum_{i \in S} u_i = \sum_{i \in S \setminus A} u_i y_i + \sum_{i \in S \cap A} u_i \geq D$$

$$\sum_{i \in S \setminus A} u_i y_i \geq D - \sum_{i \in S \cap A} u_i$$
$$\sum_{i \in S \setminus A} u_i y_i \geq D - \sum_{i \in A} u_i > 0$$
$$\sum_{i \in S \setminus A} u_i y_i \geq D(A)$$
$$\sum_{i \in E \setminus A} u_i y_i \geq D(A).$$

Lo que nos muestra que esta desigualdad es factible para cualquier solución factible y para cualquier conjunto A que no supere D. En los casos en que lo superen, basta con ver que $\sum_{i \in S \setminus A} u_i y_i \geq 0 = D(A)$. Finalmente, observando que este es un problema de MK con objetos $E' = E \setminus A$ y demanda $D' = D(A)$, podemos truncar cada u_i, lo que nos da que las KC-*inequalities* son desigualdades válidas. □

3.1.2. Relajación Lineal

Ahora, con estas restricciones podemos generar un modelo que es un relajación de [IP$_1$] y que tiene un gap que veremos más adelante que esta acotado por 2. Este modelo lo llamaremos [LP$_1$] y se define por

$$\begin{aligned} [\text{LP}_1] \min & \sum_{i \in E} x_i c_i \\ s.a. & \sum_{i \in E \setminus A} x_i u_i(A) \geq D(A) \qquad \forall A \subseteq E \\ & x \geq 0, \end{aligned}$$

donde x es una variable en $\mathbb{R}^n_+$ y que en cada componente define la fracción del elemento que utiliza la solución. Cabe destacar que no es necesario agregar que cada componente de x debe ser menor a 1, ya que debe existir al menos un óptimo donde cada componente es menor a 1. En caso contrario, si en el óptimo existe una componente que es mayor a 1 y como se debe cumplir que se cubra la demanda residual para cada $A \subseteq E$, en particular para el A igual a esa componente sola, entonces el resto de variables cubren la demanda

residual usando esa componente de x como 1, eso significa que si la disminuimos a 1, tenemos un punto factible y de costo menor o igual al anterior, lo que nos da que no era óptimo.

3.1.3. Algoritmo Primal-Dual

Un problema de la relajación $[\text{LP}_1]$ es que tiene una cantidad exponencial de restricciones. Sin embargo, la mayoría son redundantes y para definir un vértice solo se necesitan $n + 1$ restricciones. Con esta idea, Carnes y Shmoys (2015) crearon un algoritmo primal-dual para poder aprovechar esta estructura. En esta sección explicaremos como es el algoritmo, y por qué el algoritmo es una 2 aproximación. Estos autores no se quedaron solo con este problema, ellos proponen una generalización llamado *Single-Demand Facility Location*, para el cual proponen un conjunto de desigualdades y dan un algoritmo que mantiene la constante de aproximación. En esta tesis explicaremos de una manera distinta la construcción de estas desigualdades, para poder dar la idea principal que nos ayudará a generalizarlas aún más y lograr el objetivo de abordar funciones de costos no-lineales y discontinuas.

Volviendo al problema de *Minimum Knapsack*, podemos tomar el modelo $[\text{LP}_1]$ y calcular el dual

$$[\text{DP}_1] \text{ máx} \sum_{A \subseteq E} D(A) v_A \tag{3.1}$$

$$s.a. \sum_{A \subseteq E : i \notin A} u_i(A) v_A \leq c_i \qquad \forall i \in E, \tag{3.2}$$

$$v_A \geq 0 \qquad \forall A \subseteq E. \tag{3.3}$$

Podemos ver que la cantidad de restricciones ahora es n, pero tenemos una cantidad exponencial de variables. Sin embargo, el algoritmo mantendrá una solución dual factible con a lo mas n variables no-nulas.

El algoritmo inicia con todas las variables en 0, primales y duales, lo que produce una solución dual factible y una solución primal infactible. Iniciando con el conjunto $A = \emptyset$, podemos incrementar la variable dual v_A hasta que una restricción $i \in E$ se cumpla con igualdad y entonces agregamos el elemento i al conjunto A. Con esto, podemos asegurar que el lado izquierdo de esa restricción no seguirá aumentando, ya que las variables duales que aumentaremos de aquí en adelante siempre tendrán al elemento i en el conjunto A. Luego, se incrementará la nueva variable $v(A)$ y se repetirá el proceso. Este proceso continuará iterando mientras que $D(A) > 0$. Una vez que terminemos, nuestra solución serán los elementos en el conjunto A y lo llamaremos S. Esta solución final será integral y como $D(S) = 0$, tendremos una solución factible de $[\text{IP}_1]$. Se explica su pseudocódigo en el Algoritmo 1.

Algoritmo 1 Algoritmo primal-dual para *Minimum Knapsack*

1: $y, v \leftarrow 0$;
2: $A \leftarrow \emptyset$;
3: **while** $D(A) > 0$ **do**
4: Incrementar v_A hasta que una restricción dual se cumpla con igualdad para un ítem i;
5: $y_i = 1$;
6: $A \leftarrow A \cup \{i\}$;
7: **end while**
8: **return** $S \leftarrow A$;

Teorema 3.1. *El Algoritmo 1 termina con una solución de costo no más grande que* $2opt_{[LP_1]} \leq 2opt_{[IP_1]}$.

DEMOSTRACIÓN. Sea l el elemento ingresado en la última iteración del Algoritmo 1 y S la solución final. Podemos ver que si $v_A > 0$ entonces A fue un conjunto tomado en el algoritmo y es un subconjunto de la solución final, es decir, $A \subseteq S \setminus \{l\}$ ya que la variable v_S se mantiene como 0 y que para cada elemento en S la restricción (3.2) se cumple con igualdad.

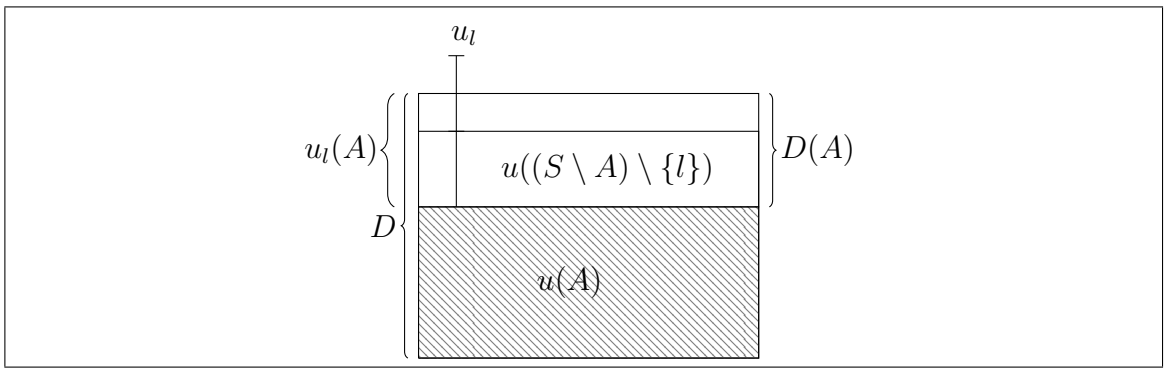

Figura 3.1. Cubrimiento de la demanda para un conjunto fijo A. Donde el área achurada es la parte cubierta por el conjunto A, el rectángulo superior a esa, es la demanda cubierta por la solución final sin el último elemento en ingresar l, la línea vertical nos muestra el cubrimiento real u_l y $u_l(A)$ es el cubrimiento truncado del elemento l cuando al fijar los elementos en A.

Luego, tomando la solución final y, tenemos que

$$\begin{aligned}\sum_{i\in E} y_i c_i &= \sum_{i\in S} c_i \\ &= \sum_{i\in S} \sum_{A\subseteq E: i\notin A} u_i(A) v_A \\ &= \sum_{A\subseteq E} \sum_{i\in S\setminus A} u_i(A) v_A \\ &= \sum_{A\subseteq E} v_A \sum_{i\in S\setminus A} u_i(A).\end{aligned}$$

Como $v_A > 0$ solo si A es el conjunto fue tomado en alguna iteración, podemos ver que para cada A la suma que acompaña a cada variable v_A representa la cantidad total de cubrimiento de los elementos que están en la solución final y que aun no son ingresados en la iteración relacionada con el conjunto A. Entonces, podemos ver en la Figura 3.1 que si sacamos de la suma el elemento l, entonces ésta no supera $D(A)$ y que $u_l(A)$ también es menor a $D(A)$ por definición. Por lo tanto tenemos que

$$\sum_{i\in E} y_i c_i = \sum_{A\subseteq E} v_A \left(\sum_{i\in (S\setminus A)\setminus\{l\}} u_i(A) + u_l(A) \right)$$

$$\leq \sum_{A \subseteq E} v_A \left(D(A) + D(A) \right)$$
$$= 2 \sum_{A \subseteq E} v_A D(A).$$

Lo que indica que nuestra solución primal es menos costosa que dos veces la solución dual. Agregando que la solución dual es menor que el óptimo del dual, y que este a su vez es menor que el óptimo primal, podemos concluir que la solución entregada por el algoritmo es una 2-aproximación. Es decir,

$$\sum_{i \in E} y_i c_i \leq 2 \sum_{A \subseteq E} v_A D(A) \leq 2opt_{[\text{DP}_1]} \leq 2opt_{[\text{LP}_1]} \leq 2opt_{[\text{IP}_1]}.$$

□

Para poder capturar al máximo esta técnica, utilizamos el hecho de que al tener un problema puro de cubrimiento (es decir, con una sola restricción de cubrimiento de demanda), entonces el dual será un problema de empaquetamiento puro. Un punto de vista que será útil más adelante, es la intuición de que las variables duales llenan cajas (restricciones) y cuando una caja se llena (restricción se cumplen con igualdad), entonces el ítem relacionado es bueno para ser ingresado a la solución final.

También podemos entender el algoritmo con teoría de dualidad lineal. Como sabemos que en el óptimo se cumplen las ecuaciones de holguras complementarias, este algoritmo trata de cumplir con igualdad un conjunto de esas ecuaciones. Esto puede notarse en que cuando una variable primal tiene valor mayor a 0, entonces la restricción del dual asociada se cumple con igualdad. Sin embargo, como no necesariamente estamos en el óptimo, tenemos que no siempre cumple que por cada variable dual distinta de cero, la restricción primal asociada se cumple con igualdad.

En cuanto al gap de integraldidad, el algoritmo nos da una cota superior de 2. Para poder ver que el gap de integralidad es exactamente 2, podemos ver el siguiente ejemplo. Si tomamos $D = 1 + \frac{1}{n-1}$, n elementos con $u_i = c_i = 1$, entonces podemos notar que cada

solución integral factible tiene al menos dos elementos en la solución. Por esto, tenemos que $opt_{[\text{IP}_1]} = 2$. Además, podemos notar que si tomamos $y_i = \frac{1}{n-1}$, entonces tenemos una solución de $[\text{LP}_1]$. Esto lo podemos notar al ver que cuando fijamos A tenemos que:

$i)$ Si $|A| \geq 2$, entonces $D(A) = 0$.

$ii)$ Si $|A| = 1$, entonces $D(A) = \frac{1}{n-1}$. Esta demanda es cubierta por cualquier elemento fuera de A.

$iii)$ Si $|A| = 0$, entonces cubrimos D con los n elementos, ya que nos queda $ny_1 = \frac{n}{n-1} = \frac{n-1+1}{n-1} = \frac{1}{n-1} + 1$.

Luego, como y es una solución factible de costo $\frac{n}{n-1}$, tenemos que $opt_{[\text{LP}_1]} \leq 1 + \frac{1}{n-1}$. Finalmente, el gap de integralidad es

$$gap = \frac{opt_{[\text{IP}_1]}}{opt_{[\text{LP}_1]}} \geq \frac{2}{1 + \frac{1}{n-1}} \xrightarrow{n\to\infty} 2.$$

3.2. Problema de Instalaciones con Demanda Simple (SDFL)

Carnes y Shmoys (2015) dan una generalización de las KCI que va en la dirección hacia donde queremos llegar. En esta sección mostraremos estas desigualdades y una interpretación que se nos haga más abordable la generalización que propondremos en el próximo capítulo. Además, mostraremos gráficamente como se ve *Minimum Knapsack Problem* (MK) y *Single-Demand Facility Location Problem* (SDFL) al ver cada item como su función de costo y así será más fácil ver que SDFL es una generalización de MK.

En SDFL se tiene un conjunto de instalaciones E donde cada instalación $i \in E$ tiene una capacidad u_i, un costo de apertura c_i y un costo g_i de servir cada unidad de demanda, las cuales deben cubrir una demanda total D. El objetivo es seleccionar un subconjunto de instalaciones $S \subseteq E$ para abrir y definir cuanta demanda cubrirá cada instalación.

La formulación entera-mixta natural es

$$[\text{IP}_2] \text{ mín} \sum_{i \in E} y_i c_i + z_i g_i$$

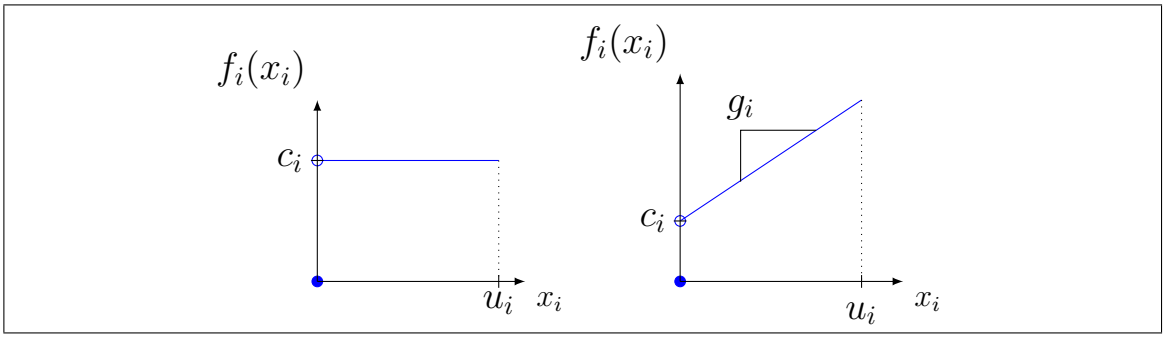

Figura 3.2. Funciones de cubrimiento MK versus SDFL

$$
\begin{aligned}
& s.a. \sum_{i \in E} z_i \geq D, \\
& z_i \leq u_i y_i \qquad \forall i \in E, \\
& y \in \{0,1\}^n, \\
& z_i \geq 0 \qquad \forall i \in E,
\end{aligned}
$$

donde y es el vector que tiene 1's en las instalaciones que se abrirán y x dice en cada componente la demanda que cubrirá cada instalación. Las restricciones nos dicen que debemos cubrir la demanda D y que si la instalación no se abre, entonces no podemos utilizarla para cubrir demanda.

Para poder entender por qué este problema es una generalización de MK, debemos ver cada instalación como un elemento o ítem y ver la comparación entre la función de costo de MK (lado izquierdo) y la de SDFL (lado derecho) como se ve en la Figura 3.2. Podemos ver en el lado izquierdo, que si pagamos c_i, entonces podemos utilizar un cubrimiento de 0 a u_i sin pagar nada extra. Siempre nos conviene utilizarla completamente, lo que nos deja solamente la decisión de si usarla o no. En el lado derecho, vemos que es similar a MK, pero que ahora debemos seguir pagando g_i por cada unidad de demanda a cubrir con un tope de u_i, pero en el caso en que $g_i = 0$ tenemos el caso de MK. De esta relación, entre MK y SDFL podemos ver que este último problema es NP-dificil, ya que es aun más difícil de resolver MK. Cabe destacar que si se tiene un algoritmo que solucione SDFL con un tiempo polinomial, entonces tendremos una forma de resolver MK también.

Para poder entender el conjunto de desigualdades válidas para este problema, mostraremos un modelo convexo intermedio que nos ayudará a construir nuestra generalización en el siguiente capítulo.

3.2.1. Relajación Convexa

La restricción que hace que nuestro modelo no sea un problema puro de cubrimiento es

$$z_i \leq u_i y_i \qquad \forall i \in E.$$

Si esta restricción no estuviera, entonces podríamos utilizar las técnicas de algoritmos primal-dual directamente, al igual que en MK.

Primero, podemos fijar un conjunto de instalaciones A, las cuales definen una demanda residual $D(A)$. La única diferencia es que acá debemos decidir el echo de que las instalaciones en A son usadas completamente, es decir, $D(A) = \max\{D - \sum_{i \in A} u_i, 0\}$. Luego, podemos truncar el cubrimiento máximo, ya que si fijamos una demanda residual, no vale la pena cubrir mas que eso en ningún caso.

Segundo, para poder incluir la propiedad de que si no se usa la instalación i, entonces no se puede cubrir demanda con esa instalación en la restricción de cubrimiento, podemos tomar el mínimo entre $u_i(A)y_i$ y z_i. Con esto nos aseguramos que en algún óptimo, si $y_i = 0$ entonces $z_i = 0$ porque sino no serviría para cubrir y estaría generando costos extras sin necesidad.

Con estas ideas, un modelo no lineal con una sola restricción de cubrimiento nos queda

$$[\text{NLP}_2] \min \sum_{i \in E} y_i c_i + z_i g_i \tag{3.4}$$

$$s.a. \sum_{i \in E \setminus A} \min\{z_i, y_i u_i(A)\} \geq D(A) \quad \forall A \subseteq E, \tag{3.5}$$

$$y \in \{0,1\}^n, \tag{3.6}$$

$$z_i \geq 0 \qquad \forall i \in E. \tag{3.7}$$

Cada punto factible de [IP$_2$] es valido en este modelo, ya que con las instalaciones que se activan, tenemos que la suma de sus z's cubren D, y como no pueden ser mayores a u_i, entonces la suma de la restricción (3.5) es al menos D.

3.2.2. Relajación Lineal

Para poder generar una relajación lineal a partir de [NLP$_2$] debemos linealizar la restricción (3.5) y la naturaleza de las variables y. Para la restricción, podemos usar el siguiente lema, donde la idea principal es generar una partición de $E \setminus A$ en dos conjuntos, E_1 y E_2, donde existirá una partición ideal para el óptimo donde las instalaciones que toman su mínimo en la expresión z_i estarán en el conjunto E_1 y las que lo alcancen en $y_i u_i(A)$ quedaran en E_2.

Lema 3.2. *Si $\alpha, \beta \in \mathbb{R}^n$, entonces*

$$\sum_{i=1}^{n} \text{mín}\{\alpha_i, \beta_i\} = \min_{E_1 \cup E_2 = \{1,..,n\}} \sum_{i \in E_1} \alpha_i + \sum_{i \in E_2} \beta_i.$$

DEMOSTRACIÓN. Sea (E_1^*, E_2^*) la partición que alcanza el mínimo en el término de la derecha, esto quiere decir que para todo $i \in E_1^*$ se tiene que $\alpha_i \leq \beta_i$ y que para todo $i \in E_2^*$ se tiene que $\alpha_i \geq \beta_i$.

Con esto, podemos ver que el término de la izquierda será,

$$\begin{aligned}
\sum_{i=1}^{n} \text{mín}\{\alpha_i, \beta_i\} &= \sum_{i \in E_1^*} \text{mín}\{\alpha_i, \beta_i\} + \sum_{i \in E_2^*} \text{mín}\{\alpha_i, \beta_i\} \\
&= \sum_{i \in E_1^*} \alpha_i + \sum_{i \in E_2^*} \beta_i \\
&= \min_{E_1 \cup E_2 = \{1,..,n\}} \sum_{i \in E_1} \alpha_i + \sum_{i \in E_2} \beta_i
\end{aligned}$$

Con lo que demostramos la igualdad. □

Este lema nos dice que podemos relajar la formulación entera-mixta [IP$_2$] utilizando una cantidad exponencial de restricciones. Utilizando el conjunto de índices $E \setminus A$, los vectores α y β como $\alpha_i = z_i$ y $\beta_i = y_i u_i(A)\ \forall i \in E \setminus A$ del Lema 3.2 y relajando la naturaleza binaria de la variable y, obtenemos la siguiente relajación lineal

$$[\text{LP}_2]\ \text{mín} \sum_{i \in E} y_i c_i + z_i g_i \tag{3.8}$$

$$s.a. \sum_{i \in E_1} z_i + \sum_{i \in E_2} y_i u_i(A) \geq D(A) \quad \forall (A, E_1, E_2) \in \mathcal{F}, \tag{3.9}$$

$$z_i, y_i \geq 0 \qquad \forall i \in E, \tag{3.10}$$

donde $\mathcal{F}$ es el conjunto que tiene las tripletas válidas de (A, E_1, E_2), es decir,

$$\mathcal{F} = \{(A, E_1, E_2) : A \subseteq E \wedge E_1 \cup E_2 = E \setminus A\}.$$

Ahora, veremos con un algoritmo primal-dual basado en esta relajación, que este modelo mantiene la constante de aproximación del poliedro de MK relajado con las KC-*inequalities* originales, ya que también alcanza una 2-aproximación.

3.2.3. Algoritmo Primal-Dual

Al calcular el dual de [LP$_2$] nos queda

$$[\text{DP}_2]\ \text{máx} \sum_{(A,E_1,E_2) \in \mathcal{F}} D(A) v_{(A,E_1,E_2)} \tag{3.11}$$

$$s.a. \sum_{\substack{(A,E_1,E_2) \in \mathcal{F}: \\ i \in E_1}} v_{(A,E_1,E_2)} \leq g_i \qquad \forall i \in E, \tag{3.12}$$

$$\sum_{\substack{(A,E_1,E_2) \in \mathcal{F}: \\ i \in E_2}} v_{(A,E_1,E_2)} u_i(A) \leq c_i \quad \forall i \in E, \tag{3.13}$$

$$v_{(A,E_1,E_2)} \geq 0 \qquad \forall (A, E_1, E_2) \in \mathcal{F}, \tag{3.14}$$

donde v es la variable dual que nos dará la idea de llenar las restricciones para saber cuando un elemento es lo suficientemente bueno para usarse en la solución primal.

Podemos interpretar cada restricción como un contenedor relacionado con la parte de la función asociada al costo, es decir, cuando la restricción (3.12) relacionada a un elemento i se alcance con igualdad significa que la parte variable de ese elemento es buena para usarse y cuando la restricción (3.13) se cumpla con igualdad es que el costo fijo es barato en relación a los demás. Otra forma similar de verlo es que estamos llenando todos los contenedores a la vez y eso es como ir aumentando el presupuesto hasta que un elemento se paga por completo por primera vez. En ese momento sabremos que ese elemento puede ingresar a la solución primal final usando el menor presupuesto, por lo tanto lo pagamos y seguimos viendo cual es el siguiente mejor, y así hasta llenar la demanda. La diferencia con la sección anterior es que para que un elemento sea bueno e ingrese a la solución final, el presupuesto debe cubrir el costo fijo y el costo variable.

Al igual que en la sección anterior, partiremos con todas las variables, primales y duales, en 0, el conjunto A como vacío y los conjuntos E_1 y E_2 como E y $\emptyset$ respectivamente. Diremos que un elemento será bueno para ingresarse a la solución final cuando ambas restricciones relacionadas al elemento se hayan alcanzado con igualdad.

Entonces, como todos los elementos parten en el conjunto E_1, cuando se cumpla con igualdad la restricción (3.12), el elemento relacionado se moverá de E_1 a E_2. Luego, cuando ese elemento cumpla con igualdad la restricción (3.13), entonces se ingresara al conjunto A y a la solución final. Este algoritmo terminará una vez que el conjunto A cubra toda la demanda D. El detalle del algoritmo se ve en Algoritmo 2.

Teorema 3.2. *(Carnes y Shmoys, 2015) El Algoritmo 2 termina con una solución de costo no más grande que* $2opt_{[LP_2]} \leq 2opt_{[IP_2]}$.

DEMOSTRACIÓN. Sea ℓ el elemento ingresado en la última iteración del Algoritmo 2 y S la solución final del algoritmo. Podemos ver que si $v_{(A,E_1,E_2)} > 0$ solo para las iteraciones validas del algoritmo, entonces $A \subseteq S \setminus \{\ell\}$ ya que las variables $v_{(S,E_1,E_2)} = 0$ para todas las combinaciones válidas $(S, E_1, E_2) \in \mathcal{F}$. Además, para cada instalación en S tenemos que la restricción (3.13) y (3.12) se cumple con igualdad.

Algoritmo 2 Algoritmo primal-dual para SDFL.

y,z, $v \leftarrow 0$;
$E_1 \leftarrow E$;
$F_2, A \leftarrow \emptyset$;
while $D(A) > 0$ **do**
    Incrementar $v_{(A,E_1,E_2)}$ hasta que una restricción dual se cumpla con igualdad para un item i;
    if $i \in E_1$ **then**
        $E_1 \leftarrow E_1 \setminus \{i\}$ y $E_2 \leftarrow E_2 \cup \{i\}$;
    else
        $z_i \leftarrow u_i(A)$;
        $y_i \leftarrow 1$;
        $E_2 \leftarrow E_2 \setminus \{i\}$ y $A \leftarrow A \cup \{i\}$;
    end if
end while
return $S \leftarrow A$;

Luego, tomando la solución final (y, z), tenemos que

$$\sum_{i \in E} y_i c_i + z_i g_i = \sum_{i \in S} c_i + z_i g_i$$
$$= \sum_{i \in S} \left[\sum_{\substack{(A,E_1,E_2) \in \mathcal{F}: \\ i \in E_2}} u_i(A) v_{(A,E_1,E_2)} + z_i \left(\sum_{\substack{(A,E_1,E_2) \in \mathcal{F}: \\ i \in E_1}} v_{(A,E_1,E_2)} \right) \right].$$

Lo que intercambiando el orden de las sumatorias nos queda

$$\sum_{i \in E} y_i c_i + z_i g_i = \sum_{(A,E_1,E_2) \in \mathcal{F}} \left[\sum_{i \in S \cap E_2} u_i(A) v_{(A,E_1,E_2)} + \sum_{i \in S \cap E_1} z_i v_{(A,E_1,E_2)} \right]$$
$$= \sum_{(A,E_1,E_2) \in \mathcal{F}} v_{(A,E_1,E_2)} \left[\sum_{i \in S \cap E_2} u_i(A) + \sum_{i \in S \cap E_1} z_i \right].$$

Ahora podemos separar el aporte del último elemento ℓ de la suma. Para esto, cabe destacar que para cada iteración del algoritmo ($v_A > 0$) el elemento ℓ pudo haber estado en E_1 o en E_2. Definimos $\beta_\ell(A, E_1, E_2)$ como el aporte de ℓ. Es decir, si $\ell \in E_1$ entonces $\beta_\ell(A, E_1, E_2) = z_\ell$ y si $\ell \in E_2$ entonces $\beta_\ell(A, E_1, E_2) = u_\ell(A)$.

Por otro lado, hay que fijarse que los elementos $i \in S \setminus \{\ell\}$ tienen $u_i(A) = u_i$ para todas las variables $v_A > 0$, ya que estos elementos ingresaron durante el algoritmo a la solución primal y si alguno hubiese estado truncado, entonces el algoritmo hubiese terminado antes. Además, como los elementos en E_1 que aportan a la suma son solo los elementos que ingresaron a S en algún momento, podemos cambiar los z_i por u_i usando el mismo argumento anterior. Recordemos que $z_i = u_i$ para todo i en $S \setminus \{\ell\}$ y que $z_i = 0$ para todo i en $E \setminus S$.

Con esto en mente, tenemos que

$$\begin{aligned}
\sum_{i \in E} y_i c_i + z_i g_i &= \sum_{(A,E_1,E_2) \in \mathcal{F}} v_{(A,E_1,E_2)} \left[\sum_{i \in (S \cap E_2) \setminus \{\ell\}} u_i(A) + \sum_{i \in (S \cap E_1) \setminus \{\ell\}} z_i + \beta_\ell(A, E_1, E_2) \right] \\
&= \sum_{(A,E_1,E_2) \in \mathcal{F}} v_{(A,E_1,E_2)} \left[\sum_{i \in (S \cap E_2) \setminus \{\ell\}} u_i + \sum_{i \in (S \cap E_1) \setminus \{\ell\}} u_i + \beta_\ell(A, E_1, E_2) \right] \\
&= \sum_{(A,E_1,E_2) \in \mathcal{F}} v_{(A,E_1,E_2)} \left[\sum_{i \in (S \cap (E_2 \cup E_1)) \setminus \{\ell\}} u_i + \beta_\ell(A, E_1, E_2) \right] \\
&= \sum_{(A,E_1,E_2) \in \mathcal{F}} v_{(A,E_1,E_2)} \left[\sum_{i \in (S \cap (E \setminus A)) \setminus \{\ell\}} u_i + \beta_\ell(A, E_1, E_2) \right] \\
&= \sum_{(A,E_1,E_2) \in \mathcal{F}} v_{(A,E_1,E_2)} \left[\sum_{i \in (S \setminus A)) \setminus \{\ell\}} u_i + \beta_\ell(A, E_1, E_2) \right].
\end{aligned}$$

Al igual que en MK, para cada A podemos ver que la sumatoria de u_i's es exactamente lo que cubre la solución final sin el último elemento, que sabemos que es menor a $D(A)$. Solo nos falta ver que $\beta_\ell(A, E_1, E_2) \leq D(A)$ y tendremos que el algoritmo entrega una 2 aproximación. Para esto, tenemos que ver dos casos

$a)$ En el caso en que $\ell \in E_1$, tenemos que $\beta_\ell(A, E_1, E_2) = z_\ell$.
Sabemos que $z_\ell = u_\ell(S \setminus \{\ell\})$ y como la definición de $u_i(A)$ es monótona decreciente en A y $v_{(A,E_1,E_2)} > 0$ solo si $A \subseteq S \setminus \{\ell\}$ tenemos que $z_\ell \leq u_l(A') \leq D(A')$ para todo A' donde $v_{(A',E_1,E_2)} > 0$.

$b)$ En el caso en que $\ell \in E_2$, tenemos que $\beta_\ell(A, E_1, E_2) = u_\ell(A)$. Por definición de u_ℓ tenemos que $u_i(A) \leq D(A)$.

Por lo tanto, $\beta_\ell(A, E_1, E_2) \leq D(A)$.

Con esto nos queda que

$$\begin{aligned}\sum_{i \in E} y_i c_i + z_i g_i &\leq \sum_{(A,E_1,E_2) \in \mathcal{F}} v_{(A,E_1,E_2)} \left[D(A) + D(A)\right] \\ &\leq 2 \sum_{(A,E_1,E_2) \in \mathcal{F}} v_{(A,E_1,E_2)} D(A).\end{aligned}$$

Juntando esto con que v nos entrega una solución factible dual, dualidad débil y que partimos de la solución primal de una relajación lineal de una relajación lineal de $\text{IP}_2]$, tenemos que

$$\sum_{i \in E} y_i c_i + z_i g_i \leq 2 \sum_{A \in E} v_{(A,E_1,E_2)} D(A) \leq 2opt_{[\text{DP}_2]} \leq 2opt_{[\text{LP}_2]} \leq 2opt_{[\text{IP}_2]}.$$

□

Como este problema es una generalización de KC, basta con tomar el mismo ejemplo que en la sección 3.1.1 para ver que la constante de aproximación 2 es ajustada.

Para poder generalizar este problema, haremos un paso intermedio en el siguiente capítulo. Tomaremos un modelo que utilice funciones continuas y lineales por tramo que partan en $(0, 0)$ y luego en el capítulo subsiguiente mostraremos una generalización de funciones no decrecientes, continuas por la izquierda y con una cantidad finita de discontinuidades.

Para esta generalización intermedia de MK, utilizaremos funciones de costos continuas y lineales por tramo. Podemos ver en la figura 4.1 como modelamos MK con esta propuesta. De esta manera, modelamos de manera exacta el problema MK, ya que podemos amplificar los valores u y D para que sean enteros y tomar un ε suficientemente pequeño, de manera que aunque usemos los saltos de todos los elementos, no alcancemos a cubrir una unidad de demanda. En el siguiente capítulo demostraremos esta propiedad para un caso más general. Podemos tomar ε como la precisión de maquina, pero basta con tomarlo menor a una unidad de demanda sobre la cantidad total de discontinuidades sumada de todas las funciones de costo, siempre que las constantes de cubrimiento y la demanda D sean enteras, en caso contrario, se pueden amplificar sin modificar las soluciones.

Notemos que si utilizamos la función del lado derecho de la figura 4.1, el costo de cada elemento i está dado por $\varepsilon g_i = \varepsilon \frac{c_i}{\varepsilon} = c_i$. Por lo tanto, el costo de un elemento modelado de esta manera, será exactamente el mismo que en MK. Luego, al igual que en MK, si es que un elemento paga el primer tramo (el salto en 0), entonces podemos utilizarlo completamente, ya que no aumenta el costo.

Teniendo lo anterior claro, el problema de la mochila lineal por partes o *Piecewise Linear Minimum Knapsack* lo definimos por una demanda D a cubrir y un conjunto de

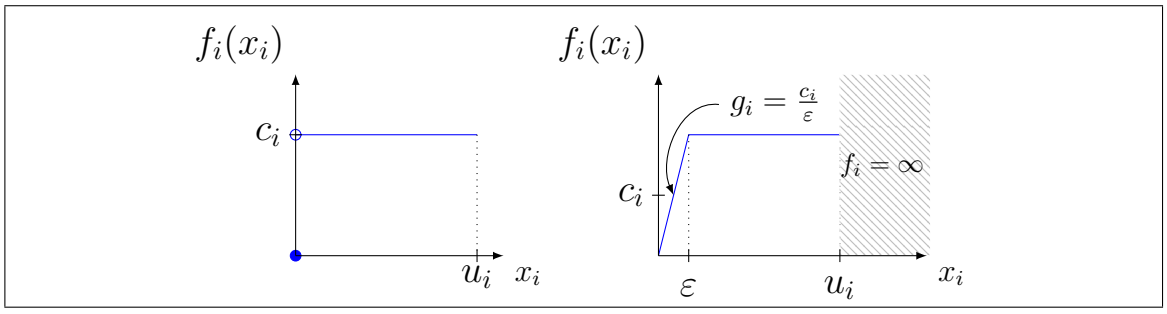

Figura 4.1. Modelamiento de MK a través de funciones lineales por tramo. El lado izquierdo muestra la función original y el lado derecho la aproximación del modelo.

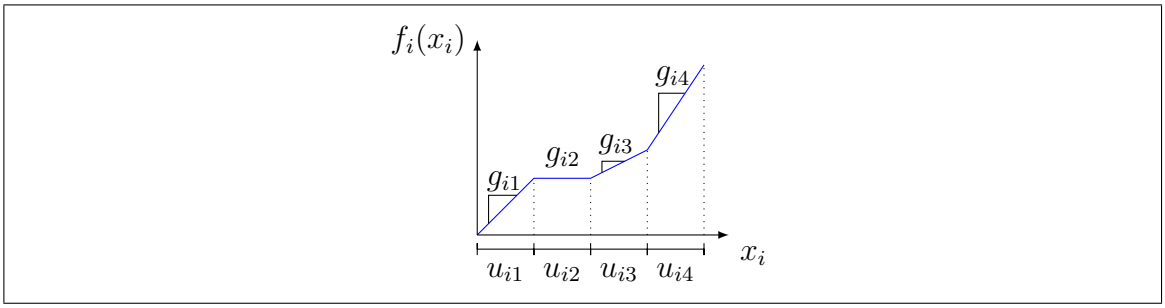

Figura 4.2. Ejemplo de función costo continua y lineal por tramos

elementos E, donde para cada elemento $i \in E$ se tiene una función de costo no decreciente, continua y lineal por tramos $f_i : \mathbb{R} \to \mathbb{R}_+ \cup \{\infty\}$. Donde si se decide cubrir una cantidad z_i de la demanda con el elemento i, entonces el costo asociado será $f_i(z_i)$. Como cada función f_i es lineal por tramo, podemos definir g_{ij} como la pendiente del elemento i en el tramo j. Como se ve en la Figura 4.2, g_{ij} es el costo unitario de cubrir una unidad de demanda con el tramo j del elemento i. El objetivo de este problema es decidir cuanto cubrir con cada elemento, de manera que cubran la demanda D minimizando los costo que estarán dados por la expresión $\sum_{i,j} f_i(z_i)$.

El modelo natural de este problema es

$$
\begin{aligned}
[\text{IP}_3]\ \text{mín} \sum_{i \in E} f_i(z_i) & \\
s.a. \sum_{i \in E} z_i \geq D, & \\
z_i \geq 0 \qquad & \forall i \in E.
\end{aligned}
$$

La complicación de este modelo es que queda no lineal, ya que f_i es lineal por tramos. Para arreglar esto, podemos usar la estructura de la función y utilizar variables auxiliares z_{ij} que definan cuanto cubriremos en cada tramo por cada elemento, por lo tanto $z_i = \sum_j z_{ij}$. Junto con esto, debemos definir u_{ij} como el largo del tramo j del elemento i, para el cuál la función tiene la misma pendiente en todo el tramo. Como se ve en la Figura 4.3, u_{ij} nos da la capacidad máxima de cubrimiento de cada variable z_{ij}.

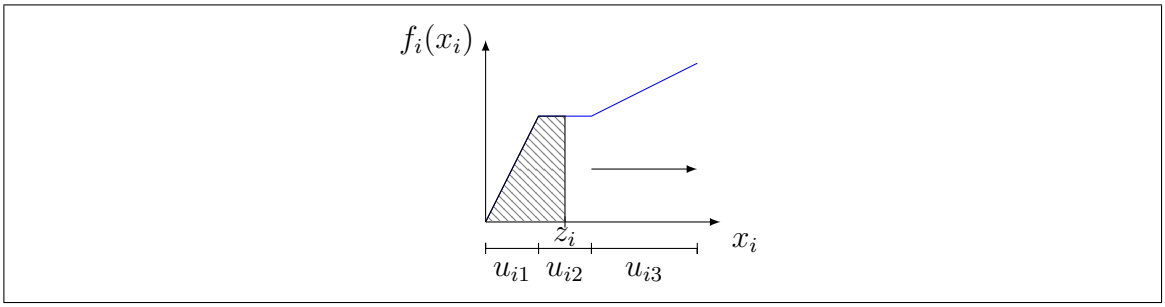

Figura 4.3. Forma de utilizar función de costo f_i.

Al reemplazar z_i por las variables z_{ij} debemos agregar una restricción de precedencia, ya que originalmente la función f_i se debe llenar como se ve en la Figura 4.3. Esta restricción debe asegurar que si una parte j se utiliza, entonces sus antecesoras también deben ser usadas. Esto se puede representar de manera resumida en que cada vez que usemos una parte j, entonces se utilice la parte $j-1$. Además, sin perdida de generalidad podemos asumir que todos los elementos tienen la misma cantidad de tramos m, en caso de que no sea así, podemos crear tramos de largo 0.

Sea $J = \{1, .., m\}$ el conjunto de las partes de la función f_i, con ésto el modelo nos queda

$$[\text{CLP}_3] \text{ mín} \sum_{i \in E} \sum_{j \in J} z_{ij} g_{ij} \tag{4.1}$$

$$s.a. \sum_{i \in E} \sum_{j \in J} z_{ij} \geq D, \tag{4.2}$$

$$z_{ij} > 0 \Rightarrow z_{i(j-1)} = u_{i(j-1)} \qquad \forall i \in E \quad \forall j \in \{2, .., m\}, \tag{4.3}$$

$$0 \leq z_{ij} \leq u_{ij} \qquad \forall i \in E \quad \forall j \in J. \tag{4.4}$$

El modelo $[\text{CLP}_3]$ es casi lineal, lo sería si pudiésemos eludir la restricción (4.3) que nos da la precedencia. Esta restricción puede ser modelada con programación entera y $[\text{CLP}_3]$ quedaría equivalente a

$$
\begin{aligned}
[\mathrm{IP}_3]\ \text{mín} & \sum_{i\in E}\sum_{j\in J} g_{ij}z_{ij} \\
s.t. & \sum_{i\in E}\sum_{j\in J} z_{ij} \geq D, \\
& z_{ij} \leq y_{ij}u_{ij} \qquad \forall i,j, \\
& y_{ij} \geq y_{i,j+1} \qquad \forall i,j, \\
& z \geq 0 \\
& y \in \{0,1\}^{n\times m},
\end{aligned}
$$

donde la variable y_{ij} es 1 si es que se usa la parte j del elemento i y 0 si no. Además, la tercera restricción nos dice que solo podemos usar la parte $j+1$ de un elemento, si es que usamos la parte anterior.

Nuestro objetivo es tomar $[\mathrm{IP}_3]$ y tratar de linealizarlo de manera que quede como un problema de cubrimiento puro. De esta manera, podríamos utilizar técnicas parecidas a las del capítulo anterior.

4.1. Relajación Convexa

Al igual que en el capítulo anterior, pasaremos por un modelo convexo que utilice mínimos para capturar la propiedad de precedencia de la restricción (4.3) y que luego podamos linealizar utilizando una cantidad exponencial de restricciones.

Existe más de una forma de relajar el modelo y generalizar las KCI, pero en esta tesis propondremos una que hace más intuitivo el algoritmo primal-dual que propondremos más adelante. Para nuestra generalización, primero fijaremos un vector $a \in \{0,..,m\}^n$, tal que si $a_i = j$ quiere decir que el elemento i se utiliza a capacidad máxima hasta la parte j, en el caso en que $j = 0$ el elemento no tiene partes utilizadas. Por ejemplo, si la componente $a_1 = 0$ quiere decir que para el primer elemento no se fijará ninguna

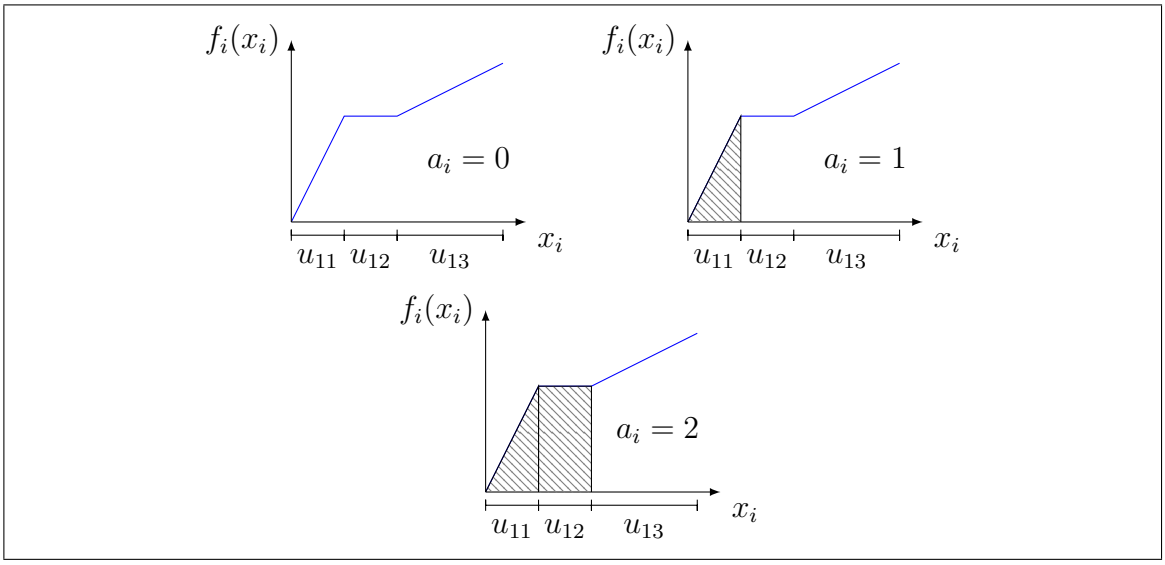

Figura 4.4. Significado del vector a en función f_i.

componente y si $a_2 = 2$ quiere decir que se toman las primeras dos partes del segundo elemento completamente. Esto queda más claro en la Figura 4.4, en donde se marca con achurado gris la parte de cada elemento que se se toma a priori con un vector a fijo.

Con este vector en mente, podemos generar nuestras definiciones para generalizar las KCI. Para esto debemos definir la demanda residual y el cubrimiento truncado. Debemos tener en cuenta que si queremos truncar una parte en un elemento que no ha sido fijado, entonces sabemos que para usar esa parte, es necesario usar las partes previas.

Definición 4.1. *Se define la* ***demanda residual*** *y el* ***cubrimiento truncado****, respectivamente, como:*

$a)$

$$D(a) = \text{máx}\left\{0, D - \sum_{i \in E} \sum_{j=0}^{a_i} u_{ij}\right\}.$$

$b)$

$$u_{ij}(a) = \text{mín}\left\{u_{ij}, D(a) - \sum_{h=a_i+1}^{j-1} u_{ih}\right\}.$$

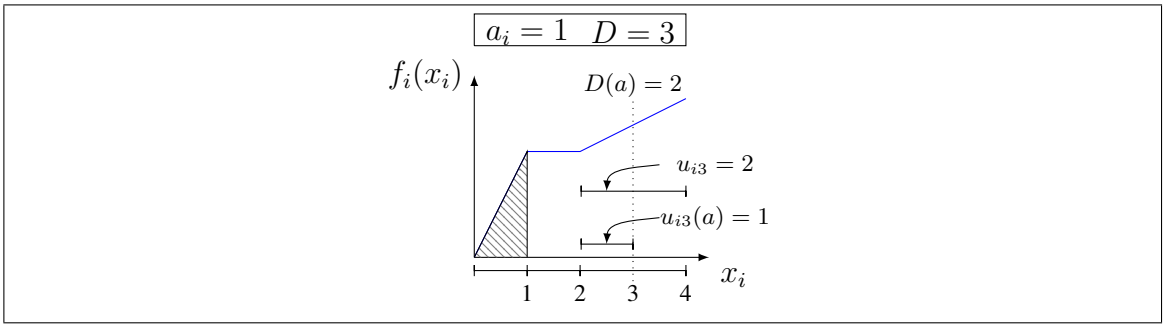

Figura 4.5. Ejemplo de como se truncan los cubrimientos de un elemento con la demanda residual.

La demanda residual quiere significa la demanda que queda una vez que se utilizan en totalidad las partes seleccionadas en a. El cubrimiento truncado $u_{ij}(a)$ nos da un largo truncado de la parte j del elemento i, de tal manera que si sigo utilizando el elemento i desde la parte $a_i + 1$ hasta $j - 1$ y aun así nos queda demanda residual por cubrir, entonces el cubrimiento u_{ij} es truncado si es que la nueva demanda que queda por cubrir es menor a u_{ij}. Esto se puede ver en el ejemplo de la Figura 4.5, en donde la tercera parte ($j = 3$) se trunca, ya que su cubrimiento es $u_{i3} = 2$ y la demanda residual es $D(a) = 2$, pero antes de usar la parte $j = 3$ debe usarse la parte anterior $j = 2$, la cuál cubre 1 unidad de demanda.

Usando estas definiciones, podemos escribir el paso intermedio de nuestra generalización de las KCI pensando en la siguiente pregunta: ¿Cuánto de la demanda residual cubre efectivamente una solución z? Para responder a esto, primero fijemos $a \in \{0, .., m\}^n$ y tomemos un elemento i cualquiera. Como tenemos fijado hasta la parte a_i pensemos en cuanto las partes que nos quedan:

$(a_i + 1)$ Como las partes anteriores fueron fijadas por a, lo que podemos cubrir con esta es

$$z_{i(a_i+1)}$$

.

$(a_i + 2)$ Esta parte puede cubrir solo si la anterior esta utilizada en completitud. Esto lo podemos relajar y decir que cubrirá tanto como la parte anterior. Sin embargo,

para que distintas partes sean comparables, debemos comparar por la fracción que utilizamos de su cubrimiento total, es decir, si la parte $a_i + 1$ se usa hasta la mitad de su cubrimiento máximo ($\frac{u_{i(a_i+1)}}{2}$), entonces la cantidad máxima a cubrir por la siguiente parte podrá ser a lo más la mitad de su cubrimiento máximo, es decir, podrá ser a lo más ($\frac{u_{i(a_i+2)}(a)}{2}$). Entonces, el cubrimiento de la parte $(a_i + 2)$ está dado por la siguiente expresión,

$$\text{mín}\left\{ z_{i(a_i+2)}, \frac{z_{i(a_i+1)}}{u_{i(a_i+1)}(a)} u_{i(a_i+2)} \right\}.$$

$(a_i + 3)$ En la siguiente, podría bastar con truncar solo con la parte anterior, pero para tener una restricción más fuerte, truncaremos el cubrimiento usando todas las partes anteriores hasta llegar a las que se ingresaron por a. Es decir, la restricción nos queda,

$$\text{mín}\left\{ z_{i(a_i+3)}, \frac{z_{i(a_i+2)}}{u_{i(a_i+2)}} u_{i(a_i+3)}(a), \frac{z_{i(a_i+1)}}{u_{i(a_i+1)}} u_{i(a_i+3)}(a) \right\}.$$

Con todo esto en mente, entonces podemos tomar la siguiente relajación convexa de nuestro problema,

$$[\text{NLP}_3] \ \text{mín} \sum_{i\in E}\sum_{j\in J} z_{ij} g_{ij} \tag{4.5}$$

$$s.a. \sum_{i\in E}\left[\sum_{j=a_i+1}^{m} \text{mín}\left\{ z_{ij}, \frac{z_{i(j-1)}}{u_{i(j-1)}} u_{ij}(a), .., \frac{z_{i(a_{i+1})}}{u_{i(a_{i+1})}} u_{ij}(a) \right\}\right] \geq D(a)$$

$$\forall a \in \{0, .., m\}^n \tag{4.6}$$

$$0 \leq z_{ij} \qquad \forall i \in E \quad \forall j \in J. \tag{4.7}$$

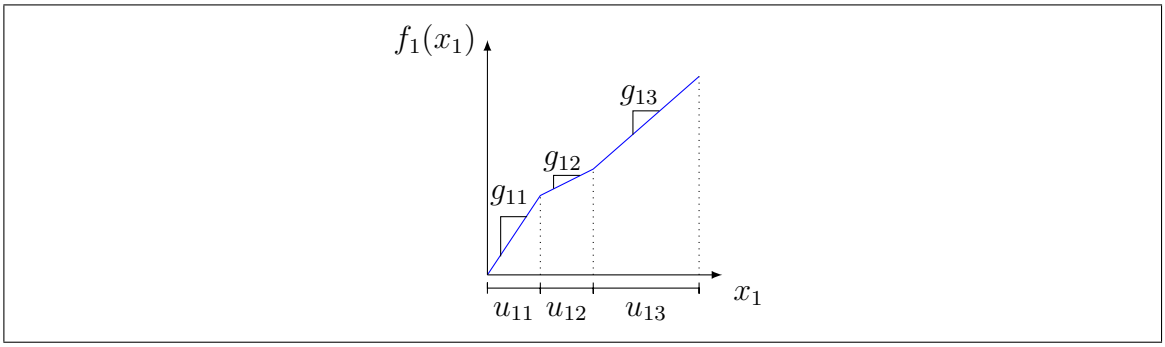

Figura 4.6. Función de costo de 3 partes.

4.2. Relajación Lineal

Para poder trabajar [NLP$_3$] debemos linealizar cada mínimo. Para guiar el procedimiento, nos vamos a centrar en un ejemplo que nos ayudará a entender nuestra generalización.

Pensemos en una instancia donde solo tenemos un elemento con una función de costo de tres partes como se ve en la Figura 4.6.

Supongamos ahora que tomamos la restricción $a = \vec{0}$. Entonces tenemos que nuestra restricción es

$$\underbrace{z_{11}}_{(I)} + \underbrace{\text{mín}\left\{z_{12}, \frac{z_{11}}{u_{11}}u_{12}(a)\right\}}_{(II)} + \underbrace{\text{mín}\left\{z_{13}, \frac{z_{12}}{u_{12}}u_{13}(a), \frac{z_{11}}{u_{11}}u_{13}(a)\right\}}_{(III)} \geq D(a), \qquad (4.8)$$

donde (I) representa la cantidad cubierta por la primera parte, (II) la que cubre la segunda y (III) la tercera. El problema es que no sabemos cuanto valdrá (II) y (III), ya que dependen de las variables de decisión, pero si sabemos sus posibles valores. Entonces, podemos tomar las combinaciones de esos valores y hacer una restricción por cada combinación, ya que si para cada término de los mínimos se cumple que la restricción es $\geq D(a)$, entonces con los mínimos se cumplirá también.

Entonces, nos quedan las siguientes restricciones:

$$
\begin{aligned}
&z_{11} + z_{12} + z_{13} \geq D(a),\\
&z_{11} + z_{12} + \frac{z_{12}}{u_{12}} u_{13}(a) \geq D(a),\\
&z_{11} + z_{12} + \frac{z_{11}}{u_{11}} u_{13}(a) \geq D(a),\\
&z_{11} + \frac{z_{11}}{u_{11}} u_{12}(a) + z_{13} \geq D(a),\\
&z_{11} + \frac{z_{11}}{u_{11}} u_{12}(a) + \frac{z_{12}}{u_{12}} u_{13}(a) \geq D(a),\\
&z_{11} + \frac{z_{11}}{u_{11}} u_{12}(a) + \frac{z_{11}}{u_{11}} u_{13}(a) \geq D(a).
\end{aligned}
$$

Como mencionamos antes, cuando todas estas restricciones se cumplen, entonces se cumple la restricción (4.8).

Cuando hacemos esto para cada vector a y cada elemento i, tendremos nuestra generalización de las KCI. Para expresarla, podemos tomar una colección de tuplas $F = (F_1, F_2, .., F_n)$ de cardinalidad n donde cada tupla $F_i = (F_i^0, F_i^1, .., F_i^{m-1})$ contiene m conjuntos, donde si $j \in F_i^k$ quiere decir que la cantidad que cubre la parte j en la restricción (4.8) viene dada por el $(k+1)$-ésimo término en el mínimo. Es decir, para tener las restricciones de antes, usamos las tuplas de la forma $(F_1^0, F_1^1, F_1^2) \in \{$ $(\{1,2,3\}, \emptyset, \emptyset)$, $(\{1,2\}, \{3\}, \emptyset)$, $(\{1,2\}, \emptyset, \{3\})$, $(\{1,3\}, \{2\}, \emptyset)$, $(\{1\}, \{2,3\}, \emptyset)$, $(\{1\}, \{2\}, \{3\})$ $\}$, respectivamente.

Otra interpretación que se le puede dar a cada conjunto es que k nos dice la cantidad de *shifts* hacia la izquierda que debemos tomar en j de la variable z_{ij} que se relaciona con la parte de la restricción y la del término del mínimo que usaré. Por ejemplo, en el último conjunto $(\{1\}, \{2\}, \{3\})$ como 1 está en F_1^0, entonces en el cubrimiento de su parte usamos su variable z_{11}, como 2 está en F_1^1 entonces no usaremos la variable z_{12} sino que tomamos un *shift* hacia la izquierda en j y usamos el término relacionado con la variable z_{11} que es $\frac{z_{11}}{u_{11}} u_{12}(a)$ y como la parte 3 está en F_1^1 entonces tomamos un *shift* de 2 hacia la izquierda y nos queda el término relacionado a la variable z_{11} que es $\frac{z_{11}}{u_{11}} u_{13}(a)$.

Por otro lado, debemos fijarnos que hay restricciones sobre las colecciones F_i's y que una parte j no puede estar en cualquier conjunto F_i^k. Como las colecciones dependen de a, definimos las tuplas válidas en el conjunto $\mathcal{F}$ por

$$\mathcal{F} = \left\{ (a, F) : \begin{array}{c} a \in \{0, .., m\}^n \wedge \bigcupdot_{k=0}^{m-1} F_i^k = \{a_i + 1, .., m\} \\ \wedge \ j \in F_i^k \Rightarrow j - a_i > k \quad \forall k, j \in J \end{array} \right\},$$

donde F es una colección de tuplas como mencionamos anteriormente. Además, cabe destacar que muchos conjuntos serán vacíos. Como $k < j - a_i$ y $j \leq m$, entonces $k \leq m - a_i - 1$, lo que usaremos para definir ciertas constantes como 0 y así no agregar variables de más en nuestro modelo lineal.

Con estas definiciones, la relajación lineal de nuestro problema nos queda

$$[\text{LP}_3] \ \text{mín} \sum_{i \in E} \sum_{j \in J} z_{ij} g_{ij} \tag{4.9}$$

$$s.a. \sum_{i \in E} \sum_{k=0}^{m-1} \left[\sum_{j \in F_i^k} \tau_a(i, j, k) z_{i(j-k)} \right] \geq D(a) \qquad \forall (a, F) \in \mathcal{F} \tag{4.10}$$

$$0 \leq z_{ij} \qquad \forall i \in E \, \forall j \in J, \tag{4.11}$$

donde $\tau_a(i, i, k)$ está definida como el término respectivo al conjunto F_i^k, es decir,

$$\tau_a(i, j, k) = \begin{cases} 1 & \text{, si } k = 0, \\ \frac{u_{ij}(a)}{u_{i(j-k)}} & \text{, si } 0 < k \leq m - a_i - 1, \\ 0 & \text{, si } k \geq m - a_i. \end{cases}$$

4.3. Algoritmo Primal-Dual

Lo primero que debemos hacer es calcular el dual, para lo cual, haremos una combinación lineal de las restricciones con las variable dual $v_{(a,F)} \geq 0$. Pero antes, cambiemos un poco la restricción para dejar mas aislado z_{ij}. Entonces, tomando el lado izquierdo de

la restricción, tenemos que

$$\sum_{i\in E}\sum_{k=0}^{m-1}\left[\sum_{j\in F_i^k}\tau_a(i,j,k)z_{i(j-k)}\right] = \sum_{i\in E}\sum_{k=0}^{m-1}\left[\sum_{j=1}^{n}\mathbb{1}_{\{j\in F_i^k\}}\tau_a(i,j,k)z_{i(j-k)}\right]$$
$$= \sum_{i\in E}\sum_{j'\in J}\sum_{k=0}^{m-1}\mathbb{1}_{\{(j'+k)\in F_i^k\}}\tau_a(i,j'+k,k)z_{ij'}.$$

Además, como la condición de que $(j'+k)\in F_i^k$ implica que $(j'+k)$ sea una parte valida, entonces $(j'+k)\leq m$. A su vez, esto significa que $k\leq m-j'$, con lo que nos queda que

$$\sum_{i\in E}\sum_{k=0}^{m-1}\left[\sum_{j\in F_i^k}\tau_a(i,j,k)z_{i(j-k)}\right] = \sum_{i\in E}\sum_{j'\in J}z_{ij'}\left(\sum_{\substack{k=0\\ j'+k\in F_i^k}}^{m-j'}\tau_a(i,j'+k,k)\right). \tag{4.12}$$

Luego, al hacer la combinación lineal de las restricciones con las variables $v_{(a,F)}\geq 0$ tenemos que,

$$\underbrace{\sum_{(a,F)\in\mathcal{F}}v_{(a,F)}\left[\sum_{i\in E}\sum_{j'\in J}z_{ij'}\left(\sum_{\substack{k=0\\ (j'+k)\in F_i^k}}^{m-j'}\tau_a(i,j'+k,k)\right)\right]}_{(\star)} \geq \sum_{(a,F)\in\mathcal{F}}D(a)v_{(a,F)},$$

donde el lado izquierdo se puede reescribir como

$$(\star) = \sum_{i\in E}\sum_{j'\in J}z_{ij'}\left[\sum_{(a,F)\in\mathcal{F}}v_{(a,F)}\left(\sum_{\substack{k=0\\ (j'+k)\in F_i^k}}^{m-j'}\tau_a(i,j'+k,k)\right)\right]$$
$$= \sum_{i\in E}\sum_{j'\in J}z_{ij'}\left[\sum_{(a,F)\in\mathcal{F}}v_{(a,F)}\left(\sum_{k=0}^{m-j'}\mathbb{1}_{\{(j'+k)\in F_i^k\}}\tau_a(i,j'+k,k)\right)\right]$$
$$= \sum_{i\in E}\sum_{j'\in J}z_{ij'}\left[\sum_{k=0}^{m-j'}\sum_{(a,F)\in\mathcal{F}}v_{(a,F)}\mathbb{1}_{\{(j'+k)\in F_i^k\}}\tau_a(i,j'+k,k)\right]$$

$$= \sum_{i \in E} \sum_{j' \in J} z_{ij'} \left[\sum_{k=0}^{m-j'} \sum_{\substack{(a,F) \in \mathcal{F}: \\ (j'+k) \in F_i^k}} v_{(a,F)} \tau_a(i, j'+k, k) \right].$$

Donde cabe destacar que al usar la indicatriz, la igualdad se mantiene aunque k se restrinja hasta $(m - j')$, ya que el resto hacen que la indicatriz valga 0.

Luego, el dual es directo y nos queda,

$$[\text{DP}_3] \text{ máx} \sum_{(a,F) \in \mathcal{F}} D(a) v_{(a,F)} \tag{4.13}$$

$$s.a. \sum_{k=0}^{m-j} \sum_{\substack{(a,F) \in \mathcal{F}: \\ (j+k) \in F_i^k}} \tau_a(i, j+k, k) v_{(a,F)} \leq g_{ij} \qquad \forall i \in E, \ \forall j \in J, \tag{4.14}$$

$$v_{(a,F)} \geq 0 \qquad \forall (a,F) \in \mathcal{F}, \tag{4.15}$$

donde las variables del dual son $v_{(a,F)}$ para cada $(a, F) \in \mathcal{F}$ y son no-negativa. Acá, nuevamente podemos tener una idea de empaquetamiento, pero para ver bien la estructura de $[\text{DP}_3]$ veremos un ejemplo con elementos que tienen funciones de costo de 3 partes.

Al tomar $m = 3$ y separar las ecuaciones relacionada con las partes de las funciones de costo nos queda

$$\text{máx} \sum_{(a,F) \in \mathcal{F}} v_{(a,F)} D(a) \tag{4.16}$$

$$s.a. \sum_{\substack{(a,F) \in \mathcal{F}: \\ 1 \in F_i^0}} v_{(a,F)} + \sum_{\substack{(a,F) \in \mathcal{F}: \\ 2 \in F_i^1}} v_{(a,F)} \frac{u_{i2}(a)}{u_{i1}} + \sum_{\substack{(a,F) \in \mathcal{F}: \\ 3 \in F_i^1}} v_{(a,F)} \frac{u_{i3}(a)}{u_{i1}} \leq g_{i1} \quad \forall i \in E \tag{4.17}$$

$$\sum_{\substack{(a,F) \in \mathcal{F}: \\ 2 \in F_i^0}} v_{(a,F)} + \sum_{\substack{(a,F) \in \mathcal{F}: \\ 3 \in F_i^1}} v_{(a,F)} \frac{u_{i3}(a)}{u_{i2}} \leq g_{i2} \qquad \forall i \in E, \tag{4.18}$$

$$\sum_{\substack{(a,F) \in \mathcal{F}: \\ 3 \in F_i^0}} v_{(a,F)} \leq g_{i3} \qquad \forall i \in E, \tag{4.19}$$

$$v_{(a,F)} \geq 0 \qquad \forall (a, F) \in \mathcal{F}, \quad (4.20)$$

donde la restricción (4.17), (4.18) y (4.19) se generan al tomar $j = 1$, $j = 2$ y $j = 3$, respectivamente.

Como mencionamos antes, al correr nuestro algoritmo primal-dual podemos interpretar las restricciones como contenedores que se irán llenando, donde la restricción (4.19) es el contenedor de la última parte de la función de costos de cada elemento y que cuando éste se llene, entonces esa parte es lo suficientemente buena para ser utilizada. Al ver la restricción (4.18), vemos que además del término que se ve en la restricción anterior, aparece un nuevo término que tiene que ver con la parte 3. Este término será utilizado en el algoritmo para llenar más rápido el contenedor 2, siempre y cuando sepamos que la parte 3 es buena. Esto viene de la idea de que si la última parte es muy barata (por ejemplo costo cero), entonces la segunda parte será más atractiva para agregarla en la solución final que antes. La constante que aparece en este nuevo término hace que el apoyo de la parte 3 en la parte 2 sea proporcional al cubrimiento máximo de la parte 3 e inversamente proporcional al de la parte 2. Esto significa que si es mucho lo que se cubre de manera eficiente en la parte tres, entonces el beneficio que gano al utilizar esta parte es muy alto. Ademas, si la parte 2 es muy grande, entonces el costo de agregarla es muy alto y el beneficio de la parte 3 se ve aminorado.

El efecto de apoyarse, se va arrastrando hacia atrás hasta llegar a la primera parte, en donde todos podrían llegar a apoyar a la primera parte. Por ejemplo en una función cóncava, el apoyo se propaga hasta que el elemento es ingresado. Pero por otro lado, si un elemento del medio es bueno, entonces éste apoyará al anterior y puede que la parte de más adelante sea muy mala, lo que hace que ese elemento no necesariamente se utilice completamente como se ve en la figura 4.7. Podemos ver en esta figura que la segunda parte apoya a la primera y que como la tercera no es buena, puede que ni siquiera ingrese a la solución final.

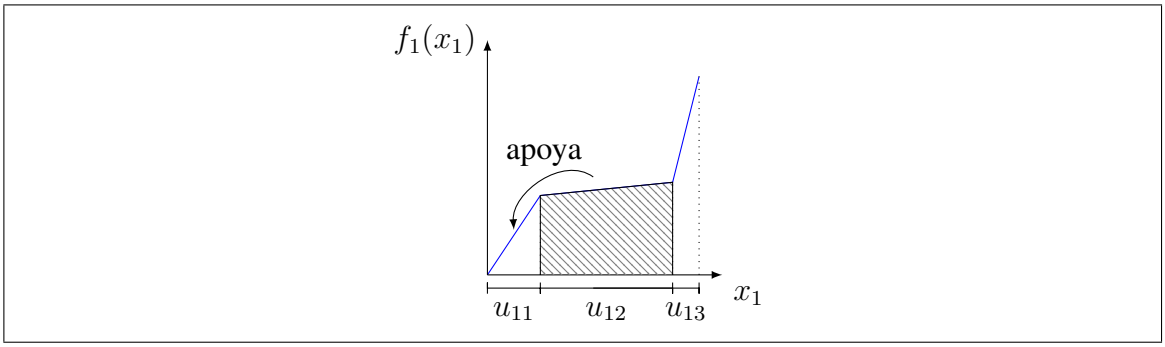

Figura 4.7. Apoyo de una parte a otra en un elemento. La parte achurada cumple su restricción con igualdad y apoya a la parte de la izquierda.

Con esta idea de apoyar podemos construir un algoritmo primal-dual, el cual partirá con una solución primal vacía e infactible y una solución dual factible con $v = \vec{0}$. La idea al igual que en los algoritmos anteriores es no tener que utilizar todas las variables (cantidad exponencial) y preocuparnos solo de las variables que sean mayores a 0, ya que estas siempre serán una cantidad polinomial. Partiremos con $a = \vec{0}$, lo que nos dará la solución inicial vacía. Además, partimos con las partes de cada elemento en los conjuntos con *shift* 0, es decir, con $F_i^0 = \{1, .., m\}$ y con los demás conjuntos F_i^k's vacíos. Las variables v y z parten como 0. x

La decripción precisa del proceso se encuentra en el Algoritmo 3, que tiene 3 partes importantes. La primera es la línea 6, la cual nos dice que la parte j-ésima del elemento i es lo suficientemente buena para ingresar al algoritmo, en el contexto en que los elementos que expresa a ya están en la solución final. La segunda, va de la línea 7 a la 17 y nos dice que si todas las partes anteriores a j del elemento i escogidas en la línea 6 ya se encuentran contempladas por a, entonces ingresaremos la parte j del elemento i a la solución final y todas las partes que la apoyaban (a menos de que la demanda se cubra antes), ya que esas partes pasaron por la línea 6 con anterioridad. Por último, en la tercera parte, de las líneas 18 a la 23, nos dice que si el elemento j' anterior a la parte j (escogida en la línea 6) del elemento i aún no forma parte de la solución final, entonces la parte j apoyará a j' junto a todas las partes que apoyaban a j. Finalmente, cuando el algoritmo sale del *while* tendremos una solución factible del problema $[\text{IP}_3]$.

Algoritmo 3 Algoritmo Primal-Dual para Knapsack Cover con funciones de costos continuas y lineales por tramos.

$z, v \leftarrow 0$;
$a \leftarrow \vec{0}$;
$F_i^k \leftarrow \emptyset \ \ \forall i, j$;
$F_i^0 \leftarrow \{1, .., m\}$;
while $D(a) > 0$ **do**
  Incrementar $v_{a,F}$ hasta que una restricción dual se cumpla con igualdad para un item i y una parte j;
  if $a_i = j - 1$ **then**
    Tomar j' como el mínimo j'' tal que $j'' > j$ y que $j'' \in F_i^0$
    for $j'' = j..(j' - 1)$ **do**
      if $D(a) > 0$ **then**
        $z_{ij''} \leftarrow u_{ij''}(a)$;
        $a_i \leftarrow j''$;
        quitar j'' del F_i^k en el que esté;
      else
        terminar **for**;
      end if
    end for
  else
    Tomar j' como el máximo j'' tal que $j'' < j$ y que $j'' \in F_i^0$;
    Tomar j''' como el mínimo j'' tal que $j'' > j$ y que $j'' \in F_i^0$;
    for $j'' = j..(j''' - 1)$ **do**
      $F_i^{(j''-j)} \leftarrow F_i^{(j''-j)} \setminus \{j''\}$ y $F_i^{(j''-j')} \leftarrow F_i^{(j''-j')} \cup \{j''\}$
    end for
  end if
end while
return $(\bar{a}, \bar{F}, \bar{z}) \leftarrow (a, F, z)$;

Teorema 4.1. *El Algoritmo 3 termina con una solución de costo no mas grande que* $2opt_{[IP_3]}$.

DEMOSTRACIÓN. Sea (q, ℓ) el par (elemento,parte) que se alcanzo con igualdad en la última iteración del Algoritmo 3, $(\bar{a}, \bar{z})$ la solución final, p la última parte utilizada de q (no necesariamente $\ell = p$, de hecho $q = \bar{a}_q$) y $(\bar{b}, \bar{F})$ el vector de la última componente de v que tomo valor mayor a cero, es decir, $\bar{b}$es igual a $\bar{a}$ sin los elementos que ingresaron en la última iteración. Podemos ver que si $v_{a,F} > 0$, entonces $a \leq \bar{b} = \bar{a}|_{\bar{a}_q = \ell - 1}$. Esto, porque las demás variables duales nunca pasaron por el algoritmo y tienen valor 0, como

ocurre para $\bar{a}$. Además, para cada par (elemento,parte) (i,j) con $\bar{z}_{ij} > 0$, tenemos que la restricción (4.14) se cumple con igualdad.

Entonces tenemos que

$$\sum_{i\in E}\sum_{j\in J}\bar{z}_{ij}g_{ij} = \sum_{i\in E}\sum_{j\in J}\bar{z}_{ij}\sum_{k=0}^{m-j}\sum_{\substack{(a,F)\in\mathcal{F}:\\(j+k)\in F_i^k}}\tau_a(i,j+k,k)v_{(a,F)}.$$

Lo que podemos trabajar para intercambiar las sumatorias

$$\begin{aligned}
\sum_{i\in E}\sum_{j\in J}\bar{z}_{ij}g_{ij} &= \sum_{i\in E}\sum_{j\in J}\bar{z}_{ij}\sum_{k=0}^{m-j}\sum_{\substack{(a,F)\in\mathcal{F}:\\(j+k)\in F_i^k}}\tau_a(i,j+k,k)v_{(a,F)} \\
&= \sum_{i\in E}\sum_{j\in J}\bar{z}_{ij}\sum_{k=0}^{m-j}\sum_{(a,F)\in\mathcal{F}}\mathbb{1}_{\{(j+k)\in F_i^k\}}\tau_a(i,j+k,k)v_{(a,F)} \\
&= \sum_{(a,F)\in\mathcal{F}}v_{(a,F)}\underbrace{\sum_{i\in E}\sum_{j\in J}\bar{z}_{ij}\sum_{k=0}^{m-j}\mathbb{1}_{\{(j+k)\in F_i^k\}}\tau_a(i,j+k,k)}_{(4.12)} \\
&= \sum_{(a,F)\in\mathcal{F}}v_{(a,F)}\underbrace{\left[\sum_{i\in E}\sum_{k=0}^{m-1}\sum_{j'\in F_i^k}\tau_a(i,j',k)\bar{z}_{i(j'-k)}\right]}_{(*)},
\end{aligned}$$

donde primero colocamos las condiciones de las sumatorias con indicatrices para poder intercambiar las sumatorias que son independientes entre sí, para luego usar la ecuación (4.12) para que nos quede la última ecuación.

Podemos ver que el algoritmo hace que $\bar{z}_{ij} = u_{ij}$ para todo $i \in E$ y $j \leq \bar{a}_i$ a excepción de $(i,j) = (q,p)$ que podría estar truncado, ya que si existe otro par (elemento,parte) truncado, entonces el algoritmo hubiese terminado antes. Con esto en mente, podemos

separar el aporte de (q,p) en la ecuación $(*)$, lo que nos queda

$$(*) = \left[\underbrace{\sum_{i\in E\setminus\{q\}} \sum_{k=0}^{m-1} \sum_{j\in F_i^k} \tau_a(i,j,k)\bar{z}_{i(j-k)} + \sum_{k=0}^{m-1} \sum_{\substack{j\in F_q^k:\\ j<p}} \tau_a(q,j,k)\bar{z}_{q(j-k)}}_{(I)} + \underbrace{\beta_{(q,p)}(a,F)}_{(II)} \right],$$

donde $\beta_{(q,p)}(a,F)$ es el aporte del par (elemento,parte) (q,p) en la suma $(*)$ y se define por

$$\beta_{(q,p)}(a,F) = \sum_{k=0}^{m-1} \mathbb{1}_{\{p\in F_q^k\}} \tau_a(q,p,k)\bar{z}_{q(p-k)}.$$

Podemos ver que por la forma de F tenemos que $\beta_{(q,p)}(a,F)$ es un solo término, y es justamente el término relacionado con el F_q^k que contiene la parte p en cada iteración del algoritmo.

Podemos ver que si logramos demostrar que $(I) \leq D(a)$ y $(II) \leq D(a)$, entonces tendremos el resultado deseado, ya que tendremos que

$$\sum_{i\in E}\sum_{j\in J} \bar{z}_{ij} g_{ij} \leq 2 \sum_{(a,F)\in\mathcal{F}} v_{(a,F)} D(a) \leq 2opt_{[\mathrm{DP}_3]} \leq 2opt_{[\mathrm{LP}_3]} \leq 2opt_{[\mathrm{IP}_3]}.$$

Para lograr esto, tenemos que fijarnos en que

$$\tau_a(i,j,k)\bar{z}_{i(j-k)} = \begin{cases} \bar{z}_{ij} & \text{, si } k = 0, \\ \frac{u_{ij}(a)}{u_{i(j-k)}} \bar{z}_{i(j-k)} & \text{, si } 0 < k \leq m - a_i - 1, \\ 0 & \text{, si } k \geq m - a_i \end{cases} \tag{4.21}$$

y usando que si z_{ij} toma valor, es u_{ij} o 0 en (I) tenemos que

$$\tau_a(i,j,k)\bar{z}_{i(j-k)} = \begin{cases} u_{ij} & \text{, si } \bar{z}_{ij} > 0 \wedge k = 0, \\ u_{ij}(a) & \text{, si } \bar{z}_{ij} > 0 \wedge 0 < k \leq m - a_i - 1, \\ 0 & \text{, si } \bar{z}_{ij} = 0 \vee k \geq m - a_i, \end{cases}$$

lo que significa que

$$\tau_a(i,j,k)\bar{z}_{i(j-k)} \leq \begin{cases} u_{ij} & \text{, si } \bar{z}_{ij} > 0 \wedge k \leq m - a_i - 1, \\ 0 & \text{, si } \bar{z}_{ij} = 0 \vee k \geq m - a_i. \end{cases} \tag{4.22}$$

Demostremos por partes:

$a)$ Tenemos que demostrar que

$$(I) = \sum_{i \in E \setminus \{q\}} \sum_{k=0}^{m-1} \sum_{j \in F_i^k} \tau_a(i,j,k)\bar{z}_{i(j-k)} + \sum_{k=0}^{m-1} \sum_{\substack{j \in F_q^k: \\ j<p}} \tau_a(q,j,k)\bar{z}_{q(j-k)} \leq D(a).$$

Sea $v(a,F) \geq 0$ la variable dual para la cual suma (I) y es fija. Usando (4.22) tenemos que

$$\begin{aligned} (I) &= \sum_{i \in E \setminus \{q\}} \sum_{k=0}^{m-1} \sum_{j \in F_i^k} \tau_a(i,j,k)\bar{z}_{i(j-k)} + \sum_{k=0}^{m-1} \sum_{\substack{j \in F_q^k: \\ j<p}} \tau_a(q,j,k)\bar{z}_{q(j-k)} \\ &\leq \sum_{i \in E \setminus \{q\}} \sum_{k=0}^{m-1} \sum_{j \in F_i^k} u_{ij} \mathbb{1}_{\{\bar{z}_{ij}>0\}} + \sum_{k=0}^{m-1} \sum_{\substack{j \in F_q^k: \\ j<p}} u_{qj} \mathbb{1}_{\{\bar{z}_{qj}>0\}} \\ &= \sum_{i \in E} \sum_{k=0}^{m-1} \sum_{j \in F_i^k} u_{ij} \mathbb{1}_{\{\bar{z}_{ij}>0\}} \mathbb{1}_{\{(i,j) \neq (q,p)\}}, \end{aligned}$$

donde la última igualdad se da porque para los $j > p$ tenemos que $z_{qj} = 0$.
Ahora, podemos ver que esta última expresión suma sobre todos los F_i^k para un elemento i dado, entonces por definición de los F validos, se tiene que cumplir que $\bigcup_{k=0}^{m-1} F_i^k = \{a_i + 1, .., m\}$. Con esto, nos queda que

$$(I) \leq \sum_{i \in E} \sum_{k=0}^{m-1} \sum_{j=a_i+1}^{m} u_{ij} \mathbb{1}_{\{\bar{z}_{ij}>0\}} \mathbb{1}_{\{(i,j) \neq (q,p)\}}.$$

Esta ecuación nos dice que (I) es menor o igual a lo que cubre la solución final $\bar{z}$ sin la última parte ingresada por el algoritmo, es decir, la parte p del elemento q.

Además, como el algoritmo no termina hasta ingresar ésta última parte, sabemos que en cada iteración, esta expresión es $\leq D(a)$.

$b)$ Tenemos que demostrar que

$$(II) = \beta_{(q,p)}(a, F) \leq D(a).$$

Sea $v(a, F)$ la variable dual para la cual suma (II) y es fija. Usando la definición de β y la ecuación (4.21) tenemos que

$$\begin{aligned}(II) &= \beta_{(q,p)}(a, F) \\ &= \sum_{k=0}^{m-1} \mathbb{1}_{\{p \in F_q^k\}} \tau_a(q, p, k) \bar{z}_{q(p-k)} \\ &\leq u_{qp}(a),\end{aligned}$$

ya que el elemento p esta en un solo F_q^k.
En el caso en que $\tau_a(q, p, k)\bar{z}_{q(p-k)} = \bar{z}_{qp}$ tenemos que $\bar{z}_{qp} = u_{qp}(\bar{b}) \leq u_{qp}(a)$ para cualquier $a \leq \bar{b}$ que como vimos antes, todas las variables que tienen $v(a, F) > 0$ cumplen con esto. Y en el caso en que $k > 0$ tenemos que $\tau_a(q, p, k)\bar{z}_{q(p-k)} = \frac{u_{qp}(a)}{u_{q(p-k)}} \bar{z}_{q(p-k)} < u_{qp}(a)$ ya que $\bar{z}_{q(p-k)} \leq u_{q(p-k)}$. Luego, por definición de cubrimiento truncado tenemos que $u_{qp}(a) \leq D(a)$.

Con esto terminamos la demostración y tenemos que la solución entregada por el algoritmo es una 2-aproximación. □

Finalmente, tenemos un algoritmo que entrega una solución que es una 2-aproximación, pero para que sea realmente útil, debemos mostrar que corre en tiempo polinomial.

Como tenemos que en cada iteración del Algoritmo 3 al menos una parte ingresa a la solución final (líneas 7-17) o al menos una parte da algún elemento se mueve de un conjunto F_i^0 a un F_i^k con $k > 0$ (líneas 18-23), podemos analizar la cantidad total de pasos. Primero, cada vez que el algoritmo pasa por las líneas 7-13 ingresa una parte de un algoritmo, por lo que la cantidad máxima de veces que puede pasar por acá esta acotado por

nm. Segundo, cada vez que el algoritmo pasa líneas 18-23 una parte j de algún elemento i deja el conjunto F_i^0, por lo que la cantidad máxima de veces que el algoritmo puede usar estas líneas es nm. Tercero, por cada iteración del *while* en la línea 5, el algoritmo deberá pasar por una de las dos partes mencionadas anteriormente, podemos ver que la cantidad máxima de iteraciones del algoritmo completo está acotado por $2nm$. Cuarto, las líneas 8,19 y 20 son polinomiales, ya que hay que buscar un mínimo de a lo mas m enteros, lo que nos da $O(mlog(m))$ que es la complejidad de ordenar una lista de tamaño m. Quinto, la línea 6 puede ser reescrita como una expresión cerrada, donde hay que buscar un mínimo en las restricciones que son mn en total, lo que nuevamente ordenando los valores para encontrar el mínimo nos da $O(nmlog(nm))$. En resumen, asumiendo que declarar, modificar o comparar variables y conjuntos es $O(1)$, tenemos que el algoritmo tiene una complejidad de $O((nm)^2 log(nm))$, lo que esta contenido en $O((nm)^3)$ que a su vez es polinomial.

Como MK es un caso particular que puede ser modelado con este modelo de manera exacta, podemos utilizar el mismo ejemplo del Capítulo 3 para demostrar el siguiente corolario.

Corolario 4.1. *El gap de integralidad* $\frac{\sup[IP_3]}{\inf[LP_3]}$ *es 2.*

En la siguiente sección mostraremos una interpretación física del algoritmo para ayudar a la comprensión de como funciona el Algoritmo 3.

4.4. Interpretación Física

Como el dual es un problema de empaquetamiento, podemos imaginarnos un compartimiento o *bin* por cada parte de cada función. La idea principal de esta interpretación es que estos contenedores se irán llenando de líquido y estarán ordenados de manera escalonada, de tal manera que cuando un contenedor se llena, el agua escurrirá hacia abajo y se irán llenando más rápido los contenedores a los cuales el agua caiga.

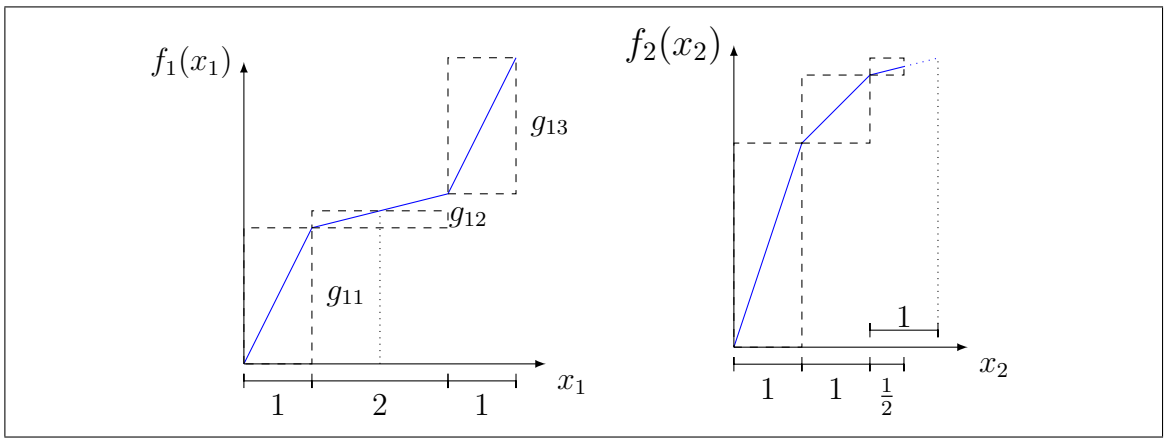

Figura 4.8. Ejemplos de contenedores.

Primero, debemos imaginar que está lloviendo y que los contenedores están ordenados de manera escalonada como se puede ver en la figura 4.8. Los contenedores tendrán un ancho de u_{ij} y alto de g_{ij} que recordemos que es la pendiente o el costo de servir una unidad de demanda en ese tramo j del elemento i. Como se ve en la figura 4.8 el alto del contenedor puede superar la altura que alcanza función si $u_{ij} < 1$, lo importante es que la parte superior de los contenedores formen una escalera.

A medida que llueve, los contenedores irán llenándose de agua, en el momento en que un contenedor se llena por primera vez, ocurrirá una de las siguientes opciones. Si por un lado, el contenedor que se llena no es el del nivel inferior de su escalera, entonces al llenarse, este elemento se mueve de conjunto en el algoritmo, lo que se traduce en que si recibe más agua por efecto de la lluvia, esta agua escurrirá por la escalera y llenará más rápido algún contenedor que esté por debajo. Por ejemplo, en el caso en que el contenedor que está directamente debajo se encuentre lleno, el agua escurrirá hacia el siguiente, y así sucesivamente. En el caso de que no esté lleno, el contenedor de abajo se beneficiará del el agua escurrida y se llenará más rápido. Este fenómeno es el que se mencionó en el algoritmo como “apoyar”, ya que cuando se que una parte del final de un elemento es bueno (se llena antes), entonces este apoya a sus predecesores para llegar lo antes posible a la solución final.

Si una parte j de un elemento i esta apoyando a otra parte j' anterior, entonces por cada unidad de altura (de agua) que suba la j', la parte j apoyará con $\frac{u_{ij}}{u_{ij'}}$ unidades de altura.

Por otro lado, si el contenedor que se llenó es el primero en la escalera que queda de algún elemento (tomando en cuenta los contenedoras que aun no se llenan), entonces se incluye inmediatamente esa parte a la solución final junto con los contenedores que lo apoyaban. De esta manera, se irá modificando el vector a hasta cubrir la demanda.

El hecho de que alguna parte j de algún elemento i se trunque cuando su cubrimiento supere la demanda residual, se traducirá en que a ese contenedor se le tapará la parte truncada en el ancho del contenedor, de tal manera que la lluvia que cae sobre esa parte escurrirá hacia los lados y no hacia abajo por la escalera. De esta manera, si este contenedor apoya a una parte anterior, la cantidad de agua que escurrirá será menor.

CAPÍTULO 5. PROBLEMA DE MK CON FUNCIONES DE COSTOS NO LINEALES

Para generalizar más nuestro problema, debemos dar un modelo para funciones más generales. Consideremos funciones de costo $f_i : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ para cada elemento $i \in E$ donde cada una es continua por la izquierda, no decreciente, acotada y con una cantidad finita de saltos. Las funciones de costos estarán dadas por un oráculo de manera implícita. Además, asumiremos que existe un polinomio p que cuando se le entrega una entrada de tamaño x al oráculo, este nos devuelve una respuesta de tamaño a lo más $p(x)$. Dado un valor v, el oráculo entregara el valor más grande del dominio tal que su imagen sea menor o igual v. Es decir, para un elemento i, el oráculo correspondiente entregará el valor

$$\mathcal{O}_i(v) = \text{máx}\{x : f_i(x) \leq v\}.$$

Además, tenemos una cantidad $\delta_i > 0$ que se asume parte de la entrada. Este valor es menor a cualquier $f_i(x)$ y cualquier x que retorne el oráculo luego de ingresarle un valor $v > 0$. Podemos pensar en δ_i como la precisión de maquina de donde se implemente el algoritmo. Finalmente, asumiremos que para cada elemento $i \in E$ tendremos un valor γ_i en la entrada que es una cota superior de f_i.

Dado $\varepsilon > 0$, la idea principal será aproximar cada función f_i por una función g_i que diste a lo más por un factor $(1 + \varepsilon)$. Es decir, que para cada i se cumpla con que

$$f_i(x) \leq g_i(x) \leq (1 + \varepsilon) f(x), \tag{5.1}$$

para cada x en el dominio de las funciones.

Posteriormente, encontraremos funciones h_i, que serán continuas y lineales por tramo y tales que aproximen a g_i de manera que sean iguales en todas las posibles soluciones que entregue el algoritmo, manteniendo así la Propiedad (5.1). Finalmente, con estas funciones podemos correr el Algoritmo 3, el cual nos entregará una solución de nuestro problema original. Es decir, si usamos el presupuesto definido para cada elemento, tendremos una solución factible del problema original ya que las funciones originales utilizarán un costo

menor o igual al entregado en algoritmo por la Propiedad (5.1). Finalmente, demostraremos que esta solución es $(2+\varepsilon)$-aproximada.

5.1. Aproximación

En esta sección mostraremos como aproximar una función f_i, lo que se extiende para todos los elementos i en I. Como mostraremos que la cantidad de variables que usaremos en el algoritmo será polinomial para cada una elemento, tendremos que la cantidad total será polinomial también.

Dado un $\varepsilon > 0$, primero debemos tomar una partición por cada función f_i dada por

$$P_i = \{0, \delta_i, \delta_i(1+\varepsilon), \delta_i(1+\varepsilon)^2, .., \delta_i(1+\varepsilon)^{k_i}\},$$

donde k_i es un numero menor a $\log_{(1+\varepsilon)}(\frac{\gamma_i}{\delta_i})$, donde γ_i es el valor máximo para cada función f_i y δ_i es el valor mínimo > 0 dado por el oráculo. Con esto, tenemos que la cantidad total de tramos es polinomial en el tamaño de la entrada y por lo tanto, tendremos una cantidad polinomial de partes, lo que se traducirán en variables en el modelo que ingresaremos al algoritmo.

Luego, podemos tomar los valores que nos entrega el oráculo para cada valor $v \in P_i$ y tendremos que g_i se definirá por

$$g_i(x) = \begin{cases} 0 & x \in [0, \mathcal{O}_i(0)], \\ \delta_i & x \in (\mathcal{O}_i(0), \mathcal{O}_i(\delta_i)], \\ \delta_i(1+\varepsilon) & x \in (\mathcal{O}_i(\delta_i), \mathcal{O}_i(\delta_i(1+\varepsilon))], \\ \quad \vdots & \\ \delta_i(1+\varepsilon)^{k_i} & x \in (\mathcal{O}_i(\delta_i(1+\varepsilon)^{(k-1)}), \mathcal{O}_i(\delta_i(1+\varepsilon)^k)], \end{cases}$$

donde cabe destacar que la función vale 0 en $x = 0$ y que si una discontinuidad traspasa más de un intervalo de la partición P_i, entonces el conjunto donde la función toma los

valores de esos intervalos será vacío, ya que dará algo de la forma $(a, a] = \emptyset$. Además, con esta definición, nos aseguramos que g_i cumple la Propiedad (5.1).

Finalmente, para cada discontinuidad, definiremos un tramo lineal de largo $\zeta > 0$ en el eje x, dejando la discontinuidad en el medio y haciendo que las funciones h_i's sean continuas. Podemos tomar $\zeta = \frac{1}{2c_{mcm}}$, donde c_{mcm} es el mínimo común múltiplo de los denominadores de los valores en el eje x donde cada las funciones g_i's tienen discontinuidades y D al tomarlos como fracciones. Con esta valor de ζ aseguramos que el Algoritmo 3 siempre terminará con una solución que no utiliza elementos que se ocupen hasta uno de estos tramos ficticios de largo ζ, como veremos en el Teorema 5.1 más adelante.

Entonces, tenemos que cuando el algoritmo nos retorne una solución $\bar{z}$, como ninguna función de costo usará estos tramos nuevos, nos queda que

$$\sum_{i \in E} h_i(\bar{z}_i) = \sum_{i \in E} g_i(\bar{z}_i),$$

lo que cumple con la Propiedad 5.1 y nos deja que

$$\sum_{i \in E} f_i(\bar{z}_i) \leq \sum_{i \in E} h_i(\bar{z}_i) \leq \sum_{i \in E} (1 + \varepsilon) f_i(\bar{z}_i).$$

Para asegurarnos que el algoritmo nos entregará una buena solución, tenemos el siguiente teorema.

Teorema 5.1. *Tomando $\zeta = \frac{1}{2c_{mcm}}$, el Algoritmo 3 siempre entregará una solución $\bar{z}$ tal que $g_i(\bar{z}_i) = h_i(\bar{z}_i)$ para todo $i \in E$.*

Donde c_{mcm} es el mínimo común múltiplo de los denominadores de los valores en el eje x donde cada las funciones g_i's tienen discontinuidades y D al tomarlos como fracciones.

DEMOSTRACIÓN. Esto se traduce en que el algoritmo nunca entregará una solución que para alguna función g_i se utilice una parte fraccionaria de una aproximación de la discontinuidad. Para ver esto, podemos tomar el valor D y los valores en el eje x y amplificarlos por la constante c_{mcm}, lo que nos asegura que todas las discontinuidades se

alcancen en valores enteros y que D sea entero también. Esto es lo mismo que amplificar ambos lados de la ecuación de cubrimiento de tal manera que D y cada u_{ij} queden con valores enteros. Luego, podemos tomar $\zeta = \frac{1}{2}$ y $\bar{z}$ la solución que entrega el algoritmo. Observemos que como la única variable, relacionada a un tramo de la función lineal por partes, que puede ser tomada menor a u_{ij} es la última en ingresar al algoritmo, entonces existen dos posibilidades:

$a)$ El último tramo en ingresar es uno ficticio creado por una discontinuidad.

$b)$ El último tramo no es uno ficticio. En este caso, tenemos de manera directa que $g_i(\bar{z}_i) = h_i(\bar{z}_i)$ para todo $i \in E$.

Analizando lo que pasa más en detalle para el caso $a)$, vamos a concluir, que realmente no puede ocurrir.

Sea (q, p) el par (elemento,parte) ingresado en la última iteración. Como D es entero, y $\bar{z}$ es una solución factible tenemos que las siguientes expresiones son equivalentes,

$$\sum_{i \in E} \sum_{j \in J} \bar{z}_{ij} \geq D$$

$$\left\lfloor \sum_{i \in E} \sum_{j \in J} \bar{z}_{ij} \right\rfloor \geq \lfloor D \rfloor$$

$$\left\lfloor \sum_{i \in E} \sum_{j \in J} \mathbb{1}_{\{i \neq p \wedge j \neq p \wedge \bar{z}_{ij} > 0\}} u_{ij} + \bar{z}_{qp} \right\rfloor \geq D$$

$$\sum_{i \in E} \sum_{j \in J} \mathbb{1}_{\{i \neq p \wedge j \neq p \wedge \bar{z}_{ij} > 0\}} u_{ij} \geq D,$$

donde $\mathbb{1}_t$ es la indicatriz que vale 1 cuando la proposición lógica t es cierta y 0 en otro caso. Podemos notar que la ultima igualdad se da porque $\bar{z}_{qp} \leq \zeta < 1$, lo que contradice el algoritmo, ya que hubiese parado antes de la última iteración.

Con esto demostramos lo pedido, ya que la solución siempre entregará una solución del escenario $b)$.

Luego, redefiniendo ζ se tiene el resultado esperado. $\square$

Utilizando el teorema anterior, tenemos que existe $\zeta > 0$ tal que el Algoritmo 3 entregará soluciones con la Propiedad (5.1). En consecuencia directa del algoritmo anterior, tenemos el siguiente lema.

Lema 5.1. *Se puede resolver el problema de Knapsack Cover con funciones de costo no decrecientes, continuas por la izquierda y con una cantidad finita de discontinuidades a través del Algoritmo 3, cuya solución es una $(2+\varepsilon)$-aproximación.*

DEMOSTRACIÓN. Sea (x^*, OPT_f) una solución óptima y el valor óptimo del problema de la mochila con funciones de costo f_i's, OPT_h el valor óptimo si usamos funciones de costo h_i y $\varepsilon > 0$ una constante fija. Ahora, podemos aproximar las funciones f_i's a funciones h_i's como se mencionó anteriormente y luego correr el Algoritmo 3 para que nos entregue una solución $\bar{z}$. Esta solución es una 2-aproximación del problema de *knapsack cover* con funciones de costo h_i's, es decir,

$$\sum_{i \in E} h_i(\bar{z}_i) \leq 2opt_h,$$

donde $z_i = \sum_{j \in J} z_{ij}$ para cada $i \in E$. Además, como se cumple la Propiedad (5.1), tenemos que

$$\begin{aligned}
\sum_{i \in E} f_i(\bar{z}_i) &\leq \sum_{i \in E} h_i(z_i) \\
&\leq 2opt_h \\
&\leq 2 \sum_{i \in E} h_i(x_i^*) \\
&\leq 2(1+\varepsilon) \sum_{i \in E} f_i(x_i^*) \\
&= 2(1+\varepsilon) opt_f.
\end{aligned}$$

Luego, reescalando ε adecuadamente, tenemos que $\bar{z}$ es una $(2+\epsilon)$-aproximación del problema de la mochila con funciones de costos f_i's. $\square$

CAPÍTULO 6. RESULTADOS EXPERIMENTALES

En esta sección queremos ver como se desempeña el algoritmo empíricamente en una simplificación de una de sus posibles aplicaciones. Para esto tomaremos como referencia el Sistema Interconectado del Norte Grande (SING), que contiene una gran cantidad de centrales de energía térmica, para la cuales existe bibliografía para poder aproximar de buena manera sus funciones de costo.

El SING tiene una capacidad instalada en energía térmica de 5.294,77 MW (*Capacidad Instalada de Generación - SING*, s.f.), lo que representa un 88,37 % de la capacidad máxima total del SING como se ve en la Tabla 6.1. Según el último reporte anual de La Asociación de Generadoras de Chile (AGC), que reúne a las principales generadoras de energía eléctrica del país, la generación eléctrica del SING fue de 19.466 GWh, del cual el 94 % fue generado por plantas térmicas, el 5 % por solares y el 1 % por eólicas. Es por esto que nos centraremos en las plantas de generación térmica de el SING.

La distribución de la capacidad máxima instalada de estas plantas es muy variado como se puede ver en la Figura 6.1, donde en el eje y esta la capacidad máxima instalada en MW y en el eje x las distintas instalaciones de energía termoeléctrica en operación al 24

Tabla 6.1. Capacidad máxima instalada en el Sistema Interconectado del Norte Grande según tipo de generación.

Tipo de Energía	**Total Suma de Potencia Bruta [MW]**
Carbón	2.670,64
Gas Natural	2.341,16
Solar	544,6
Fuel Oil Nro. 6	148,28
Petróleo Diesel	134,69
Eólica	90
Geotérmica	27,5
Cogeneración	17,5
Hidráulica Pasada	10,89
Mini Hidroeléctrica Pasada	6,25
Total SING	5.991,51

de enero del 2018. Usaremos estas plantas para ver como se comporta nuestro algoritmo. Trataremos a las dos primeras plantas como plantas grandes, ya que tienen una mayor capacidad instalada, las 5 que siguen serán medianas y el resto serán consideradas como plantas pequeñas.

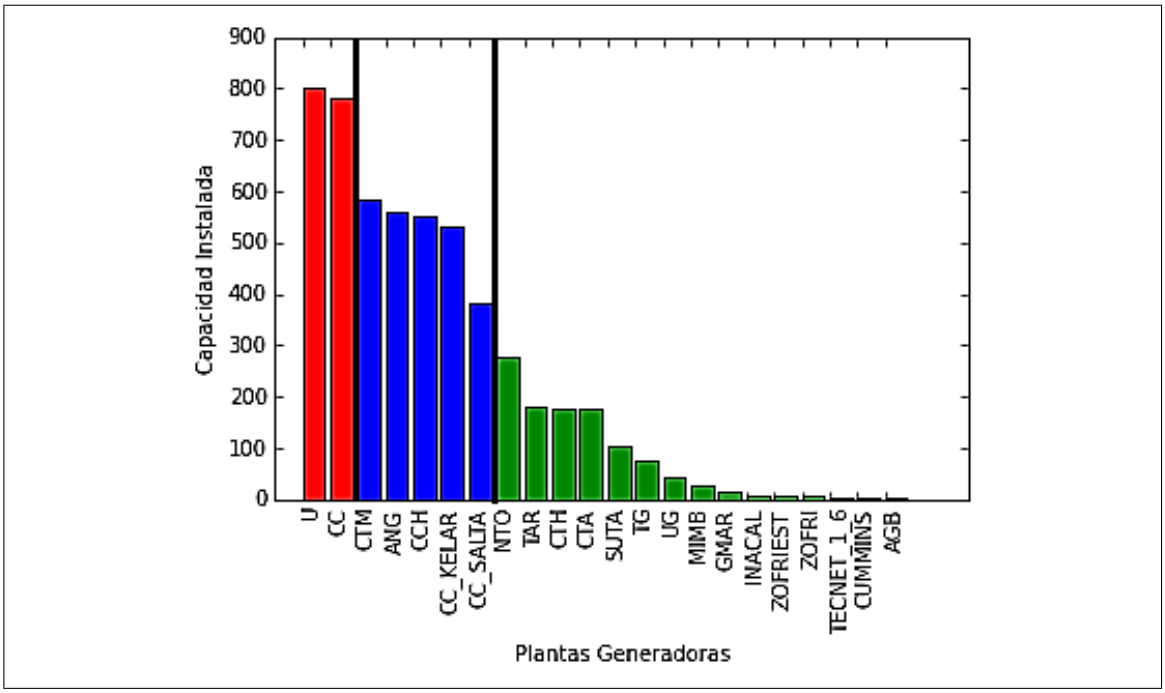

Figura 6.1. Capacidad máxima instalada de las plantas de generación termoeléctrica del SING.

La plantas termoeléctricas tienen estudios muy acabados de sus funciones de costos, también llamadas *Input-Output Characteristic* en inglés. En el área de energía se conoce que estas funciones son convexas y generalmente son no lineales (Zhu, 2015) dentro de su rango de producción. Usualmente se asume que estas funciones son cuadráticas y que tienen un mínimo y un máximo de producción técnica. Notemos que pueden haber discontinuidades en torno al mínimo técnico. La función de costo de una planta se calcula en base a una de las siguientes fuentes de información:

a) Experimentos para obtener la eficiencia de la planta.

b) Data histórica de los costos de generación de la planta.

c) Información dada por la empresa que diseña la planta de producción.

Como no contamos con esta información, ya que generalmente es de índole privada, lo que haremos es simular estas funciones de costos con distintos supuestos.

Para encontrar el mínimo técnico de producción podemos utilizar que generalmente este valor está determinado por factores de la caldera y la turbina. La limitación mínima de la caldera son generados normalmente por temas de la estabilidad de la combustión y generalmente rondan entre 25-70 % de la capacidad de diseño y la limitación de la turbina normalmente vienen dadas por restricciones del diseño y rondan entre los 10-15 % (Zhu, 2015). Por esto, en nuestras simulaciones, calcularemos el mínimo técnico de producción como $P_{min} = e_c e_t P_{max}$, donde P_{max} es la producción máxima, e_c es la eficiencia de la caldera que será una variable aleatoria uniforme entre $0{,}25$ y $0{,}7$, y e_t la eficiencia de la turbina que será una variable aleatoria uniforme entre $0{,}1$ y $0{,}15$.

Como no conocemos exactamente los parámetros de las funciones cuadráticas y el costo fijo de cada planta, propondremos supuestos y veremos como se comporta el algoritmo para cada uno de los escenarios.

6.1. Supuestos

Tomaremos los siguientes supuestos para generar nuestra simulación.

I) Las funciones de costos de cada planta tendrán su mínimo técnico en el vértice de la función cuadrática, de modo de poder tener una idea de eficiencia con el parámetro que acompaña al término cuadrático.

II) Las plantas más grandes son mas eficientes para producir energía.

III) Mientras más grande son las plantas de producción, más grande es el costo fijo.

Si $x_i^{\text{mín}}$ es el mínimo técnico de la planta i, entonces $f_i(x_i) = 0$ si $x_i \leq x_i^{\text{mín}}$. Como las funciones de costo tendrán la forma $f_i(x_i) = a_i x_i^2 + b_i x_i + c_i$ para $x_i > x_i^{\text{mín}}$, el primer

supuesto se traduce en

$$x_i^{\text{mín}} = \frac{-b_i}{2a_i}. \tag{6.1}$$

Una vez que fijemos el costo fijo cf_i de la planta i, tendremos la siguiente ecuación

$$cf_i = a_i(x_i^{\text{mín}})^2 + b_i x_i^{\text{mín}} + c_i. \tag{6.2}$$

Aún con estas ecuaciones tenemos un grado de libertad, ya que tenemos tres incógnitas y dos ecuaciones. Entonces, fijaremos la constante a_i teniendo en cuenta el supuesto $II)$ y como el costo puede verse de manera relativa, no importa si los montos que usaremos están en el mismo orden de magnitud que en el SING, ya que no especificaremos la unidad de medida del costo. Recordemos que este experimento es para ver el desempeño empírico del algoritmo en ciertos escenarios cercanos a la realidad de una posible aplicación.

Para nuestra simulación, tomaremos que el parámetro a_i viene dado por una variable uniforme con parámetros distintos para cada tamaño de plantas. Es decir, los parámetros vienen dado por instancias de las siguientes variables aleatorias:

$a)$ Plantas pequeñas: $a_i \sim U(0{,}0075; 0{,}0105)$.
$b)$ Plantas medianas: $a_i \sim U(0{,}005; 0{,}008)$.
$c)$ Plantas grandes: $a_i \sim U(0{,}0035; 0{,}0065)$.

Podemos ver que las plantas medianas se pueden cruzar en eficiencia con las grandes, esto lo tomamos de esa manera dado que las medianas no tienen tanta diferencia con las grandes en cuanto a la capacidad instalada. Las medias de a para los tipos de plantas son $0,009$, $0,0065$ y $0,005$, respectivamente.

Una vez teniendo las constantes a_i's, utilizando la ecuación (6.1) podemos calcular la constante $b_i = -2a_i x_i^{\text{mín}}$ para cada función i.

Finalmente, para fijar el costo fijo, tomaremos los siguientes escenarios:

$i)$ Costos fijos bajos y similares entre los distintos grupos de plantas. Para esto, definiremos que el costo fijo será igual al costo variable de producir el 50, 20 y 20 % de la producción máxima para las plantas pequeñas, medianas y grandes respectivamente.

$ii)$ Costos fijos bajos y distintos entre los distintos grupos de plantas, es decir, el costo fijo es proporcional al tamaño de la planta. Para esto, definiremos que el costo fijo será igual al costo variable de producir el 20 % de la producción máxima para cada planta.

$iii)$ Costos fijos altos y similares entre los distintos grupos de plantas. Para esto se tomarán los parámetros de $i)$ y se le sumara a los costos fijos, una constante igual a 10 veces el costo fijo mas alto (entre las distintas plantas).

$iv)$ Costos fijos altos y distintos entre los grupos de plantas como en $ii)$. Para esto, definiremos que el costo fijo será igual al costo variable de producir el 50 % de la producción máxima para cada planta.

En el Apéndice podemos ver ejemplos de como se ven las instancias para distintos escenarios de costos fijos. Además, en la parte superior de cada gráfico, se encuentra la semilla que puede ser utilizada para repetir el mismo experimento.

6.2. Metodología

Para realizar estos experimentos, se utilizó Python 2.7, utilizando los paquetes de Numpy para el manejo de estructuras numéricas, Mathplotlib para los gráficos y Time para medir los tiempos de ejecución. El script se corrió en un computador Asus N56vj, con un procesador Intel(R) Core(TM) i7-3630QM CPU de 2.4 GHz, 4 procesadores principales, 8 procesadores lógicos y 8 Gygabytes de memoria RAM. El detalle del código se puede encontrar en mi GitHub https://github.com/inmorales/.

Para aproximar las funciones cuadráticas f_i por funciones lineales y continuas g_i, primero tenemos que definir lo que ocurre entre 0 y el mínimo técnico. Para esto, definimos

que la nueva función valdrá 0 en $x = 0$ y tomaremos un pequeño δ tal que $g_i(x) = f_i(x_i^{\text{mín}})$ para todo $x \in [\delta, x_i^{\text{mín}}]$ y en el intervalo $(0, \delta)$ definimos una función lineal de pendiente muy alta, que haga que g_i sea continua. Luego, para el intervalo entre el mínimo y máximo técnico, tomamos una división uniforme en $m > 0$ tramos, es decir, cada tramo tendrá el tamaño $\frac{x_i^{\text{máx}} - x_i^{\text{mín}}}{m}$ y de tal manera que la función g_i vale exactamente lo mismo que la función original en los extremos de los intervalos y es una recta que une esos puntos dentro del intervalo. De esta manera, podemos asegurar que la función g_i es mayor o igual a f_i en todo punto y extiende a la función f_i desde 0 hasta el mínimo técnico de manera que incurre en los mismos costos para utilizar el mínimo técnico.

Esta no es la manera mas eficiente de aproximar la función (en el sentido de minimizar la cantidad de tramos), pero como nuestro objetivo es ver como se comporta el algoritmo en una situación real, esta aproximación nos bastará para ese objetivo. Para asegurar un error de aproximación $\varepsilon > 0$ se iterará en m aumentándolo gradualmente hasta lograr un error de a lo más un factor de $1 + \varepsilon$, en nuestros experimentos, fijamos $\varepsilon = 5\,\%$.

Para obtener el desempeño del algoritmo, vamos a ver el gap entre la solución optima y la solución entregada por nuestro algoritmo, es decir, mientras mas cercano a 1 (óptimo) sea el gap, mejor será nuestro algoritmo. Como es necesario resolver el problema entero para obtener el valor exacto del gap de integralidad, buscaremos una cota superior, de modo de saber aproximadamente el desempeño del algoritmo. Lo que haremos es tomar el valor de la solución final del dual de nuestro algoritmo, lo que nos da una cota inferior del costo de la solución óptima de la relajación, lo que a su vez es una cota inferior de la solución integral óptima. Con esto tenemos que una cota superior del gap de integralidad viene dado por tomar el costo de nuestra solución final dividido por el costo de la solución dual.

Esta idea es la misma que se utilizó en la demostración. Con esto, podemos observar empíricamente que tan buena o mala es la solución que tenemos para cada instancia, ya que la demostración nos da una 2-aproximación en el peor de los casos, pero puede que en promedio, el algoritmo tenga un mucho mejor desempeño.

6.3. Resultados

Para efectos de los resultados, tomaremos 3 escenarios de demanda, la que será el 25, 50 y 75 % de la suma de las capacidades máxima de las plantas. Esto será tomado como subescenarios 1, 2 y 3 respectivamente. Es decir, el escenario i,1) es el dado por el escenario i) en los costos fijos y con demanda del 25 % de la capacidad total.

Los resultados de cada escenario y subescenario se pueden ver en la Tabla 6.2, donde se corrieron 100 instancias para cada subescenario, con m (número de intervalos de la aproximación) definido para que el error máximo estuviese acotado por 5 %. Además, se puede observar el desempeño dado por el gap promedio, máximo y la desviación estándar, y por el error máximo de las instancias. La columna "Error" en la tabla, corresponde al valor de $\varepsilon > 0$ correspondiente al error al aproximar las funciones no lineales por lineales por tramos, interpretado como un porcentaje. Para obtener éste error se iteró en m para asegurar un error máximo de $(1 + \varepsilon)$ en la aproximación de las funciones de costos originales por funciones continuas y lineales por tramo.

Para saber el desempeño real de una instancia, podemos multiplicar el gap empírico por $1 + \varepsilon$, lo que nos daría una cota superior del error al estar tomando esa decisión o solución entregada por el algoritmo.

Primero, podemos notar que la aproximación de una función linear por tramos constante necesita más detalle para obtener el error menor al 5 % en los escenarios i) y ii), debido a que en estos casos el costo fijo es menor y por eso las funciones a aproximar parten más cercanas al eje $y = 0$. Esto ocurre porque la holgura para la distancia entre la función aproximada y la función real para un $\varepsilon > 0$ fijo al acercarnos al 0 va disminuyendo.

Segundo, podemos ver que el costo fijo afecta directamente al tiempo de ejecución, ya que las instancias i) y ii) (costo fijo bajo) tienen casi el mismo tiempo de ejecución que las instancias iii) y iv) (costos fijos altos) respectivamente, mientras que las primeras

Tabla 6.2. Resultado de los experimentos computacionales

Escenario	m	Gap			Error	Tiempo [seg]	
		Promedio	Máximo	Desv. Est.	Máximo	Promedio	Desv. Est.
i.1)	12	1,00606	1,06557	0,0126	4,9 %	3,52	0,19
i.2)	12	1,00022	1,01778	0,0018	4,91 %	4,04	0,12
i.3)	12	1	1	0	4,91 %	4,08	0,11
ii.1)	12	1,00631	1,05099	0,0118	4.92 %	1,41	0,04
ii.2)	12	1	1	0	4,92 %	1,48	0,04
ii.3)	12	1	1	0	4,91 %	1,54	0,03
iii.1)	5	1,04118	1,21702	0,0652	4,95 %	3,9	1.68
iii.2)	5	1,02101	1,08892	0,0251	4,98 %	3,94	1,74
iii.3)	5	1,00224	1,02825	0,0057	4,97 %	4,42	1,88
iv.1)	6	1,02516	1,13928	0,0349	3,85 %	0,91	0,03
iv.2)	6	1,00685	1,09342	0,0170	3,85 %	1	0,02
iv.3)	6	1,00021	1,02083	0,0021	3,85 %	1.01	0,01

tienen un m de 12 (lo que significa mayor cantidad de variables) y las segundas tienen m de 5 y 6. Por lo tanto, si corremos las instancias de $iii)$ y $iv)$ con m de 12, podríamos ver que su error sería muy bajo, pero a costas de un alto tiempo de ejecución. Los costos fijos (se puede observar en el Apéndice) de los escenarios $i)$ y $ii)$ rondan entre los 50 y 100 en el eje y, del escenario $iv)$ está entre los 100 y 1000, y del escenario $iii)$ ronda los 1500. La diferencia en el tiempo de ejecución viene dado porque al tener un costo fijo tan alto, el algoritmo debe iterar más veces antes de comenzar a ingresar elementos a la solución, mientras que en los escenarios $i)$ y $ii)$ los elementos se van ingresando casi inmediatamente a la solución y se logra encontrar una solución factible más deprisa. La estructura de las soluciones en el escenario $iii)$ son de ingresar elementos completos o casi completos hasta llegar a una solución factibles. Por otro lado, podemos ver que el algoritmo no aumenta tanto en el tiempo de ejecución al aumentar la demanda exigida para cubrir.

Tercero, podemos ver que el algoritmo funciona notablemente mejor que la cota teórica del peor caso, ya que en promedio el gap es muy bueno (cercano a 1) y si vemos el peor de los casos, no supera el $7\,\%$, $6\,\%$, $22\,\%$ y $14\,\%$ en los distintos escenarios respectivamente, mientras que en promedio, no supera el 4,2 %. Además, si observamos la desviación

estándar, podemos ver que los valores altos de error máximo (en los escenarios iii,1) y iv,1)) se dan con poca probabilidad, ya que el promedio es mucho menor y la desviación estándar es muy baja. Ahora si vemos el error total del algoritmo, en promedio es cercano al 9,2 % dado por un 4 % en el gap de aproximación y un 5 % en la aproximación de las funciones. Mientras que resolver esto no tarda más de 5 segundos por instancia.

En resumen, podemos concluir que este tipo de desigualdades funciona realmente bien en la práctica para funciones cuadráticas, que es el la forma de modelar las centrales termoeléctricas. Estos resultados nos dan una idea de que es recomendable seguir esta línea de investigación para analizar como extender este algoritmo o el uso de estas desigualdades para otros problemas que podrían ser mas complejos, como por ejemplo el problema de generación eléctrica en un horizonte de tiempo dado, donde las decisiones que se toman en un periodo de tiempo afectan en el punto de partida del siguiente periodo, ya que los costos de prender las maquinas que ya están prendidas se vería disminuido o podría ser 0 y podría agregarse un costo de modificación de las maquinas actualmente andando.

Además, cabe destacar que para problemas donde la demanda a cubrir es un gran porcentaje de la suma de las capacidades máximas, entonces este algoritmo puede llegar a encontrar el óptimo del problema lineal por tramos, y que el único error vendría dado por la aproximación de las funciones de costos reales.

CAPÍTULO 7. CONCLUSIONES

En esta tesis de magister hemos generalizado las famosas *knapsack cover inequalities* para poder abordar un problema de cubrimiento de demanda por elementos de tamaño variable según una función de costos no decreciente y continua por la izquierda. Éste tipo de funciones nos dan mucha más flexibilidad para poder utilizar estas desigualdades generalizadas en nuevas aplicaciones o poder modelar, de manera más cercana a la realidad, las aplicaciones donde ya se utilizan las *knapsack cover inequalities* originales.

Con el algoritmo primal-dual mostrado, podemos asegurar que el poliedro, que queda al usar nuestras desigualdades sobre el problema de la mochila con funciones de costos no lineales, tiene un gap de integralidad acotado por $(2 + \varepsilon)$, con $\varepsilon > 0$ definido antes de correr el algoritmo. Además, como vimos en el capítulo de resultados experimentales, esta aproximación da un mejor desempeño cuando se utilizan funciones cuadráticas, como ocurre al decidir qué plantas termoeléctricas utilizar para generar energía. Como vimos, el gap de integralidad con las funciones lineales por partes nos dio un error promedio que no supero el 4 %, y fijando $\varepsilon = 5\,\%$, entonces el error total es de un 9,2 % y que cuando la demanda es más cercana a la capacidad total (como en el caso de la demanda eléctrica en Chile), el algoritmo funcionaba mucho mejor, llegando a veces a encontrar el óptimo en muy poco tiempo.

Esperamos que este tipo de desigualdades se puedan utilizar para mejorar la relajación de muchos problemas que utilicen este tipo de restricciones de cubrimiento con funciones generales. Cuando este tipo de restricción aparece, pueden ser reemplazadas con este set de desigualdades que planteamos para poder luego buscar un algoritmo que logre resolver el modelo relajado, siempre teniendo en cuenta que el algoritmo no puede utilizar todas las variables, ya que aumentan en una cantidad exponencial.

Por ejemplo, una posible aplicación directa son los problemas que utilizan las *knapsack cover inequalities* que mencionamos en la Introducción. Un ejemplo más tangible

de esto es la red inteligente para autos eléctricos, donde al utilizar nuestras desigualdades, permitirían que los agregadores del modelo puedan recibir funciones de costo mas compleja de parte de los generadores o de los mismos automóviles que tienen energía almacenada y que no la necesitan utilizar en el corto plazo.

Además, cabe destacar que nuestras desigualdades mantienen la limitación, al igual que las desigualdades originales, de que la constante de aproximación es ajustada. Otra limitación actual de estas desigualdades es que solo sirven para problemas 1-dimensionales, pero esperamos que esto se pueda extender para tener restricciones de cubrimiento más generales.

7.1. Trabajo Futuro

El trabajo futuro viene dado por dos grandes líneas principales. La primera es poder identificar los problemas donde este tipo de desigualdades puede ser un aporte al momento de resolver problemas tanto teóricos como aplicados. La dificultad de esto es poder analizar y encontrar algoritmos que puedan resolver los poliedros relajados que aparecen, ya que normalmente los problemas no tendrán solo una restricción, lo que nos dejará que el dual puede no ser tan abordable, lo que a primeras dificulta el utilizar un algoritmo primal dual como el que se presenta en esta tesis. Por esta razón, sería interesante contar con otra manera de explotar estas desigualdades. Por ejemplo, queda la interrogante si es que podemos desarrollar un algoritmo basado en redondeo de una solución fraccionaria para encontrar una $(2 + \varepsilon)$-aproximación. Esta línea de ideas ha sido fructifera al atacar distintos problemas con relajaciones basadas en las *knapsack cover inequalities*, por ejemplo para el Generalized Scheduling Problem (Bansal y Pruhs, 2014), por lo que también tiene potencial para atacar problemas basados en nuestra relajación. Notemos por otra parte que para poder utilizar técnicas de redondeo necesitamos poder resolver un programa lineal con una cantidad exponencial de restricciones, por lo que necesitamos además desarrollar algoritmos para separar nuestra familia de restricciones en tiempo polinomial, al menos de manera approximada.

La segunda línea de trabajo viene dada por aplicar nuestro algoritmo en problemas con dependencias temporales, al igual que el algoritmo primal dual para el problema de *Minimum Knapsack* fue extendido para resolver el problema de *Unsplittable Flow-Cover Problem on a Path* (Bar-Noy et al., 2001; Höhn et al., 2014) (UFP-Cover), donde alcanzan una 4-aproximación al utilizar de manera inteligente las *knapsack cover inequalities* y la estructura del problema.

Notemos que el problema de UFP-Cover con funciones de costos no lineales tiene una cierta semejanza con el problema de generación de energía dinámico, donde se debe decidir el nivel de energía generado por cada planta en cada instante de tiempo. Esperamos que para UFP-Cover con funciones de costos no lineales se encuentre un algoritmo $(4 + \varepsilon)$- aproximado, utilizando técnicas similares a las utilizadas para UFP-Cover, pero con nuestras nuevas desigualdades. De ser así sería interesante ver si es que es posible obtener un algoritmo de aproximación con un factor constante para el problema de generación eléctrica dinámico.

BIBLIOGRAFIA

Agrawal, A., Klein, P., y Ravi, R. (1995). When trees collide: An approximation algorithm for the generalized steiner problem on networks. *SIAM Journal on Computing*, *24*(3), 440–456.

Bansal, N., y Pruhs, K. (2014). The Geometry of Scheduling. *SIAM Journal on Computing*, *43*(5), 1684-1698.

Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., y Schieber, B. (2001). A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, *48*(5), 1069–1090.

Bar-Yehuda, R., y Even, S. (1981). A linear-time approximation algorithm for the weighted vertex cover problem. *Journal of Algorithms*, *2*(2), 198–203.

Bertsimas, D., y Teo, C.-P. (1998). From valid inequalities to heuristics: A unified view of primal-dual approximation algorithms in covering problems. *Operations Research*, *46*(4), 503–514.

Capacidad instalada de generación - SING. (s.f.). `http://datos.energiaabierta.cl/datastreams/94351/capacidad-instalada-de-generacion-sing/`. (Acceso: 2018-25-01)

Carnes, T., y Shmoys, D. B. (2015). Primal-dual schema for capacitated covering problems. *Mathematical Programming*, *153*(2), 289–308.

Carr, R. D., Fleischer, L. K., Leung, V. J., y Phillips, C. A. (2000). Strengthening integrality gaps for capacitated network design and covering problems. En *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 106–115).

Cheung, M., Mestre, J., Shmoys, D. B., y Verschae, J. (2017). A Primal-Dual Approximation Algorithm for Min-Sum Single-Machine Scheduling Problems. *SIAM Journal on Discrete Mathematics*, *31*(2).

Chvatal, V. (1979). A greedy heuristic for the set-covering problem. *Mathematics of operations research*, *4*(3), 233–235.

Dantzig, G. B., Ford Jr, L. R., y Fulkerson, D. R. (1956). *A primal–dual algorithm* (Inf. Téc.). RAND CORP SANTA MONICA CA.

Goemans, M. X., y Williamson, D. P. (1995). A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, *24*(2), 296–317.

Goemans, M. X., y Williamson, D. P. (1997). The primal-dual method for approximation algorithms and its application to network design problems. *Approximation algorithms for NP-hard problems*, 144–191.

Höhn, W., Mestre, J., y Wiese, A. (2014). How unsplittable-flow-covering helps scheduling with job-dependent cost functions. En *International Colloquium on Automata, Languages, and Programming* (pp. 625–636).

Karp, R. M. (1972). Reducibility among combinatorial problems. En *Complexity of Computer Computations* (pp. 85–103). Springer.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, *2*(1-2), 83–97.

Levi, R., Roundy, R. O., y Shmoys, D. B. (2006). Primal-dual algorithms for deterministic inventory problems. *Mathematics of Operations Research*, *31*(2), 267–284.

Li, S. (2017). Constant approximation algorithm for non-uniform capacitated multi-item lot-sizing via strong covering inequalities. En *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (pp. 2311–2325).

McCormick, S. T., Peis, B., Verschae, J., y Wierz, A. (2017). Primal–dual algorithms for precedence constrained covering problems. *Algorithmica*, *78*(3), 771–787.

Takazawa, Y., y Mizuno, S. (2017). A 2-approximation algorithm for the minimum knapsack problem with a forcing graph. *Journal of the Operations Research Society of Japan*, *60*(1), 15–23.

Zhong, W., Xie, K., Liu, Y., Yang, C., y Xie, S. (2017). Efficient auction mechanisms for two-layer vehicle-to-grid energy trading in smart grid. En *IEEE International Conference on Communications* (pp. 1–6).

Zhu, J. (2015). *Optimization of power system operation* (Vol. 47). John Wiley & Sons.

ANEXOS

A. ANEXOS

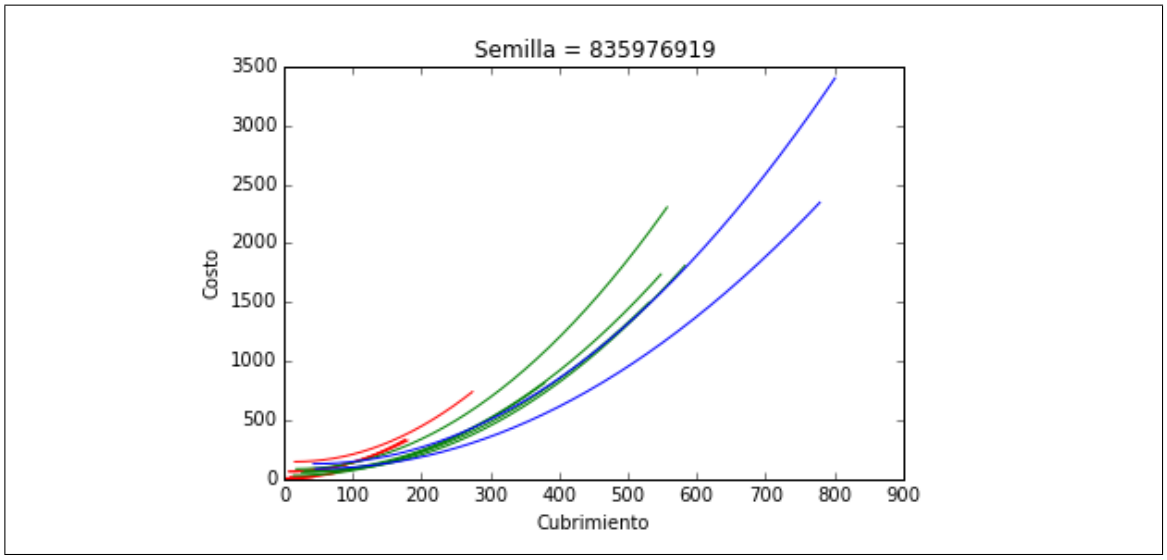

Figura A.1. Ejemplo de escenario i,1)

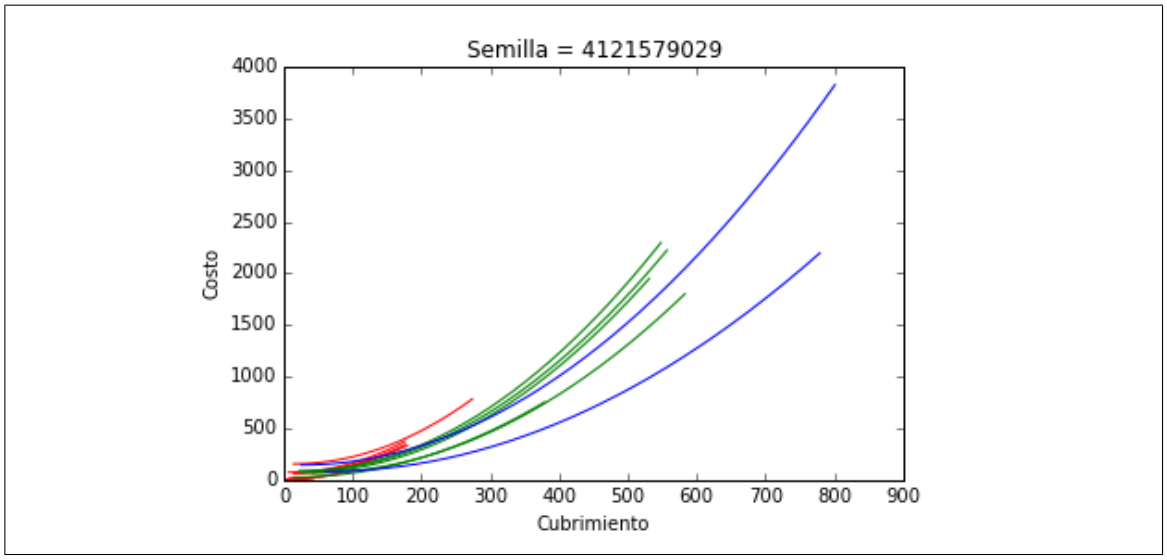

Figura A.2. Ejemplo de escenario i,2)

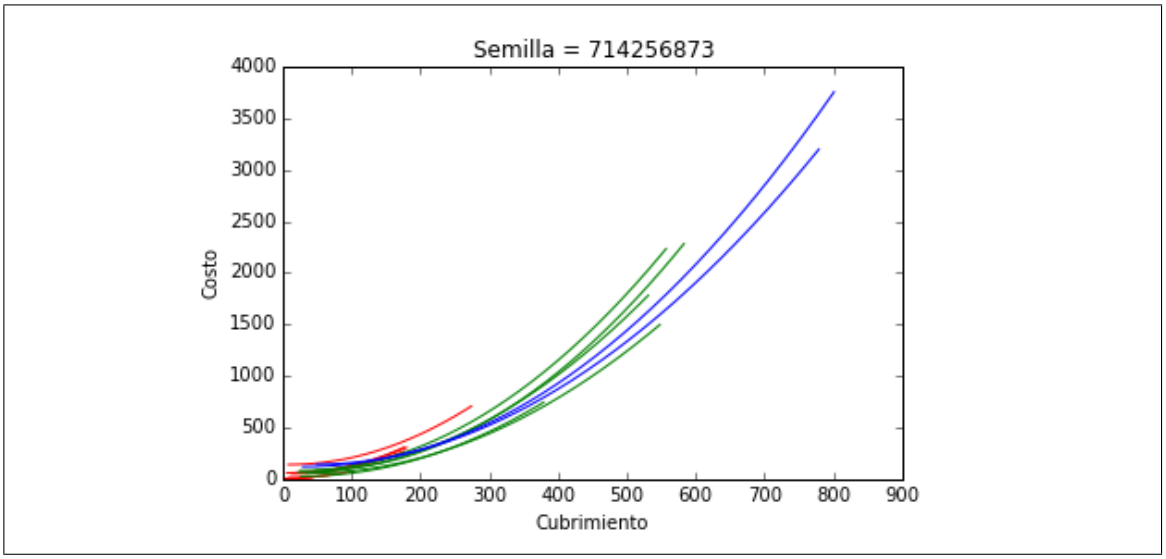

Figura A.3. Ejemplo de escenario i,3)

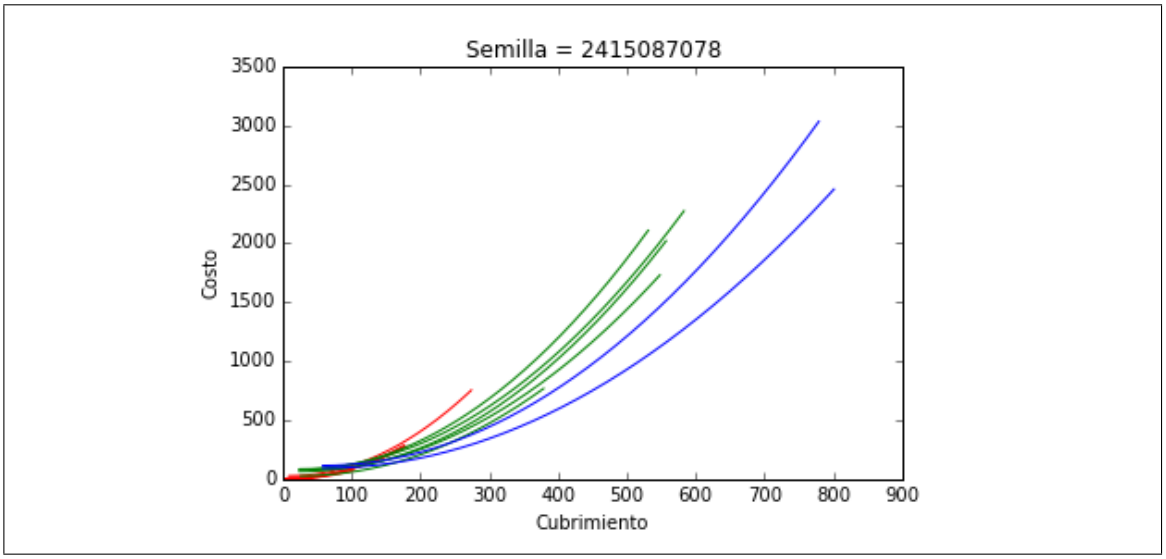

Figura A.4. Ejemplo de escenario ii,1)

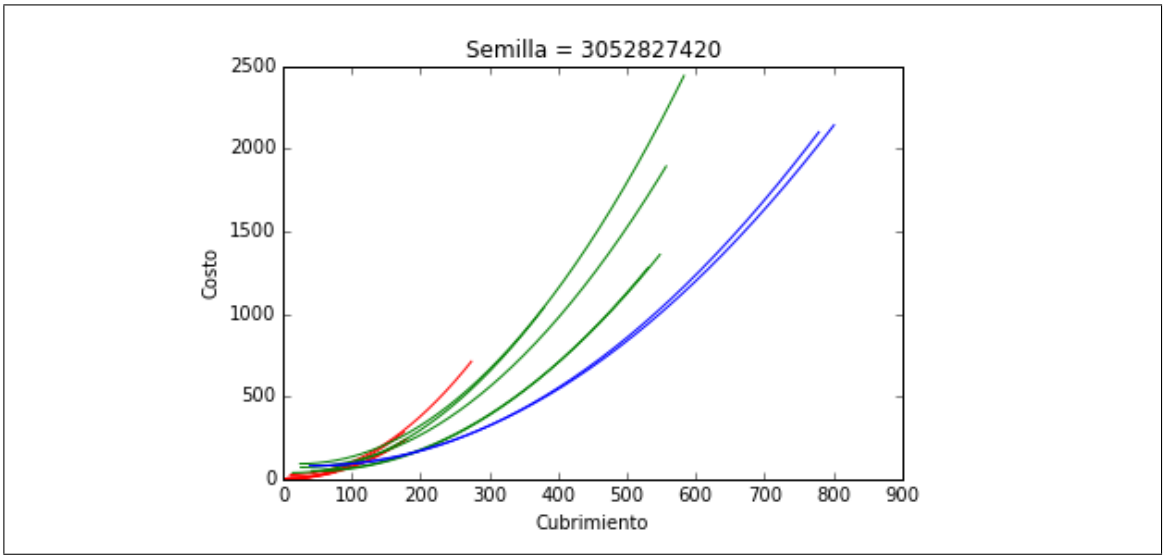

Figura A.5. Ejemplo de escenario ii,2)

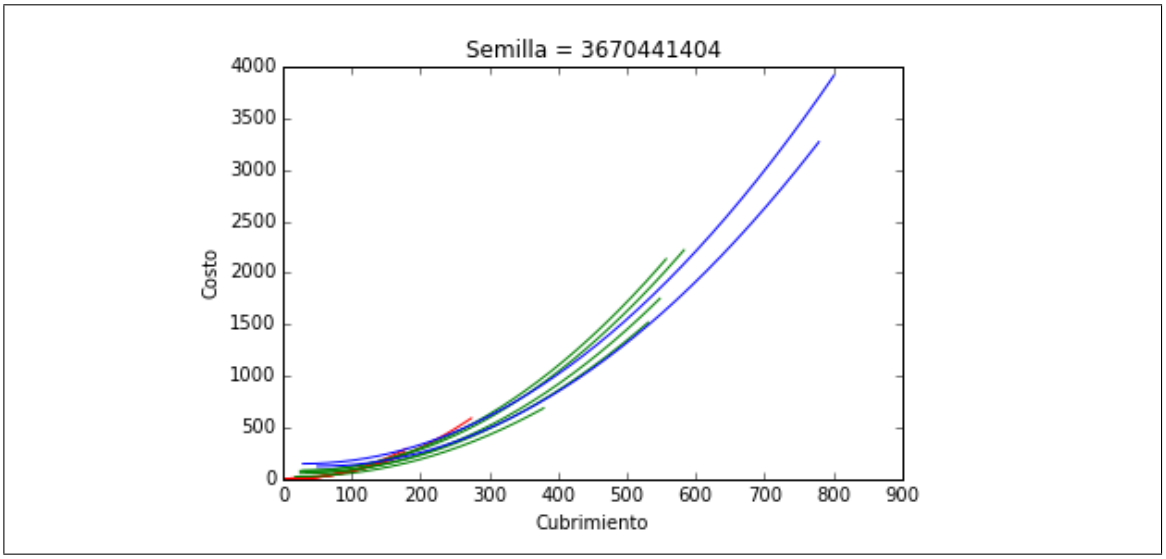

Figura A.6. Ejemplo de escenario ii,3)

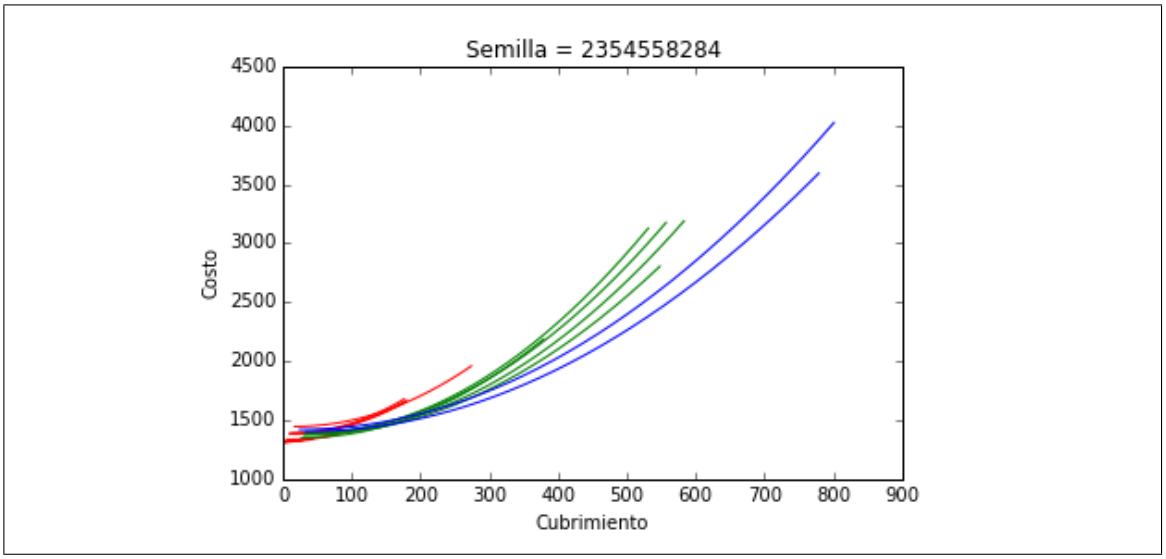

Figura A.7. Ejemplo de escenario iii,1)

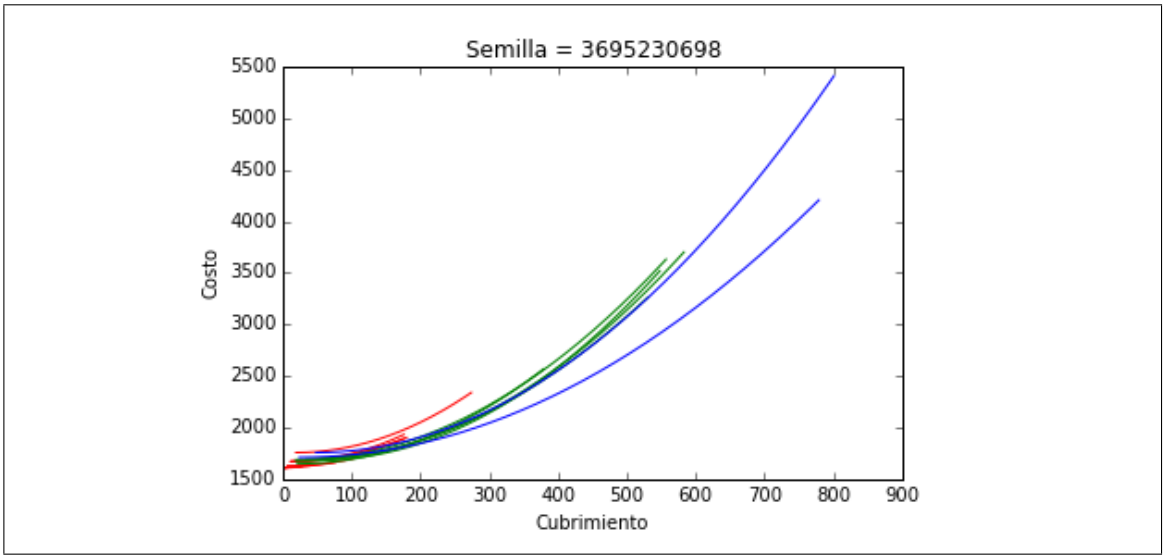

Figura A.8. Ejemplo de escenario iii,2)

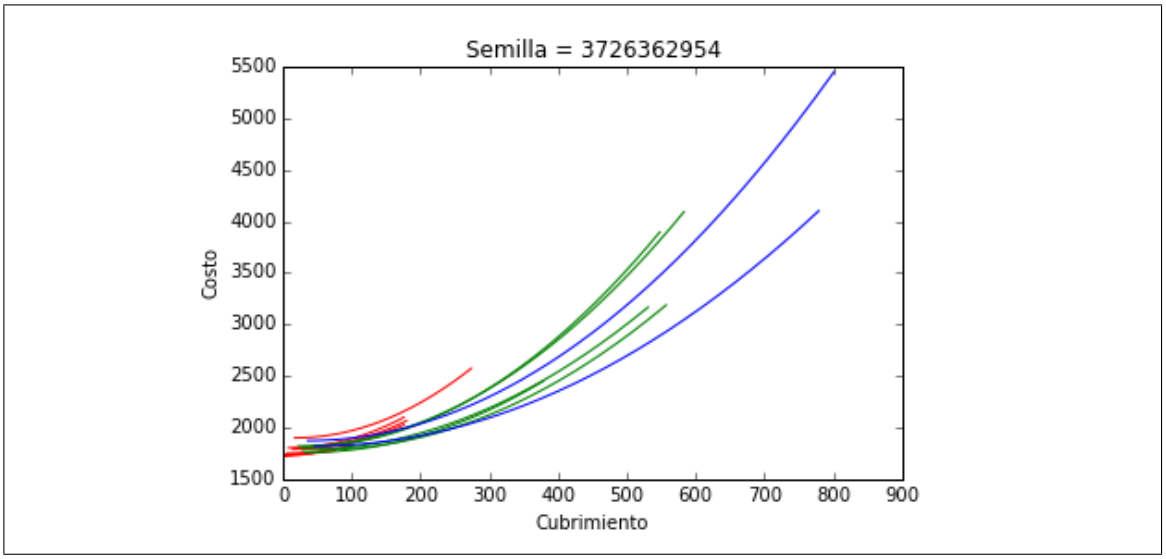

Figura A.9. Ejemplo de escenario iii,3)

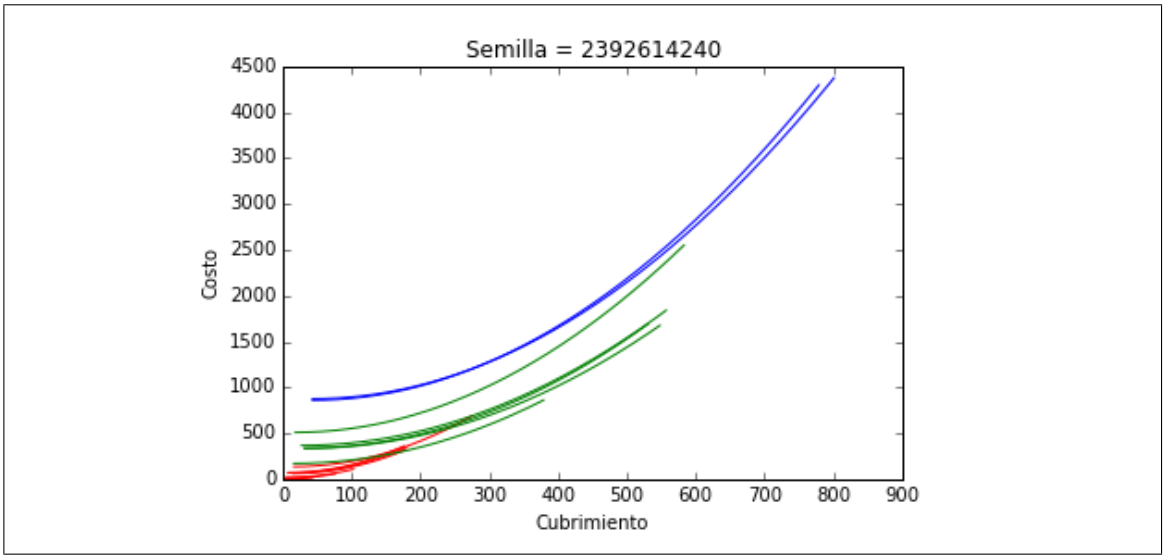

Figura A.10. Ejemplo de escenario iv,1)

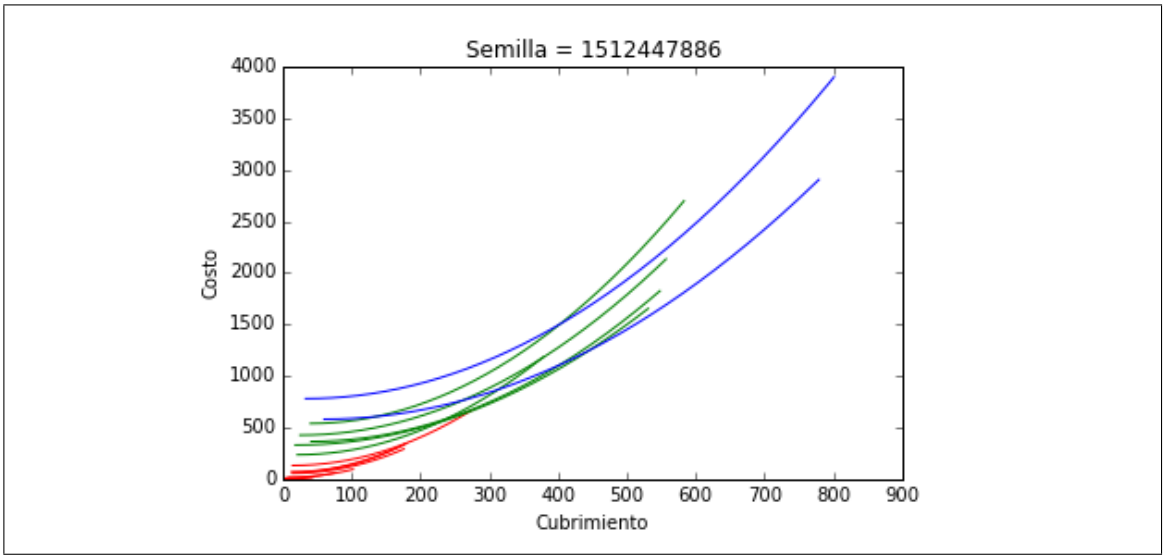

Figura A.11. Ejemplo de escenario iv,2)

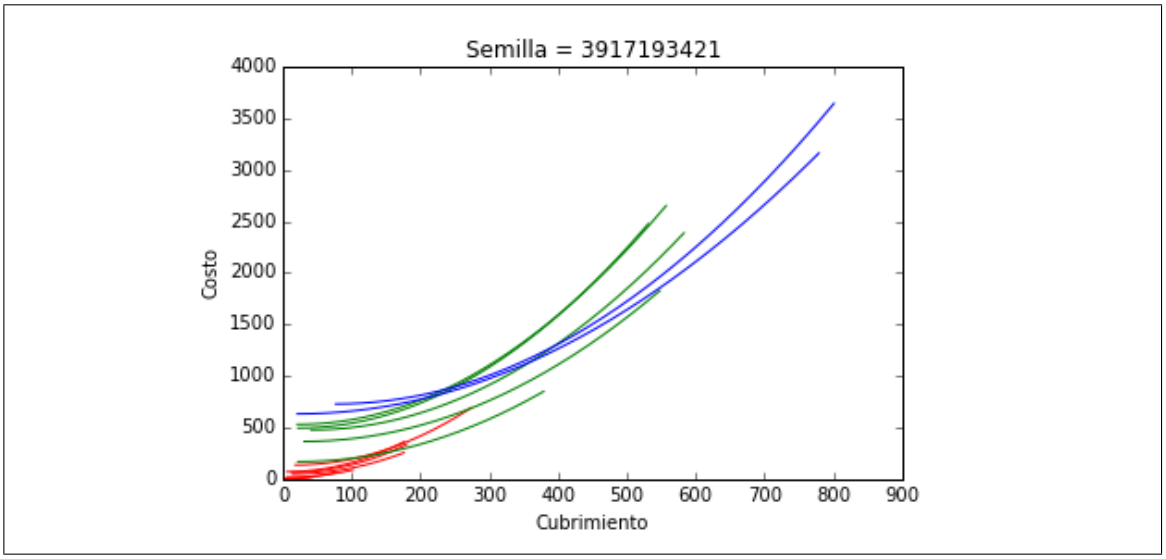

Figura A.12. Ejemplo de escenario iv,3)