ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE CIENCIAS

ASIGNACIÓN ÓPTIMA DEL PERSONAL PARA ATENCIÓN CIUDADANA USANDO PROGRAMACIÓN ENTERA MIXTA

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO MATEMÁTICO

PROYECTO DE INVESTIGACIÓN

JORGE EMILIO PÉREZ MONTALVO
emilio.perez@nivaleph.com

Director: DR. RAMIRO DANIEL TORRES GORDILLO
ramiro.torres@epn.edu.ec

QUITO, MARZO 2022

DECLARACIÓN

Yo JORGE EMILIO PÉREZ MONTALVO, declaro bajo juramento que el trabajo aquí escrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual, correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normatividad institucional vigente.

Jorge Emilio Pérez Montalvo

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por JORGE EMILIO PÉREZ MONTALVO, bajo mi supervisión.

Dr. Ramiro Daniel Torres Gordillo
Director del Proyecto

AGRADECIMIENTOS

A Ramiro Torres, mi tutor, por haberme guiado en este trabajo de titulación con sus conocimientos, sabiduría y paciencia. Por ser el precursor de que me involucre en este maravilloso mundo de la investigación de operaciones. A Luis Miguel Torres, por brindarme apertura en el proyecto de investigación PIGR-19-11 del cual es fruto este trabajo y por haber sido un excelente maestro que con sus enseñanzas permitió reafirmar mi amor a esta ciencia. También agradecer a Sandra Gutiérrez, María Fernanda Salazar, Estéfano Viteri y Fernando Jiménez investigadores del proyecto PIGR-19-11 que contribuyeron con el desarrollo de este trabajo.

A mi madre, por su inmenso e incondicional apoyo a lo largo de mi carrera universitaria y sobre todo a lo largo de mi vida. A mis hermanas por siempre estar a mi lado. A mi padre por sus enseñanzas y consejos. A mis abuelitos y tíos que siempre me motivaron a seguir adelante.

A todos mis amigos, con los que compartimos a diario aprendizajes, aventuras, vivencias e historias que siempre las llevaré conmigo.

A Nathalie, quien siempre me brindó su amor, apoyo incondicional y me ayudó a crecer como persona.

Y sobre todo a mí mismo, por confiar en mis capacidades y por nunca haberme rendido pese a las adversidades.

Gracias a todos.

DEDICATORIA

A mi madre, Cecilia, y mis hermanas, Camila y Victoria, que son las personas más especiales que tengo en mi vida y son fuente de inspiración, amor y apoyo incondicional. Este trabajo es para ustedes.

Índice general

Resumen x

Abstract xi

Notaciones xii

1. Introducción 1

2. Definiciones Preliminares 5

2.1. Programación Lineal Entera 5

2.2. Teoría de Grafos 8

2.2.1. Grafos Dirigidos 9

2.2.2. Redes de flujo 11

2.3. Calendarización de máquinas 13

3. Calendarización de máquinas en paralelo 16

3.1. Primera formulación 16

3.2. Segunda formulación 19

4. Desigualdades válidas y cotas inferiores 23

4.1. Cotas inferiores para el problema MPCM 23

4.2. Desigualdades válidas y cotas inferiores para CMC 29

4.3. Desigualdades válidas y cotas inferiores para CMD 32

5. Resultados Computacionales 35

5.1. Instancias 35

5.2. Experimentos computacionales usando CMC 37
5.2.1. Algoritmo Branch & Cut 39
5.3. Experimentos computacionales usando CMD 40
5.4. Comparación de los mejores escenarios de CMC y CMD 42
5.5. Soluciones de la mejor estrategia de CMD en instancias reales 43

6. Conclusiones y Recomendaciones **44**

Bibliografía **46**

Índice de figuras

2.1. Grafo simple. 8
2.2. Subgrafos obtenidos mediante la eliminación de una arista y un nodo. 9
2.3. Grafo dirigido. 10
2.4. Red de flujo. 12
2.5. Intervalos de atención en el horizonte de tiempo para el Ejemplo 1. . 14
2.6. Solución óptima para el Ejemplo 1. 15

3.1. Intervalos de atención en el horizonte de tiempo para el Ejemplo 2. . 21
3.2. Grafo de eventos discretizados del ejemplo 1. 22

Índice de cuadros

2.1. Información de tiempos e intervalos de atención. 14

5.1. Información de instancias muestreadas 36
5.2. Información de instancias modificadas 36
5.3. Información de instancias completas 37
5.4. Soluciones de CMC 38
5.5. Soluciones de CMC con la inclusión de desigualdades asociadas a los Teoremas 4.9 y 4.14 40
5.6. Soluciones de CMD 41
5.7. Comparación de los mejores escenarios de CMC y CMD 42
5.8. Soluciones de la mejor estrategia de CMD en instancias reales 43

Resumen

En el presente trabajo se estudia un problema multi-período de calendarización de máquinas paralelas que está motivado por el problema de asignación de personal a ventanillas de servicio al cliente en las distintas agencias del Sistema de Rentas Internas del Ecuador (SRI). Los funcionarios a cargo de cada una de las agencias deben decidir cuántas ventanillas (máquinas) de servicio deben estar activas en cualquier momento para atender el flujo entrante de clientes (trabajos) y minimizar el número total de horas-hombre necesarias para atender cada una de las ventanillas. Asumiendo como conocida la hora de llegada de los clientes, el tiempo de duración de cada uno de los trámites (procesamiento del trabajo) y un parámetro de calidad impuesto por el SRI definiendo que los usuarios no pueden esperar más de 20 minutos para ser atendidos, se proponen dos modelos de programación lineal entera junto con cotas inferiores y familias de desigualdades válidas para ambas formulaciones. Así, la primera formulación consiste en una versión continua en la que cada trabajo se procesa dentro de una ventana de tiempo definida por su hora de llegada y su última hora de inicio permitida y la segunda formulación consiste de una versión discretizada de las ventanas de tiempo en la que los conjuntos de intervalos de trabajo se dan explícitamente. Finalmente, se reportan extensos resultados computacionales usando instancias del mundo real.

Palabras clave: Programación entera, calendarización, máquinas paralelas, flujos sobre redes, instancias del mundo real.

Abstract

In the present work, a multi-period problem of scheduling on parallel machines is studied. The problem is motivated by personnel assignment to customer service counters in the different agencies of the Internal Revenue System of Ecuador (SRI). The officials in charge of each of the agencies must decide how many service counters (machines) must be active at any time to serve the incoming flow of customers (jobs) and minimize the total number of man-hours that are necessary to serve such counters. Each job is characterized by an arrival time, a latest processing start time, and a processing time. Two integer linear programming models together with lower bounds and several families of valid inequalities for both formulations are provided. Moreover, the SRI has imposed a quality parameter defining that users cannot wait more than 20 minutes to be served. Thus, the first formulation consists of a continuous version in which each job is processed within a time window defined by its arrival time and its last allowed start time, and the second formulation consists of a discretized version of the time window in which the sets of work intervals are given explicitly. Finally, extensive computational results using real-world instances are reported.

Keywords: Integer Programming, scheduling, parallel machine, network flows, real-world instances.

Notaciones

J	Conjunto de trabajos.
M	Conjunto de máquinas.
P	Conjunto de períodos de tiempo.
C	Conjunto de trabajos compatibles.
V	Conjunto de nodos.
A	Conjunto de arcos.
Q	Conjunto de eventos.
n	Número de trabajos($\|J\|$).
m	Número de máquinas ($\|M\|$).
$D = (V, A)$	Grafo dirigido.
(v_i, v_j)	Arco donde el nodo v_i es el predecesor del nodo v_j.
$\delta^+_{v_i} := \{(v_j, v_i) : (v_j, v_i) \in A\}$	Conjunto de arcos entrantes al nodo $v_i \in V$.
$\delta^-_{v_i} := \{(v_i, v_j) : (v_i, v_j) \in A\}$	Conjunto de arcos salientes del nodo $v_i \in V$.

Capítulo 1

Introducción

En la sociología, ciencias administrativas y administración pública es común encontrarse con el término «burocracia». En su análisis sociológico, Weber consideró a la burocracia como el modelo de organización eficiente por excelencia, donde se realza la precisión, velocidad, claridad, regularidad exactitud [19]. El Servicio de Rentas Internas del Ecuador (SRI) en su afán de estar a la vanguardia y aplicar la definición de Weber, ha adoptado el objetivo estratégico de fortalecer una de las áreas fundamentales de su institución: el servicio al cliente.

Diariamente a las oficinas del SRI ubicadas en distintos puntos del país, acuden miles de clientes a realizar trámites de declaración de impuestos u otras obligaciones fiscales que demandan una gran cantidad de recursos materiales y humanos para su atención. Esto ha generado una problemática de toma de decisiones para los altos funcionarios responsables de cada agencia, pues deben gestionar la apertura o no de ventanillas de servicio en cualquier momento para brindar atención a los clientes, garantizando altos estándares de calidad en atención y reducir el uso del recurso humano durante un día de trabajo.

En cada jornada laboral, las ventanillas de servicio en cualquier agencia pueden ser dinámicamente abiertas o cerradas (períodos) dependiendo del flujo de clientes. Para simplificar la planificación, los funcionarios de administración han decidido dividir la jornada laboral en períodos de una hora. Si la ventanilla se encuentra abierta en un período, el oficial atenderá a los clientes sin importar el tipo de trámite, caso contrario puede dedicarse a otras actividades de la agencia. La institución considera que un parámetro de calidad es que los usuarios no pueden esperar más de 20 minutos para ser atendidos. Así, cada agencia buscará atender a los clientes minimizando el número de períodos de trabajo de los operadores y respetando el

criterio de calidad de la atención de los clientes, es decir, las diferentes agencias se enfrentan constantemente a un problema de optimización en línea que permita determinar el número óptimo de ventanillas abiertas en cada período, manteniendo un límite máximo de tiempo de espera de cada cliente.

En este trabajo de titulación se considera una versión offline de este problema. Si se asume como conocida la hora de llegada de los clientes y el tiempo de duración de cada uno de los trámites, entonces la asignación de personal puede ser enfrentada desde el punto de vista determinístico. Así, esta toma de decisión se la puede tratar como una versión multi-período del problema de calendarización de máquinas paralelas sin interrupción de los trabajos [26], donde se dispone de un conjunto de clientes (trabajos) y un conjunto de ventanillas de servicio (máquinas) que deben atender (procesar) a los clientes dentro de un horizonte de tiempo dado y la jornada de trabajo se divide en períodos de tiempo de una hora. Cada trabajo se caracteriza por una hora de llegada, un plazo para el inicio del procesamiento y un tiempo de procesamiento. Todas las máquinas son idénticas y cada máquina puede estar activa (ventanilla abierta) o inactiva (ventanilla cerrada) en cada período. El objetivo es decidir qué máquinas deben estar activas en qué períodos y programar todos los trabajos a las máquinas activas disponibles, tal que el número de períodos-máquina (es decir, la suma del número de períodos activos en todas las máquinas) sea minimizado, mientras que el procesamiento de cada trabajo comienza dentro de su plazo. Una vez iniciado el procesamiento de un trabajo este no debe ser interrumpido.

En los últimos 50 años, la calendarización ha recibido gran atención por parte de la comunidad científica en el campo de la optimización combinatoria. La investigación ha sido motivada por diversas preguntas que han surgido en áreas como la planificación de la producción [5], simulación integrada e inteligencia artificial para la fotografía [2], informática e ingeniería industrial [13], defensa naval [16], entre otras. Todos estos trabajos se enfocan en un proceso de asignación de recursos a tareas durante períodos de tiempo determinados, tal que una o varias actividades sean optimizadas. Entre los objetivos de la optimización se puede mencionar la minimización del tiempo de finalización de la última tarea, la minimización del número de tareas completadas después de sus respectivas fechas de vencimiento o la minimización de los recursos necesarios para procesar las tareas [25].

En [7] los autores consideran el problema de calendarización sin interrupción de los trabajos considerando tiempos de inicio y tiempos límite de ejecución en un número mínimo de máquinas (idénticas) (SRDM). Si los tiempos son iguales para todas las tareas, los autores reportan que el problema puede ser resuelto usando un

algoritmo tipo Bin Packing, mientras que cuando se permite que los trabajos tengan holguras como máximo uno, el problema se puede resolver de manera eficiente a través de una formulación basada en flujo. La holgura de un trabajo se define como la diferencia entre su tiempo de inicio y el último tiempo posible en que se puede iniciar sin dejar de cumplir su plazo de inicio. En la literatura se han descrito varias variantes de SRDM. En [27], se propone un algoritmo de aproximación 2 y un algoritmo de aproximación 6 para los casos específicos en los que todos los trabajos tienen un tiempo de liberación común y cuando todos los trabajos requieren el mismo tiempo de procesamiento, respectivamente. Una variante del problema central de esta propuesta consiste en incluir capacidades de carga de trabajo sobre las máquinas [24]. Los autores abordan el problema considerando restricciones de tiempo de ejecución de manera continua y discreta. En la versión continua cada tarea debe ser ejecutada dentro de una ventana de tiempo y en la versión discreta se tendrá explícitamente conjuntos de intervalos de procesamiento de cada una de las tareas. En este trabajo se proponen varias heurísticas basadas en el problema de Bin Packing y un método exacto tipo Branch & Price. En [22] se propone un modelo entero mixto para el problema con una sola máquina con tiempos iniciales de procesamiento para los trabajos, el artículo es complementado con la presentación y discusión de cotas inferiores, varias clases de familias de desigualdades válidas y algoritmos de separación.

En [3], la programación de oficiales de servicio de atención al cliente en un centro de atención telefónica es reportado, los autores proponen un enfoque de solución que combina simulación con programación entera. En [1] se centran en el problema de atención al cliente desde el punto de vista de la economía y enriquece la literatura mostrando distintas perspectivas y áreas de aplicación del problema estudiado. En un estudio reciente presentado en [17] se busca mejorar la atención por parte de los oficiales de ventanilla a los clientes de un banco. El problema se centra en la calendarización de turnos de ventanilla de oficiales en la sección de servicio al cliente, considerando como estocástico el tiempo de llegada del cliente y la duración del servicio, tal que la pérdida relacionada con la espera del cliente y el costo generado por los oficiales de servicio sea minimizado; para resolver este problema de manera eficiente, se utiliza un algoritmo adaptado de optimización de ondas de agua [28].

Con todos estos antecedentes se procede a recapitular la propuesta que se hace en este proyecto de titulación: abordar el problema como una versión multi-período del problema de calendarización de máquinas paralelas sin interrupción de los trabajos, de tal manera que el número de períodos activos totales sea minimizado, res-

petando el criterio de calidad de atención al cliente impuesto por el SRI. El problema es modelizado usando distintos enfoques (tiempo de atención continuo y tiempo de atención discreto) y se han comparado los resultados obtenidos. Además, se encontraron cotas inferiores y varias familias de desigualdades validas, junto con un algoritmo exacto tipo Branch & Cut para la solución del problema estudiado.

El presente trabajo de titulación se encuentra organizado de la siguiente forma: en el Capítulo 2 se presentan las definiciones básicas necesarias para abordar el problema. En el Capítulo 3 se proporcionan dos formulaciones de programación entera para el problema de calendarización multi-período sin interrupción de los trabajos con máquinas paralelas. La primera es una versión continua en la que se permite que el procesamiento de un trabajo comience dentro de una ventana de tiempo, mientras que la segunda corresponde a una formulación basada en flujos considerando una discretización de las ventanas de tiempo. Algunas cotas inferiores y desigualdades válidas para ambas formulaciones son expuestas en el Capítulo 4. En el Capítulo 5 se presentan ampliamente los resultados computacionales y finalmente en el Capítulo 6 se exponen las conclusiones obtenidas del trabajo realizado.

Capítulo 2

Definiciones Preliminares

2.1. Programación Lineal Entera

En esta sección se repasan los conceptos más importantes para el desarrollo de este trabajo de titulación. Las definiciones han sido tomadas de [14].

DEFINICIÓN 2.1. *Sean $A \in \mathbb{R}^{m\times n}$ una matriz, $b \in \mathbb{R}^m$ y $c \in \mathbb{R}^n$ dos vectores. Un problema de programación lineal (LP) consiste en encontrar $x \in \mathbb{R}^n$ tal que $Ax \leq b$ y $c^T x$ sea maximizado, o decidir que $\{x \in \mathbb{R}^n : Ax \leq b\}$ es vacío, o decidir que para todo $\alpha \in \mathbb{R}$ hay un $x \in \mathbb{R}^n$ con $Ax \leq b$ y $c^\top x > \alpha$.*

Aquí $c^\top x$ denota el producto escalar entre vectores y se lo denomina función objetivo y $Ax \leq b, x \geq 0$ representan el conjunto de restricciones. A menudo se escribe un programa lineal de las siguiente manera:

$$\begin{aligned} &\text{máx } c^T x \\ &\text{s.r.} \\ &Ax \leq b \\ &x \geq 0 \end{aligned}$$

Una solución factible de un LP máx$\{cx : Ax \leq b\}$ es un vector x con $Ax \leq b$. Una solución factible que alcanza el máximo se llama solución óptima.

La programación lineal se ocupa de maximizar o minimizar una función objetivo lineal de un número finito de variables sujetas a un número finito de desigualdades lineales. Entonces, el conjunto de soluciones factibles es la intersección de un núme-

ro finito de semiespacios. Tal conjunto se llama poliedro:

DEFINICIÓN 2.2. *Un poliedro en $\mathbb{R}^n$ es un conjunto $\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b\}$ para alguna matriz $A \in \mathbb{R}^{m \times n}$ y algun vector $b \in \mathbb{R}^m$. Si A y b son racionales, entonces P es un poliedro racional. Un poliedro acotado se llama politopo.*

DEFINICIÓN 2.3. *Dado un politopo $\mathcal{P}$ y dos vectores α y β. Si se satisface para todo $x \in \mathcal{P}$ la desigualdad de la forma $\alpha x \leq \beta$, entonces esta es una desigualdad válida para $\mathcal{P}$.*

Si se imponen restricciones de integrabilidad sobre las variables del LP, entonces se tendrán los problemas de programación lineal entera. Si se dispone de variables enteras y continuas, entonces aparecen los problemas de programación lineal mixtos. Finalmente, cuando las variables enteras están restringidas exclusivamente a dos posibles valores $\{0, 1\}$, entonces surge la programación binaria.

DEFINICIÓN 2.4. *Un programa lineal entero (IP) es un problema de programación lineal, sujeto a la restricción adicional de que las variables de decisión x tomen valores enteros:*

$$
\begin{aligned}
&\max c^T x \\
&\text{s.r.} \\
&Ax \leq b \\
&x \geq 0 \\
&x \in \mathbb{Z}^n
\end{aligned}
$$

En 1947 George Dantzig [21] introdujo la programación lineal y el algoritmo Simplex como un método que podría usarse para resolver problemas del mundo real. El éxito del algoritmo condujo a una amplia gama de especializaciones y generalizaciones que han dominado la investigación de operaciones durante más de medio siglo. Este algoritmo permite resolver LPs de forma eficiente en la práctica, pero se pueden construir ciertas instancias donde este algoritmo no trabaja adecuadamente. En 1979, L. Khachiyan presentó por primera vez que los problemas de programación lineal podrían ser resueltos en tiempo polinomial adoptando el método de la elipsoide usado para resolver problemas no lineales. Desafortunadamente la apli-

cabilidad computacional fue muy limitada e inaplicable en la práctica. Unos años después, en 1984, Karmarkar [12] presentó el Método de Punto Interior que resuelve problemas lineales en tiempo polinomial y además presentó una implementación computacional competitiva con el Simplex. El artículo de Karmarkar condujo a un notable incremento de trabajos asociados a la programación lineal y muchas otras áreas relacionadas como la programación convexa.

Para los IP, de manera general, no existe un algoritmo que los resuelva en tiempo polinomial puesto que la Programación Lineal Entera es un problema NP-Completo. Sin embargo, muchos trabajos han aportado en la generación de algoritmos de solución. Así, en [8] se estudió el Problema del Agente Viajero, donde los autores presentaron el Algoritmo de Planos Cortantes que consiste en cortar iterativamente ciertas caras del poliedro relajado sin perder las soluciones enteras. En 1958, Gomory [11] demuestra la gran importancia de los planos cortantes para resolver problemas generales de la programación entera, describiendo un procedimiento que identifica una secuencia correcta de planos cortantes que garantiza una solución entera en finitas iteraciones (aunque exponencial). Un par de años más tarde, en [10] se propone el algoritmo fraccionario de Gomory que consiste en resolver el problema sin considerar las restricciones de integrabilidad de las variables y se propone añadir desigualdades válidas que reducen el conjunto de soluciones del problema lineal continuo asociado, sin excluir ninguna solución entera.

Por otra parte se tiene el algoritmo Branch & Bound [15] que consiste en linealizar el modelo de Programación Entera, es decir, resolver éste como si fuese un modelo de Programación Lineal y luego generar cotas en caso que al menos una variable de decisión entera adopte un valor fraccionario. Finalmente, se tiene el método Branch & Cut [18] que consisten en una combinación del algoritmo de planos cortantes y el algoritmo Branch & Bound. El método resuelve el programa lineal sin restricciones enteras utilizando el algoritmo símplex. Cuando se obtiene una solución óptima, y esta solución tiene un valor no entero para una variable que se supone que es entera, se puede usar un algoritmo de planos cortantes para encontrar restricciones lineales adicionales que se satisfacen con todos los valores enteros factibles, pero que se violan con la regla.

2.2. Teoría de Grafos

La segunda formulación de este trabajo de titulación está basado en flujos, por lo que es fundamental conocer ciertos conceptos básicos de Teoría de Grafos. Las definiciones para esta sección han sido tomadas de [14, 20].

DEFINICIÓN 2.5. *Un grafo no dirigido $G = (V, E)$ es un par ordenado compuesto por un conjunto finito de puntos $V = \{v_1, ..., v_2\}$ llamados nodos o vértices y un multiconjunto, de pares desordenados de V, $E = \{\{v_i, v_j\} : v_i, v_j \in V\}$ llamados aristas.*

El conjunto de nodos de un grafo G también es notado como $V(G)$ y el conjunto de aristas como $E(G)$. Además, se notará a $|V| = n$ como el número de nodos y $|E| = m$ como el número de aristas del grafo G.

Si dos aristas tienen los mismos extremos se dicen aristas paralelas y una arista que conecta un nodo consigo mismo se llama lazo. Un grafo simple es aquel que no tiene aristas paralelas ni lazos. En la Figura 2.1 se puede apreciar un grafo simple.

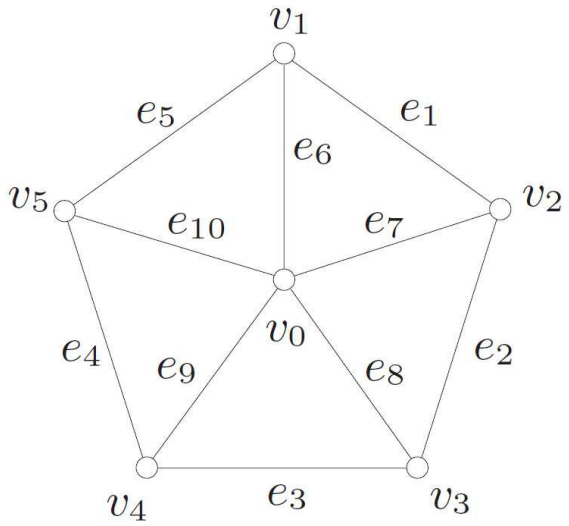

Figura 2.1: Grafo simple.

Un grafo simple en el que cada par de nodos se encuentran conectados por una arista se llama grafo completo.

DEFINICIÓN 2.6. *Sea $G = (V, E)$ un grafo no dirigido, $W \subseteq V$ y $F \subseteq E$. Si $H = (W, F)$ es un grafo, se dice que H es un subgrafo de G.*

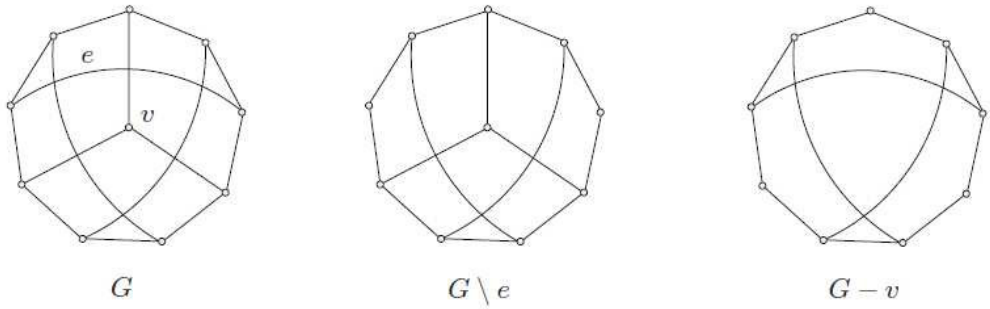

Figura 2.2: Subgrafos obtenidos mediante la eliminación de una arista y un nodo.

En la Figura 2.2 se puede apreciar un grafo y dos subgrafos obtenidos al eliminar una arista y un nodo respectivamente.

DEFINICIÓN 2.7. *Sea $G = (V, E)$ un grafo no dirigido. Se dice que los nodos $v_i, v_j \in V$ son adyacentes si $\{v_i, v_j\} \in E$. Los nodos adyacentes son conocidos como vecinos.*

DEFINICIÓN 2.8. *El conjunto de todos los nodos vecinos a un cierto nodo $v_i \in V$ se denomina vecindad, esto es:*

$$N(v_i) = \{v_j : \{v_i, v_j\} \in E\}$$

DEFINICIÓN 2.9. *Sea $G = (V, E)$ un grafo no dirigido. El conjunto de todas las aristas incidentes a cierto nodo $v_i \in V$ es representado por:*

$$\delta(v_i) = \{\{v_i, v_j\} : \{v_i, v_j\} \in E\}$$

La cardinalidad de $\delta(v_i)$ se conoce como el grado del nodo y corresponde al número de aristas incidentes al nodo v_i. Un nodo de grado 0 se denomina nodo aislado.

2.2.1. Grafos Dirigidos

Aunque muchos problemas se pueden abordar con teoría de grafos, el concepto de grafo que se introdujo anteriormente a veces no es del todo adecuado. Cuando se trata de problemas de flujo de tráfico, por ejemplo, es necesario saber qué caminos de la red son de un solo sentido y en qué dirección se permite el tráfico. En este caso un grafo no dirigido no es de mucha utilidad. Claramente lo que se necesita en esta situación es un grafo en el que a cada arista se le asigne una dirección, es decir, un grafo dirigido.

DEFINICIÓN 2.10. *Un grafo dirigido $D = (V, A)$ es un par ordenado compuesto por un conjunto finito de puntos $V = \{v_1, ...v_n\}$ llamados nodos o vértices y un multiconjunto de pares ordenados de nodos $A = \{(v_i, v_j) : v_i, v_j \in V\}$ llamados arcos.*

En un grafo dirigido se tiene que $(v_i, v_j) \neq (v_j, v_i)$. Por otra parte, dado un arco $a = (v_i, v_j)$, se dice que v_i es el predecesor de v_j o que v_j es el sucesor de v_i. Además, se llama a v_i como el nodo inicial del arco a y a v_j como el nodo final. En la Figura 2.3 se puede apreciar un grafo dirigido.

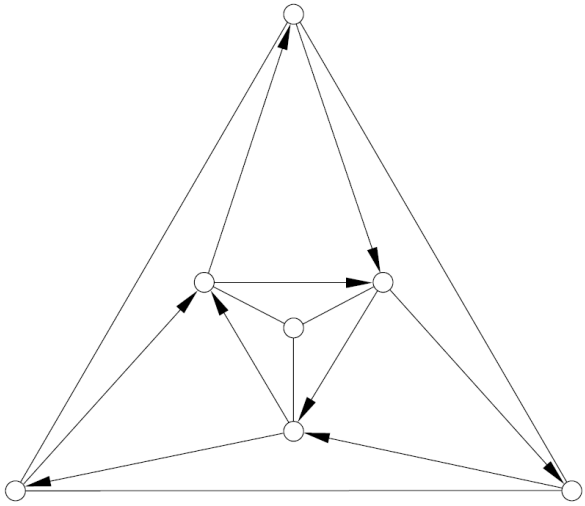

Figura 2.3: Grafo dirigido.

DEFINICIÓN 2.11. *Sean a_1 y a_2 dos arcos del grafo dirigido D. Se dice que a_1 y a_2 son arcos paralelos si tienen los mismos nodos iniciales y finales. Por otro lado, a_1 y a_2 son antiparalelos si el nodo inicial de a_1 es el final de a_2 y el nodo inicial de a_2 es el final de a_1.*

DEFINICIÓN 2.12. *Un arco se dice que es un lazo si su nodo inicial y final es el mismo.*

DEFINICIÓN 2.13. *Un grafo dirigido simple es aquel que no tiene arcos paralelos ni lazos.*

Para las nociones de vecindad y grado de los nodos es necesario precisar las direcciones en que actúan los arcos. Así, se tienen las siguientes definiciones:

DEFINICIÓN 2.14. *La vecindad predecesora de un nodo $v_i \in V$ se define como:*

$$N^+(v_i) = \{v_j : (v_j, v_i) \in A\}$$

DEFINICIÓN 2.15. *La vecindad sucesora de un nodo $v_i \in V$ se define como:*

$$N^-(v_i) = \{v_j : (v_i, v_j) \in A\}$$

DEFINICIÓN 2.16. *El conjunto de arcos entrantes para un nodo $v_i \in V$ se define como:*

$$\delta^+(v_i) = \{(v_j, v_i) : (v_j, v_i) \in A\}$$

DEFINICIÓN 2.17. *El conjunto de arcos salientes para un nodo $v_i \in V$ se define como:*

$$\delta^-(v_i) = \{(v_i, v_j) : (v_i, v_j) \in A\}$$

En cuanto a la cardinalidad de estos conjuntos de arcos se define a $|\delta^+(v_i)|$ como el grado entrante de $v_i \in V$ y a $|\delta^-(v_i)|$ como el grado saliente de $v_i \in V$.

Una observación importante es que todo grafo no dirigido puede ser transformado en un grafo dirigido dividiendo cada arista en arcos antiparalelos.

Finalmente, se procederá a definir qué es un camino.

DEFINICIÓN 2.18. *Un camino $P = (V, E)$ es un grafo no vacío de la forma $V = \{v_1, ..., v_k\}$, $E = \{\{v_1, v_2\}, ..., \{v_{k-1}, v_k\}\}$ donde los elementos de V son distintos.*

Los nodos v_1 y v_k son llamados nodos extremos del camino y los nodos $v_2, ..., v_{k-1}$ son llamados nodos internos. El número de aristas del camino es la longitud del mismo. A menudo es usual encontrar en la literatura que un camino es una secuencia de nodos $P = v_1, v_2, ..., v_k$ o que es una secuencia alternada de nodos y aristas $P = v_1, e_1, v_2, ..., e_{k-1}, v_k$ que va desde el nodo v_1 hasta el nodo v_k.

2.2.2. Redes de flujo

DEFINICIÓN 2.19. *Una red capacitada $R = (V, A, u)$ consiste en un grafo dirigido $D = (V, A)$ y una función no negativa de capacidades sobre los arcos $u : A \to \mathbb{R}_+$.*

Además, se trabajará con dos nodos específicos s y t, denominados nodo fuente y nodo sumidero, respectivamente. Adicionalmente, se supondrá que $\delta^+(s) = \emptyset$ y $\delta^-(t) = \emptyset$, es decir, no existen arcos entrantes a s o salientes de t.

DEFINICIÓN 2.20. *Un flujo de s a t $(st - flujo)$ sobre una red capacitada es una función $x : A \to \mathbb{R}_+$ que satisface las siguientes dos propiedades:*

- *1. Conservación de flujo*

$$\sum_{(i,j)\in\delta^-(i)} x_{ij} - \sum_{(j,i)\in\delta^+(i)} x_{ji} = 0,$$

para todo $i \in V\backslash\{s,t\}$.

- *2. Límite de capacidades*

$$0 \leq x_{ij} \leq u_{ij}$$

para todo $(i,j) \in A$.

Con la definición anterior se puede introducir una medida de discrepancia sobre el flujo que se mueve sobre el conjunto de arcos.

DEFINICIÓN 2.21. *Sea* $x : A \to \mathbb{R}_+$. *El flujo neto de* s *a* t *en la red capacitada* $R = (V, A, u)$ *está dado por:*

$$|x| := x(\delta^-(s)) - x(\delta^+(s)) = \sum_{(s,i)\in A} x_{si} - \sum_{(i,s)\in A} x_{is}.$$

Toda red tiene al menos un flujo factible, pues tomando $x_{ij} = 0$ para todo $(i,j) \in A$, se satisface la definición de conservación de flujo y capacidad.

En la figura 2.4 se presenta un ejemplo de un flujo. Sobre cada arco del digrafo se presenta la información relacionada al flujo y a la capacidad.

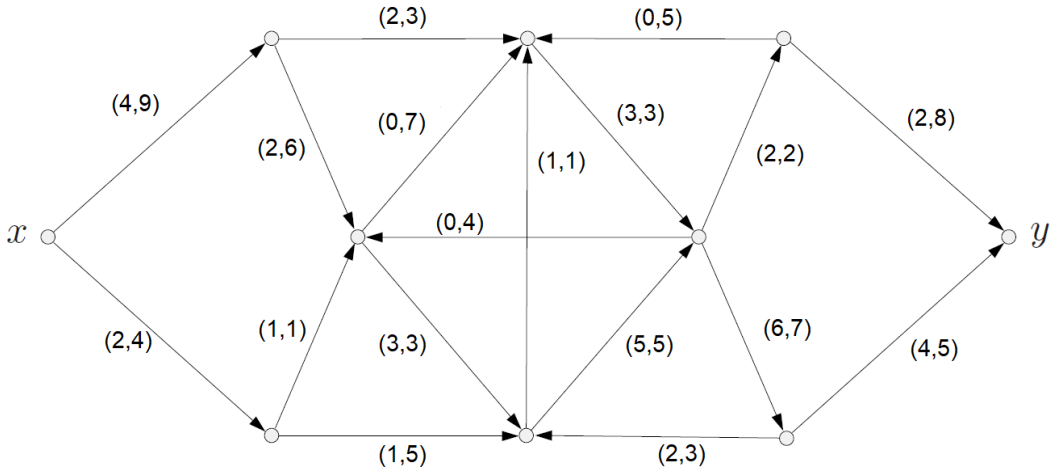

Figura 2.4: Red de flujo.

2.3. Calendarización de máquinas

En esta sección se presenta la notación sobre calendarización de máquinas que se usará a lo largo de este trabajo y se define formalmente el problema a estudiar. Los fundamentos y notación de este apartado se pueden encontrar en [6, 24, 9].

Sean $J = \{1, \ldots, n\}$ un conjunto de trabajos y $M = \{1, \ldots, m\}$ un conjunto de máquinas. Cada trabajo $i \in J$ tiene asociada una hora de llegada $a_i \in \mathbb{R}_+$ y un tiempo de procesamiento $t_i > 0$. El procesamiento del trabajo debe comenzar a lo sumo en $\gamma \in \mathbb{Z}_+$ unidades de tiempo una vez que el trabajo ha llegado. La última hora de inicio permitida de procesamiento de un trabajo i es denotado por $b_i := a_i + \gamma$. El horizonte temporal se divide en un conjunto finito P de períodos de tiempo de igual duración L. Una máquina puede estar activa o inactiva en cada período y puede procesar un trabajo solo mientras está activa. Además, una máquina activa puede procesar como máximo un trabajo al mismo tiempo y no hay límite en la cantidad de trabajos que puede procesar una máquina.

El problema multi-período de calendarización de máquinas paralelas (MPCM) consiste en decidir qué máquinas deben estar activas en cada período y programar todos los trabajos en las máquinas activas, de manera que el procesamiento de los trabajos se realice dentro de los plazos correspondientes. El objetivo del problema apunta a minimizar la cantidad total de períodos-máquina, es decir, la suma del número de máquinas activas durante todos los períodos. Para una mejor compresión del problema MPCM se presenta el siguiente ejemplo:

EJEMPLO 1. Consideremos una jornada que consta de 3 períodos, cada uno con una longitud de $L = 5$ unidades. Se deben programar cinco trabajos dentro de este horizonte en tres máquinas paralelas. La información del tiempo de llegada y el tiempo de procesamiento de cada uno de los trabajos se encuentra en las columnas 2 y 3 del Cuadro 2.1 respectivamente. El tiempo máximo de espera para el inicio del procesamiento es $\gamma = 1$ unidad.

Cuadro 2.1: Información de tiempos e intervalos de atención.

i	a_i	t_i	$[a_i, a_i + t_i]$	$[a_i + 1, a_i + 1 + t_i]$
1	0	2	[0, 2]	[1, 3]
2	1	2	[1, 3]	[2, 4]
3	6	3	[6, 9]	[7, 10]
4	6	5	[6, 11]	[7,12]
5	12	2	[12, 14]	[13, 15]

Por lo tanto, cada trabajo puede ser procesado en dos posibles intervalos de tiempo que están indicados en las dos últimas columnas del Cuadro 2.1. En la Figura 2.5 se pueden observar los intervalos de tiempo de atención de los cinco trabajos, en el horizonte de tiempo con los tres períodos.

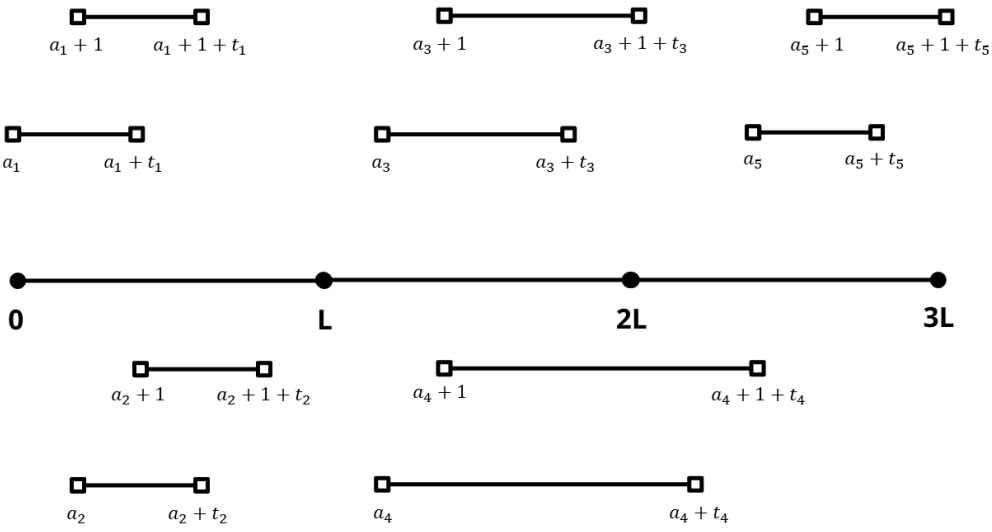

Figura 2.5: Intervalos de atención en el horizonte de tiempo para el Ejemplo 1.

En la Figura 2.6 se puede observar la solución óptima para este problema. Se han usado dos máquinas: la máquina 1 estará activa durante los tres períodos y procesará los trabajos 1, 2, 4 y 5, mientras que la máquina 2 estará activa durante un solo período y procesará el trabajo 3. La tercera máquina no es usada en la solución. Así, el valor óptimo para el ejemplo del problema MPCM es la suma de los períodos activos de las dos máquinas, es decir, el valor es 4.

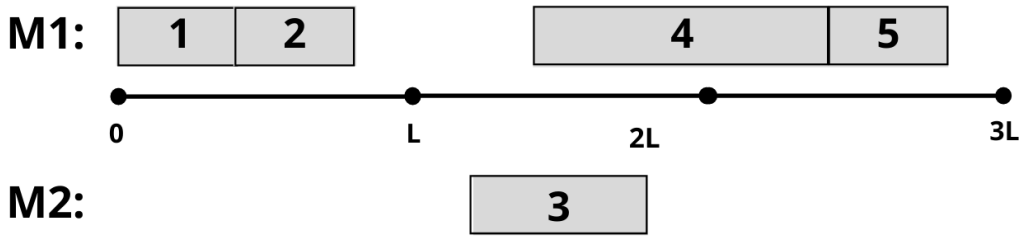

Figura 2.6: Solución óptima para el Ejemplo 1.

El problema propuesto está estrechamente relacionado con los problemas de calendarización de máquinas paralelas sin interrupción de los trabajos con restricciones de intervalo. Estos problemas se pueden clasificar en términos generales en dos grupos: (i) una versión discreta en la que los conjuntos de intervalos de trabajo se dan explícitamente y (ii) una versión continua en la que cada trabajo se procesa dentro de una ventana de tiempo definida por su hora de llegada y su última hora de inicio permitida [6].

Capítulo 3

Calendarización de máquinas en paralelo

Para este trabajo de titulación se asume como conocida la hora de llegada de los usuarios (trabajos) y el tiempo de duración de cada uno de los trámites (procesamiento del trabajo), entonces la asignación de personal (asignación de trabajos a máquinas) se la puede tratar como una versión multi-período del problema de calendarización de máquinas paralelas.

Como se mencionó en capítulos anteriores, este problema se abordará con dos modelos de programación enteros mixtos.

3.1. Primera formulación

En [9] se estudia el Problema del Agente Viajero (TSP) con ventanas de tiempo (TSPTW) que puede ser formulado de la siguiente forma: dado un conjunto de ciudades, una ventana de tiempo sobre cada ciudad y las distancias entre cada par de ciudades. El problema consiste en encontrar la ruta de distancia mínima visitando cada ciudad exactamente una sola vez. Además, el servicio en una ciudad debe comenzar dentro de la ventana de tiempo, es decir, no se permite que un vehículo llegue a una ciudad después de la última hora para comenzar el servicio. Sin embargo, si un vehículo llega demasiado pronto, se le permite esperar hasta que la ciudad esté lista para el inicio del servicio. Este tipo de problemas se pueden encontrar en una variedad de aplicaciones del sector industrial y de servicios. El modelo formulado para TSPTW, ha servido como punto de partida para formular el modelo continuo de calendarización de máquinas sin interrupción de los trabajos, donde

el procesamiento del trabajo $i \in J$ debe comenzar dentro de la ventana de tiempo $[a_i; b_i]$.

Para describir este primer modelo se define el grafo dirigido $D = (V, A)$. Se dispone de un conjunto de nodos $V := J \cup \{o\} \cup \{d\}$ que contiene todos los trabajos del conjunto J más un nodo origen y un nodo de depósito. Para definir el conjunto de arcos, se introduce la siguiente definición:

DEFINICIÓN 3.1. *Un par ordenado de trabajos $(i, j) \in J \times J$ se llaman compatibles si el trabajo j puede ser procesado después del trabajo i en la misma máquina, es decir, si se cumple que $a_i + t_i \leq b_j$.*

El conjunto $C \subset J \times J$ denotará al conjunto de trabajos compatibles y el conjunto de arcos del digrafo D está dado por $A := C \cup (\{o\} \times J) \cup (J \times \{d\})$.

La formulación de este modelo considera variables enteras, binarias y continuas. Se define x_{ij} como una variable de decisión binaria que toma el valor de uno si el arco $(i, j) \in A$ es usado en la solución y será cero en caso contrario. Esta variable intenta determinar el orden de procesamiento de los trabajos, ya que si $x_{ij} = 1$ implica que el trabajo j es procesado inmediatamente después del trabajo i. Para cada nodo $i \in J$ se define la variable continua $T_i \in \mathbb{R}_+$ que indica el inicio de procesamiento del trabajo i. Además, se definen las variables enteras $y_i^1 \in P$ que representa el período en el que el trabajo i inicia su procesamiento y $y_i^2 \in P$ denota el período en el que el trabajo i finaliza su procesamiento. Finalmente, para cada par de trabajos compatibles $(i, j) \in C$, se define la variable binaria z_{ij} que es igual a uno si las siguientes condiciones se satisfacen: (i) el trabajo j es procesado inmediatamente después del trabajo i en la misma máquina; (ii) el trabajo i finaliza en un período q y el trabajo j empieza en un período r tal que $r > q$. Caso contrario esta variable toma el valor de 0.

El modelo continuo de calendarización de máquinas $(\mathcal{CMC})$ viene dado por:

$$\text{mín} \sum_{i \in J} x_{oi} + \sum_{i \in J} (y_i^2 - y_i^1) + \sum_{(i,j) \in C} z_{ij} \tag{3.1}$$

$$\sum_{(i,j) \in A} x_{ij} = 1, \qquad \forall i \in J, \tag{3.2}$$

$$\sum_{(j,i) \in A} x_{ji} - \sum_{(i,j) \in A} x_{ij} = 0, \qquad \forall i \in J, \tag{3.3}$$

$$\sum_{i \in J} x_{oi} \leq |M|, \tag{3.4}$$

$$T_i + t_i - T_j \leq K(1 - x_{ij}), \qquad \forall (i, j) \in C, \tag{3.5}$$

$$Ly_i^2 \geq T_i + t_i - L, \qquad \forall i \in J, \tag{3.6}$$

$$Ly_i^1 \leq T_i, \qquad \forall i \in J, \tag{3.7}$$

$$|P|z_{ij} \geq (y_j^1 - y_i^2) - |P|(1 - x_{ij}), \qquad \forall (i,j) \in C, \tag{3.8}$$

$$a_i \leq T_i \leq b_i, \qquad \forall i \in J, \tag{3.9}$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A,$$

$$z_{ij} \in \{0,1\}, \qquad \forall (i,j) \in C,$$

$$T_i \in \mathbb{R}_+, y_i^1, y_i^2 \in \mathbb{Z}_+, \qquad \forall i \in J.$$

donde K es un número suficientemente grande.

La función objetivo minimiza el número total de períodos-máquina. Las restricciones (3.2) aseguran que cada trabajo sea procesado por exactamente una sola máquina y (3.3) son restricciones de conservación de flujo asegurando que la solución se forme de un conjunto de caminos dirigidos de o a d, de modo que cada nodo $i \in J$ esté contenido exactamente en un camino. Cada uno de estos caminos representa la secuencia de trabajos procesados en una máquina específica. La restricción (3.4) limita el número de dichos caminos para que sean menores o iguales al número de máquinas disponibles. Las restricciones (3.5) establecen los tiempos de inicio de procesamiento de acuerdo con las secuencias de trabajo asignadas a cada máquina. Las restricciones (3.6) y (3.7) determinan los períodos de inicio y finalización de cada trabajo. Las restricciones (3.8) cuentan la cantidad de veces que dos trabajos compatibles consecutivos comienzan y terminan en períodos diferentes, es decir, si $(y_j^1 - y_i^2) > 0$ y $x_{ij} = 1$, entonces $z_{ij} = 1$. Finalmente, las restricciones (3.9) son restricciones de ventana de tiempo para la hora de inicio del procesamiento de cada trabajo. Tanto el número de variables como el número de restricciones tienen un orden igual a $O(n^2)$.

El valor de la constante K puede ser calculada de forma adecuada dependiendo de la instancia de entrada. Se tiene el siguiente resultado:

PROPOSICIÓN 3.1. *La restricción* (3.5) *puede ser reescrita como:*

$$T_i + t_i - T_j \leq K_{ij}(1 - x_{ij}), \forall (i,j) \in C$$

donde $K_{ij} = \max\left\{b_i + t_i - a_j, 0\right\}, (i,j) \in C$.

Demostración. Puesto que los trabajos $i, j \in J$ son compatibles, entonces existen dos escenarios: (i) para una misma máquina el trabajo j sólo puede procesarse después del trabajo i, es decir, $(i,j) \in C$ y $(j,i) \notin C$; o, (ii) para una misma máquina cual-

quiera de los dos trabajos puede procesarse primero, es decir, $(i,j) \in C$ y $(j,i) \in C$. Para el primer escenario se tiene que $b_i + t_i \leq a_j$ y por consecuencia $K_{ij} = 0$. Con esto $T_i + t_i \leq T_j$, lo cual siempre es cierto pues para una misma máquina se debe procesar primero el trabajo i. En cuanto al segundo escenario existen dos posibles casos. Si el arco (i,j) es seleccionado en la solución, entonces $x_{ij} = 1$ y por tanto $T_i + t_i \leq T_j$, lo cual es cierto pues el trabajo j se procesará después del trabajo i. Por otro lado, si el arco (i,j) no es seleccionado en la solución, entonces $x_{ij} = 0$ y así $T_i + t_i - T_j \leq b_i + t_i - a_j$. De esto se tiene que $a_j - T_j \leq b_i - T_i$, ya que $T_j \geq a_j$ y $b_i \geq T_i$. □

3.2. Segunda formulación

En esta sección se presenta una versión discretizada del problema de calendarización de máquinas sin interrupción de los trabajos, en la que se discretizan los intervalos de tiempo asociados a los trabajos y se definen explícitamente conjuntos de intervalos de tiempo en los que pueden ser procesados los trabajos. Para esto se hará uso del término «evento» el cual representa el inicio o final del procesamiento de un trabajo, así como también el inicio o final de un período de tiempo. Para cada trabajo $i \in J$, el conjunto Q_i^1 contiene todos los puntos de tiempo (discretos) en los que puede iniciar o terminar el procesamiento del trabajo i, es decir, todos los tiempos posibles $a_i, a_i + 1, \ldots, b_i$ donde puede comenzar el procesamiento de los trabajos i, junto con todos los tiempos posibles $a_i + t_i, a_i + t_i + 1, \ldots, b_i + t_i$ en los que puede finalizar el procesamiento del trabajo. Además, se define $Q^2 := \{0, L, \ldots, |P|L\}$ como el conjunto de puntos en el tiempo en las que los períodos pueden comenzar o terminar. En el caso en el que existan eventos que se ejecuten en el mismo tiempo, entonces los valores a_i, t_i y b_i tendrán una pequeña perturbación sin afectar la solución óptima del problema, esto será, $a_i + \epsilon * i$, $t_i + \epsilon * i$ y $b_i + \epsilon * i$ con $\epsilon > 0$. Finalmente, se define el conjunto de eventos como $Q = \bigcup_{i \in J} Q_i^1 \cup Q^2$ con $|Q| = 2(\gamma + 1)|J| + |P| + 1$. Así, el problema de calendarización discretizado se puede formular como una instancia del problema de flujo de costo mínimo.

Sea $D = (V, A)$ un digrafo que asocia un nodo para cada evento y dos nodos adicionales que representan el origen y el depósito, es decir, $V = \{1, \ldots, |Q|\} \cup \{o, d\}$. Además, sea h una función que asigna a cada nodo $j \in \{1, \ldots, |Q|\}$ el tiempo $h(j) \in Q$ del evento representado por j. Los nodos asociados a Q han sido etiquetados de forma que $j < l$ si y solo si $h(j) < h(l)$.

El conjunto de arcos del digrafo D es la unión de 3 subconjuntos A_1, A_2, A_3, donde:

$$A_1 = \{(j, j+1) : 1 \leq j < |Q|\} \cup \{(o,1), (|Q|, d), (o,d)\}$$
$$A_2 = \{(j,l) \ : \ h(j), h(l) \in Q^2, h(l) = h(j) + L\}$$
$$A_3 = \cup_{i \in J} A_3^i, \text{ con } A_3^i = \{(j,l) \ : \ h(j), h(l) \in Q_i^1, \ h(l) = h(j) + t_i\}.$$

Además, se tiene que $|A_3| = (\gamma + 1)|J|$.

Por otra parte, se define la función de costo sobre los arcos $c : A \to \mathbb{R}_+$. Todos los arcos en A_1 de la forma $(j, j+1)$, donde $h(j) \in Q^2$ tienen costos iguales a uno, los arcos restantes en A_1 tienen costos iguales a cero. Todos los arcos en A_2 tienen costos iguales a cero, mientras que el costo de cada arco $(j,l) \in A_3$ se establece como $c_{jl} = \lfloor \frac{h_l}{L} \rfloor - \lfloor \frac{h_j}{L} \rfloor$. Además, la capacidad u_{jl} de un arco (j,l) se define como $|M|$, si $(j,l) \in A_1 \cup A_2$, y es igual a uno si $(j,l) \in A_3$. Finalmente la demanda de los nodos se establece en la siguiente función:

$$g_j = \begin{cases} -|M|, \text{ if } j = o, \\ |M|, \text{ if } j = d, \\ 0, \text{ si } j \in V \setminus \{o, d\}. \end{cases}$$

El modelo discreto de calendarización de máquinas ($\mathcal{CMD}$) viene dado por:

$$\text{mín} \sum_{(j,l) \in A} c_{jl} x_{jl} \tag{3.10}$$
$$\sum_{(j,l) \in A_3^i} x_{jl} = 1, \qquad \forall i \in J, \tag{3.11}$$
$$\sum_{(l,j) \in A} x_{lj} - \sum_{(j,l) \in A} x_{jl} = g_j, \qquad \forall j \in V, \tag{3.12}$$
$$x_{jl} \leq u_{jl}, \qquad \forall (j,l) \in A, \tag{3.13}$$
$$x_{jl} \in \mathbb{Z}_+, \qquad \forall (j,l) \in A.$$

La función objetivo busca la minimización del número total de períodos-máquina. Las restricciones (3.11) aseguran que cada trabajo se procese en solo uno de sus intervalos dados. Las restricciones (3.12) y (3.13) son de conservación de flujo y capacidades de los arcos, respectivamente.

Para facilitar la comprensión de cómo se construye el grafo anteriormente des-

crito, se propone el siguiente ejemplo:

EJEMPLO 2. Consideremos una jornada que consta de 2 períodos, cada uno con una longitud de $L = 7$ unidades de tiempo. Se deben programar dos trabajos dentro de este horizonte en tres máquinas paralelas: el primer trabajo llega en el tiempo $a_1 = 1$ y requiere un tiempo de procesamiento de $t_1 = 5$ unidades de tiempo, mientras que el tiempo de llegada del segundo trabajo es $a_2 = 4$ y su tiempo de procesamiento es de $t_2 = 6$ unidades. El tiempo máximo de espera para el inicio del procesamiento es $\gamma = 1$ unidad. Por tanto, el primer trabajo debe procesarse en uno de los intervalos $[1;6]$ o $[2;7]$, mientras que el procesamiento del segundo trabajo debe ocurrir en uno de los intervalos $[4;10]$ o $[5;11]$. En la Figura 3.1 se pueden observar los intervalos de tiempo de atención de los dos trabajos, en el horizonte de tiempo con los dos períodos.

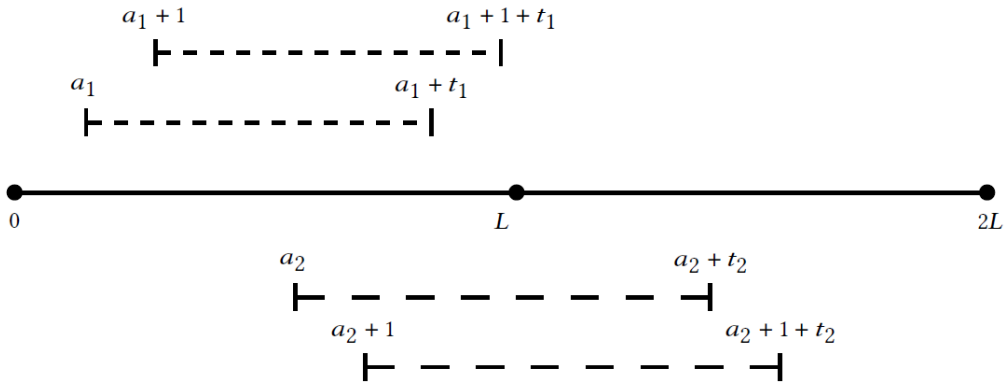

Figura 3.1: Intervalos de atención en el horizonte de tiempo para el Ejemplo 2.

Ahora, dado que ya se conocen los intervalos de atención de cada uno de los trabajos se exponen los conjuntos relacionados a cada uno de estos. Se tienen los conjuntos $Q_1^1 = \{1,2,6,7\}$, $Q_2^1 = \{4,5,10,11\}$ y $Q^2 = \{0,7,14\}$. Notar que el evento 7 aparece tanto en Q_1^1 como en Q^2. En consecuencia, todos los eventos relacionados al inicio y al final del evento en conflicto son perturbados, esto es, $Q_1^1 = \{1+\epsilon*1, 2+\epsilon*1, 6+\epsilon*1, 7+\epsilon*1\}$. Así, se tiene el conjunto ordenado temporalmente $Q = \{0, 1+\epsilon*1, 2+\epsilon*1, 4, 5, 6+\epsilon*1, 7, 7+\epsilon*1, 10, 11, 14\}$. Ahora, dado que $|Q| = 2(\gamma+1)|J| + |P| + 1 = 11$, entonces $V = \{1,\ldots,11\} \cup \{o,d\}$ y se tienen los valores asociados a la función h : $h(1) = 0, h(2) = 1+\epsilon, \ldots, h(11) = 14$. Finalmente, considerando los criterios de construcción de arcos, costos y capacidades se tiene la Figura 3.2 donde las etiquetas (c_{ij}, u_{ij}) en los arcos indican el costo y capacidad, respectivamente.

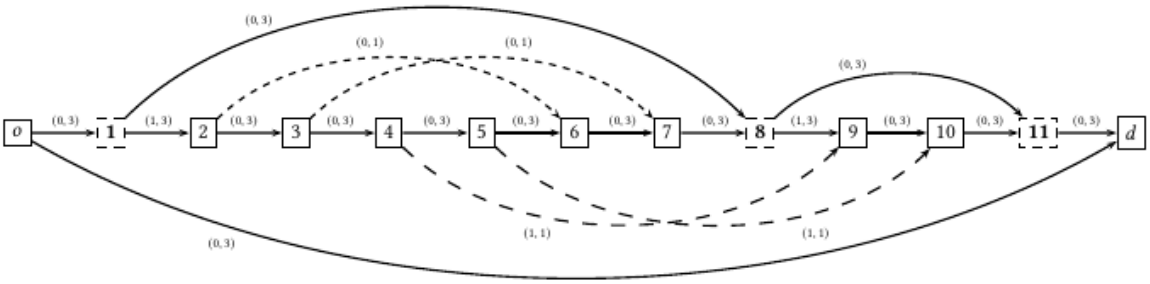

Figura 3.2: Grafo de eventos discretizados del ejemplo 1.

Si se disponen de tres máquinas, una solución factible consiste en procesar cada trabajo en el primer momento posible en una máquina diferente. El costo de la solución es tres, ya que la máquina que procesa el primer trabajo está activa durante el primer período, mientras que la máquina que procesa el segundo trabajo debe estar activa durante los dos períodos. Esta solución se representa en la Figura 3.2 como los caminos:

$$P_1 : o, 1, 2, 6, 7, 8, 11, d.$$
$$P_2 : o, 1, 2, 3, 4, 9, 10, 11, d.$$
$$P_0 : o, d.$$

Los caminos P_1 y P_2 indican la secuencia de trabajos procesados por las dos primeras máquinas, mientras que el camino P_3 señala que la tercera máquina no tiene ningún trabajo asignado en la solución. Los costos de los caminos corresponden al número de períodos activos de las máquinas: 1, 2 y 0 para los tres caminos dados.

Como se observa en el ejemplo 1, cualquier solución factible para CMD se puede descomponer en $|M|$ caminos en D. Para cualquier camino $\hat{P}$, los arcos en $A(\hat{P}) \cap A_3$ determinan la secuencia de trabajos que debe procesar una de las máquinas.

Capítulo 4

Desigualdades válidas y cotas inferiores

En este capítulo se presentan varias cotas inferiores y desigualdades válidas para los modelos CMC y CMD asociados al problema MPCM. Algunas desigualdades válidas serán incorporadas en el nodo raíz y otras en un algoritmo tipo Branch & Cut.

4.1. Cotas inferiores para el problema MPCM

A continuación se exponen algunas definiciones y resultados preliminares asociados al problema MPCM, que permitirán obtener cotas inferiores y desigualdades válidas para los modelos.

DEFINICIÓN 4.1. *Sea $S \subseteq J$, se define la longitud del conjunto S por $l(S) = f_S - a_S$, con $f_S = \text{máx}_{i \in S}\{b_i + t_i\}$ y $a_S = \text{mín}_{i \in S}\{a_i\}$.*

DEFINICIÓN 4.2. *Sea $S \subseteq J$, se llama tiempo de procesamiento del conjunto S a $t(S) = \sum_{i \in S} t_i$.*

TEOREMA 4.1. *Sea $S \subseteq J$ y $|S| \geq 2$. Si $l(S) < t(S)$ entonces los trabajos en S deben ser procesados por al menos 2 máquinas.*

Demostración. Supongamos que los trabajos se procesan en una sola máquina. Sin pérdida de generalidad asumimos que el primer trabajo inicia su procesamiento en el tiempo a_S y que una vez finalizado un trabajo el siguiente será ejecutado inmediatamente después. Así, se tiene que $a_S + t(S) \leq f_S$, pues f_S es el tiempo máximo de

finalización de los trabajos en S. Claramente lo anterior genera una contradicción, pues se tiene que $t(S) \leq f_S - a_S = l(S)$. En consecuencia, los trabajos deben ser procesados por al menos 2 máquinas. □

La idea de compatibilidad de trabajos (Definición. 3.1) puede ser expuesta de forma equivalente por el siguiente resultado:

TEOREMA 4.2. *Sean i, j dos trabajos en J. Si $\gamma \geq a_j - a_i + t_j$ o $\gamma \geq a_i - a_j + t_i$ entonces los trabajos i y j pueden ser procesados por una sola máquina y serán llamados trabajos compatibles.*

Demostración. Las expresiones $\gamma \geq a_j - a_i + t_j$ y $\gamma \geq a_i - a_j + t_i$ son equivalentes a $b_i \geq a_j + t_j$ y $b_j \geq a_i + t_i$, respectivamente. La primera expresión implica que el trabajo j debe ser procesado primero, ya que al menos uno de sus tiempos de finalización es menor o igual que uno de los tiempos de inicio del trabajo i, es decir, el trabajo i puede ser procesado por la misma máquina después del trabajo j. Además, se verifica que si $b_i = a_j + t_j$ y se procesa primero la tarea j, entonces solo existe una forma de procesar los trabajos y es atender uno inmediatamente después del otro. Para el caso discreto si, $b_i > a_j + t_j$, entonces existen al menos $b_i - a_j + t_j + 1$ formas de procesar los trabajos. En la segunda expresión se tiene el mismo análisis, pero primero se procesa el trabajo i. □

El resultado anterior permite obtener un par de corolarios que serán los cimientos para la construcción de varias familias de desigualdades válidas.

COROLARIO 4.3. *Sean i, j dos trabajos en J. Si $\gamma < a_j - a_i + t_j$ y $\gamma < a_i - a_j + t_i$, entonces los trabajos i y j no pueden ser procesados por una sola máquina y serán llamados trabajos incompatibles.*

Demostración. El resultado se obtiene al negar el teorema anterior, por consecuencia queda demostrado. □

COROLARIO 4.4. *Sean $S \subseteq J$ con $|S| \geq 2$ y $t \in S$ un punto de tiempo. Si para todo trabajo $i \in S$ se tiene que $b_i < t$ y $a_i + t_i \geq t$, entonces S es un conjunto de trabajos incompatibles.*

Demostración. La demostración es un resultado inmediato del corolario anterior. □

A partir del concepto de incompatibilidad es posible encontrar cotas inferiores para el número de máquinas en cada uno de los períodos de trabajo del sistema. Para esto primero se presentan las siguientes definiciones:

DEFINICIÓN 4.3. *Sea $p \in P$. Se define $J_p^1 = \{i \in J : Lp \leq a_i < b_i < L(p+1)\}$ como el conjunto de trabajos que deben iniciar su procesamiento en el período p.*

DEFINICIÓN 4.4. *Sean $p \in P$ y un tiempo $t \in [Lp, L(p+1))$. Se llamarán conjuntos de inicialización de máquinas en un tiempo t a los $W_p^t \subseteq J_p^1$, tal que todo par de trabajos distintos $i, j \in W_p^t$ son incompatibles en dicho tiempo t.*

DEFINICIÓN 4.5. *Sea $p \in P$. Se define el conjunto mínimo de inicialización de máquinas para trabajos incompatibles de un período p por:*

$$W_p = \text{máx}_{t \in [Lp, L(p+1))}\{W_p^t\}.$$

TEOREMA 4.5. *Para cada $p \in P$ se necesitan al menos $|W_p|$ máquinas.*

Demostración. Puesto que W_p contiene trabajos que deben empezar en p y además ningún par de trabajos en W_p puede ser atendido en una misma máquina, entonces cada trabajo deberá ser procesado por una máquina distinta. Así, se necesitan al menos $|W_p|$ máquinas para procesar estos trabajos en el período p. □

COROLARIO 4.6. *El número mínimo de máquinas necesarias para procesar todos los trabajos de una jornada con $|P|$ períodos es igual a $max_{p \in P}|W_p|$.*

Demostración. Por el teorema anterior se tiene que en cada $p \in P$ se necesitan al menos $|W_p|$ máquinas para procesar los trabajos. Por consecuencia, el máximo de dichos valores es el mínimo número de máquinas necesarias para procesar todos los trabajos de una jornada con $|P|$ períodos . □

Usando el mismo criterio de incompatibilidad y considerando la finalización del procesamiento de un trabajo en lugar de la inicialización, se plantean dos nuevas cotas inferiores para el número de máquinas. Primero se introducen las siguientes definiciones:

DEFINICIÓN 4.6. *Se define $J_p^2 = \{i \in J : Lp \leq a_i + t_i < b_i + t_i < L(p+1)\}$ como el conjunto de trabajos que deben ser finalizados en el período $p \in P$.*

DEFINICIÓN 4.7. *Sean $p \in P$ y un tiempo $t \in [Lp, L(p+1))$. Se llamarán conjuntos de finalización de máquinas en un tiempo t a los $F_p^t \subseteq J_p^2$, tal que todo par de trabajos distintos $i, j \in F_p^t$ son incompatibles en dicho tiempo t.*

DEFINICIÓN 4.8. *Sea $p \in P$. Se define el conjunto mínimo de finalización de máquinas para trabajos incompatibles de un período p por:*

$$F_p = \text{máx}_{t \in [Lp, L(p+1))} \{F_p^t\}.$$

Con esto se proponen dos resultados adicionales:

TEOREMA 4.7. *Para cada $p \in P$ se necesitan al menos $|F_p|$ máquinas.*

Demostración. Puesto que F_p contiene trabajos que deben finalizar en p y además ningún par de trabajos en F_p puede ser atendido en una misma máquina, entonces cada trabajo deberá ser procesado por una máquina distinta. Así, se necesitan al menos $|F_p|$ máquinas para procesar estos trabajos en el período p. □

COROLARIO 4.8. *El número mínimo de máquinas necesarias para procesar todos los trabajos de una jornada con $|P|$ períodos es igual a $max_{p \in P}|F_p|$.*

Demostración. En el teorema anterior se demostró que para cada $p \in P$ se necesitan al menos $|F_p|$ máquinas para procesar los trabajos. Por consecuencia, el máximo de dichos valores es el mínimo número necesario de máquinas para procesar todos los trabajos de una jornada con $|P|$ períodos . □

Los siguientes resultados se enfocan en el comportamiento del sistema sobre cada uno de los períodos de trabajo. Se introducen las siguientes definiciones:

DEFINICIÓN 4.9. *Sea $p \in P$. Se define a $J_p = \{i \in J : Lp \leq a_i < b_i + t_i < L(p+1)\}$ como el conjunto de trabajos que deben ser procesados en el período p.*

DEFINICIÓN 4.10. *Sea $p \in P$. Se define el conjunto de trabajos que se procesan en el período p como $\theta_p = J_p \cup J_p^1 \cup J_p^2$.*

DEFINICIÓN 4.11. *Se define $\eta_p = \lceil \frac{|\theta_p|}{|J|} \rceil$ como el número minimal de máquinas para el período $p \in P$.*

Con todas estas definiciones y teoremas anteriores se procede a plantear los siguientes resultados:

TEOREMA 4.9. *Sean $S \subseteq J$, W_p^S y F_p^S los conjuntos mínimos de inicialización y finalización de máquinas en un período $p \in P$ usando los trabajos en el conjunto S, respectivamente. Además, sean J_p^S el subconjunto de tareas de S que deben ser procesadas en el período p y η_p^S el número minimal de máquinas de un período p para*

el conjunto S. El número mínimo de períodos-máquinas necesarios para procesar los trabajos en S es:

$$\text{máx}\left\{\left\lceil\frac{t(S)}{L}\right\rceil, \sum_{p=\left\lfloor\frac{a_S}{L}\right\rfloor}^{\left\lceil\frac{f_S}{L}\right\rceil} \text{máx}\left\{|W_p^S|, |F_p^S|, \eta_p^S, \left\lceil\frac{t(J_p^S)}{L}\right\rceil\right\}\right\}$$

Demostración. Se tiene que $p \in [\lfloor\frac{a_S}{L}\rfloor, \lfloor\frac{f_S}{L}\rfloor]$. Puesto que W_p^S es el conjunto de trabajos no compatibles 2 a 2 en el período p considerando trabajos en S, entonces $|W_p^S|$ es una cota inferior para el número de máquinas necesarias para procesar los trabajos de S en dicho período. Análogamente, $|F_p^S|$ también es una cota inferior para el número de máquinas para procesar los trabajos de S en dicho período. Por otra parte, es claro que η_p^S es el número mínimo de máquinas necesarias en el período p, pues si se debe procesar al menos un trabajo de S en dicho período, entonces $\eta_p^S = 1$ y en el caso de que no se procese ningún trabajo en dicho período se tendrá $\eta_p^S = 0$. Además, es fácil notar que para procesar los trabajos del conjunto J_p^S, las máquinas necesarias deben cubrir al menos el tiempo de procesamiento de dichos trabajos lo que implica que la suma de los tiempos de procesamiento de los trabajos sobre este conjunto son una cota inferior para el número de máquinas necesarias para procesar los trabajos de S en cada período p. Por consecuencia, al ser todas cotas mínimas en un período p, el máximo de estos valores es la mejor cota inferior para dicho período y al sumarlos se tendrá una cota inferior del número de períodos-máquinas necesarios para procesar los trabajos en S. Finalmente, si todas las máquinas procesan los trabajos en S sin tener tiempos inactivos, entonces $\left\lceil\frac{t(S)}{L}\right\rceil$ corresponde a una segunda cota inferior válida. Por tanto, el mayor de estos dos valores es una cota para el óptimo del problema MPCM.

DEFINICIÓN 4.12. *Sean i y j dos trabajos en J. Se define a $S_2 = \{i, j\}$ como el conjunto generador de trabajos compatibles si i es compatible con j, pero j no lo es con i.*

DEFINICIÓN 4.13. *Se define $S_n \subseteq J$, con $|S_n| = n \geq 3$, como el conjunto básico de trabajos compatibles, si contiene al menos un conjunto generador de trabajos compatibles $\{i, j\}$ y además $a_i + t_i \leq a_k + t_k$ y $b_k \leq b_j$ para todo $k \in S_n \setminus S_2$.*

TEOREMA 4.10. *Sean S_n y un conjunto generador de trabajos compatibles $S_2 = \{i, j\} \subset S_n$. Si se satisface que:*

- $\bigcap_{k \in S_n \setminus S_2} [a_k, b_k] \bigcap [a_i + t_i, b_j] = [I_1, I_2] \neq \emptyset$,

- $I_2 - I_1 \geq t(S_n \setminus S_2)$,

entonces todos los trabajos en S_n pueden ser procesados por una sola máquina.

Demostración. El intervalo $[a_i + t_i, b_j]$ es el espacio de tiempo que se genera al procesar primero el trabajo i y al final el trabajo j. Claramente existe un espacio de tiempo mayor a 0, ya que de no ser el caso, la intersección de la primera condición sería vacía. Este intervalo marca el espacio de tiempo donde podrán ser procesados el resto de trabajos. Por otra parte, al realizar $\bigcap_{k \in S_n \setminus S}[a_k, b_k]$ se tendrá una ventana de tiempo común de inicios de procesamiento para los trabajos k. Así, la expresión $\bigcap_{k \in S_n \setminus S_2}[a_k, b_k] \bigcap [a_i + t_i, b_i] = [I_1, I_2] \neq \emptyset$ indica que existe un espacio de tiempo entre el procesamiento de los trabajos en S_2, donde los trabajos en $S_n \setminus S_2$ pueden iniciar su procesamiento. Finalmente, basta con garantizar que el tamaño del intervalo (en términos de tiempo) sea mayor o igual a la suma de los tiempos de procesamientos de los trabajos, esto es $I_2 - I_1 \geq t(S_n \setminus S_2)$. Así, todos los trabajos en S_n pueden ser procesados por una sola máquina. □

El resultado anterior abre paso al estudio de cotas inferiores para el problema MPCM. Por ejemplo, si se considera $|S_n| = 3$ y su conjunto generador de trabajos compatibles $S_2 = \{i, j\} \subset S_n$ y $I_2 - I_1 < t_k$, entonces para este conjunto se necesitarán al menos 2 períodos-máquinas para el procesamiento de los trabajos. Podrán darse varios enfoques acorde a la cardinalidad del conjunto básico de trabajos compatibles o al tiempo de procesamiento de los trabajos que no están en su conjunto generador de trabajos compatibles. Así, se presenta el siguiente resultado:

TEOREMA 4.11. *Sean $S_n \subseteq J$ y $S_2 = \{i, j\} \subset S_n$ un conjunto generador de trabajos compatibles. Si para todo $k \in S_n \setminus S_2$ se tiene que $b_j - (a_i + t_i) < t_k$, entonces se necesitan al menos $\lceil \frac{n}{2} \rceil$ períodos-máquina para procesar los trabajos de S_n.*

Demostración. Notar que en el mejor de los casos todo par de trabajos distintos l, h en $\in S_n \setminus S_2$ pueden ser compatibles. Así, entre este par de trabajos se genera un intervalo de tiempo $[a_l + t_l, b_h]$ donde pueden ser procesados otros trabajos. Puesto que $a_i + t_i \leq a_k + t_k$, $b_k \leq b_j$ y $b_j - (a_i + t_i) < t_k$ para todo $k \in S_n \setminus S_2$, entonces $b_h - (a_l + t_l) < t_k$ para todo $l \neq h \neq k$. De lo anterior, ningún subgrupo de trabajos de cardinalidad mayor a 2 puede ser procesado por una sola máquina y ningún trabajo $k \in S_n \setminus S_2$ puede ser procesado por la misma máquina que procesa los trabajos i, j. Puesto que se ha asumido el mejor escenario, la compatibilidad de trabajos, se concluye que al menos se necesitarán $\lceil \frac{n}{2} \rceil$ máquinas para procesar los trabajos en

S_n. Claramente, si es una cota inferior para el número de máquinas también lo será para el número de períodos-máquina. □

TEOREMA 4.12. *Para cada $p \in P$ se necesitan al menos $\left\lceil \frac{t(J_p)}{L} \right\rceil$ máquinas.*

Demostración. Puesto que el conjunto J_p contiene todos los trabajos que deben ser procesados en el período p entonces el número de máquinas necesarias para procesar estos trabajos deben cubrir al menos su tiempo de atención. □

COROLARIO 4.13. *El número mínimo de máquinas necesarias para procesar todos los trabajos de una jornada con $|P|$ períodos es igual a* $\max_{p \in P} \left\{ \left\lceil \frac{t(J_p)}{L} \right\rceil \right\}$.

Demostración. En el teorema anterior se demostró que para cada $p \in P$ se necesitan al menos $\left\lceil \frac{t(J_p)}{L} \right\rceil$ máquinas para procesar los trabajos. Por consecuencia, el máximo de dichos valores es el mínimo número necesario de máquinas para procesar todos los trabajos de una jornada con $|P|$ períodos . □

4.2. Desigualdades válidas y cotas inferiores para CMC

En esta sección se presentan desigualdades válidas y cotas inferiores para el problema MPCM en su versión continua. Algunos resultados son consecuencia de lo reportado en la Sección 4.1.

TEOREMA 4.14. *El valor óptimo del modelo CMC satisface la siguiente desigualdad:*

$$\sum_{i \in J} x_{oi} + \sum_{i \in J} (y_i^2 - y_i^1) + \sum_{(i,j) \in C} z_{ij} \geq \left\lceil \frac{t(J)}{L} \right\rceil$$

Demostración. El resultado es inmediato puesto que el número total de períodos-máquina en el horizonte de tiempo deben cubrir al menos la suma de los tiempos de procesamiento de todos los trabajos en J. □

TEOREMA 4.15. *La desigualdad:*

$$\sum_{i \in J} (y_i^2 - y_i^1) \geq 0$$

es válida para CMC

Demostración. El resultado es inmediato, pues el período en el que finaliza el procesamiento de un trabajo $i \in J$ debe ser siempre mayor o igual al período en el que llega el trabajo. □

Los siguientes resultados permiten identificar el período en el que un trabajo inicia y finaliza su procesamiento.

TEOREMA 4.16. *Dado un trabajo $i \in J$ tal que $\lfloor a_i/L \rfloor = \lfloor b_i/L \rfloor$, entonces la ecuación:*

$$y_i^1 = \left\lfloor \frac{a_i}{L} \right\rfloor$$

es válida para el modelo CMC.

Demostración. Dado que $a_i \leq T_i \leq b_i$ y por la restricción (3.7), se prueba el resultado. □

Este teorema indica que el trabajo será procesado exactamente en el mismo período que llegó.

TEOREMA 4.17. *Sea $i \in J$ un trabajo tal que $\lfloor (a_i + t_i)/L \rfloor = \lfloor (b_i + t_i)/L \rfloor$, entonces la ecuación:*

$$y_i^2 = \left\lfloor \frac{a_i + t_i}{L} \right\rfloor$$

es válida para el modelo CMC.

Demostración. Dado que $a_i \leq T_i \leq b_i$ y además se tiene la restricción (3.6), entonces el resultado es inmediato. □

El siguiente resultado identifica los trabajos que inician y terminan en el mismo período. Es consecuencia de los dos teorema anteriores.

TEOREMA 4.18. *Dado un trabajo $i \in J$ tal que $\lfloor a_i/L \rfloor = \lfloor (b_i + t_i)/L \rfloor$, entonces la ecuación:*

$$y_i^1 = y_i^2 = \left\lfloor \frac{a_i}{L} \right\rfloor$$

es válida para CMC.

Ahora se presentarán resultados que reducen el número de variables binarias.

TEOREMA 4.19. *Dados dos trabajos $i, j \in J$, tal que $(i, j) \in C$ y $\lfloor a_i/L \rfloor = \lfloor b_j/L \rfloor$, entonces*

$$y_j^1 - y_i^2 \leq 0.$$

Además, la ecuación:

$$z_{ij} = 0$$

se cumple para la solución óptima de CMC.

Demostración. Dado que $\lfloor a_i/L \rfloor = \lfloor b_j/L \rfloor$, el procesamientos de los trabajos i y j empiezan en el mismo período. Por otra parte, se tiene que $a_i < T_i + t_i$ y por la restricción (3.6) se tiene que:

$$y_j^1 \leq \left\lfloor \frac{T_j}{L} \right\rfloor \leq \left\lfloor \frac{b_j}{L} \right\rfloor = \left\lfloor \frac{a_i}{L} \right\rfloor \leq \left\lceil \frac{T_i + t_i - L}{L} \right\rceil \leq y_i^2,$$

Además, dado que el lado derecho de (3.8) es menor o igual a cero, $z_{ij} = 0$ se cumple para la solución óptima de CMC. □

Algunas combinaciones de arcos en D representan soluciones infactibles, por lo que deben ser eliminadas del modelo. Así, se presentan los siguientes resultados.

TEOREMA 4.20. *Sean $i, j \in J$ dos trabajos compatibles, entonces se tiene la siguiente desigualdad válida para CMC:*

$$z_{ij} \leq x_{ij}$$

Demostración. El resultado es inmediato pues si el arco $(i, j) \in C$ no es seleccionado en la solución, entonces por definición $x_{ij} = z_{ij} = 0$. □

TEOREMA 4.21. *Dados dos trabajos $i, j \in J$, tal que $(i, j), (j, i) \in C$, entonces las desigualdades*

$$z_{ij} + z_{ji} \leq 1, \quad x_{ij} + x_{ji} \leq 1$$

son válidas para el modelo CMC.

Demostración. De (3.5) se tiene que x_{ij} y x_{ji} no pueden tomar el valor de uno simultáneamente. Por consecuencia y por el teorema anterior, z_{ij} y z_{ji} no pueden tomar el valor de uno simultáneamente. □

Ahora, se presentan los resultados derivados del teorema 4.5 y el corolario 4.6.

TEOREMA 4.22. *Para todo período $p \in P$, la desigualdad*

$$\sum_{\substack{j \in J_p^1: \\ i \notin J_p^1}} x_{ij} \geq |W_p|$$

es válida para el modelo CMC.

COROLARIO 4.23. *La desigualdad*

$$\sum_{i \in J} x_{oi} \geq \max_{p \in P} |W_p|$$

es válida para CMC.

Además, se tienen los casos particulares para el conjunto mínimo de finalización de un período p obtenidos en el teorema 4.7 y corolario 4.8.

TEOREMA 4.24. *Para todo período $p \in P$, la desigualdad*

$$\sum_{\substack{j \in J_p^2: \\ i \notin J_p^2}} x_{ij} \geq |F_p|$$

es válida para el modelo CMC.

COROLARIO 4.25. *La desigualdad*

$$\sum_{i \in J} x_{oi} \geq \max_{p \in P} |F_p|$$

es válida para CMC.

A continuación se presentan los resultados derivados del teorema 4.12 y el corolario 4.13:

TEOREMA 4.26. *Para todo período $p \in P$, la desigualdad*

$$\sum_{\substack{j \in J_p: \\ i \notin J_p}} x_{ij} \geq \left\lceil \frac{t(J_p)}{L} \right\rceil$$

es válida para el modelo CMC.

COROLARIO 4.27. *La desigualdad*

$$\sum_{i \in J} x_{oi} \geq \max_{p \in P} \left\{ \left\lceil \frac{t(J_p)}{L} \right\rceil \right\}$$

es válida para CMC.

4.3. Desigualdades válidas y cotas inferiores para CMD

A continuación se presenta un resultado derivado del teorema 4.14:

TEOREMA 4.28. *La desigualdad*

$$\sum_{(i,j)\in A} c_{ij}x_{ij} \geq \left\lceil \frac{t(J)}{L} \right\rceil$$

es válida para CMD.

Ahora se reportarán los resultados derivados del teorema 4.12 y del corolario 4.13, para esto es necesario introducir la siguiente definición:

DEFINICIÓN 4.14. *Se define a* $V_p = \{j \in V \setminus \{o,d\} \,:\, Lp \leq h(j) < L(p+1)\}$ *como el conjunto de nodos que representan eventos que ocurren dentro del período* $p \in P$.

TEOREMA 4.29. *Para cada período* $p \in P$*, la desigualdad*

$$\sum_{\substack{(j,l)\in A_3:\\ j\notin V_p, l\in V_p}} x_{jl} + x_{k,k+1} \left\lceil \frac{t(J_p)}{L} \right\rceil$$

es válida para el modelo CMD. El nodo k *representa el inicio del período* p*, es decir,* $h(k) = Lp \in Q^2$.

COROLARIO 4.30. *La desigualdad*

$$x_{o1} \geq \max_{p\in P} \left\{ \left\lceil \frac{t(J_p)}{L} \right\rceil \right\}$$

es válida para el modelo CMD.

A continuación se presentan las desigualdades válidas a partir del conjunto mínimo de inicialización que se obtuvieron en el teorema 4.5 y corolario 4.6.

TEOREMA 4.31. *Para todo período* $p \in P$*, la desigualdad*

$$\sum_{\substack{(j,l)\in A_3:\\ j\notin V_p, l\in V_p}} x_{jl} + x_{k,k+1} \geq |W_p|$$

es válida para el modelo CMD.

COROLARIO 4.32. *La desigualdad*

$$x_{o1} \geq \max_{p\in P} |W_p|$$

es válida para el modelo CMD.

Así, se presentan las desigualdades válidas a partir del conjunto mínimo de finalización para CMD obtenidas en el teorema 4.7 y corolario 4.8.

TEOREMA 4.33. *Para todo período $p \in P$, la desigualdad*

$$\sum_{\substack{(j,l)\in A_3:\\ j\notin V_p, l\in V_p}} x_{jl} + x_{k,k+1} \geq |F_p|$$

es válida para el modelo CMD.

COROLARIO 4.34. *La desigualdad*

$$x_{o1} \geq \max_{p\in P} |F_p|$$

es válida para el modelo CMD.

Capítulo 5

Resultados Computacionales

En este capítulo se reportan los resultados de 100 experimentos computacionales llevados a cabo con las formulaciones CMC y CMD propuestos en el Capítulo 3. Se estudia el impacto que tienen en los modelos la inclusión de las desigualdades válidas descritas en el Capítulo 4. Se presentan los resultados del modelo continuo con desigualdades válidas incluidas en un algoritmo exacto tipo Branch & Cut. Finalmente, se presentan resultados computacionales sobre instancias reales de días completos para el modelo discreto con la elección de la mejor estrategia de solución.

Los modelos de programación entera se resolvieron utilizando el solver de optimización Gurobi 9.1.1 [23] y la interfaz del lenguaje de programación Python. Todos los experimentos se realizaron en un computador portátil Intel Core i7 2.7GHz con 16 GB de RAM bajo el sistema operativo Windows 10 Home.

5.1. Instancias

Los experimentos computacionales se realizaron sobre un conjunto de instancias de prueba considerando datos del mundo real (anonimizados) proporcionados por el SRI desde su base de datos. Se consideraron varias agencias representativas del país y se seleccionaron varios días del mes de octubre de 2017 y abril y junio de 2018. Las primeras 11 instancias se obtuvieron muestreando un número parcial de clientes de estos días, las siguientes 9 instancias son modificaciones de las instancias de prueba muestreadas que incluyen conjuntos de clientes que satisfacen el Teorema 4.9 y las últimas 6 son instancias de pruebas completas, es decir, abarcan toda la jornada laboral de las agencias. Los datos incluyen tiempos de llegada de clientes, tiempos de espera y tiempos de procesamiento.

El Cuadro 5.1 describe las características de las primeras 11 instancias. La primera columna de la tabla hace referencia al nombre abreviado de la instancia que se usará en los experimentos, las columnas 2 y 3 indican el número de clientes y la cantidad de períodos, respectivamente. El resto de columnas indican el nombre de la agencia, año, mes, día y números de período de los cuales se han tomado los datos.

Cuadro 5.1: Información de instancias muestreadas

Instancia	|J|	|P|	Agencia	Año	Mes	Día	Períodos
I_1	30	1	Plataforma Gubernamental Amazonas	2018	06	01	8 a 9
I_2	50	9	Plataforma Gubernamental Amazonas	2018	06	01	8 a 16
I_3	100	4	Centenario	2018	06	01	8 a 11
I_4	100	5	Chone	2018	06	01	8 a 12
I_5	100	4	Guayaquil Centro	2018	06	01	8 a 11
I_6	248	9	Plataforma Gubernamental Amazonas	2018	06	01	8 a 16
I_7	287	4	Plataforma Gubernamental Amazonas	2018	06	01	8 a 11
I_8	372	9	Plataforma Gubernamental Amazonas	2018	06	01	8 a 16
I_9	457	5	Plataforma Gubernamental Amazonas	2018	06	01	13 a 17
I_{10}	507	6	Plataforma Gubernamental Amazonas	2017	10	01	8 a 13
I_{11}	556	6	Plataforma Gubernamental Amazonas	2017	10	01	8 a 13

El cuadro 5.2 presenta las 9 instancias modificadas con la inclusión de clientes que satisfacen el Teorema 4.9 y tiene la misma estructura que el cuadro 5.1.

Cuadro 5.2: Información de instancias modificadas

Instancia	|J|	|P|	Agencia	Año	Mes	Día	Períodos
IS_1	35	1	Plataforma Gubernamental Amazonas	2018	06	01	8 a 9
IS_2	75	9	Plataforma Gubernamental Amazonas	2018	06	01	8 a 16
IS_3	95	9	Plataforma Gubernamental Amazonas	2018	06	01	8 a 16
IS_4	100	4	Centenario	2018	06	01	8 a 11
IS_5	100	5	Chone	2018	06	01	8 a 12
IS_6	100	4	Guayaquil Centro	2018	06	01	8 a 11
IS_7	155	9	Plataforma Gubernamental Amazonas	2018	06	01	8 a 16
IS_8	205	9	Plataforma Gubernamental Amazonas	2018	06	01	8 a 16
IS_9	305	9	Plataforma Gubernamental Amazonas	2018	06	01	8 a 16

En el Cuadro 5.3 se muestra la información de las instancias de prueba considerando días completos y tiene la misma estructura que los dos cuadros anteriores.

Cuadro 5.3: Información de instancias completas

Instancia	|J|	|P|	Agencia	Año	Mes	Día	Períodos
IC_1	499	9	Centenario	2018	04	02	8 a 16
IC_2	533	10	Manta	2018	04	02	8 a 17
IC_3	598	9	Salinas y Santiago	2018	04	02	8 a 16
IC_4	697	10	Galo Plaza	2018	04	02	8 a 17
IC_5	853	10	Plataforma Gubernamental Amazonas	2018	04	02	8 a 17
IC_6	931	9	Cuenca	2018	04	02	8 a 16

Considerando los estándares de calidad y la información de las agencias proporcionada por el SRI se tiene que: (i) el tiempo de espera es de veinte minutos, por consecuencia $\gamma = 20/60$ horas, (ii) el número promedio de períodos en los que las agencias ofrecen servicio al cliente es $|P| = 12$, (iii) la duración de cada uno de estos períodos es $L = 1$ hora y (iv) el número máximo de ventanillas disponibles en la jornada (máquinas activas en cada período) es de $|M| = 25$.

5.2. Experimentos computacionales usando CMC

El objetivo de esta sección es estudiar el desempeño del modelo CMC y el impacto de la inclusión de las desigualdades válidas estudiadas en el capítulo 4. Para esto se han considerado 4 escenarios: (i) CMC, que es la experimentación computacional del modelo continuo sin incluir desigualdades válidas, (ii) $CMC + BDC1$, que considera el modelo continuo junto con las desigualdades obtenidas de los Teoremas 4.14, 4.15, 4.20 y 4.21, (iii) $CMC + BDC2$, que corresponde al modelo continuo con la inclusión de las desigualdades obtenidas de los Teoremas 4.14, 4.18, 4.19, 4.26 y 4.25 y (iv) $CMC + BDC3$, que contiene todas las desigualdades de los bloques $BDC1$ y $BDC2$.

En el Cuadro 5.4 se presentan los resultados de los experimentos computacionales considerando las instancias del Cuadro 5.1. Las primeras tres filas contienen la información de cada una de las instancias. Cada escenario contiene un bloque de filas incluyendo el valor de la función objetivo, la mejor cota inferior, la brecha de optimalidad en porcentaje (GAP), el tiempo de cómputo en segundos, la cantidad de nodos explorados por el algoritmo Branch and Bound, la cantidad de variables y el número de restricciones. El tiempo de cálculo se limitó a 3600 segundos para cada instancia y se usó la configuración por defecto de Gurobi. Todas las instancias consideran 25 máquinas idénticas.

Se puede observar un mal comportamiento del primer escenario, ya que en todos los casos el GAP supera el 58 % e incluso en la instancia I_{10} se supera el 97 %. En los siguientes 3 escenarios, la inclusión de las desigualdades válidas genera una drástica reducción del GAP en todas las instancias (en promedio 68,96 %). Se puede observar que en las instancia I_{11}, el GAP del primer escenario en relación al cuarto escenario se reduce en más de un 88 %. Finalmente, podemos afirmar que el escenario 4 es el mejor de todos los escenarios considerados, ya que se obtienen mejores resultados, principalmente en las instancias de gran tamaño. De esta forma, se ha comprobado computacionalmente el efecto positivo de la inclusión de desigualdades válidas en CMC.

Cuadro 5.4: Soluciones de CMC

	Instancia	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}
	|J|	30	50	100	100	100	248	287	372	457	507	556
	|P|	1	9	4	5	4	9	4	9	5	6	6
	Obj	7	12	28	15	27	44	52	69	98	103	104
	Cota	1	5	9	2	7	4	5	7	7	3	4
	GAP(%)	85.71	58.33	67.86	86.67	74.07	90.91	90.38	89.86	92.86	97.09	96.15
CMC	Tiem. (S)	3600.33	3600.19	3600.08	3600.13	3600.24	3600.18	3600.14	3600.28	3600.17	3600.21	3600.16
	# Nodos	146331	128087	5533	64518	3242	2211	277	104	1	1	1
	# Var	1572	2920	11334	11800	11120	65496	91639	146374	227165	282553	337016
	# Res	1603	2971	11435	11901	11221	65745	91927	146747	227623	283061	337573
	Obj	7	12	28	14	26	43	49	69	81	87	94
	Cota	6	8	24	12	25	37	44	56	68	68	75
CMC	GAP(%)	14.29	33.33	14.29	14.29	3.85	13.95	10.2	18.84	16.05	21.83	20.21
+	Tiem. (S)	3600.24	3600.05	3600.12	3600.06	3600.08	3600.17	3600.31	3600.14	3600.18	3600.4	3600.61
BDC1	# Nodos	652330	172252	41383	50828	51144	1416	1216	401	1250	89	277
	# Var	1572	2920	11334	11800	11120	65496	91639	146374	227165	282553	337016
	# Res	2846	4481	16854	17553	16533	100661	145006	225245	340065	423072	504693
	Obj	7	12	27	14	27	42	49	66	81	80	96
	Cota	6	8	24	12	25	37	44	56	68	68	75
CMC	GAP(%)	14.29	33.33	11.11	14.29	7.41	11.9	10.2	15.15	16.05	15	21.88
+	Tiem. (S)	3600.15	3600.11	3600.15	3600.07	3600.08	3602.11	3600.15	3605.37	3600.22	3600.27	3600.22
BDC2	# Nodos	448353	209720	68441	120371	90312	2624	1626	1674	1976	1876	52
	# Var	1572	2920	11334	11800	11120	65496	91639	146374	227165	282553	337016
	# Res	1606	2982	11441	11908	11227	65757	91934	146759	227631	283069	337582
	Obj	7	12	28	14	27	42	49	63	78	80	91
	Cota	6	8	24	12	25	37	44	56	68	68	75
CMC	GAP(%)	14.29	33.33	14.29	14.29	7.41	11.9	10.2	11.11	12.82	15	17.58
+	Tiem. (S)	3600.05	3600.26	3600.06	3600.18	3600.12	3600.06	3600.14	3600.13	3600.89	3600.27	3600.79
BDC3	# Nodos	813901	189518	35892	95529	49442	3590	2623	1825	1537	1195	914
	# Var	1572	2920	11334	11800	11120	65496	91639	146374	227165	282553	337016
	# Res	2848	4491	16859	17559	16538	100672	145012	225256	340072	423079	504701

5.2.1. Algoritmo Branch & Cut

En esta sección se estudia el desempeño de CMC con la inclusión de desigualdades asociadas a los Teoremas 4.9 y 4.14. Para la experimentación computacional se usan las instancias reportadas en el Cuadro 5.2.

En el Cuadro 5.5 se presentan los resultados de este escenario. En dicho cuadro, las tres primeras columnas contienen la información de cada una de las instancias. En las siguientes columnas se expone el valor de la función objetivo, la cota inferior, la brecha de optimalidad en porcentaje (GAP), el número de desigualdades asociadas al Teorema 4.9, el tiempo de cómputo en segundos, la cantidad de nodos explorados por el algoritmo B&B, la cantidad de variables y el número de restricciones. El tiempo de cálculo se limitó a 3600 segundos para cada instancia y se desactivaron los planos cortantes que usa Gurobi para el proceso de optimización en su configuración por defecto. Todas las instancias consideran 25 máquinas idénticas.

La rutina de separación es ejecutada sobre cada nodo dependiendo de la solución fraccionaria de CMC y generando una enumeración exhaustiva de conjuntos de tamaño 3, 4 y 5 nodos. Para cada conjunto S, se verifica si la desigualdad asociada al Teorema 4.9 es violada. Si es el caso, la restricción correspondiente es incluida en el LP.

Se puede observar que en todas las instancias existe al menos una desigualdad válida ingresada al LP y que entre más grande es la instancia se necesita explorar una menor cantidad de nodos. Podemos observar que la instancia IS_6 es la de mejor GAP (7,14 %), en donde se exploran 12336 nodos en el algoritmo Branch & Bound y se ingresan dos desigualdades válidas al LP. El GAP promedio de todas las instancias es de 13,18 % que es un buen indicador y denota el beneficio de incluir las desigualdades asociadas al Teorema 4.9 y la desigualdad del Teorema 4.14.

La presente estrategia no fue considerada dentro de los experimentos anteriores ya que el algoritmo de separación usado genera altos tiempo de cálculo y no es competitiva con las otras estrategias.

Cuadro 5.5: Soluciones de CMC con la inclusión de desigualdades asociadas a los Teoremas 4.9 y 4.14

			CMC + Planos							
Instancia	|J|	|P|	Obj	Cota	GAP(%)	Planos	Tiem. (S)	# Nodos	# Var	# Res
IS_1	35	1	8	7	12.50	1	3600.04	1386813	2061	2098
IS_2	75	9	16	14	12.50	3	3600.06	91766	6489	6566
IS_3	95	9	19	16	15.79	3	3604.26	73249	9977	10074
IS_4	100	4	28	25	10.71	1	3601.97	112764	11140	11242
IS_5	100	5	19	16	15.79	3	3600.48	68903	11446	11548
IS_6	100	4	28	26	07.14	2	3602.04	128336	10950	11052
IS_7	155	9	33	28	15.15	2	3604.16	4881	26403	26560
IS_8	205	9	44	37	15.91	1	3605.14	1667	45319	45526
IS_9	305	9	61	53	13.11	1	3601.71	3710	99707	100014

5.3. Experimentos computacionales usando CMD

Al igual que en la sección anterior se han realizado 4 escenarios para estudiar el modelo CMD y el efecto producido por la inclusión de desigualdades válidas. Se tienen los 4 siguientes escenarios: (i) CMD, que corresponde a la experimentación computacional del modelo discreto sin incluir desigualdades válidas, (ii) $CMD + BDD1$, que corresponde al modelo discreto con la inclusión de las desigualdades obtenidas de los Teoremas 4.29 y 4.30, (iii) $CMD + BDD2$, relacionado al modelo discreto junto con la inclusión de desigualdades obtenidas del teorema 4.28 y (iv) $CMD + BDD3$, que contiene todas las desigualdades de los bloques $BDD1$ y $BDD2$.

En el Cuadro 5.6 se presentan los resultados de los experimentos computacionales considerando las instancias del Cuadro 5.1. Las primeras tres filas contienen la información de cada una de las instancias. Cada escenario contiene un bloque de filas incluyendo el mejor valor de la función objetivo, la brecha de optimalidad en porcentaje (GAP), el tiempo de cómputo en segundos, la cantidad de nodos explorados en el proceso de optimización, la cantidad de variables y el número de restricciones. El tiempo de cálculo no se ha limitado para ninguna instancia y se usó la configuración por defecto de Gurobi. Todas las instancias consideran 25 máquinas idénticas.

Cuadro 5.6: Soluciones de CMD

	Instancia	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}	I_{11}
	|J|	30	50	100	100	100	248	287	372	457	507	556
	|P|	1	9	4	5	4	9	4	9	5	6	6
CMD	Obj	7	12	26	13	26	39	46	60	74	74	83
	GAP(%)	0	0	0	0	0	0	0	0	0	0	0
	Tiem. (S)	0.05	0.39	1.42	1.01	0.38	456.01	25.93	683.36	82.68	62.72	19.04
	# Nodos	1	1	1	1	1	14816	307	4863	532	1642	31
	# Var	1235	2535	4385	4485	4342	10595	11833	16102	18404	20149	22073
	# Res	634	1490	2375	2470	2327	5580	6072	8607	9243	9981	10925
CMD + BDD1	Obj	7	12	26	13	26	39	46	60	74	74	83
	GAP(%)	0	0	0	0	0	0	0	0	0	0	0
	Tiem. (S)	0.16	0.43	1.39	1.32	1.21	1225.15	12.39	4085.02	181.59	114.43	48.04
	# Nodos	1	1	1	1	1	33703	794	65913	1780	1241	30
	# Var	1235	2535	4385	4485	4342	10595	11833	16102	18404	20149	22073
	# Res	636	1500	2380	2476	2333	5591	6079	8618	9250	9989	10933
CMD + BDD2	Obj	7	12	26	13	26	39	46	60	74	74	83
	GAP(%)	0	0	0	0	0	0	0	0	0	0	0
	Tiem. (S)	0.11	0.30	1.32	1.26	1.55	4040.64	27.16	2260.30	129.40	96.95	69.42
	# Nodos	1	1	1	1	1	273001	498	21747	743	303	1056
	# Var	1235	2535	4385	4485	4342	10595	11833	16102	18404	20149	22073
	# Res	635	1491	2376	2471	2328	5581	6073	8608	9244	9982	10926
CMD + BDD3	Obj	7	12	26	13	26	39	46	60	74	74	83
	GAP(%)	0	0	0	0	0	0	0	0	0	0	0
	Tiem. (S)	0.05	0.49	4.60	1.32	2.04	1258.08	22.95	3532.36	178.20	15.25	11.98
	# Nodos	1	19	122	1	1	68474	357	42228	1487	1	1
	# Var	1237	2535	4385	4485	4342	10595	11833	16102	18404	20149	22073
	# Res	637	1501	2381	2477	2334	6195	6080	8619	9251	9990	10934

Todos los escenarios alcanzan la optimalidad en todas las instancias. El modelo sin la inclusión de desigualdades válidas tiene un excelente comportamiento ya que se dispone de una buena cota inferior obtenida de la relajación lineal de CMD. Se observa que en la mayoría de los casos se alcanza la optimalidad en menor tiempo en el escenario 3. En las instancias I_6 y I_8 se consigue el valor óptimo en menor tiempo y se explora menos nodos. Por otro lado, para las primeras instancias del cuarto escenario se tiene un comportamiento similar al escenario CMD, incluso se percibe un mejor desempeño en instancias de gran tamaño. En las dos últimas instancias, el cuarto escenario es ampliamente superior al primero, ya que requiere menos tiempo para llegar al óptimo explorando un sólo nodo en el algoritmo de B& B. Debido

a esto y a que las bases de datos del SRI comprenden instancias extremadamente grandes, CMD+BDD3 es seleccionado como la mejor estrategia de solución.

5.4. Comparación de los mejores escenarios de CMC y CMD

En el Cuadro 5.7 se presenta una comparación entre los mejores escenarios de cada unos de los modelos reportados en el presente trabajo. En dicho cuadro, las tres primeras columnas contienen la información de cada una de las instancias. Para cada escenario se expone el valor de la función objetivo, la cota inferior, la brecha de optimalidad en porcentaje (GAP), el tiempo de cómputo en segundos, la cantidad de nodos explorados por el algoritmo B&B, la cantidad de variables y el número de restricciones. El tiempo de cálculo se limitó a 3600 segundos para cada instancia y se usó la configuración por defecto de Gurobi. Todas las instancias consideran 25 máquinas idénticas.

Se puede observar que la formulación CMD se desempeña claramente mejor que CMC, incluso el peor escenario de CMD tiene mejor desempeño que CMC. Esto puede explicarse debido a la diferencia en el número de variables y en el número de restricciones en cada uno de los modelos, donde CMC claramente tiene mayor tamaño generando hasta 17, 6 veces más variables y 38, 6 veces más restricciones que CMD.

Cuadro 5.7: Comparación de los mejores escenarios de CMC y CMD

			CMC+BDC3							CMD + BDD3					
Instancia	|J|	|P|	Obj	Cota	GAP(%)	Tiem. (S)	# Nodos	# Var	# Res	Obj	GAP(%)	Tiem. (S)	# Nodos	# Var	# Res
I_1	30	1	7	6	14.29	3600.05	813901	1572	2848	7	0	0.05	1	1237	637
I_2	50	9	12	8	33.33	3600.26	189518	2920	4491	12	0	0.49	19	2535	1501
I_3	100	4	28	24	14.29	3600.06	35892	11334	16859	26	0	4.60	122	4385	2381
I_4	100	5	14	12	14.21	3600.18	95529	11800	17559	13	0	1.32	1	4485	2477
I_5	100	4	27	25	7.41	3600.12	49442	11120	16538	26	0	2.04	1	4342	2334
I_6	248	9	42	37	11.9	3600.06	3590	65496	100672	39	0	1258.08	68474	10595	6195
I_7	287	4	49	44	10.2	3600.14	2623	91639	145012	46	0	22.95	357	11833	6080
I_8	372	9	63	56	11.1	3600.13	1825	146374	225256	60	0	3532.36	42228	16102	8619
I_9	457	5	78	68	12.82	3600.89	1537	227165	340072	74	0	178.20	1487	18404	9251
I_{10}	507	6	80	68	15.00	3600.27	1195	282553	423079	74	0	15.25	1	20149	9990
I_{11}	556	6	91	75	17.58	3600.79	914	337016	504701	83	0	11.98	1	22073	10934

5.5. Soluciones de la mejor estrategia de CMD en instancias reales

En este apartado se reportan los resultados computacionales de aplicar la mejor estrategia de CMD (CMD+BDD3) en instancias reales que considera toda la jornada laboral de una agencia.

En el Cuadro 5.8 se presentan las soluciones de los experimentos computacionales considerando las instancias del Cuadro 5.3. Las tres primeras columnas de dicho cuadro contienen la información de cada una de las instancias. Para cada escenario se expone el valor de la función objetivo, el tiempo de cómputo en segundos, la cantidad de nodos explorados por el algoritmo B&B, la cantidad de variables y el número de restricciones. No se limitó el tiempo de cálculo y se usó la configuración por defecto de Gurobi. Todas las instancias consideran 25 máquinas idénticas.

En todas las instancias se alcanza la optimalidad. Se puede observar que la instancia IC_3 fue aquella que necesitó mayor tiempo para alcanzar la optimalidad (2590.92 segundos) y fue la segunda instancia en la que se exploró la mayor cantidad de nodos en el algoritmo B&B. Por otra parte, todas las instancias fueron resueltas en menos de 40 minutos. Lo anterior evidencia que CMD+BDD3 es una estrategia adecuada para afrontar el problema MPCM, ya que resuelve instancias con más de 900 trabajos y alrededor de 10 períodos en un tiempo razonable.

Cuadro 5.8: Soluciones de la mejor estrategia de CMD en instancias reales

			CMD + BDD3				
Instancia	|J|	|P|	Obj	Tiem. (S)	# Nodos	# Var	# Res
IC_1	499	9	84	130.38	2468	20933	10910
IC_2	533	10	75	1344.48	40556	22132	11429
IC_3	598	9	91	2590.92	40418	24567	12604
IC_4	697	10	105	544.11	4345	28370	14387
IC_5	853	10	124	807.30	3877	34110	17007
IC_6	931	9	122	2399.59	11473	37094	18431

Capítulo 6

Conclusiones y Recomendaciones

En el presente trabajo se estudió el problema multi-períodico de calendarización de máquinas con aplicación al problema de asignación de personal a ventanillas de servicio al cliente en las distintas agencias del Sistema de Rentas Internas del Ecuador, respetando criterios de calidad impuestos por dicha institución. Para esto se plantearon dos modelos de programación lineal entera y se diseñaron algoritmos exactos de solución. El primer modelo de programación lineal entera mixta, se formuló considerando que cada cliente sea atendido dentro de una ventana de tiempo continua (CMC). El segundo modelo de programación entera, se formuló considerando que cada cliente sea atendido dentro de una ventana de tiempo discretizada (CMD).

Para el modelo CMC, se identificaron varias familias de desigualdades válidas y cotas inferiores. Se implementaron computacionalmente 4 distintos escenarios, considerando el modelo y la inclusión de distintas familias de desigualdades válidas y cotas inferiores. La implementación consideró 11 instancias reales de prueba y en ninguna de las instancias se llegó a la optimalidad. Sin embargo, la efectividad de las desigualdades válidas se evidenció significativamente al obtener reducciones de más del 50 % en la brecha de optimalidad en todos los casos. La mejor estrategia para un mejor desempeño computacional fue $CMC + BDC3$.

Para el modelo CMD también se experimentó con varias familias de desigualdades válidas y cotas inferiores. Se consideraron las mismas 11 instancias de prueba que en CMC y se consideraron 4 escenarios para la implementación computacional. Se planteó comparar el modelo CMC frente al modelo CMD incluyendo varias familias de desigualdades válidas y cotas inferiores. En todas las instancias y escenarios se llegó a la optimalidad. El escenario CMD tuvo un excelente desempeño, incluso

mejor que $CMD + BDD1$ y $CMD + BDD2$ donde se exploró una menor cantidad de nodos y se tuvo un menor tiempo de ejecución. El excelente desempeño de CMD se debe a que el valor de la función objetivo de la relajación lineal es cercana al valor óptimo del modelo entero. Además, se pudo observar que entre más grande es la instancia, la inclusión de las desigualdades válidas y cotas inferiores reporta mejores resultados. Esto se puede verificar ya que usando la estrategia $CMD + BDD3$ se reportan menores tiempos de ejecución y menor número de nodos explorados en el proceso de optimización.

En conclusión, se puede notar que la inclusión de desigualdades válidas y cotas inferiores que se reportaron en este trabajo son de amplia utilidad para abordar el problema MPCM. Además, $CMD + BDD3$ es la formulación que mejor desempeño mostró.

Finalmente se mencionarán algunos de los trabajos que se pueden realizar a futuro:

- Estudiar a fondo desigualdades válidas asociadas al poliedro de cada una de las formulaciones.
- Estudiar detenidamente los efectos de los parámetros de configuración de Gurobi, tales como desactivar o activar familias de determinados planos cortantes.
- Extender el estudio de MPCM realizado para el Sistema de Rentas Internas del Ecuador a otras instituciones que ofrezcan servicios de atención al cliente en ventanillas.

Bibliografía

[1] Aksin, Z., Armony, M., y Mehrotra, V. The modern call center: A multi-disciplinary perspective on operations management research. *Production and operations management*, 16(6):665–688, 2007.

[2] Arisha, A. y Young, P. Intelligent simulation-based lot scheduling of photolithography toolsets in a wafer fabrication facility. En *Proceedings of the 2004 Winter Simulation Conference, 2004.*, volume 2, págs. 1935–1942. IEEE, 2004.

[3] Avramidis, A. N., Chan, W., Gendreau, M., L'ecuyer, P., y Pisacane, O. Optimizing daily agent scheduling in a multiskill call center. *European Journal of Operational Research*, 200(3):822–832, 2010.

[4] Bland, R. G., Goldfarb, D., y Todd, M. J. The ellipsoid method: A survey. *Operations research*, 29(6):1039–1091, 1981.

[5] Cakici, E. y Mason, S. Parallel machine scheduling subject to auxiliary resource constraints. *Production Planning and Control*, 18(3):217–225, 2007.

[6] Chuzhoy, J. y Codenotti, P. Resource minimization job scheduling. En *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, págs. 70–83. Springer, 2009.

[7] Cieliebak, M., Erlebach, T., Hennecke, F., Weber, B., y Widmayer, P. Scheduling with release times and deadlines on a minimum number of machines. En *Exploring new frontiers of theoretical informatics*, págs. 209–222. Springer, 2004.

[8] Dantzig, G., Fulkerson, R., y Johnson, S. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[9] Desrosiers, J., Dumas, Y., Solomon, M. M., y Soumis, F. Time constrained routing and scheduling. *Handbooks in operations research and management science*, 8:35–139, 1995.

[10] Gomory, R. An algorithm for the mixed integer problem. Technical report, RAND CORP SANTA MONICA CA, 1960.

[11] Gomory, R. E. An algorithm for integer solutions to linear programs. princeton ibm mathematics research project. *Techn. Report,(1)*, 1958.

[12] Karmarkar, N. A new polynomial-time algorithm for linear programming. En *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, págs. 302–311, 1984.

[13] Keha, A. B., Khowala, K., y Fowler, J. W. Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, 56(1):357–367, 2009.

[14] Korte, B. y Vygen, J. *Combinatorial optimization: theory and algorithms*. Springer Science & Business Media, 2009.

[15] Lawler, E. L. y Wood, D. E. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

[16] Lee, Y. y Sherali, H. D. Unrelated machine scheduling with time-window and machine downtime constraints: An application to a naval battle-group problem. *Annals of Operations Research*, 50(1):339–365, 1994.

[17] Lu, X., Wu, C., Yang, X., Zhang, M., y Zheng, Y. Adapted water wave optimization for integrated bank customer service representative scheduling. *International Journal of Production Research*, págs. 1–16, 2021.

[18] Mitchell, J. E. Branch-and-cut algorithms for combinatorial optimization problems. *Handbook of applied optimization*, 1:65–77, 2002.

[19] Morgan, G. Imágenes de la organización (rama, trad.). *México, DF: Alfaomega*, 1991.

[20] Murty, U. y Bondy, A. Graph theory (graduate texts in mathematics 244), 2008.

[21] Nash, J. C. The (dantzig) simplex method for linear programming. *Computing in Science & Engineering*, 2(1):29–31, 2000.

[22] Nemhauser, G. y Savelsbergh, M. W. A cutting plane algorithm for the single machine scheduling problem with release times. En *Combinatorial optimization*, págs. 63–83. Springer, 1992.

[23] Optimization, I Gurobi and others. Gurobi optimizer reference manual, 2018. *URL http://www. gurobi. com*, 2018.

[24] Osorio-Valenzuela, L., Pereira, J., Quezada, F., y Vásquez, Ó. C. Minimizing the number of machines with limited workload capacity for scheduling jobs with interval constraints. *Applied Mathematical Modelling*, 74:512–527, 2019.

[25] Pinedo, M. *Scheduling*, volume 29. Springer, 2012.

[26] Unlu, Y. y Mason, S. J. Evaluation of mixed integer programming formulations for non-preemptive parallel machine scheduling problems. *Computers & Industrial Engineering*, 58(4):785–800, 2010.

[27] Yu, G. y Zhang, G. Scheduling with a minimum number of machines. *Operations Research Letters*, 37(2):97–101, 2009.

[28] Zheng, Y.-J. Water wave optimization: a new nature-inspired metaheuristic. *Computers & Operations Research*, 55:1–11, 2015.

Anexos

Código 1. Librerías.

```
from time import perf_counter
from   gurobipy   import *
import numpy  as np
import itertools
import pandas as pd
import math as math
import xlsxwriter
import networkx as nx
import matplotlib.pyplot as plt
```

Código 2. Lectura de la base de datos.

```
base = pd.read_csv("Instancia.csv")
base = base.iloc[:]
hora_llegada=(np.array(base)[:,1])
minuto_llegada=(np.array(base)[:,2])
tiempo_atencion=(np.array(base)[:,3])
```

Código 3. Extracción de nformación inicial para el modelo *CMC*.

```
n = (np.array(base)[:,0]).size
N=[]
for i in range(1,n+1):
    N.append(i)
o=n+1
d=n+2
V=N[:]
V.append(o)
V.append(d)
t_espera=20/60
a=[]
t=[]
for i in N:
    a.append(hora_llegada[i-1]+(minuto_llegada[i-1])/60)
    t.append(tiempo_atencion[i-1]/3600)
```

```
b=[]
for i in N:
    b.append(a[i-1]+t_espera)
k=25
```

Código 4. Construcción de conjuntos de arcos del modelo *CMC*

```
I = []
for i in N:
    for j in N:
        if a[i-1]+t[i-1] <= b[j-1]:
            I.append( (i,j) )
A = []
for i in N:
    for j in N:
        if a[i-1]+t[i-1] <= b[j-1]:
            A.append( (i,j) )
    A.append( (o,i) )
    A.append( (i,d) )
```

Código 5. Constante adecuada para la restricción 3.5 del modelo *CMC*

```
M = {}
for i,j in I:
    M[(i,j)] = max( b[i-1]+t[i-1]-a[j-1] , 0)
bt=[]
for i in N:
    bt.append(b[i-1])
M1= math.floor( max(bt))-math.floor( min(a))
```

Código 6. Implementación de *CMC*

```
m1 = Model()
x, T, y1, y2, z = {}, {}, {}, {}, {}

x       = m1.addVars( A, vtype = GRB.BINARY     , name='x'
            )
```

```
T       = m1.addVars( N, vtype = GRB.CONTINUOUS, name='T' , lb = 0  )
y1      = m1.addVars( N, vtype = GRB.INTEGER   , name='y1', lb = 0  )
y2      = m1.addVars( N, vtype = GRB.INTEGER   , name='y2', lb = 0  )
z       = m1.addVars( I, vtype = GRB.BINARY    , name='z'
             )
m1.update()
for i,j in A:
    x[i,j].BranchPriority = 4;

for i in N:
    y1[i].BranchPriority = 1;

for i in N:
    y2[i].BranchPriority = 1;

for i,j in I:
    z[i,j].BranchPriority = 1;
m1.setObjective(   x.sum(o,'*')   + ( quicksum( y2[i]-y1[i] for i in N ) ) + quicksum( z[i,j] for i,j in I),
                GRB.MINIMIZE);
m1.addConstr( (  (quicksum( x[o,i] for i in N)) + ( quicksum( y2[i]-y1[i] for i in N ) )
              +quicksum( z[i,j] for i,j in I)>= math.ceil((
                 quicksum(t[i-1] for i in N)).getValue())  ),
                 name="teo_1");
m1.addConstrs( ( x.sum(i,'*') == 1 for i in N), name="res_2");
m1.addConstr(    (quicksum( x[o,i] for i in N)) <= k          , name="res_3");
m1.addConstrs( ( x.sum('*',i) - x.sum(i,'*')  == 0  for i in N) , name="res_4");
m1.addConstrs( (T[i]+t[i-1]-T[j] <= M[i,j]*(1-x[i,j]) for i,j in I) , name="res_5");
m1.addConstrs( ( y2[i] >= T[i] + t[i-1] - 1  for i in N) , name="res_6");
m1.addConstrs( ( y1[i] <= T[i]    for i in N) , name="res_7");
m1.addConstrs( ( M1*z[i,j]  >= (y1[j]-y2[i]) - M1*(1-x[i,j]) for
```

```
    i,j in I) , name="res_8");
m1.addConstrs( ( a[i-1] <= T[i]  for i in N) , name="res_9_1");
m1.addConstrs( ( T[i] <= b[i-1]  for i in N) , name="res_9_2");
```

Código 7. Desigualdades válidas para *CMC*

```
m1.addConstrs( (x[i,j] + x[j,i] <= 1 for i in V for j in V if (i
   ,j) in I if (j,i) in I) , name="res_10");
m1.addConstr( ( quicksum( y2[i]-y1[i] for i in N ) )>= 0, name="
   res_0");
m1.addConstrs( ( z[i,j] <= x[i,j] for i,j in I) , name="res_12")
   ;
for i in N:
    if math.floor( a[i-1])== math.floor( b[i-1]+t[i-1]):
        y1[i].lb=math.floor( a[i-1]);
        y1[i].ub=math.floor( a[i-1]);
        y2[i].lb=math.floor( a[i-1]);
        y2[i].ub=math.floor( a[i-1]);
for i,j in I:
    if math.floor( a[i-1])== math.floor(b[j-1]):
        z[i,j].lb=0;
        z[i,j].ub=0;
m1.addConstr( (  (quicksum( x[o,i] for i in N)) + ( quicksum( y2
   [i]-y1[i] for i in N ) )
              +quicksum( z[i,j] for i,j in I)>= math.ceil((
                 quicksum(t[i-1] for i in N)).getValue())  ),
                 name="teo_1");
p_set=[]
for i in range(math.floor( min(a)),math.ceil( max(bt))):
    p_set.append(i)
Jp={}
for p in p_set:
    Jp[p]=[]
    for i in N:
        if p<=a[i-1]<bt[i-1]<p+1:
            Jp[p].append(i-1)
tp={}
for p in p_set:
    tp[p]=[]
```

```
    s=0
    for i in Jp[p]:
        s=s+t[i]
    tp[p].append(math.ceil(s))
t_max_aux=[]
for p in p_set:
    for i in tp[p]:
        t_max_aux.append(i)
t_max=max(t_max_aux)
arcos_periodo={}
for p in p_set:
    arcos_periodo[p]=[]
    sum_arcos=0
    for j in Jp[p]:
        for i in V:
            if i not in Jp[p] and (i,j) in A:
                sum_arcos=sum_arcos+x[i,j]
    arcos_periodo[p]=sum_arcos
m1.addConstrs( (arcos_periodo[p] >=  tp[p][0] for p in p_set),
   name="teo_6");
m1.addConstr( (  (quicksum( x[o,i] for i in N)) >=  t_max ),
   name="teo_7");
```

Código 8. Optimización de *CMC*

```
m1.params.TimeLimit= 3600
m1.params.MIPFocus = 2
m1.optimize()
```

Código 9. Construcción de conjuntos de arcos en los conjuntos S.

```
arcos_s={}
count=0
for s in S:
    arcos_s[s]=[]
    for (j,k) in A:
        if j not in S[s] and k in S[s]:
            arcos_s[s].append((j,k))
```

Código 10. Función Callback para el algoritmo tipo Branch & Cut.

```
def mycallback(model,where):
    contador_callback=0
    if where == GRB.Callback.MIPNODE:
        status = model.cbGet(GRB.Callback.MIPNODE_STATUS)
        if status == GRB.OPTIMAL:
            x_= model.cbGetNodeRel(model._x)
            for s in S:
                verificador_callback=0
                corte=0
                for (i,j) in arcos_s[s]:
                    verificador_callback=verificador_callback+x_
                       [i,j]
                    corte=corte+model._x[i,j]
                if verificador_callback<2:
                    model.cbCut(corte>=2)
                    contador_callback=contador_callback+1
```

Código 11. Modificación de la configuración por defecto de Gurobi para implementación del algoritmo tipo Branch & Cut.

```
m1.Params.LazyConstraints = 0
m1.Params.PreCrush = 1
m1.Params.Cuts=0
m1.params.TimeLimit= 1800
m1.params.MIPFocus = 2
m1.Params.MIPGap = 0.1
m1.optimize()
```

Código 12. Algoritmo para asignación de clientes a ventanillas en el modelo *CMC*.

```
vx = m1.getAttr('x', x)
vt = m1.getAttr('x', T)
for i,j in A:
    if vx[i,j] > 0.9:
        print('{} -> {}: {}'.format(i, j, round(vx[i,j],4)))
def siguiente(i,V,vx,A):
    for i1 in V:
```

```
        if (i,i1) in A and vx[i,i1]>0.9:
            return(i1)
Primero_de_cada_ruta = []
for j in N:
    if vx[o,j] > 0.9:
        Primero_de_cada_ruta.append(j)
Resultados={}
ruta = 1
decimales = 10
cont=1
for prim in Primero_de_cada_ruta:
    p = prim
    print("VENTANILLA NUMERO {}".format(ruta))
    print('{:10} {:20} {:22} {:20} {:20} {:20}'.format("Cliente"
        ,"Hora de llegada(a_i)","Duraci n tr mite(t_i)","Inicio
        de Atenci n(T_i)","Fin de atenci n", "Error de
        asignaci n"))
    s=0
    while p != d:
        solapamiento=vt[p]-s
        error= 'sin error'
        if solapamiento<-0.0000000001:
            error='con error'
        s=vt[p]+t[p-1]
        print('{:10} {:20} {:22} {:20} {:20} {:20}'.format(p,
            round(a[p-1],decimales),round(t[p-1],decimales),round(
            vt[p],decimales),round(vt[p]+t[p-1],decimales),error))
        Resultados[n,cont]=[p,round(a[p-1],decimales),round(t[p
            -1],decimales),round(vt[p],decimales),round(vt[p]+t[p
            -1],decimales),error,ruta]
        cont = cont +1
        p = siguiente(p,V,vx,A)
    ruta=ruta+1
```

Código 13. Exportación de los resultados del modelo *CMC* en una hoja de excel.

```
objeto = pd.DataFrame.from_dict(Resultados, orient='index').
    rename(columns=
{0:'Cliente',1:'Hora de llegada(a_i)',2:'Duraci n tr mite(t_i)
```

```
    ',3:'Inicio de Atenci n(T_i)',4:'Fin de atenci n', 5:'Error'
    ,6:'Numero de Ventanilla'})
objeto.to_excel('Ventanas_Instancia_Junio1_5.xlsx', sheet_name='
    resultados')
```

Código 14. Adaptación de la información inicial al modelo *CMD*

```
n = (np.array(base)[:,0]).size
N=[]
for i in range(0,n):
    N.append(i)
t_espera = 20
hora_llegada=(np.array(base)[:,1])
minuto_llegada=(np.array(base)[:,2])
a=[]
for i in N:
    a.append(hora_llegada[i]+(minuto_llegada[i])/60)
tiempo_atencion=(np.array(base)[:,3])
t=[]
for i in N:
    t.append(tiempo_atencion[i]/3600)
K=25
J=[]
for j in range(0,t_espera+1):
    J.append( j/60)
```

Código 15. Construcción del conjunto de eventos Q y el conjunto de nodos V

```
T_i={}
for i in N:
    T_i[i]=[]
    for j in J:
        T_i[i].append((a[i]+j,a[i]+j+t[i]))
T=[]
for i in N:
    for (h,k) in T_i[i]:
        T.append((h,k))

```

```
H_prima=[]
for i,j in T:
    H_prima.append(i)
    H_prima.append(j)
H_p=set(H_prima)
A_min=[]
for i in N:
    A_min.append(a[i])
a_min=math.floor( min(A_min))
bt=[]
for i in N:
    bt.append(a[i]+t[i]+(t_espera/60))
bt_max=math.ceil(max(bt))
I=[]
for i in range(a_min,bt_max+1):
    I.append(i)
I.sort()
CI=set(I)
I_aux=I[:]
I_aux2=I[:]
I_aux.pop()
del I_aux2[0]
I_aux2.pop()
CH=H_p | CI
OH=[]
for i in CH:
    OH.append(i)
OH.sort()
OHlen=len(OH)
DH={}
for i in range(1,OHlen+1):
    DH[i]=OH[i-1]
H,valor_H= multidict(DH)
H_n=[]
for i in range(1,len(H)):
    H_n.append(i)
H_n.sort()
o=len(H)+1
d=len(H)+2
```

```
V=H[:]
V.append(o)
V.append(d)
V.sort;
```

Código 16. Construcción del conjunto de arcos A.

```
DHI={}
contador_HI=1
for i in OH:
    DHI[i]=contador_HI
    contador_HI=contador_HI+1
DHI_aux,indices= multidict(DHI)
DT={**DHI}
for i in DHI_aux:
    if i not in H_p:
        del DT[i]
DT_claves,indices_dt=multidict(DT)
DCI={**DHI}
for i in DHI_aux:
    if i not in CI:
        del DCI[i]
DCI_claves,indices_dci=multidict(DCI)
A1=[]
for i in H_n:
    A1.append((i,i+1))
A1.append((o,1))
A1.append((len(H),d))
A1.append((o,d))
A2=[]
for i in DCI_claves:
    for j in DCI_claves:
        if i<j:
            A2.append((DCI[i],DCI[j]))
CA1=tupledict()
for (i,j) in A1:
    if i==o and j==1:
        CA1[i,j]=(0,K)
    elif i==len(H) and j==d:
```

```
            CA1[i,j]=(0,K)
        elif i==o and j==d:
            CA1[i,j]=(0,K)
        elif DH[i] in I and j==i+1:
            CA1[i,j]=(1,K)
        else:
            CA1[i,j]=(0,K)
CA2=tupledict()
for (i,j) in A2:
    CA2[i,j]=(0,K)
Ti={}
for i in N:
    Ti[i]={}
    for (j,l) in T_i[i]:
        Ti[i][(DT[j],DT[l])]={}
        if j in I:
            Ti[i][(DT[j],DT[l])]=(1+math.floor(l)-math.floor(j)
                ,1)
        else:
            Ti[i][(DT[j],DT[l])]=(math.floor(l)-math.floor(j),1)
N_aux=range(len(N)+2)
Ti[len(N)]={**CA1}
Ti[len(N)+1]={**CA2}
A=[]
c=[]
u=[]
for i in N_aux:
    A.append(0)
    c.append(0)
    u.append(0)
for i in N_aux:
    A[i],c[i],u[i]=multidict(Ti[i])
```

Código 17. Implementación del modelo *CMD*.

```
m1 = Model('flujo')
SV=[]
for i in N_aux:
    SV.append('x{}'.format(i))
```

```
Variables={}
for i in N_aux:
    Variables[SV[i]]= m1.addVars(A[i], vtype = GRB.INTEGER    ,
       name='x{}'.format(i),ub=u[i]      );
m1.update()
for l in N_aux:
    for i,j in A[l]:
        Variables['{}'.format(SV[l])][(i,j)].BranchPriority = 4;
objetivo=0
for i in N_aux:
    objetivo=objetivo+Variables['{}'.format(SV[i])].prod(c[i], '
       *')
m1.setObjective(objetivo, GRB.MINIMIZE)
m1.addConstrs((Variables['{}'.format(SV[i])].sum()==1 for i in N
   ), "rest1");
sum_izq={}
sum_der={}
for i in V:
    sum_izq[i]=[]
    sum_der[i]=[]
for l in N_aux:
    for (j,k) in Ti[l]:
        sum_der[j].append((l,(j,k)))
        sum_izq[k].append((l,(j,k)))
flujo= {}
for i in V:
    flujo[i] = -K if i==o else (K if i==d else 0)
L_i=[]
L_d=[]
for i in V:
    L_i.append(0)
    L_d.append(0)
for i in V:
    L_i[i-1]=0
    L_d[i-1]=0
    for l in sum_der[i]:
        L_d[i-1]=L_d[i-1]+Variables['{}'.format(SV[l[0]])][l[1]]
    for l in sum_izq[i]:
        L_i[i-1]=L_i[i-1]+Variables['{}'.format(SV[l[0]])][l[1]]
```

```
41 m1.addConstrs( (L_i[i-1]- L_d[i-1] == flujo[i] for i in V), name
      ="flujo");
```

Código 18. Desigualdades válidas para *CMD*

```
1  m1.addConstr(( objetivo >= math.ceil((quicksum(t[i] for i in N))
       .getValue())  ), name="teorema_1");
2  Jp={}
3  for p in I_aux:
4      Jp[p]=[]
5      for i in N:
6          if p<=a[i]<bt[i]<=p+1:
7              Jp[p].append(i)
8  tp={}
9  for p in I_aux:
10     tp[p]=[]
11     s=0
12     for i in Jp[p]:
13         s=s+t[i]
14     tp[p].append(math.ceil(s))
15 t_max_aux=[]
16 for p in I_aux:
17     for i in tp[p]:
18         t_max_aux.append(i)
19 t_max=max(t_max_aux)
20 Vp={}
21 for p in I_aux:
22     Vp[p]=[]
23     for i in H:
24         if p<=DH[i]<p+1:
25             Vp[p].append(i)
26 arcos_periodo={}
27 for p in I_aux:
28     arcos_periodo[p]=[]
29     for i in N_aux:
30         for (l,k) in Ti[i]:
31             if l not in Vp[p] and k in Vp[p]:
32                 arcos_periodo[p].append((i,(l,k),c[i][l,k]))
33 arcos_periodo_p={}
```

```
for p in I_aux:
    arcos_periodo_p[p]=[]
    for i in N_aux:
        for (l,k) in Ti[i]:
            if l in H and DH[l]==p and k in Vp[p]:
                arcos_periodo_p[p].append((i,(l,k),c[i][l,k]))
suma_1_t2={}
for p in I_aux:
    sum_cota=0
    for l in arcos_periodo[p]:
        sum_cota=sum_cota+Variables['{}'.format(SV[l[0]])][l
           [1]]*l[2]
    suma_1_t2[p]=sum_cota
suma_2_t2={}
for p in I_aux:
    sum_cota=0
    for l in arcos_periodo_p[p]:
        sum_cota=sum_cota+Variables['{}'.format(SV[l[0]])][l[1]]
    suma_2_t2[p]=sum_cota
m1.addConstrs( (suma_1_t2[p]+suma_2_t2[p] >= tp[p][0] for p in
   I_aux), name="teo_2");
m1.addConstr( Variables['{}'.format(SV[len(N)])][(o,1)] >= t_max
   , name="teo_3");
m1.optimize()
```

Código 19. Algoritmo para asignación de clientes a ventanillas en el modelo *CMD*

```
contador_ventanillas=[]
vx=[]
for i in N_aux:
    vx.append(0)
for i in N_aux:
    vx[i]=m1.getAttr('x', Variables['{}'.format(SV[i])])
A_verificador=[]
N_verificador=[]
for i in N:
    for j,k in A[i]:
        if vx[i][j,k] >=0.01:
            A_verificador.append((j,k))
```

```
            N_verificador.append(j)
if len(A_verificador)==n:
    print('Asignaci n correcta')
else:
    print('Asignaci n incorrecta')
    print(len(A_verificador),n)
N_v_usados=N_verificador[:]
N_v_usados.sort()
A_usados=[]
f_usados=[]
N_=[]
for i in N_aux:
    for j,k in A[i]:
        if vx[i][j,k] >=0.01:
            A_usados.append((j,k))
            f_usados.append(vx[i][j,k])
            N_.append(j)
            N_.append(k)
N_orden=set(N_)
N_usados=[]
for i in N_orden:
    N_usados.append(i)
N_usados.sort()
A_usados_flujo=[]
f_aux= f_usados[:]
for (i,j) in A_usados:
    if i!=o and i!=len(H):
        while(f_aux[A_usados.index((i,j))]>=0.01):
            A_usados_flujo.append((i,j))
            f_aux[A_usados.index((i,j))]=f_aux[A_usados.index((i
               ,j))]-1
A_clientes=A_verificador[:]
A_clientes.sort()
ventanilla={}
num_ventanilla=1
N_v_us=N_v_usados[:]
A_us=A_usados_flujo[:]
prueba=0
for i in N_v_us:
```

```
    for (j,k) in A_clientes:
        if j==i:

            ventanilla[num_ventanilla]=[]
            ventanilla[num_ventanilla].append((j,k))
            A_clientes.remove((j,k))
            A_us.remove((j,k))
            while(k!=len(H)):
                for(l,m) in A_us:
                    if l==k:
                        if (l,m) in A_clientes:
                            ventanilla[num_ventanilla].append((l
                                ,m))
                            A_clientes.remove((l,m))
                            A_us.remove((l,m))
                            N_v_us.remove(l)
                            k=m
                            break
                        else:
                            k=m
                            A_us.remove((l,m))
                            break
            num_ventanilla=num_ventanilla+1
            break
verificador2=0
for i in ventanilla:
    verificador2=verificador2+len(ventanilla[i])
if verificador2==n:
    print('Las {} personas han sido asignadas a una ventanilla'.
        format(n))
else:
    print('Error {} {}'.format(n,verificador2))
print('y')
print('Se han usado {} ventanillas'.format(len(ventanilla)))
clientes={}
inicio_atencion={}
fin_atencion={}
for i in ventanilla:
    clientes[i]=[]
```

```
88      inicio_atencion[i]=[]
89      fin_atencion[i]=[]
90      for j,k in ventanilla[i]:
91          for l in N:
92              if (DH[j],DH[k]) in T_i[l]:
93                  inicio_atencion[i].append(DH[j])
94                  fin_atencion[i].append(DH[k])
95                  clientes[i].append(l)
96                  break
```

Código 20. Exportación de la asignación de clientes a ventanillas para el modelo *CMD* en un archivo de Excel.

```
1   decimales=3
2   Excel={}
3   for i in ventanilla:
4       print('Clientes asignados en la ventanilla {}'.format(i))
5       print('{:10} {:10} {:10} {:10} {:10} {:10} {:10}'.format("
           Cliente",   "Hora de llegada",   "Inicio de atenci n",
           'Final de atenci n',   'Duraci n del tramite',   'Tiempo
            de espera', 'ERROR'))
6       s=0
7       for j in clientes[i]:
8           solapamiento=inicio_atencion[i][clientes[i].index(j)]-s
9           error = 'sin error'
10          if solapamiento<-0.0000000001:
11              error='con error'
12          s=fin_atencion[i][clientes[i].index(j)]
13          print('{:10} {:15} {:15} {:15} {:15} {:25} {:25}'.format
               (j+1,round(a[j],decimales),round(inicio_atencion[i][
               clientes[i].index(j)],decimales),round(fin_atencion[i
               ][clientes[i].index(j)],decimales),round(t[j],
               decimales),round(inicio_atencion[i][clientes[i].index(
               j)]-a[j],decimales),error))
14          Excel[i,j]=[i,j+1,round(a[j],decimales),round(
               inicio_atencion[i][clientes[i].index(j)],decimales),
               round(fin_atencion[i][clientes[i].index(j)],decimales)
               ,round(t[j],decimales),round(inicio_atencion[i][
               clientes[i].index(j)]-a[j],decimales),error]
```