COMPARACIÓN DE TÉCNICAS DE PLE PARA EL PROBLEMA DE MOCHILA MÚLTIPLE CON COLORES CON APLICACIÓN A LA DISTRIBUCIÓN DE VEHÍCULOS NUEVOS EN VENEZUELA

Por

Esnil J. Guevara M.

Tesis sometida en cumplimiento parcial de los requerimientos para obtener el título de

LICENCIADO EN MATEMÁTICA

UNIVERSIDAD DE CARABOBO
FACULTAD EXPERIMENTAL DE CIENCIAS Y TECNOLOGÍA

Octubre, 2008

Aprobada por:

Jurado Prof. Nelson Hernández	Fecha
Jurado Prof. Luis Rodríguez	Fecha
Tutor Prof. Víctor Griffin	Fecha
Director del Departamento Prof. José Marcano	Fecha

COMPARACIÓN DE TÉCNICAS DE PLE PARA EL PROBLEMA DE MOCHILA MÚLTIPLE CON COLORES CON APLICACIÓN A LA DISTRIBUCIÓN DE VEHÍCULOS NUEVOS EN VENEZUELA

Resumen

Esnil J. Guevara M.

Octubre 2008

Tutor: Prof. Victor Griffin
Departamento: Matemática

Los problema de mochila múltiple con colores pertenecen a la clase de problema NP-duro en fuerte sentido, es decir, que no se conoce un algoritmo eficiente para grandes instancias del mismo. Motivados por una aplicación real para la empresa transportista CLOVER INTERNATIONAL C.A. de Venezuela, y basados en la descomposición realizada por Cuadrado M., se propone usar técnicas recientes de programación lineal entera para lograr conocer hasta que tamaño del problema de asignación de cargas a unidades de transporte se puede lograr obtener una solución buena en un tiempo razonable mediante los modelos de mochila múltiple con o sin restricción de color.

Palabras claves: Optimización multiobjetivos, software libres, asignación de cargas a unidades de transporte, problema de mochila multiple, generación de columnas y cortes.

A mi Dios y Padre celestial que reina por siempre y para siempre, a el sea la gloria Salmos 136. A mis padres: Pedro Jesus Guevara y Vilma Mendoza, y mis hermanos: Jesús D. Guevara y Ruth S. Guevara.

AGRADECIMIENTOS

A Dios primeramente por haberme creado para sus propósitos eternos, a mis padres Pedro Guevara y Vilma Mendoza; porque me instruyeron en la palabra de Dios desde niño, enseñandome buenos valores, dándome amor y apoyo en todo tiempo. A mis hermanos Jesus Guevara y Ruth Guevara por estar conmigo para compartir, jugar llorar y reír; de verdad los quiero a todos.

Quiero agradecer a Eduardo Sánchez porque fue el único que se ofreció en ayudarme incondicionalmente, estando atento a los adelantos del proyecto a nivel de programación y enseñandome muchas cosas, no había conocido a alguien como el, éxitos Eduardo vas a llegar lejos, y sabes que cuentas conmigo.

A todos mis profesores de básica, secundaria y de la universidad, entre ellos al Dr. Victor Griffin que además de ser un excelente maestro, me ofreció tiempo y grandes ideas para el proyecto en la cual me siento motivado para seguir aprendiendo, y al profesor Jesus Parras por haberme enseñado que la topología es bella.

Por ultimo a todos mis amigos sinceros que con cariño me apoyaron, a la célula de la casa de la hermana Ana de Sánchez por estar siempre atentos, a mis amigos de la universidad a los cuales estimo y aprecio mucho, a todos mis tíos y familiares.

A todos ellos, Gracias...

Índice general

pagina

Índice de cuadros

Tabla pagina

Índice de figuras

Figura pagina

LISTA DE ABREVIATURAS

MMRC	Mochila múltiple con restricción de color.
MMMC	Mochila múltiple multiobjetivos con colores.
MMC	Mochila múltiple con colores.
MMC_{GC}	Nueva formulación del modelo mochila múltiple con colores para resolver via generación de columnas.
BB	Algoritmo de ramificación y acotamiento (Branch & Bound).
BBF	Algoritmo de ramificación y acotamiento con estrategia de ramificar sobre la variables mas fraccionaria.
GC	Algoritmo de generación de columnas y cortes.
PLE	Programación lineal entera.
PL	Programación lineal.
PM	Problema maestro (Master Problem).
PAG	Problema de asignación generalizado.
BCP	Branch, cut and price.
GLPK	GNU linear programming kit.
SYMPHONY	Single or multi process optimization over networks.
GAP	La brecha entre la solución de la relajación PL y la solución del PLE.
POM	Problema de optimización multiobjetivo.
PSO	Problemas de optimización de un solo objetivo.

Capítulo 1
EL PROBLEMA

1.1. Introducción

En el enfoque científico de toma de decisiones se utiliza uno o mas modelos matemáticos para elegir entre diferentes opciones la que mas se ajuste y supla las necesidades de un proyecto. Estos son representaciones matemáticas de situaciones reales que por lo usual se utilizan para entender mejor la situación y cuando se habla de optimización se refiere al mejoramiento de dicha situación; sin embargo, en el contexto científico se puede decir que la optimización es el proceso de tratar de encontrar la mejor solución posible a un determinado problema.

Como referencia de ello, se asume una investigación aplicada a la distribución de vehículos nuevos en Venezuela (Caso: Clover International C.A.) Cuadrado, M. [1], la cual revela un (1) modelo matemático de optimización combinatoria, cuyos objetivos eran mejorar los tiempos de entregas, minimizar los vehículos en el patio con altos niveles de espera y minimizar el número de paradas de las unidades de transporte. Para solucionar este modelo de planificación a corto plazo se propuso un heurístico de dos pasos basado en una combinación de métodos de descomposición y construcción. El **Paso 1** determina las cantidades de viajes de la flota de unidades de transporte usando un modelo relajado para asignar las nodrizas a las rutas, descomponiendo el problema general en subproblemas uno por cada ruta (patio-region), ésto hace que para cada ruta se tenga un modelo de mochila múltiple con colores. El **Paso 2** es resolver los modelos de mochila múltiple con colores, un

modelo por cada ruta, para así optimizar la asignación de las cargas a las unidades de transporte. Dado que el problema de mochila múltiple con o sin colores pertenece a la clase de problemas NP-duro, se propuso la utilización de métodos heurísticos específicamente Búsqueda Tabu para obtener una solución buena, a pesar de que los problemas de cada patio-región no eran muy grandes se podría haber probado los métodos exactos de programación lineal entera (PLE).

Las técnicas de ramificación y acotamiento son efectivos para problemas pequeños. Combinando las técnicas de generación de columnas y cortes con ramificación y acotamiento se ha logrado durante los últimos 10 años resolver problemas muchos mas grandes.

La presente investigación propone combinar las técnicas de generación de columnas o generación de cortes con las técnicas de ramificación y acotamiento, y utilizar plataformas de software libre para resolver PL y LPE con el propósito de determinar hasta que tamaño de problemas se puede lograr resolver en forma exacta o por lo menos hasta saber que la solución obtenida está dentro de cierto porcentaje del óptimo por ejemplo menor que un porciento.

Dentro de los software libre existentes se buscaron y estudiaron los que proporcionan mejores herramientas posibles para el buen desarrollo y funcionamiento del proyecto, tales como: GLPK, BCP y SYMPHONY, ya que permiten combinar métodos clásicos de ramificación y acotamiento con técnicas de generación de cortes y de columnas con los cuales se han logrado resolver problemas mas y mas grandes.

Generalmente hablando, las herramientas de software libre no pueden igualarse en rapidez o robustez a las comerciales, pero éstos ofrecen una alternativa viable para los usuarios que no tengan a su alcance una herramienta comercial. Para ciertas aplicaciones, las herramientas de software libre pueden también ser mas flexibles y fáciles de modificar que las comerciales en las cuales la flexibilidad puede ser limitada por la interfaz que es expuesta al usuario.

1.2. Planteamiento del Problema

La tesis de grado presentada por Cuadrado, M. (2007) [1], desarrolla un modelo aplicado a la distribución de vehículos nuevos en Venezuela, que aportaría solución a la planificación a corto plazo de la asignación de las unidades de carga(nodrizas) a las rutas (patio de origen a región de destino) y a la asignación de las cargas (vehículos) a unidades de transporte, minimizando o limitando el número de paradas para cada unidad de transporte y maximizando la carga de vehículos críticos.

Para resolver este modelo, se propuso un heurístico de dos pasos: primeramente asignar las nodrizas a rutas y así descomponer el problema en sub-problemas, uno por cada ruta (patio-región) y segundo, por cada ruta asignar la carga de vehículos de dicha ruta entre las nodrizas asignadas a la ruta en el primer paso. El problema para cada ruta es un problema de mochila múltiple con colores (MMC) ya que posee un conjunto de artículos (conjunto de vehículos listos para cargar), un conjunto de mochilas (conjunto de unidades de transporte asignados a la ruta) y un atributo color asignado a cada vehículo que denota el concesionario a donde debe ser entregado. Marlyn Cuadrado propuso un heurístico constructivo para resolver el problema MMC de cada ruta.

La siguiente tabla (ver cuadro 1.1) presenta para dos rutas típicas, la cantidad de unidades de transporte, la cantidad de vehículos a ser cargados y la cantidad de concesionarios donde deben ser llevados los vehículos. Aunque el problema de mochila múltiple con o sin color pertenece a la clase de problemas NP-duro, lo cual implica que no es probable que exista un algoritmo eficiente para resolver grandes instancias del mismo, se observa que estos dos sub-problemas son pequeños, dando la posibilidad de ser resueltos en forma exacta o aproximada, utilizando técnicas de PLE. Por supuesto, si se divide el país en menos regiones para facilitar la solución del primer paso, el problema MMC de cada ruta sería más grande. De manera que la presente investigación propone el uso de las técnicas de programación lineal

entera para explorar hasta que tamaño del problema se pueden resolver los modelos de mochila múltiple con colores mediante las herramientas de software libre de optimización.

Cuadro 1–1: Conjuntos de autos y nodrizas en cada patio-región

Patio - Región	Concesionarios	Autos	Nodrizas
Ford Nacional - 1	8	21	3
Ford Importados - 4	7	6	2

Fuente: Cuadrado M. Modelos Matemáticos de Optimización (2005)

Existen paquetes de software libre que utilizan las técnicas de ramificación y acotamiento en forma de **caja negra** para resolver modelos de PLE. Sin embargo las técnicas de generación de columnas y de cortes combinadas con ramificación y acotamiento conocidos en inglés como **Branch, Cut y Price** (BCP) han mostrado ser eficaces en resolver problemas mucho más grandes. Tales métodos normalmente requieren crear algoritmos específicos para cada tipo de problema pero tal **customización** ha sido facilitado por la creación de plataformas para optimización que permiten el uso de subrutinas pre-programadas. Se propone hacer análisis comparativo de las diferentes técnicas.

Hay más de una manera de incluir el atributo color en los modelos de MMC. Se puede incluir restricciones que limiten la cantidad de colores en cada mochila o se puede incluir la cantidad de colores como una segunda función objetiva a ser minimizada además de maximizar la entrega de vehículos críticos. No se pretende entrar en detalle sobre las técnicas para programación lineal entera multi-objetiva pero si presentar un análisis comparativo de las soluciones de algunos casos específicos.

1.3. Objetivos

1.3.1. Objetivo General

Utilizar técnicas recientes de PLE para obtener soluciones de forma eficiente y en tiempo razonable del modelo de mochila múltiple con atributo color (MMC) considerando aplicaciones en la distribución de vehículos nuevos en Venezuela.

1.3.2. Objetivos Específicos

- Formular el problema de asignación de cargas a unidades de transporte como un problema de mochila múltiple con colores.
- Desarrollar diferentes modelos de PLE para el problema de mochila múltiple con colores (MMC).
- Establecer algunas herramientas de software libre de optimización adecuados a la solución de los modelos propuestos.
- Comprobar la efectividad de los algoritmos en la solución de cada unos de los modelos propuestos.
- Seleccionar los modelos y algoritmos que proporcionan soluciones de alta calidad en un plazo de tiempo razonable entre los modelos y algoritmos estudiados.

1.3.3. Justificación e Importancia

Resolver modelos grandes de mochila multiple con color, optimizando el problema de asignación de cargas a unidades de transporte de la distribución de vehículos nuevos en Venezuela, propiciaría de manera rápida la asignación de los vehículos a las nodrizas, maximizando la entrega de autos críticos en inventario y minimizando el número de paradas para cada unidad de transporte. Buscar un equilibrio entre los dos pasos del modelo de planificación a corto plazo de Cuadrado M. [1] puede en cierto modo determinar que tan grande pueden ser los subproblemas patio-region para un conjunto de patios y regiones, ya que si las cantidades de patios y regiones son grandes, el **Paso 1** sería difícil de resolver pero cada problema patio-region en el **Paso 2** sería pequeño. Si la descomposición es muy gruesa, es decir, hay pocos

patios y pocas regiones en el **Paso 1**, este sería mas fácil de resolver pero cada subproblema patio-region sería mucho mas grandes.

La elaboración del modelo propuesto en esta investigación es importante ya que no solo aportaría solución al modelo de asignación de cargas a unidades de transportes, sino que también se pudiera aplicar a cualquier área que requiera de envíos y cargas de artículos. Por otra parte se estaría contribuyendo y estimulando la manipulación de los softwares libres mediante la combinación de algoritmos.

1.3.4. Revisión Bibliográfica

Antecedentes del Problema

El estudio reciente que se hizo en base a la distribución de vehículos nuevos en Venezuela presentado por Cuadrado, M. [1], refiere a una tesis de grado para obtener el titulo de Licenciando en Matemática en la Universidad de Carabobo (2007) el cual se denomina Modelos Matemático para la Optimización Combinatoria de la distribución de vehículos nuevos en Venezuela (Caso: Clover International C.A.), allí se propone un modelo matemático (Modelo de planificación a corto plazo) para solucionar la asignación de los autos a las unidades de trasporte en la distribución de vehículos nuevos en Venezuela, de manera que minimice el número de autos críticos en el patio de cada región, y reducir los costos que generan las paradas de cada unidad de transporte. Se demostró en la primera corrida del modelo relajado que la suma de las criticidades de los vehículos (876) y los ingresos para la empresa en la realidad (Bs. 30,241,528), para el mismo día de planificación (27 y 28 de septiembre del 2005) presentaron ser un poco menos del doble de la suma de criticidades e ingresos generados por el modelo con criticidad e ingresos de 1589 y Bs. 87,771,100 respectivamente.

Los datos recolectados en esa investigación son de gran importancia para este proyecto, ya que serán considerados como datos secundarios, para la formulación de los modelos propuestos.

Antecedentes de la Investigación

La primera publicación que relata el problema de mochila múltiple con colores, fue presentada por Jayant K. y Milind D. [2] con el titulo de The Multiple Knapsack Problem with Color Constraints, como un informe de investigación para la pronta diseminación de su contenido (1998), donde a través de un problema que surgió por motivo de una aplicación real por parte de la industria "steel"[3], generalizan el problema de mochila múltiple agregando un nuevo atributo llamado color, donde este color puede asociarse a uno o varios artículos y por medio de una restricción de color determinan el número de colores el cual pueden ser asignados a cada mochila. Como aporte a la investigación propuesta se toma la idea del atributo color que se añade a cada artćulo y se enfoca en este proyecto para indicar el concesionario a donde fue asignado el vehículo de carga.

Otra investigación asociada a los problemas de mochila múltiple donde usan el atributo color fue realizada por John J. Forrest, Jayant Kalagnanam y Laszlo Ladanyi [4], la cual se titula A Column-Generation Approach to the Multiple Knapsack Problem with Color Constraints, esta fue presentada por INFORMS (Instituto para Investigación de Operaciones y las Ciencias de Dirección) en Linthicum, Maryland, EE.UU. (2006), quienes demostraron obtener la solución óptima del problema de mochila múltiple con restricción de color de una forma muy eficaz, proponiendo una nueva formulación para poder generar columnas con costo reducido positivo, y obtienen cotas superiores a través del algoritmo de descomposición de Dantzig-Wolfe. La implementación de este modelo usa BCP (Branch-and-cut-and-price), el cual es parte de COIN-OR, la infraestructura computacional de software libre para investigación de operaciones.

De allí que la explicación de la labor realizada en la investigación considerada antecedente a esta investigación presenta la posibilidad de utilizar métodos y algoritmos como via para resolver problemas grandes de mochila múltiple con colores.

Además la implementación de sus códigos pueden ser modificados de acuerdo a la necesidad de otros problemas parecidos.

1.3.5. Metodología de la Investigación

El objetivo central es usar las técnicas de PLE para determinar hasta que tamaño del problema se puede lograr obtener un valor óptimo o muy bueno del problema de asignación de cargas a unidades de transporte mediante el modelo de mochila múltiple con colores, y en la construcción de este la incorporación de softwares libre que se consideraron indispensables para la solución del mismo. De manera que se establece los pasos lógicos para el desarrollo de la presente investigación:

1. Se establece los pasos para la formulación del problema de MMC a través de los datos secundarios. En esta etapa se describirán todas las características que deben presentar los modelos de MMC (con restricción de color, sin restricción de color) en el lenguaje GMPL de programación.
2. Esta sección comprende todo el desarrollo práctico del presente trabajo, en la cual se resolverá cada unos de los modelos propuestos por medio de de los software y algoritmos (PL y PLE) estudiados. Se tratará de construir en GLPK un programa simple personalizado, para resolver problemas solo para estos tipos de modelos combinando algoritmos de programación lineal.
3. En este paso, una vez realizadas las pruebas o corridas de los software y algoritmos, se seleccionarán los algoritmos que proporcionen mejores tiempos de ejecución y mejores valores en la función objetivo.

Capítulo 2
MARCO TEÓRICO DE LA INVESTIGACIÓN

2.1. Optimización Combinatoria

La optimización combinatoria estudia el modelado y solución algorítmica de problemas donde se busca maximizar (o minimizar) una función de varias variables definidas sobre un conjunto discreto [5].

Definición 1. ***(Problema de Optimización Combinatoria):*** *Dado X finito y $F \subseteq X$ (una familia de subconjuntos de X); dados $C_x \in \mathbb{R}$, $\forall x \in X$ el problema de optimización combinatoria consiste en encontrar $F' \in F$ tal que $\sum_{x \in F'} C_x$ se optimice.*

2.2. Problema de Optimización Combinatoria

En la optimización combinatoria confluyen la matemática discreta, la teoría de algoritmos, y la programación lineal y lineal-entera. Un problema de programación lineal consiste en hallar el valor óptimo de una función objetivo lineal cuyas variables están sujetas a restricciones lineales. Si además se exige que las variables tomen valores enteros, entonces se tiene un problema de programación lineal-entera. Varios problemas de optimización combinatoria pueden ser resueltos en tiempo polinomial utilizando los métodos y teoría de la programación lineal. En el caso de los problemas NP-Completos, los métodos más eficaces en la actualidad para encontrar una

solución exacta a muchos de estos problemas utilizan, típicamente, la programación lineal-entera.

A continuación se dará una breve descripción de los problemas de optimización combinatoria de interés para la presente investigación.

2.2.1. Programación Lineal (PL)

La Programación Lineal (PL) es una herramienta para resolver problemas de optimización. Desde que George Dantzig en el año 1947 desarrolló el método simplex para resolver problemas de programación lineal, esta comienza a ser empleada para resolver problemas en diversas industrias, como los bancos, la educación, petróleo y transporte de carga, entre otros.

Un problema de PL consta de una función objetivo (lineal) por maximizar o minimizar, sujeta a ciertas restricciones en la forma de igualdades o desigualdades. Matemáticamente, un problema de programación lineal consiste en encontrar el óptimo (máximo o mínimo) de una función lineal en un conjunto que puede expresarse como la intersección de un número finito de hiperplanos y semiespacios en $\mathbb{R}^n$. La definición formal se da a continuación.

Definición 2. ***(Programación Lineal):*** *la forma más general de un problema de PL consiste en:*

$$\textit{Optimizar } Z = f(x) = \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

sujeto a

$$\begin{aligned} \sum_{j=1}^{n} a_{ij} x_j &= b_i, \quad i = 1, \ldots, p-1. \\ \sum_{j=1}^{n} a_{ij} x_j &\geq b_i, \quad i = p, \ldots, q-1. \\ \sum_{j=1}^{n} a_{ij} x_j &\leq b_i, \quad i = q, \ldots, m. \end{aligned} \tag{2.2}$$

donde $p, q, m \in \mathbb{Z}$ *tales que* $1 \leq p \leq q \leq m$

Donde 2.1 representa la función objetivo (función lineal) que se quiere optimizar y 2.2 representa las restricciones (desigualdades lineales) del modelo PL.

Definición 3. ***(Teorema Fundamental de la Programación lineal):*** *Un hiperplano en* $\mathbb{R}^n$ *es una variedad lineal de codimensión 1, es decir,* $\{x \in \mathbb{R}^n \ / \ a{\cdot}x = k\}$*, donde* $a = (a_1, \ldots, a_n) \in \mathbb{R}^n$,$k \in \mathbb{R}$ *y* $a{\cdot}x = a_1x_1 + \cdots + a_nx_n$*. Cada hiperplano determina dos semiespacios:* $\{x \in \mathbb{R}^n \ / \ a{\cdot}x \leq k\}$ *y* $\{x \in \mathbb{R}^n \ / \ a{\cdot}x \geq k\}$*. Un poliedro es la intersección de un número finito de semiespacios.*

Observación 1. *Un problema de maximización puede convertirse en uno de minimización cambiando el signo de la función objetivo de la siguiente manera:*

$$Maximizar \quad Z_{max} = \sum_{i=1}^{n} c_j x_j$$

es equivalente al problema

$$Minimizar \quad Z_{min} = -\sum_{i=1}^{n} c_j x_j$$

sometidos ambos a las mismas restricciones.

En base a la observación 1 y sin pérdida de generalidad, se considerarán ahora problemas de minimización.

Definición 4. ***(Región factible):*** *La región factible para un PL, es el conjunto de todos los puntos que satisfacen las limitaciones y las restricciones del signo de la PL.*

Definición 5. ***(Solución óptima):*** *Para un problema de maximización, una soluci ó óptima para una PL es un punto con el valor de la función objetivo más grande en la región factible. De igual manera, para un problema de minimización, una solución óptima es un punto con el valor de la función objetivo más pequeño en la región factible.*

2.2.2. Programación Lineal Entera (PLE)

Una PLE en la cual se requiere que todas las variables tienen que ser enteros se denomina **problema puro de programación con enteros**.

Una PLE en la cual se require que solo algunas de las variables sean números enteros, se llama **problema combinado de programación con enteros**.

Un problema de programación lineal entera en la cual todas las variables tienen que ser iguales a 0 o a 1 recibe el nombre de PE $0-1$ o PE binaria. La fomulación de problema de programación lineal entera se planteara brevemente.

Definición 6. ***(Programación Lineal Entera):*** *la forma más general de un PLE consiste en:*

$$\textit{Optimizar } Z = f(x) = \sum_{j=1}^{n} c_j x_j \tag{2.3}$$

sujeto a

$$\begin{aligned}
&\sum_{j=1}^{n} a_{ij} x_j = b_i, \quad i = 1, 2, \ldots, p-1. \\
&\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \quad i = p, \ldots, q-1. \\
&\sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = q, \ldots, m. \\
&x_j \geq 0, \qquad j = 1, \ldots, n. \\
&x_j \in \mathbb{Z}^+, \quad j = 1, \ldots, n.
\end{aligned} \tag{2.4}$$

donde $p, q, m \in \mathbb{Z}$ *tales que* $1 \leq p \leq q \leq m < n$.

2.2.3. Problema de Programación Multicriterio

El problema clásico de la programación matemática lineal consiste en la búsqueda de un óptimo de una función lineal de varias variables restringido por un conjunto de ecuaciones lineales. Un enfoque más reciente y realista, no solo en programación lineal sino también en optimización, consiste en la optimización simultánea de varias funciones objetivo que en muchos casos pueden ser conflictivas, la cual no es posible

reducir a una única función objetivo y por tanto hay que tratarlas de una manera conjunta. Este enfoque se denomina **programación multiobjetivo** (o mas general, de desición multicriterio) [6].

Se dice que las soluciones de un problema con objetivos múltiples son óptimas porque ninguna otra solución, en todo el espacio de búsqueda, es superior a ellas cuando se tienen en cuenta todos los objetivos al mismo tiempo, es decir, ningún objetivo puede mejorarse sin degradar a los demás. Al conjunto de estas soluciones óptimas se conoce como soluciones *Pareto óptimas*. A partir de este concepto se establece como requisito para afirmar que una situación es mejor que otra, en la que en ella no se disminuya a nadie, pero se mejore a alguno.

Definición 7. ***Problema de Optimización Multiobjetivo:*** *La formualción general del problema de programación multiobjetivo con* n *variables de decisión* x_j, m *restricciones y p objetivos es: determinar* $x_1, \ldots, x_n$ *que*

$$\begin{aligned} &\textit{Maximicen} && \boldsymbol{z} = (z_1(x), z_2(x), \ldots, z_p(x)) \\ &\textit{Sujeto a} && x \in F \end{aligned} \tag{2.5}$$

con $F \subset \mathbb{R}^n$, *F región factible del espacio de decisiones (o soluciones)* $\mathbb{R}^n$ *y* $Z = z(F) \subset \mathbb{R}^p$, *Z región factible del espacio de objetivos (consecuencias o resultados)* $\mathbb{R}^p$.

En muchas aplicaciones el conjunto F se puede expresar

$$F = \{x \in \mathbb{R}^n : g_i(x) \leq 0, x_i \geq 0, \forall i, j\}$$

donde las funciones g_i son las restricciones. A las funciones z_k se les denomina funciones objetivos o simplemente objetivos. Se puede suponer que el problema de optimización es de la forma de maximización (maximización de cada función objetivo).

Si en la formulación 7 todas las funciones z_k y g_i (que definen F) son lineales, el problema se denomina de programación lineal multiobjetivo.

Definición 8. ***(Optimalidad Pareto):*** *dado un vector de decisión* $\boldsymbol{x} \in \boldsymbol{F}$ *y su correspondiente vector objetivo* $\boldsymbol{y} = z(\boldsymbol{x}) \in \boldsymbol{Z}$ *, se dice que* $\boldsymbol{x}$ *es no dominado respecto a un conjunto* $\boldsymbol{A} \subseteq \boldsymbol{F}$ *si y solo si*

$$\forall \boldsymbol{a} \in \boldsymbol{A} : (\boldsymbol{x} \succ \boldsymbol{a} \vee \boldsymbol{x} \sim \boldsymbol{a}) \tag{2.6}$$

En caso de que $\mathbf{x}$ sea no dominado respecto a todo el conjunto $\mathbf{F}$ y solo en ese caso, se dice que $\mathbf{x}$ es una solución Pareto óptima ($\mathbf{x} \in \mathbf{F}_{true}$ el conjunto Pareto óptimo real).

En otras palabras, esta definición dice que $\mathbf{x}$ es un óptimo de Pareto si no existe ningún vector factible de variables de decisión $\mathbf{x}^*$ que decremente algún criterio sin causar un incremento simultaneo en al menos un criterio.

Definición 9. ***(Conjunto Pareto óptimo y frente Pareto óptimo):*** *dado el conjunto de vectores de decisión factibles* $\boldsymbol{F}$*. Se denomina* $\boldsymbol{F}_{true}$ *al conjunto de vectores de decisión no dominados que pertenecen a* $\boldsymbol{F}$*, es decir:*

$$\boldsymbol{F}_{true} = \{\boldsymbol{x} \in \boldsymbol{F} | \quad \boldsymbol{x} \quad \textit{es no dominado con respecto a } \boldsymbol{F}\}$$

Desafortunadamente, el concepto de solución de Pareto óptimo casi siempre produce no una solución única sino un conjunto de ellas a las que se les llama conjunto de óptimos de Pareto y su correspondiente vector $\mathbf{y}$, que es parte del frente Pareto óptimo real $\mathbf{Z}_{true}$. Estos términos se definirán a continuación.

El conjunto $\mathbf{F}_{true}$ también es conocido como el **conjunto Pareto óptimo**. Mientras que el conjunto correspondiente de vectores objetivo $\mathbf{Z}_{true} = \mathbf{z}(\mathbf{F}_{true})$ constituye el **frente Pareto óptimo**.

2.2.3.1. Métodos tradicionales para la resolución de POM

Numerosos métodos clásicos existen para el tratamiento de problemas de optimización multiobjetivo. Todos ellos se caracterizan por operar en dos fases. Primeramente se generan, a partir del problema de objetivos múltiples, un PSO. En la

segunda fase se aplica un método de optimización tradicional al PSO generado en la fase anterior y obtienen sus resultados. Ambas fases son independientes entre sí, es decir, una vez obtenido el PSO se puede resolver con cualquier técnica típica.

Varios tipos de métodos se emplean para la reducción realizada en la primera fase, cada uno de ellos tiene sus propias ventajas y desventajas. En gran parte de ellos, la técnica consiste en disponer de las funciones objetivos de modo tal que se puedan unir en una única función parametrizada.

Los parámetros de la función varían de modo sistemático para realizar múltiples corridas del procedimiento de optimización (2da. fase) para conseguir un conjunto que se aproxime al conjunto Pareto óptimo real. Algunas técnicas representativas son:

Método de las ponderaciones: Fue propuesto por Zadeh en 1963. Se basa en la idea de ponderar los objetivos y generar el conjunto eficiente con una adecuada variación paramétrica de los mismos.

El método de las ponderaciones se formula planteando el programa lineal

$$\begin{aligned} \text{máx } f(x) &= \sum_{i=1}^{p} \lambda_k f_k(x) \\ \text{Sujeto a} \qquad & x \in F \end{aligned} \tag{2.7}$$

con λ_k peso asociado al objetivo f_k. Se denomina a este problema P(λ), con $\lambda = (\lambda_1, \ldots, \lambda_n)$ vector de pesos o ponderaciones, de manera que se cumpla $\sum_{j=1}^{k} \lambda_j = 1$.

El peso λ_k para el objetivo f_k se interpreta como la importancia o peso relativo del k-ésimo objetivo en relación con el resto de los objetivos. De esta forma, si los pesos $\lambda_1, \ldots, \lambda_n$ expresan las preferencias del decisor sobre cada objetivo y este es capaz de asignarlos de una manera razonable, la solución óptima de P(λ) es la solución de mejor compromiso para él.

Teorema 1. *Si $\lambda_k > 0$ para todo k=1,...,n, entonces cualquier solucíon óptima x^* de P(λ) es eficiente.*

El método de las ponderaciones no resulta muy adecuado para obtener la representación completa del citado conjunto. Hay que considerar de forma sistemtica una serie de conjuntos de pesos positivos. Usualmente, se empieza por la optimización individual de cada objetivo, que equivale a tomar los pesos (1,0,...,0), (0,1,...,0),..., (0,0,...,1), para después introducir una variación sistemática de éstos con una tasa de aumento prefijada que hay que estimar como adecuada.

Método de las ϵ-restricciones El método de ϵ-restricciones consiste en optimizar una función objetivo $f_l(x)$ de f, que se supone mas importante que las otras, introduciéndose además números reales ϵ (k $\neq$ l) para las restantes funciones objetivo que corresponden a cotas inferiores propuestas por el decisor.

$$\text{Maximizar} \quad f_l(\mathbf{X}) \tag{2.8}$$

Sujeto a

$$f_l(\mathbf{X}) \geq \varepsilon_j; \quad 1 \leq j \leq k \quad \mathbf{x} \in \mathbf{F}$$

y que denominaremos $P_l(\epsilon)$.

La elección de la función objetivo f_l y las cotas ϵ, representan preferencias subjetivas del decisor, de manera que si no existiera una solución para $P_l(\epsilon)$ habria que relajar al menos una de las cotas, Chankong y Haimes (1983.

Programación por metas La Programación por Metas (Goal Programming) fue inicialmente introducida por Charnes y Cooper en los años 50. Desarrollada en los años 70 por Ljiri, Lee, Ignizio y Romero, es actualmente uno de los enfoques multicriterio que más se utilizan.

En principio fue dirigida a resolver problemas industriales, sin embargo posteriormente se ha extendido a muchos otros campos como la economía, agricultura, recursos ambientales, recursos pesqueros, etc.

La idea del método consiste en proponer metas o niveles de aspiración para los distintos objetivos que desea alcanzar y una solución óptima se define como aquélla que minimiza la desviación a las metas propuestas.

Modelo de programación por metas: Determinar x $\in \mathbb{R}^n$ tal que

$$\begin{aligned} \text{mín } f &= \sum_{i=1}^{p} \left| f_i(X) - \hat{f}_k \right| \\ \text{Sujeto a} \quad & x \in F \end{aligned} \tag{2.9}$$

donde $\hat{f}_k$ es la meta establecida por el decisor para el i-ésimo objetivo f_i y F la región factible definida por las restricciones lineales.

El criterio es, por tanto, hacer mínima la suma de los valores absolutos de las desviaciones o diferencias entre los valores de los objetivos y sus metas.

2.3. Dualidad

Cada problema de programación lineal tiene un segundo problema asociado con el. Uno se denomina primal y el otro dual. Los 2 poseen propiedades muy relacionadas, de tal manera que la solución óptima a un problema proporciona información completa sobre la solución óptima para el otro.

Las relaciones entre el primal y el dual se utilizan para reducir el esfuerzo de computo en ciertos problemas y para obtener información adicional sobre las variaciones en la solución óptima debidas a ciertos cambios en los coeficientes y en la formulación del problema. Esto se conoce como análisis de sensibilidad o post-optimidad.

Definición 10. ***(Problema Dual):*** *Dado el problema de programación lineal minimizar*

$$\text{mín } Z = c^T x \tag{2.10}$$

sujeto a

$$\begin{aligned} Ax &\geq b \\ x &\geq 0 \end{aligned} \tag{2.11}$$

Su problema dual es maximizar

$$\text{máx}\, Z = b^T y \tag{2.12}$$

sujeto a

$$\begin{aligned} A^T y &\leq c \\ y &\geq 0 \end{aligned} \tag{2.13}$$

Se denomina al primer problema problema primal, y al segundo, su dual. Obsérvese que los mismos elementos (la matriz A, y los vectores b y c) configuran ambos problemas. El problema primal no se ha escrito en forma estándar, sino en una forma que nos permite apreciar la simetría entre ambos problemas, y mostrar así que el dual del dual es el primal .

2.3.1. Obtención del Problema Dual

Un problema de programación lineal de la forma de la definición (2) tiene asociado un problema dual que puede formularse según las reglas siguientes: **Regla 1**. Una restricción de igualdad en el primal (dual) hace que la correspondiente variable dual (primal) no esté restringida en signo. **Regla 2**. Una restricción de desigualdad $\geq$ ($\leq$) en el primal (dual) da lugar a una variable dual (primal) no negativa. **Regla 3**. Una restricción de desigualdad $\leq$ ($\geq$) en el primal (dual) da lugar a una variable dual (primal) no positiva. **Regla 4**. Una variable no negativa primal (dual) da lugar a una restricción de desigualdad $\leq$ ($\geq$) en el problema dual (primal). **Regla 5**. Una variable primal (dual) no positiva da lugar a una restricción de desigualdad $\geq$ ($\leq$) en el problema dual (primal). **Regla 6**. Una variable no restringida en signo del problema primal (dual) da lugar a una restricción de igualdad en el dual (primal).

2.3.2. Teorema de Dual

Dualidad débil

Si se elige cualquier solución factible para el primal y cualquier solución factible para el dual, el valor w para la solución factible del dual sera por lo menos tan grande como el valor z para la función factible del primal. Este resultado se establece formalmente en el lema (1).

Lema 1. ***(Dualidad débil):***

Sea

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

cualquier solución factible para el primal y $\boldsymbol{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_m \end{bmatrix}$ *cualquier solución factible para el dual. Entonces, (valor* z *para* $\boldsymbol{x}$*)* $\leq$ *(valor* w *para* $\boldsymbol{y}$*).*

Lema 2. *Sea*

$$\bar{\boldsymbol{x}} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_n \end{bmatrix}$$

una solución factible para el primal y sea $\bar{\boldsymbol{y}} = \begin{bmatrix} \bar{y}_1 & \bar{y}_2 & \dots & \bar{y}_m \end{bmatrix}$ *una solución factible para el dual. Si* $c\bar{\boldsymbol{x}} = \bar{\boldsymbol{y}}b$*, entonces* $\bar{\boldsymbol{x}}$ *es óptimo para el primal y* $\bar{\boldsymbol{y}}$ *es óptimo para el dual.*

Teorema del dual

El principal interés de este teorema es la relación entre el primal y el dual cuando el primal tiene solución óptima. Si el primal tiene solución óptima, entonces

el siguiente resultado (el teorema del dual) describe la relación entre el dual y el primal.

Teorema 2. ***(Teorema del dual):***

Suponga que BV es una base óptima para el primal. Entonces $c_{BV}B^{-1}$ es una solución óptima del dual. Asimismo, $\bar{z} = \bar{w}$.

Observación 2. *Cuando encontramos una solución óptima para el primal mediante la apliación del algoritmo simplex, también encontramos la solución óptima del dual.*

Suponga que el primal es un problema de maximización normal con m restricciones para aclarar que la observación es cierta. Para poder usar el simplex para resolver este problema se debe sumar una variable de holgura s_i a la i-ésima restricción del primal. Suponga que BV es una base óptima para el primal. Entonces, el teorema del dual establece que $c_{BV}B^{-1} = \begin{bmatrix} y_1 & y_2 & \dots & y_m \end{bmatrix}$ es la solución óptima para el dual. pero recuerde que, y_i es el coeficiente de s_i en el renglón 0 del tableau del primal (BV) óptimo. Por lo tanto, si el primal es un problema de maximización normal, entonces el valor óptimo de la variable i-ésima del dual es el coeficiente de s_i en el renglón 0 del tableau del primal óptimo.

2.4. Obtención de una solución para los problemas de Optimización Combinatoria

Al resolver un problema de optimización combinatoria siempre se trata de encontrar una solución buena ó satisfactoria. Para ello, se utiliza de un algoritmo que sea capaz de hallar un valor que verifique todas las restricciones maximizando (minimizando) la función objetivo. Una clasificación de los algoritmos de optimización combinatoria es la siguiente:

- **Exactos:** son capaces de localizar la solución óptima, en la que la función objetivo alcanza su valor extremo.

- **Heurísticos:** encuentran una buena solución, pero no se garantiza que sea la óptima.

2.5. Algoritmos Exactos Utilizados para Resolver Problemas de Optimización Combinatoria

2.5.1. Algoritmo Simplex

En teoría de optimización matemática, el algoritmo simplex de George Dantzig es una técnica popular para dar soluciones numéricas del problema de la programación lineal. Este método está basado fundamentalmente en el siguiente concepto:

El método simplex emplea un proceso iterativo que principia en un punto extremo factible, normalmente el origen, y se desplaza sistemáticamente de un punto extremo factible a otro, hasta que se llega por último al punto óptimo.

En la actualidad el Algoritmo simplex es una herramienta estándar de programación lineal y su uso en otros sectores de la sociedad avanza rápidamente.

2.5.2. Algoritmo Simplex Dual

Este algoritmo fue desarrollo en 1954 por C. E. Lemke. Las reglas para el método simplex dual son muy parecidas a las del método simplex. De hecho, una vez que se inician, la única diferencia entre ellos es el criterio para elegir las variables básicas que entran y salen y la regla para detener el algoritmo. Para dar inicio al método simplex dual todos los coeficientes en la ecuación (0) deben ser no negativos (de manera que la solución básica sea super óptima). El método continua haciendo que el valor de la función objetivo disminuya, y conserva siempre coeficientes no negativos en la ecuación (0), hasta que todas las variables sean no negativas.

2.5.3. Algoritmo de Punto Interior

Gracias a la inquietud de grandes matemáticos, que se esforzaron en hacer mas eficiente el tiempo de solución de problemas a mediana y grande escala surgen los

métodos de punto interior los cuales han sido utilizados durante los últimos 10 años con gran éxito para solucionar problemas de programación lineal.

Concretamente el algoritmo de punto interior de Narendra karmarkar parte de un punto del interior de la región factible y se va moviendo en la dirección en la que mas rápidamente mejora la función objetivo, manteniendo la factibilidad. Este proceso converge a la solución óptima.

2.5.4. Algoritmo de Ramificación y Acotamiento

Hasta el momento solo se ha visto métodos de resolución de problemas lineales continuos. Sin embargo es muy frecuente que la naturaleza del problema exija que las variables sean enteras o binarias. Una alternativa es simplemente aproximar la solución hacia el entero mas cercano, pero esta estratégia podría ser bastante mala como de hecho ocurre para las variables binarias.

El algoritmo de ramificación y acotamiento (o de branch and bound) comienza con una relajación del problema (no considerar restricciones de integralidad) y construye un árbol con soluciones enteras particionando el conjunto de soluciones factibles de modo de descartar soluciones fraccionarias. Sin embargo, este solo hecho de descomponer puede llevar a un problema inmanejable por lo que se debe podar el árbol de manera inteligente.

2.5.5. Algoritmo de los Planos de Corte de Gomory

En matemática, y más en concreto en optimización, el método de los planos de corte es un procedimiento para encontrar soluciones enteras de un problema lineal.

Este método fue introducido por Gomory y funciona resolviendo un programa lineal no entero, después comprobando si la optimización encontrada es también una solución entera. Si no es así, es añadida una nueva restricción que corta la solución no entera (este corte no elimina ningún punto de la regiún factible para el PLE) hasta que se encuentra la solución entera óptima.

2.5.6. Algoritmo de Ramificación y Corte

El Algoritmo de ramificación y corte (Branch and Cut) es una generalización del algoritmo de ramificación y acotamiento (Branch and Bound) donde, después de resolver la relajación LP, y no habiendo sido acertado en la poda del nodo sobre la base de la solución LP, se trata de generar un corte. Si uno o varios cortes son generados, ellos son añadidos a la formulación y el LP es solucionado otra vez. Si ninguno es encontrado, se ramifica.

2.5.7. Algoritmo de Ramificación y Precio

El algoritmo de ramificación y precio es una generalización de LP basado en ramificación y acotamiento especialmente diseñado para manejar los PLE que contienen un enorme número de variables. Este algoritmo utiliza el método de generación de columnas en cada nodo del árbol de ramificación y acotamiento. Las columnas con costos reducido favorables son añadidas a la formulación para ser resuelta la PL hasta que ninguna columna puede ser genera, de esta manera se obtiene el valor óptimo del PL. Si las variables de la solución óptima obtenido por la generación de columna no cumplen con las condiciones de integralidad, estas se ramifican hasta alcanzar el valor óptimo del PLE.

2.6. Algoritmos Avanzados de Programación Lineal

2.6.1. Algoritmo Simplex Revisado

Cuando un problema es de considerables dimensiones (m y n muy grandes), no resulta practico en cada iteración del método simplex resolver, partiendo de cero, los sistemas de ecuaciones lineales $B^T\pi = c_{BV}$ y $By = a_j$.

Para resolver esos sistema cuando su dimensiones son muy grandes, se plantea que en cada iteración solo se genere aquella información estrictamente necesaria

manteniéndose una representación explícita de la matriz B^{-1}, la cual se recalcula después de cada cambio de base. El método utilizado para actualizar B^{-1} se explicara a continuación.

Suponga que se esta resolviendo un PL con m restricciones. Suponga, además, que se ha encontrado que x_k debería entrar a la base en el renglón r. Sea la columna para x_k en la base actual $y = B^{-1}a_k$

$$\begin{bmatrix} \bar{a}_{1k} \\ \bar{a}_{2k} \\ \vdots \\ \bar{a}_{mk} \end{bmatrix}$$

Se define la matriz E $m \times m$:

$$E = \begin{pmatrix} 1 & 0 & \dots & \frac{\bar{a}_{1k}}{\bar{a}_{rk}} & \dots & 0 & 0 \\ 0 & 1 & \dots & \vdots & \dots & 0 & 0 \\ \vdots & \vdots & & \frac{\bar{a}_{2k}}{\bar{a}_{rk}} & & \vdots & \vdots \\ 0 & 0 & \dots & \frac{1}{\bar{a}_{rk}} & \dots & 0 & 0 \\ \vdots & \vdots & & \frac{\bar{a}_{m-1,k}}{\bar{a}_{rk}} & & \vdots & \vdots \\ 0 & 0 & \dots & \vdots & \dots & 1 & 0 \\ 0 & 0 & \dots & \frac{\bar{a}_{mk}}{\bar{a}_{rk}} & \dots & 0 & 1 \end{pmatrix} \tag{2.14}$$

En pocas palabras, E es simplemente I_m con la columna r reemplazada por el vector columna

$$\begin{bmatrix} \frac{\bar{a}_{1k}}{\bar{a}_{rk}} \\ \vdots \\ \frac{\bar{a}_{2k}}{\bar{a}_{rk}} \\ \frac{1}{\bar{a}_{rk}} \\ \frac{\bar{a}_{m-1,k}}{\bar{a}_{rk}} \\ \vdots \\ \frac{\bar{a}_{mk}}{\bar{a}_{rk}} \end{bmatrix}$$

A la matriz E se le suele denominar matriz de coeficientes eta o matriz eta.

En general si B^{-1} es la inversa de la matriz básica B en una determinada iteración, la adaptación de la misma para la siguiente iteración se puede expresar como

$$\bar{B}^{-1} = EB^{-1}$$

Si como ocurre habitualmente el procedimiento simplex se inicia con un matriz B igual a la identidad, después de k iteraciones la matriz inversa de la base, B^{-1}, se puede expresar en una forma producto, de la siguiente manera:

$$B_k^{-1} = E_k E_{k-1} \cdots E_1 B^{-1}$$

En esta expresión cada matriz elemental E_i tiene la forma (2.14). Para su implementación en un ordenador (computadora) este esquema operativo únicamente requiere almacenar los valores de los elementos de la columna de la matriz E_i que la hace diferente de la matriz identidad, y del propio valor del índice i.

De manera que le método del simplex revisado se puede reescribir, introduciendo la forma producto a la inversa de la base, como se muestra a continuación:

Algoritmo simplex revisado en la forma de producto de la inversa

Paso 1 Calcular los multiplicadores simplex a partir de

$$\pi^T = C_{BV}^T B^{-1} = (((c_{BV}^T E_k)E_{k-1}) \cdots E_1)$$

Se determina los costos reducidos de las variables no básica a partir de

$$\bar{c}_j = c_j - \pi^T a_j, \forall j \in N$$

si $\bar{c}_j \geq 0$ para todo j que no pertenece a B. Entonces el tableau actual es óptimo.

Paso 2 Escoger un $q \in N$ tal que $\bar{c}_q = \text{mín}_{j \in N}\, \bar{c}_j < 0$. Calcular

$$y = B^{-1} a_q$$

si $y \leq 0$ el algoritmo se detiene ya que el problema es no acotado.

Paso 3 Si $x_j = B^{-1}b$, se establece la variable básica x_{j_p} que sale de la base determinando la fila p sobre la que pivota la columna q a partir de la relación:

$$\frac{x_{j_p}}{y_p} = \min_{1 \leq i \leq m} \left\{ \frac{x_{j_i}}{y_i}, y_i > 0 \right\}$$

Paso 4 Se adapta la matriz inversa de la base a la solución a partir de

$$E_{k+1} = \begin{pmatrix} 1 & 0 & \dots & \frac{-y_1}{y_p} & \dots & 0 & 0 \\ 0 & 1 & \dots & \vdots & \dots & 0 & 0 \\ \vdots & \vdots & & \frac{-y_{p-1}}{y_p} & & \vdots & \vdots \\ 0 & 0 & \dots & \frac{1}{y_p} & \dots & 0 & 0 \\ \vdots & \vdots & & \frac{-y_{p+1}}{y_p} & & \vdots & \vdots \\ 0 & 0 & \dots & \vdots & \dots & 1 & 0 \\ 0 & 0 & \dots & \frac{-y_m}{y_p} & \dots & 0 & 1 \end{pmatrix}$$

Donde la inversa de la base será $B^{-1} = E_{k+1}E_k \cdots E_2E_1$ y la nueva solución $x_j = E_{k+1}x_j$.

La gran ventaja de esta forma de implementar el método simplex radica en su eficacia para abordar estructuras dispersas de matrices A de grandes dimensiones.

2.6.2. Algoritmo de Generación de Columnas

Considere el siguiente problema:

minimizar

$$z = cx$$

sujeto a

$$Ax \geq b$$

$$x \geq 0$$

donde x $\in \Re^n$, b $\in \Re^m$ y n >> 0. Esta ultima condición implica que el problema original descrito tiene un número muy grande de variables.

Este número grande de variables implica que guardar esta matriz A en memoria puede no ser posible.

En muchos problemas de optimización a gran escala, la mayoría de las columnas de A nunca entran a la base, entonces para que generarlas?.

La idea principal es desarrollar un método que descubra las variables x_j con costo reducido $r_j < 0$ (j $\in NBV$). Las columnas A_j asociadas con estas variables se incluyen en una nueva matriz $\bar{A}$ que forma el siguiente problema maestro:

minimizar

$$z = \bar{c}\bar{x}$$

sujeto a

$$\bar{A}\bar{x} \geq b$$

$$\bar{x} \geq 0$$

donde $\bar{x} \in \Re^n$ es un subconjunto de variables en x, y $\bar{c}$, sus respectivos costos provenientes del vector de costos original c. Claramente, $\bar{n} <<$ n. El método de generación de columnas hace esto, mediante la solución del siguiente problema auxiliar de optimización:

$$min \quad r^* \quad = min_{j \in NBV} \quad r_j$$

El menor r_j , r^* (y su respectivo índice j^*), puede ser encontrado sin enumerar exhaustivamente todos los r_j , j $\in NBV$. Con base en r^* (y su respectivo índice j^*), se procede as:

Caso 1 Si $r^* \geq 0$, no hay variables candidatas a entrar a la base y el problema original es óptimo.

Caso 2 Si $r^* < 0$, la variable x_{j^*} debe entrar a la base del problema original. Es decir la columna A_{j^*} puede formar parte de la base.

El método de generación de columnas procede tal y como se muestra en la Figura 2–1

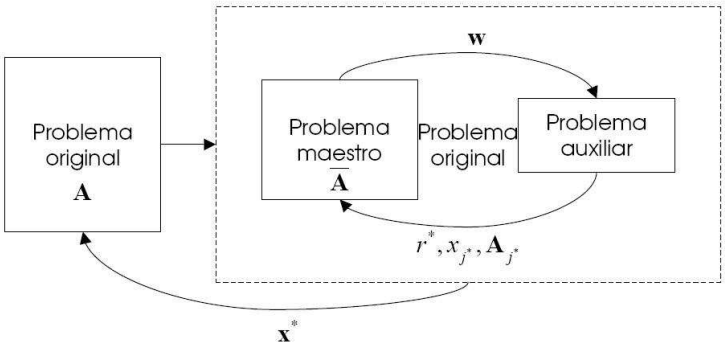

Figura 2–1: Marco general del método de generación de columnas

2.6.3. Algoritmo de Descomposición Dantzig-Wolfe

Este método se aplica a problemas lineales en que existe un grupo de restricciones generales que involucran todas las variables y grupos de restricciones que

afectan a subconjuntos distintos de variables (estructura diagonal en bloques). Formalmente, sea:

$$
\begin{array}{llrcrcrcrcl}
(P) & min & z & = & c^1 \cdot x^1 & & +c^2 \cdot x^2 & & + \ldots & & +c^r \cdot x^r & \\
s.a & & & & & & & & & & & \\
 & & & & A^1 \cdot x^1 & & +A^2 \cdot x^2 & & +... & & +A^r \cdot x^r & = & b^0 \\
 & & & & B^1 \cdot x^1 & & & & & & & = & b^1 \\
 & & & & & & B^2 \cdot x^2 & & & & & = & b^2 \\
 & & & & & & & & \ddots & & & & \\
 & & & & & & & & & & B^r \cdot x^r & = & b^r \\
 & & & & & & & & & & x^1, x^2, ..., x^r & \geq & 0
\end{array}
$$

donde hay m_0 restricciones y m_j que involucran a la variable (vectorial) x^j y además:

$$
\begin{array}{ll}
c^j, x^j \in \mathbb{R}^{n_j} & A^j \in \mathbb{R}^{m_0 \times n_j} \\
b^0 \in \mathbb{R}^{m_0} & b^j \in \mathbb{R}^{m_j} \\
B^j \in \mathbb{R}^{m_j \times n_j} &
\end{array}
$$

Se se puede aprovechar la estructura particular del problema para generar una descomposición mas eficiente: la idea es en vez de resolver un solo gran subproblema, resolver r subproblemas mas pequeños.

Si el poliedro P^j asociado al conjunto de restricciones de la variable x^j es acotado para todo j, se puede escribir cada punto en uno de estos poliedros como combinación lineal convexa de sus vértices (para el caso no acotado no se debe incorporar direcciones extremas).

$$P^j = \{x \in \mathbb{R}^{n_j} \mid B^j x^j = b^j, x^j \geq 0\}$$

Para cada $x^j \in P^j$ puede escribir:

$$\begin{aligned} x^j &= \sum_{k=1}^{N_j} \lambda_k^j x_k^j \\ \sum_{k=1}^{N_j} \lambda_k^j &= 1 \\ \lambda_k^j &\geq 0 \forall j \end{aligned}$$

Notar que N_j representa el número de vértices que tiene el problema P^j. Así, el problema maestro queda como:

$$\begin{aligned} (PM) \qquad min \quad z \; &= \left(\sum_{k=1}^{N_1} c^1 x_k^1\right) \lambda_k^1 \; + \ldots \; + \left(\sum_{k=1}^{N_r} c^r x_k^r\right) \lambda_k^r \\ s.a \\ \left(\sum_{k=1}^{N_1} A^1 x_k^1\right) \lambda_k^1 \; &+ \ldots \; + \left(\sum_{k=1}^{N_r} A^r x_k^r\right) \lambda_k^r = b^0 \\ \sum_{k=1}^{N_1} \lambda_k^1 \; &= 1 \\ &\ddots \\ \sum_{k=1}^{N_r} \lambda_k^r \; &= 1 \\ \lambda_k^j \; &\geq 0 \;\; \forall k, j \end{aligned}$$

Luego, para resolver:

- Suponga que se tiene una solución básica factible para (PM) y sea B la matriz básica asociada.
- Se debe ver si la solución es óptima o si existe un vértice que deba entrar a la base del (PM). Al igual que en el principio general de descomposición, se resuelven subproblemas para evaluar la optimalidad.

  Sea $(\omega, \alpha) = c_{BV} \cdot B^{-1}$ donde:

  - $\omega \in \mathbb{R}^{m_0}$:asociado a las restricciones mantenidas.

- $\alpha \in \mathbb{R}^r$: asociado a las restricciones de convexidad de cada uno de los r poliedros P^j.

Sea $\bar{c}_k^j$ el costo reducido asociado a la variable λ_k^j. Entonces:

$$\begin{aligned} \bar{c}_k^j &= c^j x^j - (\omega, \alpha) \begin{pmatrix} A^j \cdot x^j \\ e^j \end{pmatrix} \\ &= (c^j - \omega \cdot A^j)\, x_k^j - \alpha_j \end{aligned}$$

Para que (PM) sea óptimo, se debe verificar que $\bar{c}_k^j \geq 0 \ \forall$ k,j, pero equivalentemente podemos buscar min $\bar{c}_k^j$ y verificar que sea positivo, dando origen así al siguiente subproblema:

$$\begin{aligned} \underset{k,j}{min} \ \bar{c}_k^j &= \underset{1 \leq j \leq r}{min} \left\{ \underset{1 \leq k \leq N_j}{min} \left[(c^j - \omega A^j)\, x_k^j - \alpha_j \right] \right\} \\ &= \underset{1 \leq j \leq r}{min} \left\{ \underset{x^j \in P^j}{min} \left[(c^j - \omega A^j)\, x^j - \alpha_j \right] \right\} \end{aligned}$$

Entonces, encontrar min c_k^j es equivalente a resolver r subproblemas del tipo:

$$\begin{aligned} (SP^j) \quad min f_j &= (c^j - \omega A^j)\, x^j - \alpha_j \\ s.a \qquad B^j x^j &= b^j \\ x^j &\geq 0 \end{aligned}$$

Luego:

- Si el valor óptimo de $(SP^j) \geq 0 \ \forall$ j, (PM) es óptimo y por tanto (P) es óptimo.
- Si el valor óptimo de $(SP^j) < 0$ para algún q, se tienen variables que entra a la base de (PM): el peso λ_s^q asociado a la solucin óptima de (SP^q),x_s^q. Si existen varios subproblemas con valor óptimo negativo, por convención se elige el más negativo para que entre a la base.

- Si no es óptimo y λ_s^q entra a la base, se debe escoger una variable para que salga.

  Sean:

  - $\bar{A}_s^q = B^{-1} \cdot A_s^q = B^{-1} \begin{pmatrix} A^q \cdot x_s^q \\ e^q \end{pmatrix}$
  - $\bar{b} = B^{-1} \cdot \mathrm{b} = B^{-1} \cdot \begin{pmatrix} b^0 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$

  Luego, el índice t de la variable que sale viene dado por

$$t = argmin \left\{ \frac{\bar{b}_i}{\left(\bar{A}_s^q\right)_i} | \left(\bar{A}_s^q\right)_i > 0 \right\}$$

- Si es óptimo, se debe reconstruir la solución para (P):

$$x^j = \sum_{k=1}^{N_j} \lambda_k x^k$$

2.7. Construcción de Modelos

No existe una metodología muy concreta acerca de como se debe modelar matemáticamente un problema, este asunto tiene mucho que ver con la intuición y arte.

Una forma sencilla y bastante general de ordenar el proceso de modelación, consiste en dividirlo en tres partes:

1. Definición de variables de decisión.
2. Planteamiento de restricciones.
3. Planteamiento de la función objetivo.

2.7.1. Definición de variables

Como primer paso para poder modelar ordenadamente un problema de optimización se debe distinguir que variables son aquellas sobre las que se puede tomar decisiones en el problema y darles un nombre, es decir, saber que variables están bajo control. A veces es necesario incluir variables que si bien no se puede ejercer una decisión directa sobre ellas, pero sirven como herramienta auxiliar ya sea para plantear restricciones o para escribir la función objetivo. Serán variables de decisión por ejemplo la cantidad de producto a enviar desde el centro de producción i hasta el centro de consumo j (llámese x_{ij}), la cantidad de insumos a adquirir en el período t (llámese y_t), el número de horas destinada de la máquina i a trabajar en el proceso j en el período t (llámese z_{ij}), etc.

2.7.2. Planteamiento de función objetivo

En general se puede decir que en un problema de optimización se intenta encontrar el mejor valor de un objetivo. Es por esto que se necesita especificar que criterio utilizar para decir que una solución es mejor que otra. Para ello deberemos especificar una función de $\mathbb{R}^n$ a $\mathbb{R}$ en que una combinación de variables será mejor que otra si genera un mayor valor de la función en el caso de maximización y un menor valor de la función en el caso de minimización. Ejemplos típicos de funciones objetivos vienen dados por maximización de utilidades y minimización de costos, los que deben ser escritos en función de las variables del problema.

2.7.3. Planteamiento de restricciones

En un problema de optimización, se intenta buscar combinaciones de variables de decisión que generen un mejor valor de la función objetivo, pero en la práctica el problema esta limitado por un gran número de restricciones físicas, económicas, técnicas, etc. Es por esto que en el planteamiento del problema se debe especificar que limitantes tienen los valores que puedan tomar las variables de decisión. En

síntesis, en esta parte se debe escribir matemáticamente las limitaciones que impone por naturaleza el problema.

2.8. Aplicación Especial para Problemas de Programación Lineal Entera

Problema de Asignación Generalizada (PAG)

En el PAG el objetivo es encontrar un beneficio máximo al asignar m tareas a n máquinas, tal que cada tarea sea asignada precisamente a una máquina y estén sujetas a las restricciones de capacidad de cada máquinas.

Las variables y parámetros del problema se presentan a continuación:

Cuadro 2–1: Lista de anotaciones de PAG

p_{ij} : Beneficio asociado al asignar la tarea i la máquina j.
w_{ij} : Cantidad de la capacidad de máquina j usada por la tarea i.
d_j : Capacidad de la máquina j.
x_{ij} : 1 si la tarea i es asignada a máquina j; 0 en otro caso.

Fuente: Elaboración propia (2008)

Formulación Natural

$$\text{máx}\, PAG = \sum_{1\leq i\leq m} \sum_{1\leq j\leq n} p_{ij} z_{ij} \tag{2.15}$$

sujeto a

$$\sum_{1\leq j\leq n} z_{ij} = 1 \qquad 1 \leq i \leq m \tag{2.16}$$

$$\sum_{1\leq i\leq m} w_{ij} z_{ij} \leq d_j \qquad 1 \leq j \leq n \tag{2.17}$$

$$x_{ij} \in \{0,1\} \qquad 1 \leq i \leq m, 1 \leq j \leq n$$

Formulación para Generación de Columna (Maestra Restringida)

$$\text{máx}\, PAG_{GC} = \sum_{1\leq j\leq n} \sum_{1\leq k\leq K_j} \left(\sum_{1\leq i\leq m} p_{ij} y_{ik}^j \right) \lambda_k^j \tag{2.18}$$

sujeto a

$$\sum_{1\leq j\leq n} \sum_{1\leq k\leq K_j} y_{ik}^j \lambda_k^j = 1 \qquad 1 \leq i \leq m \tag{2.19}$$

$$\sum_{1\leq k\leq K_j} \lambda_k^j \leq 1 \qquad 1 \leq j \leq n \tag{2.20}$$

$$\lambda_k^j \in \{0,1\} \qquad 1 \leq k \leq K_j, 1 \leq j \leq n$$

donde cada

$$y_k^j = (y_{1k}^j, y_{2k}^j, ..., y_{mk}^j)$$

satisface

$$\sum_{1\leq i\leq m} y_{ik}^j w_{ij} = d_j$$

la relajación LP del problema maestro es más compacta que la relajación LP de la formulación natural porque ciertas soluciones fraccionarias son eliminadas. Específicamente, todas las soluciones de soluciones fraccionarias que no son las combinaciones convexas de soluciones 0-1 con las restricciones de mochila.

Así para obtener la solución óptima de relajación PL se resuelven la relajación LP de:

- Problema Maestra Restringida.
- Problema Precio.

Problema Precio

Para resolver la maestra restringida se puede usar del algoritmo simple revisado y la generación de columnas. Como se esta resolviendo un problema de maximización, se quiere encontrar una columna que tenga un valor positivo (valor negativo si $c_j - c_{BV}B^{-1}a_j$). Una columna esta representada como:

$$\begin{aligned}\bar{c}^j - c_{BV} \cdot B^{-1} \cdot a_j = \bar{c}^j - c_{BV} \cdot B^{-1} \cdot \begin{bmatrix} z^j_{1k} \\ \vdots \\ z^j_{nk} \\ 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ = \bar{c}^j - \sum_{1 \leq i \leq n} \upsilon_i + \nu_j \\ = \sum_{1 \leq i \leq n} p_{ij} - \sum_{1 \leq i \leq n} \upsilon_i + \nu_j \\ = \sum_{1 \leq i \leq n} (p_{ij} - \upsilon_i) - \nu_j\end{aligned} \tag{2.21}$$

donde υ_i es la variable dual asociada con las restricciones de tarea i y ν_j es la variable dual asociada con la restricción de la máquina j, el valor dual ν_j es constante para una maquina, así maximizando el costo reducido sobre todas las maquinas es equivalente al siguiente subproblema:

$$\max_{1 \leq j \leq m} \{z(KP_j) - \nu_j\}$$

y $z(KP_j)$ es el valor de la solución óptima de

$$\text{máx} \sum_{1 \leq i \leq n} (p_{ij} - \upsilon_i)\, x_i^j$$

sujeto a

$$\sum_{1 \leq i \leq n} w_{ij} x_i^j \leq d_j$$

$$x_i^j \in 0, 1 \quad i \in 1, \ldots, n$$

Ramificación

La ramificación estándar sobre las variables λ crea problemas sobre la ramificación cuando ésta es igual a 0. La solución conveniente es ramificar sobre las variables originales:

$$x_{ij} = \sum_{1 \leq k \leq K_j} y_{ik}^j \lambda_k^j$$

Al ramificar sobre x_{ij} por ser una variable binaria se tiene

- $x_{ij} = 0$
    1. No hay ninguna columna para la máquina j con un 1 en la fila i.
- $x_{ij} = 1$
    1. Todas las columnas para la máquina j tiene un 1 en la fila i.
    2. Todas las columnas para la máquina con $k \neq j$ tiene un 0 en la fila i.

2.9. Implicaciones Computacionales y Recursos de Software para la Resolución de Problemas de Optimización de PL y PLE

2.9.1. Conceptos de Programación Estructurada

Introducción

En la década del sesenta salieron a la luz publica los principios de lo que más tarde se llamo Programación Estructurada, posteriormente se libero el conjunto de las llamadas "técnicas para mejoramiento de la productividad en programación" (Improved Programming Technologies - IPTs), siendo la Programación Estructurada una de ellas.

Los programas computarizados pueden ser escritos con un alto grado de estructuración, lo cual les permite ser mas fácilmente comprensibles en actividades tales como pruebas, mantenimiento y modificación de los mismos.

La programación estructurada es una forma de escribir programas de computadora de forma clara, para ello utiliza únicamente tres estructuras: secuencial, selectiva e iterativa; siendo innecesario y no permitiéndose el uso de la instrucción o instrucciones de transferencia incondicional (GOTO).

Hoy en día las aplicaciones informáticas son mucho más ambiciosas que las necesidades de programación existentes en los años 60, principalmente debido a las aplicaciones gráficas, por lo que las técnicas de programación estructurada no son suficientes. Ello ha llevado al desarrollo de nuevas técnicas tales como la programación orientada a objetos y el desarrollo de entornos de programación que facilitan la programación de grandes aplicaciones.

Inconvenientes de la Programación Estructurada

El principal inconveniente de este método de programación, es que se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático su manejo, esto se resuelve empleando la programación modular, definiendo módulos interdependientes programados y compilados por separado. Un método un poco más sofisticado es la programación por capas, en la que los módulos tienen una estructura jerárquica muy definida y se denominan capas.

Modularidad

La modularidad es la capacidad que tiene un sistema de ser estudiado, visto o entendido como la unión de varias partes que interactúan entre sí y que trabajan para alcanzar un objetivo común, realizando cada una de ellas una tarea necesaria para la consecución de dicho objetivo. Cada una de esas partes en que se encuentre dividido el sistema recibe el nombre de módulo. Idealmente un módulo debe poder cumplir las condiciones de caja negra, es decir, ser independiente del resto de los módulos y comunicarse con ellos (con todos o sólo con una parte) a través de unas entradas y salidas bien definidas.

Funciones y Procedimientos

En algunas ocasiones se debe llamar un bloque de código mas de una vez, una forma de hacerlo es escribir las instrucciones tantas veces como se necesite, tornando de esta manera programas con exceso de código y dificultad para descubrir posibles errores, la otra es meter las instrucciones en subprogramas que se invocan cada vez que se necesiten.

Procedimientos

Un procedimiento es un grupo de sentencias que realizan una tarea concreta. En lugar de reescribir el código completo de esa tarea cada vez que se necesite,

únicamente se hace una referencia al procedimiento. Estos pueden recibir o enviar información el programa principal, técnicamente se conoce como PASO DE PARÁMETROS, que puede ser POR REFERENCIA.

Funciones

Las funciones son, al igual que los procedimientos, un conjunto de sentencias que se ejecutan constantemente, la diferencia entre éstas y los procedimientos es que las funciones regresan un valor.

Interfaces de Programación de Aplicaciones

(Application Programming Interface (API) - Interfaz de Programacin de Aplicaciones). Grupo de rutinas (conformando una interfaz) que provee un sistema operativo, una aplicación o una biblioteca, que definen cómo invocar desde un programa un servicio que éstos prestan. En otras palabras, una API representa un interfaz de comunicación entre componentes software.

El software que provee la funcionalidad descrita por una API se dice que es una implementación del API. El API en sí mismo es abstracto, en donde especifica una interfaz y si da detalles de implementación.

Archivos de Cabecera

Los archivos de descripción de interface, también llamados archivos de encabezado (header) ó (archivos .h). Los archivos de encabezado contienen las declaraciones de constantes, variables y funciones de las que consta el módulo, así como llamadas a otros archivos de encabezado necesarios

2.9.2. SYMPHONY

2.9.2.1. Introducción

SYMPHONY es un biblioteca adaptable de código abierto para resolver problemas lineales enteros mixtos por ramificación, corte y precio. La ultima version esta implementado como una biblioteca de llamadas que se puede acceder a través de

llamadas a la interfaz de programación de aplicaciones C, o a través de la clase de interfaz C++ derivado de la COIN-OR [7].

2.9.2.2. SYMPHONY como Optimizador de Caja Negra

SYMPHONY puede ser llamado desde el comando de linea de un terminal en cualquier sistema operativo. Lo siguiente es la secuencia que se llama para cargar en un archivo de formato MPS y resolverlo.

```
./symphony -F ejemplo.mps
```

Para leer y solucionar un modelo en el formato de LP, el comando podría ser

```
./symphony -L ejemplo.lp
```

Para leer y solucionar un modelo de GMPL y el fichero de datos asociado, el comando podría ser

```
./symphony -F ejemplo.mod -D ejemplo.dat
```

Además de la especificación del nombre del archivo de formato, la mayor parte de los parámetros comunes también pueden ser puestos sobre la línea de comandos añadiendo varios maniobras. Llamando SYMPHONY con el simple argumento $-h$ mostrara la lista de todas las opciones.

También se pude usar SYMPHONY como un interactivo instructor para programación lineal entera mixta, solamente se ejecuta el ejecutable sin ningún argumento en el comando de linea, esto es:

```
./symphony
```

El usuario entrará en un ambiente de comando *shell* donde le incitarán para las entradas. El interfaz de usuario consiste en un menú principal, donde un caso es leído y solucionado, un menú de configuraciones, donde los parámetros son puestos, y un menú de exhibición, donde los resultados, la estadística y valores de parámetro son mostrados.

2.9.2.3. Aplicabilidad

SYMPHONY 5.0 es la primera version de SYMPHONY a ser implementado como una biblioteca de llamadas con una nueva interfaz derivado de COIN-OR open solver interface (OSI). Este cambio mejoró notablemente el uso y flexibilidad de SYMPHONY.

SYMPHONY como una biblioteca de llamadas consiste de un completo conjunto de subrutinas para cargar y modificar los datos del problema, configuración de parámetros e invocación de algoritmos. El usuario invoca estas subrutinas a través de la API especificado en el archivo cabecera `symphony_api.h`. Algunos de los comandos básicos son descritos brevemente:

sym open environment(): Abre un nuevo ambiente, y devuelve un indicador de ello. Este indicador entonces tiene que ser pasado como un argumento a todas las otras subrutinas API, este indicador es mantenido para el usuario.

sym parse command line(): Invoca el analizador gramatical de comando de linea para configurar los parámetros usados comúnmente.

sym load problem(): Lee los datos de problema y establece el subproblema raíz.

sym solve(): Soluciona el problema actualmente cargado desde el principio.

sym warm solve(): Soluciona el problema actualmente cargado de un principio.

sym mc solve(): Soluciona el problema actualmente cargado como un problema de multicriterios.

sym close environment(): Libera todos los datos de problema y suprime el ambiente.

Por defecto, SYMPHONY lee un archivo MPS o GMPL especificado por el usuario, aunque este comportamiento pueda ser anulado poniendo en práctica una función de usuario que lee los datos de un archivo en un formato personalizado. Como un ejemplo del empleo de las funciones de biblioteca, la Figura (2.9.2.3) muestra el código para poner en práctica una genérica aplicación para un problema lineal

```
int main(int argc, char **argv)
{
   sym_environment *env = sym_open_environment();
   sym_parse_command_line(env, argc, argv);
   sym_load_problem(env);
   sym_solve(env);
   sym_close_environment(env);
}
```

Figura 2–2: Fuente: Manual 5.1.7 SYMPHONY, MILP (2007)

entero mixto con la configuraciones de los parámetros estándar. Durante la llamada a `sym_parse_command_line()`, SYMPHONY determina que el usuario quiere leer en un archivo MPS. Durante la llamada subsecuente de `sym_load_problem()`, el archivo es leído y los datos de problema almacenados. Para leer un archivo MPS llamado sample.mps y solucionar usando este programa, el comando siguiente sería emitido como:

```
./milp -F ejemplo.mps
```

2.9.3. GLPK

2.9.3.1. Introducción

GLPK (Kit de programación lineal GNU) [8] es un conjunto de rutinas escrita bajo el lenguaje de programación ANSI C y organizada de forma de una biblioteca en donde pueden ser llamadas. Esto esta encaminado para resolver problemas de programación lineal LP, programación lineal entera LPE y otros problemas relacionados.

2.9.3.2. GLPK como optimizador de caja negra

GLPK al igual que SYMPHONY posee de un optimizador llamado glpsol para ser llamado por comandos de linea. Al Llamar el optimizador de GLPK con el simple argumento $-help$ mostrará la lista de todas las opciones y parámetros. Para leer el

modelo y la data de un problema PL ó LPE se utiliza el siguiente comando:

```
.\glpsol -m model.mod -d data.dat
```

donde $-m$ lee el archivo del modelo y $-d$ lee la data del modelo.

2.9.3.3. Aplicabilidad

Problema de Programación Lineal

GLPK asume la siguiente formulación para problemas de programación lineal LP

minimice (or maximice)

$$z = c_1 x_{m+1} + c_2 x_{m+2} + \cdots + c_n x_{m+n} + c_0 \tag{1,1}$$

sujeto a las restricciones lineales

$$\begin{aligned} x_1 &= a_{11}x_{m+1} + a_{12}x_{m+2} + \cdots + a_{1n}x_{m+n} \\ x_2 &= a_{21}x_{m+1} + a_{22}x_{m+2} + \cdots + a_{2n}x_{m+n} \\ &\cdots\cdots\cdots\cdots\cdots\cdots \\ x_m &= a_{m1}x_{m+1} + a_{m2}x_{m+2} + \cdots + a_{mn}x_{m+n} \end{aligned} \tag{1,2}$$

y cotas de variables

$$\begin{aligned} l_1 \leq \quad x_1 \quad &\leq u_1 \\ l_2 \leq \quad x_2 \quad &\leq u_2 \\ &\cdots\cdots \\ l_{m+n} \leq x_{m+n} &\leq u_{m+n} \end{aligned} \tag{1,3}$$

Donde: $x_1, x_2, \ldots, x_m$ son las variables auxiliares; $x_{m+1}, x_{m+2}, \ldots, x_{m+n}$ son las variables estructurales; z es la función objetivo; $c_1, c_2, \ldots, c_n$ son los coeficientes de las variables de la función objetivo; c_0 es el término constante de la función objetivo;

$a_{11}, a_{12}, \ldots, a_{mn}$ son los coeficientes de las restricciones; $l_1, l_2, \ldots, l_{m+n}$ son las cotas inferiores de las variables; $u_1, u_2, \ldots, u_{m+n}$ son las cotas superiores de las variables.

Las variables auxiliares también pueden ser llamadas filas, porque ellas corresponden a las filas de la matriz de restricciones. Similarmente, las variables estructurales son también llamadas columnas, porque ellas corresponden a las columnas de la matriz de restricciones.

Las variables acotadas pueden ser tanto finitas como infinitas. Además, las cotas inferiores y superiores pueden ser equivalentes en cada lado. Así, los siguientes tipos de variables son posibles:

Cuadro 2–2: Tipos de variables

Cotas de variables	tipo de variables
$-\infty < x_k < +\infty$	Variable no acotadas
$l_k \leq x_k < +\infty$	Variable con una cota inferior
$-\infty < x_k \leq u_k$	Variable con una cota superior
$l_k \leq x_k \leq u_k$	Variable con doble cota
$l_k = x_k = u_k$	Variable fija

Fuente: GNU Linear Programming Kit, (2008)

Note que los tipos de variables muestran que son aplicables tanto para variables estructurales y auxiliares.

Para resolver el problema LP (1.1)–(1.3) es buscar los valores de todas las variables estructurales y auxiliares, tales que:

Satisfaga todas las restricciones lineales, y estén dentro de sus cotas, y provee el mas pequeño (en caso de minimización) o el mas grande (en caso de maximización) valor en la función objetivo.

Problema de Programación Lineal Entera Mixta

El problema de programación lineal entera LPEM es un problema LP en el cual algunas variables requieren ser adicionalmente enteras.

GLPK asume que el problema PLE tiene la misma formulación ordinaria del problema PL (1.1)–(1.3), incluyendo las variables auxiliares y estructurales, las cuales pueden tener cotas inferiores y/o superiores. Sin embargo, en caso de un

problema de PLE algunas variables pueden requerir ser enteras. Esta adicional restricción significa que los valores de cada variable entera debe ser solamente un número entero.

Ejemplo Simple

Para poder entender que es GLPK desde el punto de vista usuario, considere el siguiente problema PL:

maximice

$$z = 10x_1 + 6x_2 + 4x_3$$

sujeto a

$$\begin{aligned} x_1 + \ \ x_2 + \ \ x_3 &\leq 100 \\ 10x_1 + 4x_2 + 5x_3 &\leq 600 \\ 2x_1 + 2x_2 + 6x_3 &\leq 300 \end{aligned}$$

Donde todas las variables son no negativas

$$x_1 \geq 0, \ x_2 \geq 0, \ x_3 \geq 0$$

Como primer paso este problema LP podría ser transformado a la forma estándar (1.1)–(1.3). Esto se puede hacer fácil introduciendo las variables auxiliares, por cada una de las desigualdades de la restricción original. Así, el problema puede ser reformulado como sigue:

maximice

$$z = 10x_1 + 6x_2 + 4x_3$$

sujeto a

$$\begin{aligned} p &= \ \ x_1 + \ \ x_2 + \ \ x_3 \\ q &= 10x_1 + 4x_2 + 5x_3 \\ r &= \ \ 2x_1 + 2x_2 + 6x_3 \end{aligned}$$

y cotas de variables

$$-\infty < p \leq 100 \qquad 0 \leq x_1 < +\infty$$
$$-\infty < q \leq 600 \qquad 0 \leq x_2 < +\infty$$
$$-\infty < r \leq 300 \qquad 0 \leq x_3 < +\infty$$

Donde p, q, r son variables auxiliares (filas), y x_1, x_2, x_3 son las variables estructurales (columnas).

El ejemplo del programa en C mostrado abajo usa de las rutinas API de GLPK en orden para resolver este problema PL.

```
/* sample.c */
#include <stdio.h>
#include <stdlib.h>
#include <glpk.h>
int main(void)
{     glp_prob *lp;
      int ia[1+1000], ja[1+1000];
      double ar[1+1000], z, x1, x2, x3;
s1:   lp = glp_create_prob();
s2:   glp_set_prob_name(lp, "sample");
s3:   glp_set_obj_dir(lp, GLP_MAX);
s4:   glp_add_rows(lp, 3);
s5:   glp_set_row_name(lp, 1, "p");
s6:   glp_set_row_bnds(lp, 1, GLP_UP, 0.0, 100.0);
s7:   glp_set_row_name(lp, 2, "q");
s8:   glp_set_row_bnds(lp, 2, GLP_UP, 0.0, 600.0);
s9:   glp_set_row_name(lp, 3, "r");
s10:  glp_set_row_bnds(lp, 3, GLP_UP, 0.0, 300.0);
s11:  glp_add_cols(lp, 3);
```

```
s12:  glp_set_col_name(lp, 1, "x1");
s13:  glp_set_col_bnds(lp, 1, GLP_LO, 0.0, 0.0);
s14:  glp_set_obj_coef(lp, 1, 10.0);
s15:  glp_set_col_name(lp, 2, "x2");
s16:  glp_set_col_bnds(lp, 2, GLP_LO, 0.0, 0.0);
s17:  glp_set_obj_coef(lp, 2, 6.0);
s18:  glp_set_col_name(lp, 3, "x3");
s19:  glp_set_col_bnds(lp, 3, GLP_LO, 0.0, 0.0);
s20:  glp_set_obj_coef(lp, 3, 4.0);
s21:  ia[1] = 1, ja[1] = 1, ar[1] =  1.0; /* a[1,1] =  1 */
s22:  ia[2] = 1, ja[2] = 2, ar[2] =  1.0; /* a[1,2] =  1 */
s23:  ia[3] = 1, ja[3] = 3, ar[3] =  1.0; /* a[1,3] =  1 */
s24:  ia[4] = 2, ja[4] = 1, ar[4] = 10.0; /* a[2,1] = 10 */
s25:  ia[5] = 3, ja[5] = 1, ar[5] =  2.0; /* a[3,1] =  2 */
s26:  ia[6] = 2, ja[6] = 2, ar[6] =  4.0; /* a[2,2] =  4 */
s27:  ia[7] = 3, ja[7] = 2, ar[7] =  2.0; /* a[3,2] =  2 */
s28:  ia[8] = 2, ja[8] = 3, ar[8] =  5.0; /* a[2,3] =  5 */
s29:  ia[9] = 3, ja[9] = 3, ar[9] =  6.0; /* a[3,3] =  6 */
s30:  glp_load_matrix(lp, 9, ia, ja, ar);
s31:  glp_simplex(lp, NULL);
s32:  z = glp_get_obj_val(lp);
s33:  x1 = glp_get_col_prim(lp, 1);
s34:  x2 = glp_get_col_prim(lp, 2);
s35:  x3 = glp_get_col_prim(lp, 3);
s36:  printf("\nz = %g; x1 = %g; x2 = %g; x3 = %g\n",
         z, x1, x2, x3);
s37:  glp_delete_prob(lp);
```

```
      return 0;
} /* eof */
```

La declaración `s1` crea un problema objeto. Por los momento el objeto creado esta inicialmente vacío. La declaración `s2` asigna el nombre simbólico a el problema objeto.

La declaración `s3` llama la rutina `glp_set_obj_dir` en orden para configurar la dirección de la función objetivo, donde `GLP_MAX` significa maximización.

La declaración `s4` añade tres filas al problema objeto.

La declaración `s5` asigna un nombre simbólico `p` para la primera fila, y la declaración `s6` configura el tipo de cota de la primera fila, donde `GLP_UP` significa que la fila tiene una cota superior. La declaración `s7`, `s8`, `s9` y `s10` son usados en el mismo orden para asignar los nombres simbólicos `q` y `r` para la segunda y tercera fila y configurar sus tipos de cotas.

La declaración `s11` añade tres columnas al problema objeto.

La declaración `s12` asigna el nombre simbólico `x1` para la primera columna, la declaración `s13` configura el tipo de cota de la primera columna, donde `GLP_LO` significa que la columna tiene una cota inferior, y la declaración `s14` configura el coeficiente de la columna en la función objetivo. Las declaraciones `s15--s20` son usados en el mismo orden para asignar los nombres simbólicos `x2` y `x3` para la segunda y tercera columna y configurar sus tipos, cotas y coeficientes de la función objetivo.

La declaración `s21--s29` prepara los elementos no nulos de la matriz de restricciones. Los indices de filas en cada elemento son almacenados en el vector `ia`, los indices de columnas indica que son almacenados en el vector `ja`, y los valores númericos correspondientes a los elementos son almacenados en el vector `ar`. Entonces la declaración `s30` llama la rutina `glp_load_matrix`, el cual carga la información de estos vectores sobre el problema objeto.

De esta forma toda la data entra al problema objeto, por consiguiente la declaración `s31` llama la rutina `glp_simplex`, el cual es una aplicación al método simplex, como ultimo paso para resolver el problema. Esta rutina busca un valor óptimo y almacena todas la información relevante devuelta sobre el problema objeto.

La declaración `s32` obtiene un valor calculado de la función objetivo, y la declaración `s33`—`s35` obtiene los valores calculados de las variables estructurales (columnas), el cual corresponde a la óptima solución básica encontrada por el optimizador.

La declaración `s36` escribe la solución óptima sobre la salida estándar. Imprimir la salida es parecido a lo siguiente:

```
*     0:   objval =   0.000000000e+00   infeas =   0.000000000e+00 (0)
*     2:   objval =   7.333333333e+02   infeas =   0.000000000e+00 (0)
OPTIMAL SOLUTION FOUND
z = 733.333; x1 = 33.3333; x2 = 66.6667; x3 = 0
```

Finalmente, La declaración `s37` llama la rutina `glp_delete_prob`, el cual libera toda la memoria asignado al problema objeto.

2.9.4. Ambiente de Desarrollo

En detalles de experimentos e implementación, se utilizaron de las herramientas ó software abajo descritas:

Editor: Kate v.-2.5.8, Lenguaje: ANSI C, Compilador: GNU/GCC v.-3.3.6 y del Sistema Operativo: GNU Linux Ubuntu v.- 7.10.

Capítulo 3
DESARROLLO METODOLÓGICO

3.1. Datos del Problemas

La información relevante obtenida de los datos recolectados por Cuadrado M., fueron la criticidad o tiempo que dura un vehículo en el patio antes de ser llevado a un concesionario, la longitud de los vehículos y las longitudes de los tipos de unidades de transporte, estas se muestran en las siguientes tablas:

Cuadro 3–1: Longitudes de vehículos

Tipo de vehículo	Longitud (milímetro)
1	4813
2	3677
3	4362
4	4152
5	4228

Fuente: Cuadrado M. Modelos Matemáticos de Optimización (2005)

La criticidad del vehículo se mide en dias y su valor asociado varia entre 1 y 15. Estos datos serán utilizados para generar los modelos dividiendo el número de vehículos, nodrizas y concesionarios a medida que se particionan el problema general en mas regiones.

Cuadro 3–2: Longitudes de las unidades de transporte

Tipo de unidad de transporte	Longitud (milímetro)
1	23950
2	31500
3	31048
4	30970
5	28600
6	34080
7	14000
8	30776
9	29480
10	31080

Fuente: Cuadrado M. Modelos Matemáticos de Optimización (2005)

3.2. La Construcción General del Problema de Mochila Múltiple con Colores (MMC)

En esta sección se explicara brevemente el problema general, el cual se descompone en subproblemas, uno por cada ruta (patio-región), para construir un problema de mochila múltiple con colores, que describe de manera precisa la situación real del problema de asignación de las cargas a las unidades de transportes. La representación matemática ésta basada en la construcción de modelos descrita en la sección (2.7).

3.2.1. Descomposición del Problema por Patio-Región

La descomposición es realizada cuando el pais se dívide en regiones y se resuelve la asignanción de las rutas a las unidades de transporte. Una ruta representa el patio de origen y la región destino de una nodriza.

La descomposición del problema general crea un subproblema de mochila multiple con colores uno por cada ruta, y el tamaño de cada subproblema depende de que tan grueso haya sido la descomposición del problema general, esto es, si el pais se divide en mas regiones el primer paso de asignar las nodrizas a las rutas seria mas complicado de resolver, mientras que el segundo paso que es asignar las cargas a las unidades de transporte seria mas fácil de resolver, por otro lado, si el pais se divide

en menos regiones el primer paso seria mucho mas fácil de resolver y el segundo seria mas complicado. (ver Tabla 4–1)

3.2.2. Conjuntos y Parámetros del Modelo

El siguiente problema consistes en asignar las cargas de vehículos (artículos) a las unidades de transporte (mochilas) de manera que maximice la carga de los vehículos críticos y minimice el número de paradas (colores) para cada unidad de transporte. Cada vehículo tiene asociado un concesionario destino (esto es un atributo color), de manera que una parada es realizada por una unidad de transporte j, cuando este tiene que llevar un vehículo i a un concesionario c, ésta paradas por lo general representan costos para la empresa transportista.

Entonces se tiene un conjunto de vehículos N, un conjunto de nodrizas M y un conjunto de concesionarios C, donde la criticidad, longitud y el concesionario de cada vehículo serán representados como parámetros de entradas junto con la capacidad de carga de cada unidad de transporte.

3.2.3. Identificar Variables de Decisión

En cualquier modelo de programación lineal entera, las variables de decisíon deben describir por completo las decisiones que se tienen que tomar. Por tanto, las decisiones y exigencias que deben considerarse para asignar las cargas a las unidades de transporte, con las representaciones matemáticas de cada una de ella serían las siguientes:

Determinación de cargas a unidad de transporte ($x_{i,j}$): debido a la necesidad de determinar que vehículos serán cargados en la unidades de transporte para ser enviados a los diferentes concesionarios, se definen variables que determinan esta exigencia. La especificación de la carga depende de algunas consideraciones importantes; como la criticidad y las condiciones para que una carga determinada sea factible.

Parada de descarga de la unidad de transporte ($y_{c,j}$): una parada esta definida por el envío de una unidad de transporte a un concesionario. Estas variables están descritas de manera que puedan contarse el número de paradas que deben realizar las unidades de transportes para entregar los vehículos que les fueron asignados.

3.2.4. Identificación de Restricciones

Capacidad de carga: es importante definir restricciones que representen la oferta de cargas en las unidades de transporte. Para este caso, la representación se da en términos de las longitudes de los vehículos y las unidades de transporte disponibles.

$$\sum_{i \in N} l_i x_{ij} \leq L_j \qquad 1 \leq j \leq M \tag{3.1}$$

Envío de un vehículo en solo una unidad de transporte: esta restriccíon evita el envío de un vehículo en mas de una unidad de transporte. Esto es debido a que en la situación real, si un vehículo es cargado en una unidad de transporte, el mismo no debe ser cargado en otra.

$$\sum_{j \in M} x_{ij} \leq 1 \qquad 1 \leq i \leq N \tag{3.2}$$

Limite de paradas: esta restricción simplifica la función objetivo, y restringe el limite de concesionarios o paradas para cada nodriza.

$$\sum_{c \in C_j} y_{cj} \leq k \qquad 1 \leq j \leq M, k \in \mathbb{Z}^+ \tag{3.3}$$

Coloreado de vehículos: esta restricción ayuda a vincular el concesionario en que debe de ir el vehículo con la unidad de transporte.

$$x_{ij} \leq y_{c_i j} \qquad j \in M,\ i \in N \tag{3.4}$$

3.2.5. Lista General de Anotaciones

A continuación se proveerá un resumen de las características presentes en los modelos propuestos en la siguiente tabla (ver Tabla 3–3).

Cuadro 3–3: Lista de anotaciones MMRC

n	: Total número de vehículos.
m	: Total número de nodrizas.
c	: Total número de concesionarios.
N	: Conjunto de vehículos.
M	: Conjunto de nodrizas.
l_i	: Longitud del vehículo i.
Crt_i	: Criticidad del vehículo i.
L_j	: Longitud de la nodriza j.
C_j	: Conjunto de concesionarios en nodriza j.
c_i	: El concesionarios del vehículo i.
x_{ij}	: 1 si el vehículo i es cargado en la nodriza j; 0 otro caso.
y_{cj}	: 1 si hay una para en concesionario c por la nodriza j; 0 otro caso.

Fuente: Elaboración propia (2008)

3.3. La Formulación de los Modelos Matemáticos

A continuación se mostrarán los modelos propuestos en la presente investigación explicando con mas detalles su representación en el problema de asignación de cargas a unidades de transportes, variando dos distintas manera de incluir el atributo color en el modelo de MMC.

3.3.1. Mochila Múltiple con Restricción de Color (MMRC)

La caracterśtica principal de este modelo es que limitara el número de paradas (colores) en cada unidad de transporte (mochila) mediante el uso de una restricción que sera agregada a la formulación, simplificando el modelo MMRC a una sola función objetivo.

3.3.1.1. Función Objetivo

El objetivo considerado bajo este modelo es el siguiente: Maximizar la entrega de vehículos críticos. La criticidad se considera como el retardo o cantidad de dias laborales que tiene un vehículo en el patio de carga; esto define un índice de criticidad por vehículo, de manera que el índice mas alto representa el vehículo mas crítico.

3.3.1.2. Formulación del Modelo

La formulación del modelo de mochila múltiple con restricción de color, se presenta como:

$$\text{máx}\, MMRC = \sum_{i=1}^{N} \sum_{j=1}^{M} Crt_i x_{ij} \tag{3.5}$$

sujeto a

$$\sum_{i \in N} l_i x_{ij} \leq L_j \qquad j \in M \tag{3.6}$$

$$\sum_{j \in M} x_{ij} \leq 1 \qquad i \in N \tag{3.7}$$

$$\sum_{c \in C_j} y_{cj} \leq k \qquad j \in M, k \in \mathbb{Z}^+ \tag{3.8}$$

$$x_{ij} \leq y_{c_i j} \qquad j \in M,\ i \in N \tag{3.9}$$

$$x_{ij} \in \{0,1\} \qquad j \in M,\ i \in N$$

$$y_{cj} \in \{0,1\} \qquad c \in C_j,\ j \in M$$

Donde la declaración (3.5) es la función objetivo, las declaraciones (3.6)–(3.9) son las restricciones del modelo y las variables toman valores binarios.

3.3.1.3. Dimensiones del Modelo

El número total de variables en esta formulación es: $m \cdot (n + c)$, mientras que el número total de restricciones es: $2m + n + n \cdot m$.

3.3.2. Mochila Múltiple Multiobjetivo con Colores (MMMC)

Este modelo sera representado mediante el método de suma ponderada descritas en la sección (2.2.3.1) estableciendo una sola función objetivo como:

$$(1-\alpha)z_1 - \alpha z_2, \quad donde \quad \alpha \in [0,1]$$

3.3.2.1. Función Objetivo

Los objetivos considerados bajo este modelo son los siguientes:

- Maximizar la entrega de vehículos críticos.
- Minimizar la paradas de descarga. Las paradas para descargar representan costos de traslado, mano de obra y tiempos más elevados en el envío de vehículos.

3.3.2.2. Formulación del Modelo

La formulación del modelo de mochila múltiple multiobjetivo con colores (MMMC), se presenta como:

$$\text{máx}\, MMC = \beta \cdot \sum_{i\in N}\sum_{j\in M} Crt_i x_{ij} - \alpha \cdot \sum_{j\in M}\sum_{c\in C_j} y_{cj} \tag{3.10}$$

$$\alpha, \beta \in \mathbb{R}, \alpha + \beta = 1$$

sujeto a

$$\sum_{i\in N} l_i x_{ij} \leq L_j \qquad j \in M \tag{3.11}$$

$$\sum_{j\in M} x_{ij} \leq 1 \qquad i \in N \tag{3.12}$$

$$x_{ij} \leq y_{c_i j} \qquad j \in M,\ i \in N \tag{3.13}$$

$$x_{ij} \in \{0,1\} \qquad j \in M,\ i \in N$$

$$y_{cj} \in \{0,1\} \qquad c \in C_j,\ j \in M$$

Donde la declaración (3.10) es la función objetivo, las declaraciones (3.11)–(3.13) son las restricciones del modelo y las variables toman valores binarios.

3.3.2.3. Dimensiones del Modelo

El número total de variables en esta formulación es: $m \cdot (n + c)$, mientras que el número total de restricciones es: $m + n + n \cdot m$.

3.4. Formulación para Resolver Via Generación de Columnas

En esta sección se propone una formulación para el modelo de mochila múltiple con colores para resolver via generación de columnas. Esta formulación tiene significativamente mas columnas que la función original [9], pero ofrece muchas ventajas que facilita obtener el valor óptimo al resolver la relajación del PL.

3.4.1. Variables de Decisión

Esta variable representa una muestra factible de carga, sea u_k la muestra de cargas asociada a la nodriza k. Esta muestra (patrón) tiene un 1 en la fila correspondiente a la mochila (nodriza) usada en esa muestra, y tiene un conjunto de unos en las filas correspondiente a los artículos (vehículos) en la muestra de carga, esto es:

$$u_i = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ \vdots \\ v_{n_1} \\ m_1 \\ m_2 \\ \vdots \\ m_{n_2} \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ \vdots \\ 1 \end{pmatrix}$$

La variable u_i indica que hace uso de la nodriza m_{n_2} y la carga factible las constituyen los vehículos v_1, v_2, ... y v_{n_1}. Esta variable es una variable binaria que representa si una muestra factible es elegida o no en la solución.

3.4.2. Función Objetivo

Para que exista una correspondencia uno-a-uno entre las soluciones factibles de esta problema de "set-packing" y las soluciones factibles de la formulación original, se construye el coeficiente $\bar{c}_k = \sum_{i \in N_k} Crit_i$ para el modelo MMRC y el coeficiente $\bar{c}_k = (1-\alpha) \sum_{i \in N_k} Crit_i - \alpha \sum_{c \in C_k} p_c$ para el modelo MMMC, y obtener de esta forma idénticos valores objetivos.

El objetivo considerado en este modelo es el siguiente: Maximizar el número de modelos factibles.

3.4.3. Identificación de Restricciones

Vehículo único: esta restricción establece que un vehículo sea asignado solamente una vez. Note que un vehículo puede estar en diferentes muestras u_k.

Nodriza única: como una muestra hace referencia a una nodriza, es importante establecer la elección de solo una nodriza entre un conjunto de muestras factibles.

3.4.4. Lista de Anotaciones

Cuadro 3–4: Lista de anotaciones de $MMRC_{GC}$

P	: Es el conjunto de muestras de cargas factibles.
PS_j	: Es el conjunto de muestras de cargas factibles que hace uso de la nodriza j.
PO_i	: Es el conjunto de muestras de cargas factibles que contienen el vehículo i.
N_k	: Conjunto de indices de vehículos en la muestra u_k.
M_k	: La mochila usada en la muestra u_k
C_k	: Conjunto de indices de paradas en la muestra u_k

Fuente: Elaboración propia (2008)

3.4.5. Formulación del Modelo

$$\text{máx}\, MMRC_{GC} = \sum_{k \in P} \bar{c}_k u_k \tag{3.14}$$

sujeto a

$$\sum_{k \in PO_i} u_k \leq 1 \qquad i \in N \tag{3.15}$$

$$\sum_{k \in PS_j} u_k \leq 1 \qquad i \in M \tag{3.16}$$

$$u_k \in \{0, 1\} \qquad k \in P \tag{3.17}$$

3.4.6. Dimensiones del Modelo

El número de variables en esta formulación puede ser enorme debido a que hay muchas formas de establecer una muestra de carga factible. El número de restricciones del modelo es dado por: n + m.

3.4.7. Generar Columnas con Costos Reducidos Positivos

Para mejorar la solución monóticamente se necesita ser capaz de encontrar variables con costos reducidos positivos (con respecto a la solución dual de (3.14)-(3.17)) si las hay.

Para una solución dual π el costo reducido de la variable u_k con el coeficiente $\bar{c}_k$ correspondiente al modelo MMRC es simplemente:

$$\begin{aligned}
\bar{c}_k - c_{BV} \cdot B^{-1} \cdot a_k &= \bar{c}_k - \begin{bmatrix} \pi_1 & \dots & \pi_{n+m} \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ \vdots \\ v_{n+m} \end{bmatrix} \\
&= \bar{c}_k - \sum_{i=1}^{n+m} \pi_i \cdot v_i \\
&= \bar{c}_k - \sum_{i \in N_k} \pi_i + \pi_{M_k} \\
&= \sum_{i \in N_k} Crt_i - \sum_{i \in N_k} \pi_i + \pi_{M_k} \\
&= \sum_{i \in N_k} (Crt_i - \pi_i) - \pi_{M_k}
\end{aligned} \tag{3.18}$$

haciendo referencias del coeficiente que representa al modelo MMMC, el costo reducido de la variable u_k viene dado por:

$$\sum_{i \in N_k} ((1-\alpha) Crt_i - \pi_i) - \alpha \sum_{c \in C_k} p_c - \pi_{M_k}$$

Para cada nodriza j se tiene un problema de optimización que buscará la muestra u_k con $M_k = j$ que tenga el más alto costo reducido. El ultimo termino del costo reducido es constante para una nodriza fija; así, maximizando el costo reducido sobre todo $\{u_k : M_k = j\}$ es equivalente al siguiente subproblema dependiendo de los casos:

Con Restricción de Color:

$$\text{máx}\, MC = \sum_{i \in N_j} (Crt_i - \pi_i) x_{ij} \tag{3.19}$$

sujeto a

$$\sum_{i \in N_j} l_i x_{ij} \leq L_j \tag{3.20}$$

$$\sum_{c \in C_j} y_{cj} \leq k \tag{3.21}$$

$$x_{ij} \leq y_{c_i j} \qquad i \in N_j \tag{3.22}$$

$$y_{c_i j}, x_{ij} \in \{0, 1\} \qquad i \in N_j$$

Sin Restricción de Color:

$$\text{máx}\, MC = \sum_{i \in N_j} ((1 - \alpha) Crt_i - \pi_i) x_{ij} - \alpha \sum_{c \in C_j} y_{cj} \tag{3.23}$$

sujeto a

$$\sum_{i \in N_j} l_i x_{ij} \leq L_j \tag{3.24}$$

$$x_{ij} \leq y_{c_i j} \qquad i \in N_j \tag{3.25}$$

$$y_{c_i j}, x_{ij} \in \{0, 1\} \qquad i \in N_j$$

Ambos modelos hace referencia a un problema de mochila multiple con colores. La muestra generada u_k contendrá aquellos vehículos cuya correspondiente valor son uno en la solución óptima. El termino constante en el costo reducido implica que solo interesa las muestras de cargas cuyo valor objetivo (3.19) o (3.23) excedan a π_j. Resolver el PLE del problema de mochila múltiple con colores es mas simple cuando se requieren de estas cotas inferiores (Lower Bound) sobre los costos reducidos ya que puede ser de mucha ayuda para concluir que no es posible generar una muestra de cargas en esa nodriza.

3.4.8. Cota Superior

La sección (3.4.7) señaló como solucionar la relajación LP del problema maestro por iteraciones solucionando más pequeños subproblema LPE y generando columnas, pero se necesita algo más. Hay que ser capaz de sacar una cota superior sobre el valor óptimo objetivo de la relajación llena LP en cada iteración. Hay dos motivos para esto. El primero es que en un algoritmo de "Branch-cut-price"se puede sondear un nodo de árbol de búsqueda si la cota superior sobre el valor óptimo objetivo de la relajación LP en el nodo es ya inferior que el valor de una solución actualmente conocida factible. La segunda razón es que sin una cota superior sobre el valor óptimo de la relajación LP del problema lleno no se podría contar que tan cerca se esta a la optimalidad.

Pero se puede utilizar el enfoque de la descomposición Dantzig-Wolfe (Chvátal 1983, Dantzig y 1960 Wolfe, Lasdon 1970). Como la suma de todas las variables en PO_i es no más que uno, el valor objetivo de la relajación LP no puede cambiarse más que el coste reducido más alto (o una cota superior sobre aquel valor) como consecuencia del cambio de los valores de las variables en PO_i. Para conseguir una cota superior sobre el coste reducido se puede usar la relajación LP del subproblema, que es muy fácil para solucionar. Ahora sumándole todas las cotas superiores "por nodriza" al valor óptimo objetivo de la corriente LP la relajación cede una cota superior sobre el grado óptimo de la relajación llena LP.

3.5. Algoritmos Propuestos

Ramificación y Acotamiento

La configuración predeterminada del optimizador de GLPK usa como estrategia de ramificación y acotación el heurístico de Driebeck y Tomlin para escoger una variable para la ramificación, y la mejor proyección heurística para el retroceso, a

este algoritmo se denotará como BB. Otra estrategia de ramificación utilizada para resolver los modelos será denotado por BBF, el cual ramificará sobre el valor de la variable más fraccionaria.

Generación de Columnas y Cortes

Para obtener una solución óptima del problema de asignación de cargas a unidades de transporte mediante la técnica de generación de columnas, es a través del tradicional algoritmo de ramificación y precio descrito en la sección (2.5) (ver ejemplo de en la sección (2.8)) resolviendo la relajación LP de la formulación para resolver via generación de columnas en cada nodo de árbol y ramificando sobre las variables originales. El desarrollo e implementación de este algoritmo no fueron posibles para esta investigación de manera que se propone hacer las pruebas de los modelos a través de un heurístico combinando generación de columnas y ramifición y acotamiento sobre el conjuntos de columnas generadas para encontrar una solución buena.

El heurístico propuesto para resolver los modelos de MMRC y MMMC es el siguiente:

1. Primero se comienza con un pequeño número de patrones (cargas factibles) para resolver la relajación LP de la formulación del PM restringida a este conjunto de variables. Los costos de estas variables auxiliares son penalizadas con una cantidad muy negativa para que nunca sean candidatas a entrar a la base.
2. Para cada nodriza se resuelve un problema de mochila múltiple con atributo color, generando nuevas variables con costos reducido positivo, estas columnas serán añadidas a la formulación del PM si el valor objetivo excede la constante π_{M_j} (variables dual asociada a la fila de la nodriza j).
3. Si ninguna columna con costo positivo reducido es generada entonces la solución de la relajación del PM es el valor óptimo. Este valor además es una cota superior

para el nodo 0 del árbol de ramificación y una mejor cota conocida para la formulación original.

4. La segunda fase del heurístico trata de resolver la formulación del PM como un PLE restringido al conjunto de todas las columnas generadas.
5. Si en la solución obtenida de PLE de la formulación del PM no provee el uso de todas la nodrizas, se generan nuevas columnas resolviendo los subproblemas restringido al conjunto de vehículos y nodrizas que no fueron usadas.

De este modo se puede halla una solución probablemente buena para comparar en tiempos. Este método se muestra en el Apéndice (C) con las representaciones de sus dos fases. Cada subproblema de mochila sera resuelto iterativamente hasta no generar costos reducidos positivos, la técnica empleada para resolver los subproblemas de mochila individual y el problema maestro es mediante el algoritmo de ramificación y corte.

3.6. Obtención de Soluciones a partir de los Modelos Propuestos

A partir de la clasificación de los algoritmos utilizados para obtener soluciones para problemas de optimización combinatoria descritos en la sección (2.5), se ha hecho uso de alguno de ellos para la resolución de los modelos matemáticos propuestos en este capítulo. Para resolver los modelos de mochila múltiple con o sin restricción de color se emplearan los algoritmos exactos de búsqueda para resolver los PLE (Branch and Bound). Solamente para el problema de mochila múltiple con colores (sin restricción de color) se realizara un estudio mas extenso para proporcionar soluciones interesantes del frente de pareto.

De acuerdo al análisis realizado por Cuadrado Marlyn de los optimizadores de PLE se usaran los software que permitan la lectura en lenguaje GMPL, tales como GLPK y SYMPHONY. Al considerar las bondades que poseen estos algoritmos en estrategias de búsquedas y reglas de ramificación se ha decidido por el empleo de GLPK para resolver los modelo propuestos ya que posee las técnicas necesarias para resolver problemas enteros y lineales descritas en librerías mas fáciles de manejar. Otra razón es que al utilizar los optimizadores glpsol y symphony de GLPK y SYMPHONY respectivamente sin modificar los parámetros estándar para resolver un problema piloto, GLPK resulto ser mas eficiente en los tiempo de ejecución, sin embargo SYMPHONY es una plataforma que posee características deseables para lograr implementar a futuro las técnicas de ramificación presentes en la investigación y recomendadas para futuras investigaciones. Con respecto al modelo MMMC por ser un problema con dos objetivos, se desea usar una herramienta especializada para este tipo de caso. Explorando los ejemplos que ofrece symphony llamando a la biblioteca, se encontró un optimizador que provee el estudio para PL y PLE con dos objetivos ofreciendo soluciones óptimas para el frente de pareto. Se pretende el uso de esta herramienta para resolver este modelo. Para implementar el algoritmo de generación de columnas se usaran las rutinas API de GLPK para construir una pequeña aplicación que a su vez es un heurístico para atacar los modelos propuestos.

En principio se desarrollaron estos modelos (3.3.1) y (3.3.2) en el lenguaje GMPL para ser resueltos por AMPL (Modeling Language for Mathematical Programming) edición para estudiante, ya que es una herramienta de optimización de caja negra que usa los optimizadores CPLEX y MINOS ejecutables bajo Windows y permiten combinar técnicas de PL mediante un lenguaje mas simple. La versión usada fue CPLEX 10.1, pero debido al limite permitido para resolver problemas con nos mas de 300 variables y 300 restricciones, se decide explorar los softwares libre de optimización. Los códigos de los formatos para resolver los modelos de $MMRC$,

$MMMC$, $MMRC_{GC}$, $MMMC_{GC}$ en AMPL se presentan en el Apéndice (A) como ejemplo para futuras investigaciónes.

Se propone el uso de la plataforma de tecnología libre GNU/LINUX Ubuntu 7.10, para utilizar las herramientas de optimización de programación lineal y programación lineal entera que ofrecen GLPK y SYMPHONY. Los datos utilizados en la presente investigación fueron generados aleatoriamente de acuerdo al rango mostrados en las tablas (Tabla 3–1) y (Tabla 3–2) de la sección (3.1), ya que se debe de resolver varios problemas con diferentes datos y tamaños.

3.6.1. Solución del Problema de Mochila Múltiple con Restricción de Color

El objetivo de esta sección fue resolver el modelo MMRC maximizando la carga de vehíclos críticos en las unidades de transportes y determinar hasta que tamaño del problema se puede lograr conocer una solución; también se realizaron varias corridas variando el limite de paradas k en la restricción (3.8).

El desarrollo del modelo en GMPL se muestra en Apéndice (B), además se ejecutó bajo la plataforma *GNU/Linux Ubuntu 7.10*, utilizando GLPK 4.25 como un optimizador de caja negra. Para la utilización de este optimizador se hicieron de las siguientes llamadas:

```
glpsol -m <archivo_modelo.mod>
-o <archivo_salida_solucion.sol>
--tmlim nnn
```

y

```
glpsol -m <archivo_modelo.mod>
-o <archivo_salida_solucion.sol>
--tmlim nnn --mostf
```

Donde $-m$ es la sección de lectura del modelo y opcional datos, $-o$ escribe la solución a un formato de archivo de texto,$--tmlim\ nnn$ indica el limite de tiempo de la solución en nnn segundos y $--mostf$ indica que debe ramificar (método de ramificación y acotamiento) sobre la variables mas fraccionaria.

Para resolver el problema de mochila múltiple con restricción de color usando la técnica de generación de columnas, se diseño una aplicación llamada main5 usando el API de GLPK. Los detalles de este código se encuentran en el Apéndice (C). Para llamar esta función se hace lo siguiente:

```
./main5
```

Esta función pedirá el archivo.in como datos de entradas, note que no se necesita el modelo en lenguaje GMPL ya que es una aplicación personalizada, si el archivo presenta errores o una mala información del problema no se ejecutara el programa. La estructura del archivo es la siguiente:

$$
\begin{array}{ll}
n & \\
m & \\
c & \\
l_1 & crit_1 \\
l_2 & crit_2 \\
\vdots & \vdots \\
l_n & crit_n \\
L_1 & \\
L_2 & \\
\vdots & \\
L_m &
\end{array}
$$

donde n es el número de vehículos, m el número de nodrizas, c el número de concesionarios, $crit_i$ y l_i la criticidad y longitud del vehículo i y L_j la longitud de la nodriza j.

3.6.2. Solución del Problema de Mochila Múltiple Multiobjetivos con Colores MMMC

El problema de mochila múltiple con colores es un modelo de PLE bicriterio y se representa como un modelo de un solo objetivo mediante la técnica de coeficiente de peso (ponderaciones).

El desarrollo del modelo en GMPL se muestra en el Apéndice (B), además también se utilizó el optimizador glpsol de GLPK como caja negra, donde se realizaron varias corridas con coeficientes de pesos distintos para ver la sensibilidad de los mismos.

Para resolver el MMMC mediante generación de columnas se dispone de otra aplicación llamada main6 usando el API de GLPK. Esta tiene el mismo formato que main5, a diferencia que en los subproblemas para la generación de columnas la restricción de concesionario para una nodriza no esta y los coeficientes de las variables x_{ij} y y_{cj} fueron multiplicadas por pesos α, $\beta = (1\text{-}\alpha)$ respectivamente en la función objetivo de los subproblemas. El valor dual asociado a la fila que representa la nodriza j también fue multiplicada por su correspondiente peso α.

3.7. Frente de pareto para el Problema de Mochila Múltiple con Colores

Maximizar la carga de vehículo críticos y minimizar el número de paradas para cada unidad de transporte corresponden a los dos objetivos en el análisis del problema de mochila múltiple con colores. Para resolver el modelo sin generación de

columnas se utilizo la apliación bicritieria de SYMPHONY 5.1.8 que resuelve problemas enteros mixtos con dos objetivos construyendo soluciones óptimas del frente de pareto y determinar como los valores óptimos de PLE varían en función de $P(\mu)$.

Antes de realizar la llamada se debe dar la instrucción en el código fuente bicriteria.c de cuales son las variables y coeficientes que formaran parte de la segunda función objetivo. Para utilizar este optimizador simplemente se realizara la siguiente llamada:

```
./bicritieria -F model.mod -D datos.dat
```

Donde -F hace la lectura del modelo (modelo.mod) y -D hace la lectura de los datos (datos.dat).

Dado que durante una prueba realiza a un modelo no muy grande no se logro obtener soluciones en menos de 1000 segundos ya que al tratar de resolver varios problemas variando los pesos (ponderaciones) producen costos en tiempo de ejecución; se decide por correr la aplicación con varios modelo pequeños. Estos resultados no serán utilizados para hacer comparaciones de tiempo, sino que se utilizaran como ayuda para elección del mejor modelo.

Capítulo 4
RESULTADOS DE LA INVESTIGACIÓN

Para representar la descomposición del modelo general se necesita dividir la cantidad de concesionarios, autos y nodrizas entre el número de regiones en que fue dividido el pais. Estimando el promedio de vehículos en patio, nodrizas disponibles y de concesionario para una empresa cualquiera como de 600, 30 y 50 respectivamente, se desea determinar hasta que tamaño del problema se puede resolver usando los algoritmos planteados en la sección (3.5) en un limite de tiempo menor o igual a 5 minutos.

Cuadro 4–1: Información general de los modelos

Cantidad	Conjuntos			MMRC	MMMC	
de Regiones	n/r	m/r	c/r	Filas		Columnas
1	600	30	50	18660	18630	19500
2	300	15	25	4830	4815	4875
3	208	10	16	2308	2298	2240
4	156	8	12	1420	1412	1344
5	120	6	10	852	846	780
6	104	5	8	634	629	560

Fuente: Elaboración propia (2008)

La Tabla 4–1 muestra la cantidad de regiones en que puede ser dividido el pais, donde una región denota que no se hizo ninguna descomposición, construyendo el problema de mochila múltiple con colores con un tamaño de 600 autos (artículos), 30 nodrizas (mochilas) y 50 concesionarios (colores). Una información adicional referente a las filas y columnas de los modelos formados por las regiones (1)-(6) se

muestra en las siguientes 3 columnas de la tabla, estas refieren al número de restricciones (filas) y variables (columnas) del problema. Los valores de los autos y concesionarios fueron seleccionados de tal manera que el número de autos sea divisible por el número de concesionarios, esto es para repartir equivalentemente el coloreado del vehículo.

4.1. Resultados de la Aplicación del Modelo de Mochila Múltiple con Restricción de Color

Los subproblemas de mochila múltiple con restricción de color se resolvieron mediante tres algoritmos: BB, BBF y GC; además se consideraron dos diferentes limites de paradas: k=2 y k=4. La siguiente tabla (ver Tabla 4–2) muestra bajo un limite de tiempo de 300 segundos la calidad de soluciones expresada en el valor del Gap, obtenidas en los diferentes algoritmos y modelos.

Cuadro 4–2: Detalles computacionales Tiempo/GAP de los modelos MMRC

Tiempo limite de 300 seg						
n°	k = 2			k = 4		
Reg.	BB	BBF	GC	BB	BBF	GC
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	300/8.0 %	264/1.4 %	-	300/6.9 %	-
4	300/7.6 %	300/6.3 %	232/1.1 %	-	300/5.6 %	296/0.9 %
5	300/14.4 %	-	72/1.0 %	-	300/12.5 %	186/0.8 %
6	300/14.6 %	300/5.7 %	21/2.0 %	300/9.3 %	300/5.4 %	97/1.6 %

Fuente: Elaboración propia (2008)

A pesar de que los modelos muestran ser relativamente sencillos no se logró obtener un valor óptimo en ninguno de los casos, esto es un Gap = 0 %, además el método tradicional de ramificación y acotamiento no resulto ser muy eficiente aun con modelos mas pequeños, no ofreciendo para ninguno de los casos soluciones

de muy buena calidad. Usando la técnica de ramificación sobre la variables mas fraccionaria se pudo reducir el gap significativamente pero no fue superior al resultado ofrecido por el algoritmo de generación de columnas y cortes siendo este un heurístico. La combinación de estas dos técnicas empleadas en GC fueron capaces de obtener una buena solución en menos tiempo del exigido.

4.2. Resultados de la Aplicación del Modelo de Mochila Múltiple Multiobjetivo con Colores

Al quitar la restricción del limite de paradas para utilizar una sola función objetivo mediante la técnica de sumas ponderadas con objeto de maximizar la carga de vehículos críticos y minimizar el número de paradas para cada unidad de transporte, se lograron resolver en el mismo limite de tiempo los modelos con 4 y 5 regiones utilizando los algoritmos BB y BBF en los modelos donde no se obtuvieron soluciones con la restricción de color.

Cuadro 4–3: Detalles computacionales Tiempo/GAP de los modelos MMMC

Tiempo limite de 300 seg						
n°	$\alpha = 0.8$			$\alpha = 0.3$		
Reg.	BB	BBF	GC	BB	BBF	GC
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	300/13.1 %	136/4.1 %	-	300/7.6 %	-
4	300/14.7 %	300/11.0 %	56/0.3 %	300/6.6 %	300/5.9 %	193/0.5 %
5	300/33.2 %	300/12.8 %	25/1.5 %	300/20.4 %	300/10.2 %	9.8/0.8 %
6	300/32.6 %	300/22.0 %	11/1.6 %	300/12.7 %	300/5.3 %	27/0.8 %

Fuente: Elaboración propia (2008)

La Tabla 4–3 presenta información de los pesos α asociados a la segunda función objetivo descrita en la sección 3.3.2 del capítulo 3, y demuestra que los algoritmos lograron alcanzar en 300 segundos mas soluciones factibles pero hubo en la mayoría

de los casos un desmejoramiento en la calidad de la solución en los algoritmos BB y BBF. Sin embargo el heurístico GC fue mas eficiente al proporcionar casi siempre soluciones de buena calidad en mejor tiempo de ejecución y resultaron ser mejores que los alcanzados con la restricción de color. Estos algoritmos no lograron alcanzar el valor óptimo, pero a través de los modelos lograron alcanzar mas soluciones factibles y para el caso de GC mejorar los tiempo de ejecución. Los pesos fueron tomados de tal manera que se le da a un objetivo mas importancia que al otro y viceversa, para comparar la asiganción de cargas entres los parámetros de MMMC y medir las paradas realizadas por cada nodriza en todos los modelos.

4.3. Estructura de los Resultados de Asignación

El propósito de formular varios modelos con diferentes parámetros fue el de mostrar los resultados para analizar la asignación de cargas a las unidades de transporte, también se tiene interés en estudiar el comportamiento de la función objetivo variando los pesos, para conocer que tanto influyen en la solución minimizar el número de paradas.

Para los modelos de mochila con restricción de color no se tuvo ningún problema en determinar el número de paradas de cada unidad de transporte ya que estaban sujetas a un máximo de 2 o 4 paradas. La Tabla 4–4 muestra el máximo y mínimo de paradas alcanzado por cada unidad de transporte con sus respectivos promedios para los modelos de peso α y de número de regiones n.

Los valores representados describe que para $\alpha = 0.8$ el promedio de paradas es para dos concesionarios, pero algunas nodrizas pueden realizar de 3 y hasta 4 paradas; para $\alpha = 0.3$ se tiene un promedio de 3 paradas con un máximo de 4. Estos valores comparados con la restricción de color resultan ser mejores ya que en el caso de k=2 sólo se puede realizar dos paradas por nodriza limitando posibles cargas de

Cuadro 4–4: Estadística paradas/nodrizas para el MMMC

n°	$\alpha = 0.8$		$\alpha = 0.3$	
Reg.	max/min	med	max/min	med
1	-	-	-	-
2	-	-	-	-
3	4/1	1.9	4/1	2.5
4	3/1	1.8	4/2	2.7
5	3/1	2.1	4/2	3
6	3/1	2.2	4/1	2

Fuente: Elaboración propia (2008)

vehículos críticos, mientras que si el peso es de α=0.8 se hacen la mayoría de la veces dos paradas y algunas de 4 y 3 pero consigue maximizar la carga de vehículos críticos y minimizar el número de paradas de cada unidad de transporte. El caso de $\alpha = 0.3$ tiene mucho parecido con k=4 e iguales promedios de paradas.

4.3.1. Observaciones para el Frente de Pareto

Los resultados obtenidos al utilizar el ejecutable bicriteria de SYMPHONY se muestran en la Tabla 4–5, construyendo por medio de las soluciones óptimas el frente de pareto del problema multiobjetivo, donde el peso α es reservada para la función objetivo máx.

Se puede apreciar que dando pesos mas grandes sobre la función objetivo máx se puede lograr mejorar la función objetivo mín sin desmejorar el valor de la función objetivo máx, tal como el caso de $\alpha = 0.96$ donde se reduce el número de paradas a 24 sin desmejorar el valor 63 de la criticidad total, también se observa que maximizar la carga de los vehículos críticos depende de la cantidad de paradas total de todas las unidades de transportes.

Cuadro 4–5: Corridas del modelo MMMC usando bicriteria de SYMPHONY para formar el Frente de Pareto.

Conjunto	Pesos			Costo de la función	
n m c	Rango de α			Objetivo máx	Objetivo mín
10 2 5	1.000000	-	1.0000000	63	-31
	0.960000	-	1.0000000	63	-24
	0.880000	-	0.960000	62	-22
	0.724138	-	0.880000	60	-21
	0.607143	-	0.724138	55	-17
	0.555556	-	0.607143	52	-15
	0.461538	-	0.555556	51	-12
	0.354839	-	0.461538	49	-11
	0.275862	-	0.354839	43	-8
	0.233333	-	0.275862	42	-7
	0.146341	-	0.233333	40	-6
	0.100000	-	0.146341	28	-4
	0.073171	-	0.100000	27	-3
	0.032258	-	0.073171	25	-2
	0.015625	-	0.032258	3	-1
	0.000000	-	0.015625	0	-0
20 3 5	1.000000	-	1.000000	119	-52
	0.882353	-	1.000000	119	-30
	0.852941	-	0.882353	115	-29
	0.781250	-	0.852941	114	-25
	0.606061	-	0.781250	112	-20
	0.441860	-	0.606061	106	-19
	0.377778	-	0.441860	95	-17
	0.319149	-	0.377778	91	-15
	0.240000	-	0.319149	87	-12
	0.200000	-	0.240000	81	-11
	0.166667	-	0.200000	75	-10
	0.140625	-	0.166667	69	-9
	0.119403	-	0.140625	64	-8
	0.084507	-	0.119403	60	-6
	0.060976	-	0.084507	54	-5
	0.046512	-	0.060976	42	-4
	0.032967	-	0.046512	37	-3
	0.019608	-	0.032967	31	-2
	0.008333	-	0.019608	19	-1
	0.000000	-	0.008333	0	-0

Fuente: Elaboración propia (2008)

Capítulo 5
CONCLUSIONES Y TRABAJOS FUTUROS

Conclusiones

En el presente trabajo de investigación se ha logrado completar todas las etapas de la metodología propuesta, tales como el estudio de los datos, el desarrollo de los modelos MMRC y MMMC, la comprensión e implementación de los software libre de optimización combinatoria (GLPK, SYMPHONY), donde se construyeron los modelos para resolverlos via generación de columnas y cortes usando la biblioteca API de GLPK, y lográndose entender como es la estructura de symphony para el desarrollo de las aplicaciones.

La formulación del problema de asignación de los vehículos a las nodrizas fue presentada en la sección (3.2) del capítulo 3, donde se logra representar claramente que el problema de asignación de cargas a las unidades de transporte para una ruta es un problema de mochila múltiple con colores; relacionando los vehículos, nodrizas y concesionarios con los artículos, mochilas y colores respectivamente.

Para controlar el número de paradas de las unidades de transporte se desarrollaron los siguientes modelos: MMRC y MMMC. El modelo MMRC se utilizó para restringir el número de paradas en las nodrizas, mientras que el modelo MMMC minimiza el número de paradas mediante la implementación de una sola función objetivo con la suma ponderada de pesos.

Los software de optimización utilizados en esta investigación fueron descritos en la sección (2.9) del capitulo 2, ya que permiten el uso de lenguaje GMPL y son flexibles para crear y personalizar cualquier aplicación.

Para resolver los modelos de MMRC y MMMC se escogieron los siguientes algoritmos: Branch & Bound (BB) con la configuración de parámetros por defecto, Branch & Bound modificando el parámetro de estrategia de ramificación sobre la variables mas fraccionaria (BBF) y el heurístico propuesto GC que combina las técnicas de generación de columnas y cortes. Al correr los diferentes modelos mediante el uso de estos algoritmos se comprobó la eficiencia de cada uno de ellos en las Tablas (4–2)-(4–3) del capítulo 4. Se pudo apreciar que al modificar la estrategia de ramificación se obtuvo una mejor solución.

Con respecto a los algoritmos, ninguno resuelve el problema de mochila multiple con colores en un tiempo razonable cuando el número de regiones del pais es menor que 3, pero se logra demostrar que usar las técnicas de generación de columnas mejora los tiempos de ejecución. Además al considerar resolver los subproblemas individuales de mochila a la optimalidad usando la técnica de ramificación y corte, se redujo el tiempo de resolución de cada subproblema hasta un 90 %; el heurístico planteado también logra obtener mejores soluciones que los métodos tradicionales de BB y BBF.

Los modelos MMRC y MMMC fueron comparados enumerando las paradas de cada nodriza con diferentes parámetros (k = 2, k =4 y $\alpha = 0.8$, $\alpha = 0.3$) y se observó que el modelo MMMC ofrece mas facilidad de resolución, aunque desmejora un poco la calidad de las soluciones usando BB y BBF. Sin embargo fue mas eficiente el heurístico GC, ya que proporciona un Gap menor de 1 % en la mayoría de los casos. La ventaja de este modelo es que no limita la carga de vehículos críticos permitiendo mas paradas si esta mejora el valor de la función objetivo, ya que este valor depende del número de paradas total de todas las unidades de transportes, es decir, que

mientras mas sean las paradas, habrá un aumento en las sumas de las criticidades, donde el equilibrio deseado depende de los pesos de las funciones objetivos.

Las herramientas de software libres usadas en este trabajo de investigación fueron de gran importancia, demostrando ser confiables para resolver los modelos y se espera seguir utilizando para futuras investigaciones en relación a esta apliación.

Recomendaciones

Una vez terminado los objetivos de esta investigación se presentan las siguientes sugerencias y recomendaciones:

Para resolver estos modelos a través de las herramientas y bondades que ofrecen SYMPHONY y BCP, se propone el uso de la técnica de Ramificación y precio (Branch & Price) generando columnas y cortes para resolver la relajación del problema maestro y ramificar sobre las variables originales.

Para comprender el uso de estos software se recomienda impartir los conocimientos adquiridos en este Trabajo Especial de Grado como posible base para futuras aplicaciones en esta area de investigación.

APENDICES

Apéndice A
CODIGO AMPL

A.1. Modelo de Mochila Múltiple con Restricción de Color para Generación de Columnas

La implementación en AMPL del método de generación de columnas para problemas de programación lineal entera mixta (MILP) aplicado al modelo MMRC se describe a continuación mediante los siguientes archivos:

- mkcx.mod
- mkcx.dat
- mkcx.run

El fichero `mkcx.mod` define parámetros, variables, el problema maestro y el subproblema.

- mkcx.mod

```
1
2  ##########################################################
3  ## Patrones de produccion factibles del Problema     ##
4  ## de Mochila Multiple con Colores (MMC)             ##
5  ##########################################################
6
7  #-#  Variables Auxiliares #-#
8  # Se utiliza para sumar el tiempo de resolucion
```

```
9  # de todos los problemas
10 param tiempo >=0;
11 # Se utiliza para contar el numero de iteraciones
12 param cont integer >=0;
13 # Se utiliza para contar el numero total de autos cargados
14 param total integer >=0;
15 # Se utiliza para contar el numero nodrizas usadas
16 param total1 integer >=0;
17 # Se utiliza para ver la cota superior del problema
      original
18 param Cota_UB;
19 # Se utiliza para guardar la mejor cota superior
20 param Mejor_Cota_UB default 10000000;
21 #---------------------------#
22
23
24 param m >=0, integer; # Numero de mochilas
25 param n >=0, integer; # Numero de articulos
26 param c >=0, integer; # Numero de colores
27
28 # Conjunto de articulos
29 set Articulos:= 1..n;
30 # Conjunto de informacion de la criticidad
31 # y longitud de los articulos (vehiculos)
32 set Descripcion;
33 # Conjunto de mochilas
34 set Mochilas:= (n+1)..(m+n);
35 # Conjunto de colores
36 set Colores:= 1..c;
```

```
37
38 # Matriz de asigancion de los articulos a las mochilas
39 param Asig{Articulos,Mochilas};
40 # Longitud de las mochilas
41 param L{Mochilas};
42 # Guarda informacion de los articulos
43 param Vehiculos{Articulos,Descripcion};
44
45 param nPat integer >= 0;
46 set Patrones:= 1..nPat;
47
48 # Matriz de asignacion de la nueva formulacion para los
      articulos
49 param A{Articulos,Patrones} binary;
50 # Matriz de asignacion de la nueva formulacion para las
      mochilas
51 param M{Mochilas,Patrones} binary;
52
53 # Conjunto de patrones que coinciden con el articulo i
54 set POi{Articulos};
55 # Conjunto de patrones que coinciden con la mochila j
56 set PSj{Mochilas};
57
58 # Variable de la nueva formulacion la cual representa un
      patron
59 var u{Patrones} binary;
60 # Variable auxiliar con penalizacion
61 var Aux1;
62
```

```
63
64 #-- FUNCION OBJETIVO PROBLEM MASTER --#
65 maximize MMC:
66 sum{k in Patrones} (sum{i in Articulos: A[i,k]>0}
67      Vehiculos[i,'criticidad'])*u[k] - 1000*Aux1;
68
69 #-- Conjunto de restricciones --#
70 subject to RA {i in Articulos}: sum{k in POi[i]} u[k] +
      Aux1<=1;
71 subject to RM {j in Mochilas}: sum{k in PSj[j]} u[k] +
      Aux1<=1;
72
73 ##################################################
74 #               Mochila Subproblema              #
75 #------------------------------------------------#
76 #              GENERACION DE COLUMNAs            #
77 ##################################################
78
79 # Conjunto de colores en mochila j
80 set Cj{Mochilas};
81 # Conjunto de articulos en mochila j
82 set Nj{Mochilas};
83 # Conjunto de mochilas que carga el articulo i
84 set Mi{Articulos};
85
86 # Se define una matrix auxiliar Aux para establecer
87 # el tipo de color de cada articulo
88
89 param Aux{i in Articulos,j in Mochilas}:=
```

```
90      if Asig[i,j]>0 then 1 else 0; # Matrix auxiliar.
91  param tipocolor{i in Articulos}:=
92      floor (sum{j in Mochilas}Asig[i,j]/
93              sum{k in Mochilas: Aux[i,k]>0} Aux[i,k]);
94
95  # Conjunto de variables duales
96  set Dual:= Articulos union Mochilas;
97  # Valor de la varibale dual
98  param precio{Dual} default 0.0;
99  # Mochila que se cuenta
100 param Mochilaj integer >=0;
101 # Usada para condicion de parada
102 param repetido integer >=0;
103 # Usada para condicion de entrada de columna a la base
104 param Mejor_CR{Mochilas} default 0.0;
105
106 # 1 si el articulo i es asignado a la mochila j, 0 en otro
        caso
107 var x {Articulos,Mochilas} binary;
108 # 1 si se establece una parada en color c de la mochila j
109 var y {Colores,Mochilas} binary;
110
111
112 #-- FUNCION OBJETIVO DEL SUBPROBLEMA --#
113 minimize costo_reducido: sum{i in Nj[Mochilaj]}
114   (-Vehiculos[i,'criticidad']+precio[i])*x[i,Mochilaj]
115                                       + precio[Mochilaj];
116
117 #-- Conjunto de restricciones --#
```

```
118 subject to CM : sum{i in Nj[Mochilaj]}
119    Vehiculos[i,'longitud']*x[i,Mochilaj] <= L[Mochilaj];
120 subject to NC :
121    sum{s in Cj[Mochilaj]: s in Colores} y[s,Mochilaj] <=
          2;
122 subject to unico
123    {i in Nj[Mochilaj]}: x[i,Mochilaj] <= y[tipocolor[i],
          Mochilaj];
124
125 end;
```

- mkcx.dat

```
1
2  data;
3
4  param m:=2;
5  param n:=5;
6  param c:=3;
7
8  set Descripcion := longitud criticidad;
9
10 param Asig: 6 7 :=
11 1 1 1
12 2 1 1
13 3 2 2
14 4 2 2
15 5 3 3;
16
```

```
17
18 param L:=
19 6   48
20 7   48;
21
22
23 param Vehiculos: longitud criticidad:=
24 1   11   26
25 2   10   16
26 3   11   11
27 4   10   18
28 5   17   18
29 6   17   15;
30
31 end;
```

- mkcx.run

  El archivo necesario para leer los archivos mkcx.mod y mkcx.dat es el siguiente:

```
1
2  # ------------------------------------------------
3  # GILMORE-GOMORY para Mochila multiple con colores
4  # ------------------------------------------------
5
6  option solver cplex;
7  option solution_round 6;
8
9  # Lectura del Modelo y Datos
10
```

```
11 model MMCX.mod;
12 data MMCX.dat;
13
14 # Se definen los conjuntos Cj, Nj, Mi. Note que Mi no es
      necesario.
15
16 let {j in Mochilas} Cj[j]:= union {i in Articulos} {Asig[i
      ,j]};
17 let {j in Mochilas} Nj[j]:= {i in Articulos: Asig[i,j] in
      Colores};
18 let {i in Articulos} Mi[i]:={j in Mochilas: Asig[i,j] in
      Colores};
19
20 # Se define el Problema Master y el Subproblema que genera
       la columna
21
22 problem Formulacion_Opt: u, MMC, RA, RM;
23    option relax_integrality 1;
24    option presolve 0;
25
26 problem Patron_Gen: x, y, costo_reducido, CM, NC, unico;
27    option relax_integrality 0;
28    option presolve 1;
29
30 # Para poder comenzar resolviendo la relajacion
31 # del problema Master es necesario definir
32 # un conjunto de patrones factibles.
33 # Solo comenzaremos con 6 patrones no tan buenos.
34
```

```
35 let nPat := 6;
36
37 let {i in Articulos,k in Patrones} A[i,k] := 0;
38 let {i in Mochilas,k in Patrones} M[i,k] := 0;
39
40 # Si se quiere ver los patrones factibles, descomente
      display A y B.
41 # Observese que A[,k] forma parte del mismo patron que M[,
      k].
42
43 #display A;
44 #display M;
45
46 let repetido:=0;
47 let cont:=1;
48 let tiempo:= 0.0;
49
50 repeat {
51
52 # Se actualizan los conjuntos que utiliza la
53 # Formulacion_opt para cada iteracion.
54
55 let {i in Articulos} POi[i]:={k in Patrones: A[i,k]>0};
56 let {j in Mochilas} PSj[j]:={k in Patrones: M[j,k]>0};
57
58 # Se resuelve la relajacion LP del problema Master
59 # para obtener sus valores duales.
60
61 printf "\n ITERACION %d\n", cont;
```

```
62 printf "\n Relajacion del PM para obtener variables duales
      \n";
63 printf "\n";
64
65 solve Formulacion_Opt;
66 display _solve_time;
67 let tiempo:= tiempo + _solve_time;
68
69 let {i in Articulos} precio[i] := RA[i].dual;
70 let {j in Mochilas} precio[j] := RM[j].dual;
71
72 ### Guarda todos los costos reducidos de las m mochilas
73
74 let {i in Articulos, j in Mochilas}x[i,j] := 0;
75 let Mochilaj:=n;
76 for {k in Mochilas}{
77     let Mochilaj:= Mochilaj + 1;
78         print 'LPE sub-problema', k;
79         solve Patron_Gen;
80         display _solve_time;
81         let tiempo:= tiempo + _solve_time;
82         let Mejor_CR[k]:= costo_reducido;
83
84 ###  Busca el mejor costo reducido
85
86    if (Mejor_CR[Mochilaj] < -0.05) then {
87       let nPat := nPat + 1;
88           print 'Patron generado', nPat;
89           let {i in Articulos} A[i,nPat] := x[i,Mochilaj];
```

```
90              let M[Mochilaj,nPat] := 1;
91          let {i2 in Mochilas: i2 <> Mochilaj} M[i2,nPat] :=
                0;
92      }
93  };
94
95  printf "\n";
96  let Cota_UB:= - sum{i in Mochilas} Mejor_CR[i] + MMC ;
97  display Cota_UB;
98
99  if  Cota_UB < Mejor_Cota_UB then {
100     let Mejor_Cota_UB := Cota_UB;
101 }
102
103 display Mejor_Cota_UB;
104
105 let cont:= cont +1;
106
107 if (nPat > repetido) then {let repetido := nPat;} else
       break;
108
109 display {i in Mochilas} precio[i], {i in Mochilas} RM[i].
       slack;
110
111 }; /* Fin repeat */
112
113 option Formulacion_Opt.relax_integrality 0;
114 option Formulacion_Opt.presolve 1;
115
```

```
116 print 'Resolviendo el PLE del MMC_GC de la nueva';
117 print 'formulacion con integralidad';
118
119 solve Formulacion_Opt;
120 display _solve_time;
121 let tiempo:= tiempo + _solve_time;
122
123 display MMC;
124
125 # Para ver mas detalles descomente...
126
127 let total:=0;
128 for {k in Patrones: u[k]>0} {
129     let total := total + sum{i in Articulos}A[i,k];
130     #  display {i in Articulos: A[i,k]>0} A[i,k];
131     #  display sum{i in Articulos: A[i,k]>0} Vehiculos[i,'
          longitud'];
132     #  display {i in Mochilas: M[i,k]>0} M[i,k];
133     #  display {i in Mochilas: M[i,k]>0} L[i];
134 };
135 let total1:=0;
136 for {k in Patrones: u[k]>0} {
137    let total1 := total1 + sum{i in Mochilas}M[i,k];
138 };
139
140 print 'numero de autos asigandos: ',total;
141 print 'numero de nodrizas usadas: ',total1;
142
143 for {k in Patrones: u[k]>0} {
```

```
144      for {j in Mochilas: M[j,k]>0} {
145              printf "\n La nodriza %d tiene los siguientes
                    vehiculos\n", j;
146              for {i in Articulos: A[i,k]>0} {
147                  printf " %d ", i;
148              }
149          }
150  }
151
152  printf "\n\n";
153
154  printf "\n Tiempo total de ejecucion %f seg\n\n", tiempo;
```

Este ejemplo es para ser utilizado por AMPL.exe de amplcml a través del comando:

```
ampl: commands MMCX.run;
```

El modelo general sin generación de columna para el problema de MMRC esta dada por el siguiente ejemplo, y puede ser resuelto a través de los comandos:

```
ampl: option solver cplex;
ampl: model modelocorrida.mod;
ampl: solve;
```

- modelocorrida.mod

```
1  # Modelo de Mochila multiple con resctriccion de color
      para problema de asignacion
2  # de cargas a unidades de transporte con aplicacion a la
      distribucion de vehiculos nuevos en Venezuela.
```

```
3  # Asignacion de n autos (Articulos) a m nodrizas (Mochilas
       ).
4  # El color c hace refencia a un concesionario especifico.
5
6  # Parametros del Modelo
7
8  param m >=0, integer;  # Numero de Mochilas
9  param Lm1 >=0, integer;
10 param Lm2 >=0, integer;
11 param n >=0, integer;  # Numero de Articulos
12 param Cn1 >=0, integer;
13 param Cn2 >=0, integer;
14 param ln1 >=0, integer;
15 param ln2 >=0, integer;
16 param c >=0, integer;  # Numero de colores
17 param col:= n/c;
18
19 # Conjunto de Indices
20
21 set Articulos:= 1..n; # Conjunto de Articulos
22 set Mochilas:= 1..m;  # Conjunto de Mochilas
23 set Colores:= 1..col; # Conjunto de Colores
24
25 #  Parametros del Modelo
26
27 # Longitud de la unidad de transporte j.
28 param L{j in Mochilas}:= trunc(Uniform(Lm2,Lm1));
29 # Criticidad del vehiculo i.
```

```
30 param Criticidad{i in Articulos}:= trunc(Uniform(Cn2,Cn1))
      ;
31 # Longitud del vehiculo i.
32 param Longitud{i in Articulos}:= trunc(Uniform(ln2,ln1));
33  # Concesionario a donde debe de llegar el vehiculo i.
       Estos colores se distribuyen de n/col en n/col.
34 param tipocolor{i in Articulos}:= if i/c <= ceil(i/c) then
       ceil(i/c);
35
36 # Variables de Desicion y Restricciones No negativadad
37
38 # 1 si el auto (Articulo) i es asignado a la nodriza (
      Mochila) j. 0 en otro caso.
39 var x {i in Articulos,j in Mochilas} binary;
40 # 1 si los autos (Articulos) de concesionario c son
      enviado por la nodriza (Mochila) j. 0 en otro caso.
41 var y {s in Colores,j in Mochilas} binary;
42
43 # Funcion Objetivo
44 maximize MKCP:
45 sum{i in Articulos,j in Mochilas} Criticidad[i]*x[i,j];
46
47 # restricciones
48
49 subject to CM {j in Mochilas}:
50 sum{i in Articulos} Longitud[i]*x[i,j] <= L[j];
51
52 subject to unicidad {i in Articulos}:
53 sum{j in Mochilas}x[i,j] <= 1;
```

```
54
55  subject to NC {j in Mochilas}:
56  sum{s in Colores} y[s,j] <= 2;
57
58  subject to unico {i in Articulos, j in Mochilas}:
59  x[i,j] <= y[tipocolor[i],j];
60
61  data;
62
63  param m:=2;
64  param n:=75;
65  param c:=25;
66  param Lm1:= 300;
67  param Lm2:= 400;
68  param Cn1:= 1;
69  param Cn2:= 4;
70  param ln1:= 35;
71  param ln2:= 50;
72
73  end;
```

Apéndice B
ARCHIVOS DE LOS MODELO Y FORMATO DE ENTRADA PARA EL PROBLEMA DE MMC EN GLPK

B.1. Modelo de Mochila Múltiple con y sin Restricción de Color

```
1
2  # Modelo de Mochila multiple con resctriccion de color para
       problema de asignacion
3  # de cargas a unidades de transporte con aplicacion a la
       distribucion de vehiculos nuevos en Venezuela.
4  # Asignacion de n autos (Articulos) a m nodrizas (Mochilas).
5  # El color c hace refencia a un concesionario especifico.
6  # Se puede fijar o cambiar el limite del numero de colores en
       la restriccion NC.
7
8  # Parametros del Modelo
9
10 param ALFA;
11 # Numero de Mochilas
12 param m >=0, integer;
13 # Limite inferior del parametro longitud de mochila
```

```
#param Lm1 >=0, integer;
# Limite superior del parametro longitud de mochila
#param Lm2 >=0, integer;
# Numero de Articulos
param n >=0, integer;
# Limite inferior del parametro criticidad de articulo
param Cn1 >=0, integer;
# Limite superior del parametro criticidad de articulo
param Cn2 >=0, integer;
# Limite inferior del parametro longitud de articulo
#param ln1 >=0, integer;
# Limite superior del parametro longitud de articulo
#param ln2 >=0, integer;
# Distribucion de colores
param c >=0, integer;
# Numero de colores
param col:= n/c;

# Longitud del tipo de vehiculo
param Lv{1..5};
# Longitud del tipo de unidad de transporte
param Lut{1..10};

# Conjunto de Indices

set Articulos:= 1..n; # Conjunto de articulos
set Mochilas:= 1..m;  # Conjunto de mochilas
set Colores:= 1..col; # Conjunto de colores

```

```
43 #  Parametros del Modelo
44
45 # Longitud de la unidad de transporte j.
46 param L{j in Mochilas}:= Lut[ceil(Uniform(0,9))+1];
47 # Criticidad del vehiculo i.
48 param Criticidad{i in Articulos}:= trunc(Uniform(Cn1,Cn2));
49 # Longitud del vehiculo i.
50 param Longitud{i in Articulos}:= Lv[ceil(Uniform(0,4))+1];
51 param tipocolor{i in Articulos}:= if i/c <= ceil(i/c) then
      ceil(i/c);
52 # Concesionario a donde debe de llegar el vehiculo i.
53 # estos colores se distribuyen de n/col en n/col.
54 # Variables de Desicion y Restricciones No negativadad
55
56 var x {Articulos,Mochilas} binary;
57 # 1 si el auto (Articulo) i es asignado a la nodriza (Mochila
      ) j.
58 # 0 en otro caso.
59 var y {Colores,Mochilas} binary;
60 # 1 si hay una parada en concesionario (Color) c hecha por la
       nodriza (Mochila) j.
61 # 0 en otro caso.
62 var num >=0, integer;
63
64 # Funcion Objetivo
65 maximize MKCP:
66     sum{i in Articulos,j in Mochilas} Criticidad[i]*x[i,j];
67
68
```

```
69
70 #maximize MKCP:
71     (1-ALFA)*(sum{i in Articulos,j in Mochilas} Criticidad[i
       ]*x[i,j])
72     - ALFA*(sum{j in Mochilas,s in Colores} y[s,j]);
73 #Si va a trabajar con esta funcion objetivo (MMMC) comente la
       primera funcion y descomente esta linea
74
75
76 # restricciones
77 subject to total:
78         sum{i in Articulos, j in  Mochilas} x[i,j] = num;
79 subject to CM {j in Mochilas}:
80         sum{i in Articulos} Longitud[i]*x[i,j] <= L[j];
81 subject to unicidad {i in Articulos}:
82         sum{j in Mochilas}x[i,j] <= 1;
83 # Si va a trabajar con la segunda funcion objetivo (MMMC)
       comente esta linea de NC
84 subject to NC {j in Mochilas}:
85         sum{s in Colores} y[s,j] <= 2;
86 subject to unico {i in Articulos, j in Mochilas}:
87         x[i,j] <= y[tipocolor[i],j];
88
89 solve;
90
91 printf "%d\n",n;
92 printf "%d\n",m;
93 printf "%d\n",c;
94 printf{i in Articulos} "%d %d %d\n",Longitud[i],
```

```
95          Criticidad[i],tipocolor[i];
96  printf{j in Mochilas} "%5d\n",L[j];
97
98  data;
99
100 param n:=120;
101 param m:=6;
102 param c:=12;
103 #param Lm1:= 43;
104 #param Lm2:= 48;
105 param Cn1:= 1;
106 param Cn2:= 15;
107 #param ln1:= 6;
108 #param ln2:= 20;
109
110 # Si va a trabajar con la segunda funcion objetivo (MMMC)
111 param ALFA := 0.8;
112
113 param Lv:=
114 1 4813
115 2 3677
116 3 4362
117 4 4152
118 5 4228;
119
120 param Lut:=
121 1 23950
122 2 31500
123 3 31048
```

```
124  4 30970
125  5 28600
126  6 34080
127  7 14000
128  8 30776
129  9 29480
130  10 31080;
131
132
133  end;
```

Apéndice C
HEURÍSTICO PARA GENERACIÓN DE COLUMNAS

C.1. Esquema Heurístico

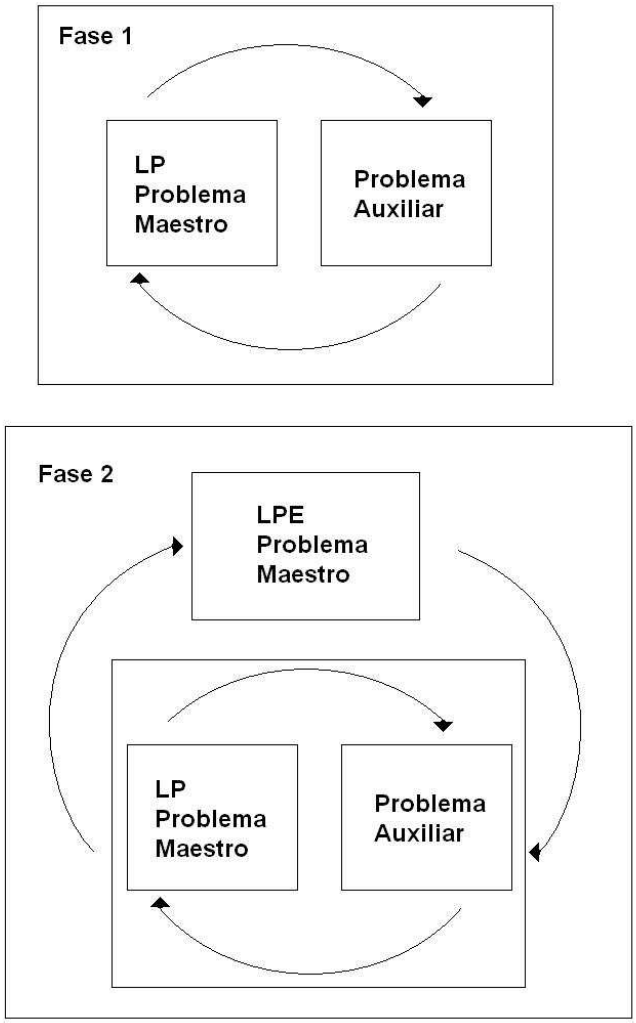

Figura C–1: Esquema heurístico para generación de columnas

C.2. Código en C

Este algoritmo solo cumple la primera fase del esquema heurístico para resolver el MMRC.

```
1
2  /* v.- 0.1. 16/09/08 17:15:42 */
3
4  #include <stdio.h>
5  #include <math.h>
6  #include "glpk.h"
7
8  /*------------------------------------------------------
9    Constantes.
10   ------------------------------------------------------*/
11
12 #define STR_LEN 64          // Longitud de las cadenas.
13 #define PENALTY -1000       // Penalizar var. artificiales.
14 #define MAX_CARROS 1500     // Cantidad maxima de vehiculos.
15 #define MAX_MOCHILAS  50    // Cantidad maxima de mochilas.
16 #define TAM_COL_X_MAS_UNO 3 // Se le suma uno por si acaso.
17
18 /*------------------------------------------------------
19   Tipos de datos.
20   ------------------------------------------------------*/
21
22 // Primero, por comodidad, definiremos una cadena como:
23
24 typedef char String[STR_LEN];
25
```

```
26 // Definimos una columna (Strip) de la siguiente manera...
27
28 typedef struct{
29   String nombreStrip;         // Nombre var. estructurales.
30   String nombreVarArtificial; // Nombre var. auxiliares
31   double dualFila;            // Valor dual.
32   unsigned int numFila;       // Numero de fila del modelo.
33 }Strip;
34
35 // Definimos un patron como sigue:
36
37 typedef struct{
38   String nombrePattern;       // Mi nombre.
39   unsigned int *stripCount;   // Valores de los subproblemas.
40   unsigned int numCol;        // Numero de columna que soy.
41   double solution;            // Costo reducido del patron.
42 }Pattern;
43
44 // Definimos un carro (articulo) como:
45
46 typedef struct{
47   unsigned int longitud;
48   unsigned int crit;
49   unsigned int concesionario;
50 }Carro;
51
52 // Y creamos un arreglo de carros para todos.
53 typedef Carro Carros[MAX_CARROS];
54
```

```
55 // Definimos la coleccion de mochilas como una coleccion de
      sus capacidades.
56 typedef unsigned int Mochilas[MAX_MOCHILAS];
57
58 /*--------------------------------------------------------
59   Prototipos.
60   --------------------------------------------------------*/
61
62 // Lectura de datos.
63 int lectura(unsigned int *mochilasRead, unsigned int *
      articulosRead, unsigned int *colorRead);
64 // Lectura inicial para obtener las variables artificiales.
65 void obtenerVariablesArtificiales(Strip *strips, unsigned int
       mochilasRead, unsigned int articulosRead);
66 // Generacion de patrones
67 Pattern *generarPatron(Strip *strips,
68                        int iter_no,
69                        int mochila, int numeroPatron,
70                        unsigned int mochilasRead,
71                        unsigned int colorRead,
72                        unsigned int articulosRead);
73
74 /*--------------------------------------------------------
75   Variables globales.
76   --------------------------------------------------------*/
77
78 Carros CARS;    // Arreglo de carros a ser leidos.
79 Mochilas MOCH;  // Arreglo de mochilas (capacidades) a ser
      leidas.
```

```
80
81  /*_____________________________________________________
82    Modulo principal.
83    _____________________________________________________*/
84
85  int DEBUG = 0;
86
87  int main(){
88
89    double *duals;    // Coleccion de variables duales.
90    glp_prob *lp;     // Coleccion de problemas.
91
92    int *ind;         // Buffer del indice de las filas.
93    double *val;      // Buffer de los valores de ese indice.
94
95    Pattern *new_pattern;       // Coleccion de patrones.
96    String model_filename;      // Nombre del modelo.
97    Strip *strips;              // Coleccion de tiras.
98    unsigned int ii;            // Iterador para...
99    unsigned int kk;            // Iterador para la k-esima
        mochila.
100   unsigned int row_count;     // Iterador para las filas.
101   unsigned int col_count;     // Iterador para las columnas.
102   unsigned int iteration;     // Contador para condicion de
        parada.
103   unsigned int nz;            // Contador de variables no
        nulas.
104   unsigned int jj;            // Index.
105   unsigned int patronesNULL;  // Contador de patrones nulos.
```

```
106
107     unsigned int mochilasRead;  //Num. de mochilas de entrada.
108     unsigned int articulosRead; //Num. de articulos de entrada
           ..
109     unsigned int colorRead;     //Num. de colores de entrada.
110
111     static int pattern_count;   // Cuenta los patrones
           generados
112     static int tmlim;           // limite de tiempo para PLE
113     int numeroPatron;           // Contador auxiliar
114     int use_cuts;
115     int lonvar;
116
117     // Se inicializan algunas variables
118     numeroPatron = 1;
119     pattern_count = 1;
120     tmlim = 600;
121     new_pattern = NULL;
122     ii = 0;
123     iteration = 1;
124     row_count = 0;
125     col_count = 0;
126     nz = 0;
127     use_cuts = 0;
128
129
130
131     if(lectura(&mochilasRead, &articulosRead, &colorRead)){
132
```

```
133     /* Inicializacion adaptativa de los arreglos utilizados
           ... */
134
135     ind = (int*) calloc(mochilasRead + articulosRead,
136           sizeof(int));
137     val = (double*) calloc(mochilasRead + articulosRead,
138           sizeof(double));
139     duals = (double*) calloc(mochilasRead + articulosRead,
140             sizeof(double));
141     strips = (Strip*) calloc(mochilasRead + articulosRead,
142              sizeof(Strip));
143
144     printf("\nDesea modo de depuracion: ");
145     scanf("%i", &DEBUG);
146     printf("\n");
147
148     if(DEBUG)
149     glp_term_out(0);
150
151     /* Si se desea vamos a mostrar los valores leidos. */
152     if(DEBUG){
153       printf("Capacidades Nodrizas:\n");
154       for(ii = 0; ii < mochilasRead; ii++)
155         printf("\tNodriza %i Cap = %i\n", ii + 1, MOCH[ii]);
156       printf("\n");
157       printf("Informacion de autos:\n");
158       for(ii = 0; ii < articulosRead; ii++)
```

```
159          printf("Auto %i: Longitud: %i Criticidad: %i
                Concesionario: %i\n", ii+1, CARS[ii].longitud,
                CARS[ii].crit, CARS[ii].concesionario);
160        printf("\n");
161      }
162
163      /* Funcion para objener las variables
            estructurales_auxiliares y artificiales  */
164      obtenerVariablesArtificiales(strips, mochilasRead,
            articulosRead);
165
166      /* Crear problema maestro. */
167      lp = glp_create_prob();
168      glp_set_prob_name(lp, "MMCGeneracionColumnas");
169
170      /* Maximizar numero total de patrones usados. */
171      glp_set_obj_dir(lp, GLP_MAX);
172
173      /* Agregar filas */
174      glp_add_rows(lp, articulosRead + mochilasRead);
175      for(ii = 0; ii < articulosRead + mochilasRead; ii++){
176        glp_set_row_bnds(lp, ii+1, GLP_UP, 0, 1.0);
177        glp_set_row_name(lp, ii+1, strips[ii].nombreStrip);
178        strips[ii].numFila = ii+1;
179      }
180
```

```
181     /*  Como no tenemos nigun patron o variable, debemos
           agregar variables artificiales en cada fila. Usaremos
           panalizacion para las variables auxiliares en funcion
           objetivo. */
182
183     glp_add_cols(lp, articulosRead + mochilasRead);
184     for(ii = 0; ii < articulosRead  + mochilasRead; ii++) {
185       glp_set_col_bnds(lp, ii+1, GLP_DB, 0, 1);
186       glp_set_col_name(lp, ii+1,
187                        strips[ii].nombreVarArtificial);
188       glp_set_obj_coef(lp, ii+1, PENALTY);
189
190       ind[1] = strips[ii].numFila;
191       val[1] = 1.0;
192
193       glp_set_mat_col(lp, ii+1, 1, ind, val);
194     }
195     /* Enfoque de generacion de columnas. */
196     int generacion = 1;
197
198     while(generacion) {
199       ind = (int*) calloc(mochilasRead + articulosRead,
200             sizeof(int));
201       val = (double*) calloc(mochilasRead + articulosRead,
202             sizeof(double));
203
204       patronesNULL = 0;
205       /* Resolver problema maestro y obtener valores duales
             ... */
```

```
206         sprintf(model_filename, "model.lp.%d", iteration);
207         if(DEBUG)
208           lpx_write_cpxlp(lp, model_filename);
209         glp_simplex(lp, NULL);
210            printf("Obj function value = %5.2f.\n",
                glp_get_obj_val(lp));
211
212         /* Escribir solucion */
213         col_count = glp_get_num_cols(lp);
214
215         /* Resetear valores duales. */
216         row_count = glp_get_num_rows(lp);
217         for(ii = 0; ii < row_count; ii++)
218           strips[ii].dualFila = glp_get_row_dual(lp, ii+1);
219
220          if(DEBUG) {
221           printf("Valores Duales de las restricciones de
                nodriza: \n");
222           for(ii = articulosRead; ii < row_count; ii++)
223             printf("%5.2f\n", strips[ii].dualFila);
224          }
225
226         /* Generar unas nuevas columnas. */
227         for(kk = 0; kk < mochilasRead; kk++){
228           new_pattern = generarPatron(strips,
229                                       iteration,
230                                       kk,
231                                       numeroPatron,
232                                       mochilasRead,
```

```
233                                     colorRead,
234                                     articulosRead);
235           if(new_pattern != NULL){
236             nz = 0;
237             for(ii=0;ii<articulosRead + mochilasRead; ii++) {
238               if(new_pattern -> stripCount[ii] != 0) {
239                 ind[nz + 1] = strips[ii].numFila;
240                 val[nz + 1] = 1.0;
241                 nz++;
242               }
243             }
244
245             if(nz == 0)
246               generacion = 0;
247
248             if(DEBUG)
249               printf("El numero de variables no nulas es %d.\n"
                     , nz);
250
251             /* Agregar nuevo patron al modelo. */
252             glp_add_cols(lp, 1);
253             col_count = glp_get_num_cols(lp);
254             glp_set_col_bnds(lp, col_count, GLP_DB, 0, 1.0);
255             glp_set_col_name(lp, col_count,
256                              new_pattern->nombrePattern);
257             glp_set_obj_coef(lp, col_count,
258                              new_pattern->solution);
259             glp_set_mat_col(lp, col_count, nz, ind, val);
260             pattern_count++;
```

```
261             numeroPatron = pattern_count;
262           }
263           else{
264             patronesNULL++;
265           }
266         }
267
268         iteration++;
269
270         if( patronesNULL == mochilasRead || iteration >= 100 ){
271         use_cuts |= LPX_C_ALL;
272         lpx_set_int_parm(lp,LPX_K_USECUTS,use_cuts);
273         // Si excede este limite de tiempo se detiene el
               algoritmo.
274         lpx_set_real_parm(lp, LPX_K_TMLIM, (double)tmlim);
275           for(ii = 0; ii < glp_get_num_cols(lp); ii++){
276           // Se indica que las variables son binarias
277             glp_set_col_kind(lp,ii+1,GLP_BV);
278           }
279           lpx_write_cpxlp(lp, "PM.lp");
280          // glp_simplex(lp,NULL);
281          // lpx_integer(lp);
282           lpx_intopt(lp);
283           printf("\n");
284           for(ii=0; ii < glp_get_num_cols(lp); ii++){
285             if(glp_mip_col_val(lp, ii+1) > 0 ){
286               lonvar = glp_get_mat_col(lp,ii+1,ind,NULL);
287               printf("Lista de autos asignados a la nodriza %i
                     :\n",ind[1]);
```

```
288               for(jj=2;jj<=lonvar;jj++){
289                 printf("%i ", ind[jj]);
290               }
291               printf("\n");
292             }
293
294           }
295
296           printf("Obj function value = %5.2f.\n",
                 glp_mip_obj_val(lp));
297           printf("\n");
298           generacion = 0;
299         }
300       }
301     }else{
302       printf("Error: archivo de entrada inexistente o
             inaccesible. El programa se detendra.\n");
303     }
304     return 0;
305   }
306   /*-------------------------------------------------------
307     Procedimiento 1: lectura
308       Lee los valores inciales...
309     Pre:
310     Post: Se obtienen los datos para comenzar la ejecucion.
311     -------------------------------------------------------*/
312
313   int lectura(unsigned int *mochilasRead,
314               unsigned int *articulosRead,
```

```
315             unsigned int *colorRead){
316
317     FILE *IN;          // Archivo de entrada.
318     unsigned int ii;   // Contador.
319     int A, B, C;       // Variables auxiliares para la lectura.
320     String fileName;   // Obvio.
321
322     printf("\nAplicacion a modelos de mochila multiple con
           colores\n");
323     printf("_____\n\n");
324     printf("Nombre archivo: ");
325     scanf("%s", &fileName);
326     IN = fopen(fileName, "r");
327     if( IN == NULL )
328       return 0;
329     else{
330       fscanf(IN, "%i", &A);
331       *articulosRead = A;
332       fscanf(IN, "%i", &B);
333       *mochilasRead = B;
334       fscanf(IN, "%i", &C);
335       *colorRead = C;
336       for(ii = 0; ii < *articulosRead; ii++)
337         fscanf(IN, "%i %i %i", &CARS[ii].longitud, &CARS[ii].
              crit, &CARS[ii].concesionario);
338       for(ii = 0; ii < *mochilasRead; ii++)
339         fscanf(IN, "%i", &MOCH[ii]);
340       fclose(IN);
341       return 1;
```

```
342     }
343   }
344
345   /*-------------------------------------------------------
346     Procedimiento 2: obtenerVariablesArtificiales
347       Lee los valores inciales para obtener las variables
348       artificiales.
349     Pre:
350     Post: Se obtiene...
351     -------------------------------------------------------*/
352
353   void obtenerVariablesArtificiales(Strip *strips,
354                                     unsigned int mochilasRead,
355                                     unsigned int articulosRead)
356                                     {
357
358     unsigned short int ii;
359
360     if(DEBUG)
361       printf("Soy la funcion de obtener variables\n");
362
363     for(ii = 0; ii < mochilasRead + articulosRead; ii++){
364       sprintf(strips[ii].nombreStrip, "c%d", ii+1);
365       sprintf(strips[ii].nombreVarArtificial, "Uaux%d", ii+1);
366     }
367   }
368
369   /*-------------------------------------------------------
370     Funcion 1: generarPatron
```

```
371     Genera...
372   Pre:  (1) colorRead debe ser par.
373         (2) articulosRead debe ser divisible
374             entre colorRead.
375   ----------------------------------------------------*/
376
377 Pattern *generarPatron(Strip *strips, int iter_no,
378                        int mochila, int numeroPatron,
379                        unsigned int mochilasRead,
380                        unsigned int colorRead,
381                        unsigned int articulosRead){
382
383   double ub;            // Upper Bound. Cota superior.
384   double mejorub;       // Mejor "Upper Bound" encontrada.
385   glp_prob *MIP;        // Coleccion de problemas.
386   Pattern *new_pattern;
387   String filename;
388   int repetido; // Control de salida en generacion de
        columnas.
389   int ii;       // Iterador.
390   int jj;       // Iterador.
391   int kk;       // Iterador.
392   int use_cuts;
393   double coef;  // Coeficiente de  los u_k.
394   int tmlim1;
395
396   // Vectores para la implementacion del modelo de
        optimizacion.
397   unsigned int IND[TAM_COL_X_MAS_UNO];
```

```
398     double VAL[TAM_COL_X_MAS_UNO];
399
400     ub = 0.0;
401     mejorub = 0.0;
402     repetido = 0;
403     ii = 0;
404     kk = 0;
405     coef = 0.0;
406     use_cuts = 0;
407     tmlim1 = 120;
408
409     new_pattern = NULL;
410
411     MIP = glp_create_prob();
412     glp_set_obj_dir(MIP, GLP_MAX);
413     glp_set_prob_name(MIP, "Subproblem");
414
415     /* Agregar fila. */
416     glp_add_rows(MIP, articulosRead + 2);
417
418     /* Preparar filas. */
419     glp_set_row_name(MIP, 1, "Capacidad");
420     glp_set_row_bnds(MIP, 1, GLP_UP, 0, MOCH[mochila]);
421     glp_set_row_name(MIP, 2, "color");
422     glp_set_row_bnds(MIP, 2, GLP_UP, 0, 2);
423
424     for(ii=0;ii<articulosRead;ii++){
425     glp_set_row_bnds(MIP, ii + 3, GLP_UP, 0, 0);
426     }
```

```
427
428     /*  Agregar columnas. Como son dos (02) mochilas
429         resto 1 para que quede
430           c1*x1+c2*x2+...cn*xn -cost*z
431         Para este ejemplo son
432           7 columnas autos = 4 + colores = 3.
433     */
434
435     glp_add_cols(MIP, articulosRead + colorRead);
436
437     for(ii = 0; ii < articulosRead; ii++){
438       glp_set_col_bnds(MIP, ii+1, GLP_DB, 0.0, 1.0);
439       glp_set_obj_coef(MIP, ii+1,
440                        CARS[ii].crit - strips[ii].dualFila);
441       glp_set_col_kind(MIP,ii+1,GLP_BV);
442     }
443     for(ii = 0; ii < colorRead; ii++){
444       glp_set_col_bnds(MIP, articulosRead + ii +1,
445                        GLP_DB, 0.0, 1.0);
446       glp_set_col_kind(MIP, articulosRead + ii +1,
447                        GLP_BV);
448     }
449
450     /* Implementacion del modelo de optimizacion. */
451
452     for(ii = 0; ii < articulosRead; ii++){
453       IND[1] = 1;
454       IND[2] = ii + 3;
455       VAL[1] = CARS[ii].longitud;
```

```
456       VAL[2] = 1.0;
457       glp_set_mat_col(MIP,ii+1,2,IND,VAL);
458     }
459
460     unsigned int *INDY;
461     double *VALY;
462     int cont;
463     int contaux;
464     int NumColor;
465
466     INDY = (unsigned int*) calloc(articulosRead,
467            sizeof(unsigned int));
468     VALY = (double*) calloc(articulosRead,
469            sizeof(double));
470
471     cont = 0;
472     contaux = 0;
473     NumColor = 0;
474
475     INDY[1] = 2;
476     VALY[1] = 1;
477
478     for(ii=articulosRead;ii<articulosRead + colorRead;ii++){
479           ++NumColor;
480       for(jj = 0; jj < articulosRead; jj++){
481           if(CARS[jj].concesionario == NumColor){
482             ++cont;
483             INDY[cont + 1] = jj+3;
484             VALY[cont + 1] = -1;
```

```
485 //         printf("%i %i\n",cont+1,INDY[cont +1]);
486           }
487       }
488       glp_set_mat_col(MIP,ii+1,cont+1,INDY,VALY);
489
490       cont=0;
491     }
492
493     use_cuts |= LPX_C_ALL;
494     lpx_set_int_parm(MIP,LPX_K_USECUTS,use_cuts);
495     lpx_set_real_parm(MIP, LPX_K_TMLIM, (double)tmlim1);
496
497     /* Resolver subproblema. */
498     sprintf(filename, "subproblem.lp.%d.%d", iter_no,mochila);
499     if(DEBUG)
500       lpx_write_cpxlp(MIP, filename);
501
502     lpx_intopt(MIP);
503    // glp_simplex(MIP,NULL);
504    // lpx_integer(MIP);
505     printf("Subproblem Obj function value = %5.2f.\n",
          glp_mip_obj_val(MIP));
506
507    // printf("New Pattern : \n");
508    // for(ii=0; ii < articulosRead + colorRead; ii++)
509    // printf("valor= %f\n", glp_mip_col_val(MIP, ii+1));
510
511     if( (glp_mip_obj_val(MIP)-strips[mochila + articulosRead].
          dualFila) > 1){
```

```
512       /* Nuevo patron */
513       new_pattern = (Pattern*) malloc(sizeof(Pattern));
514       new_pattern->stripCount = (unsigned int*) calloc(
             mochilasRead + articulosRead, sizeof(unsigned int));
515       sprintf(new_pattern->nombrePattern, "U%d", numeroPatron);
516       for(ii = 0; ii < articulosRead; ii++) {
517         if(glp_mip_col_val(MIP, ii+1) > 0 ){
518           new_pattern->stripCount[ii] = 1;
519           coef = coef + CARS[ii].crit;
520         }
521         else
522           new_pattern->stripCount[ii] = 0;
523       }
524       for(ii=articulosRead;ii<mochilasRead + articulosRead;ii
             ++){
525         if(ii == articulosRead + mochila)
526           new_pattern->stripCount[ii] = 1;
527         else
528           new_pattern->stripCount[ii] = 0;
529       }
530       new_pattern->solution = coef;
531
532     }
533     else
534      new_pattern = NULL;
535     lpx_delete_prob(MIP);
536     return new_pattern;
537   }
```

Bibliografía

[1] M. Cuadrado. *Modelos matematicos para la optimizacion combinatoria de la distribucion de vehiculos nuevos en Venezuela. (Caso: Clover International C.A.)*, 2007.

[2] Milind Dawande and Jayant Kalagnanam. *The multiple knapsack problem with color constraints*. Technical report, IBM T. J. Watson Research Center, Yorktown Heights, NY, Mar 1998. citeseer.ist.psu.edu/dawande98multiple.html.

[3] Jayant R. Kalagnanam, Milind W. Dawande, Mark Trumbo, and Ho Soo Lee. The surplus inventory matching problem in the process industry. *Oper. Res.*, 48(4):505–516, 2000. http://dx.doi.org/10.1287/opre.48.4.505.12425.

[4] J. J. Forrest, L. Ladanyi, and J. R. Kalagnanam. Column generation approach to the multiple problem with color contraints. Technical Report RC22013, IBM, 2001.

[5] Dr. Hernn Abeledo. Optimización combinatoria. http://www-2.dc.uba.ar/materias/ocom/.

[6] Centro de Estudios Ramón Areces Sixto Ríos Insua, Sixto Ríos. *Investigación operativa: Programación lineal y aplicaciones*. Ramn Areces.

[7] M. Gzelsoy T.K. Ralphs. *SYMPHONY 5.1.7 Users Manual*, 2007. https://projects.coin-or.org/SYMPHONY/.

[8] *GNU Linear Programming Kit*, 2007. http://www.gnu.org/software/glpk/.

[9] L. A. Wolsy Nemhauser, G. L. Integer and combinatorial optimization. *John Wiley and Sons, New York*, 1988.