Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Optimización aplicada a la programación ferroviaria de carga

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Federico José Canay

Directores:

- Dra. Isabel Méndez-Díaz
- Dr. Juan José Miranda Bront
- Dra. Paula Zabala

Buenos Aires, 2018

AGRADECIMIENTOS

Llegó el momento de terminar esta larga etapa de mi vida, pero nunca hubiese podido llegar hasta acá sin un montón de personas que me acompaño en este camino.

En primer lugar tengo que agradecer a Agos, mi novia, que estuvo en todas, las buenas, las malas y las peores. Sin su ayuda y acompañamiento nunca hubiese llegado hasta acá.

También tengo que agradecer a mi familia, Mamá, Papá, Eze y Nacho que gracias a ellos soy quien soy y llegué hasta acá, siempre acompañándome, respetándome y apoyándome.

Gracias a mis abuelos, Ela, Osvaldo, Lolo, Suse, mis tíos, Pipa, María, Pablo, Martín y a todos mis primos, por estar siempre.

Como olvidarse de mi otra familia, de mis hermanos del alma, Salva, Fati, Maru y Flor con quienes desde siempre compartimos la vida y esta amistad que es para siempre. También a todo el "Preju" por incontables juntadas, encuentros y asados.

Gracias a todos los que hicieron esta carrera tan divertida y llevadera, a Santi, Facu, Gonzi, Emilio, Guido, Paglia, Fede, Franco, Sacha, Willy, Gus, Mongi y a todos los Willterns.

Gracias a los frikis, Alba, Ague, Graby, Fran, Facu y Pupi, por las infinitas discusiones enriquecedoras.

Gracias a Sergio y Andrea, por recibirme en su casa incontables veces y por los desayunos de los findes para seguir avanzando con la tesis.

También quiero agradecer a Irene Loiseau y Javier Marenco por tomarse el tiempo para ser jurados de mi tesis.

Por último, y no menos importante, quiero agradecer a Isabel, Paula y Juanjo por acompañarme y ayudarme en esta etapa de la carrera. Quiero destacar más allá de la gran calidad docente y como investigadores que tienen los tres, la gran calidad humana que tienen, que fue la razón de que los eligiera como directores.

Índice general

1. INTRODUCCIÓN

El transporte de cargas es uno de los servicios vitales para el desarrollo de la economía moderna. En particular el transporte ferroviario cumple un rol clave, gracias a que es uno de los métodos más económico y eficiente para el transporte de cargas. Por este motivo, es fundamental para impulsar el desarrollo económico de distintas áreas dentro de un país, ya que favorece la descentralización de la producción, la reducción de los costos y la competitividad. Además, este medio de transporte de cargas resulta más ecológico que el transporte de cargas por camiones [8].

A pesar de su bajo costo, el transporte ferroviario es uno de los medios que necesita una mayor inversión, ya que a diferencia del transporte automotor, aéreo o marítimo, es responsable de construir, operar y mantener toda la infraestructura física necesaria para su funcionamiento, como las vías, las estaciones, y las señalizaciones.

La planificación de la operación de una empresa ferroviaria de carga es una tarea sumamente compleja e involucra la resolución de múltiples problemas de decisión. Uno de estos problemas implica gestionar la infraestructura ferroviaria, la cual consiste en las vías de trenes, las estaciones y los sistemas de señalización y control. Los trenes circulan por las vías, que tendrán distintas características: trocha (ancho de la vía), peso máximo que soporta, cantidad de carriles (vías simple, dobles), etc. Las estaciones son los lugares donde los trenes cargarán y descargarán vagones. Al conjunto de estaciones y vías de tren se lo conoce como red ferroviaria. La gestión de esta infraestructura implica el mantenimiento y las decisiones de dónde y cómo invertir en nueva infraestructura.

Otro de los problemas que tiene que enfrentar una empresa de transporte ferroviario, es la gestión de los vagones y las locomotoras, también llamados material rodante. Los vagones son donde se guarda la carga a ser transportada. Las locomotoras son las encargadas de remolcar a los vagones a través de las vías y tienen diferentes cualidades como poder de tracción, velocidad, etc. Un tren está conformado por una locomotora y un conjunto de vagones que serán tanto cargados como descargados del tren en el transcurso del viaje. Gestionar el material rodante consiste en el mantenimiento y la decisión de adquirir nuevos vagones y/o locomotoras.

También es necesario contar con personal capacitado tanto para la operación como para el mantenimiento del material rodante. Al conjunto de empleados que opera un tren se los denomina tripulación. Un tren deberá contar en todo momento con una tripulación a bordo que lo opere. Estas tripulaciones operarán los trenes dentro de ciertas zonas de la red ferroviaria, a las que llamaremos secciones de tripulación. Esto se debe a reglamentaciones a niveles nacionales, municipales y sindicales; y también tienen el objetivo de que el empleado opere los trenes en las cercanías de su hogar. Al depender de múltiples reglamentaciones, la forma en que se definen estas secciones de tripulación varían ampliamente entre diferentes países e incluso dentro del mismo país, varían entre diferentes provincias.

Otro gran desafío de una empresa ferroviaria es la planificación de los itinerarios de los trenes. Estos consisten en el recorrido del tren y sus horarios, con qué frecuencia se repetirá el servicio y en qué momento cargará o descargará los vagones.

En las redes ferroviarias, gran cantidad de las vías son simples, lo que significa que solo puede circular un tren a la vez en un punto dado de esa vía. Para poder utilizar esa vía en ambos sentidos existen puntos de sobrepaso, donde la vía se vuelve doble y pueden

circular dos trenes en direcciones contrarias al mismo tiempo. Esta característica de la red agrega un nuevo punto a la planificación, ya que un tren deberá esperar en el punto de sobrepaso a que el otro tren circulando en dirección contraria por la vía simple llegue al punto de sobrepaso para continuar con su marcha. No tener en cuenta esta característica al momento de la planificación, generará que trenes deban esperar mucho tiempo en puntos de sobrepaso.

Por último, se deberá definir el esquema de precios que la empresa cobrará a sus clientes, tanto los montos como la forma en que se cobrará, pudiendo ser un precio fijo, dependiente de la cantidad de carga, dependiente de la distancia, de la urgencia del transporte o una combinación de todas estas, entre muchas otras opciones posibles.

Todas estas problemáticas que debe afrontar una empresa ferroviaria de carga pueden planificarse teniendo en cuenta diferentes horizontes de tiempo.

La planificación a largo plazo, también conocida como planificación estratégica, comprende en general períodos de múltiples años. Al trabajar sobre un período tan largo de tiempo, se debe pasar por alto diversos detalles para simplificar ampliamente múltiples decisiones. Estos detalles que se pasan por alto tienen un impacto mínimo. Ejemplos de planificaciones estratégicas en el ámbito ferroviario son las grandes inversiones en la construcción de nuevas estaciones, la instalación de vías o la compra de nuevo material rodante, entre otros.

La planificación a mediano plazo, o planificación táctica, corresponde en general a períodos de 3 a 6 meses. A este nivel se logran considerar más detalles que a nivel estratégico, así como se pueden contemplar variables estacionales. Un ejemplo es el aumento de la frecuencia de trenes de cargas a zonas agrícolas en la época de cosecha.

Por último, la planificación a corto plazo, o planificación operativa, es la que corresponde a un horizonte de una semana. A este nivel se consideran todos los detalles necesarios para la correcta operatoria de la empresa. Un ejemplo es la sincronización de los trenes en aquellas vías que por ser únicas deben ser utilizadas por los trenes de ambos sentidos.

Un mismo problema puede ser trabajado a nivel estratégico, táctico u operativo, por lo que las características del problema variarán en función del nivel en que sea tratado. Por ejemplo, el problema de la asignación de tripulantes a diferentes trenes, donde a nivel estratégico, se puede decidir aumentar la plantilla de trabajadores para lograr incrementar la frecuencia de los trenes. A nivel táctico se puede decidir contratar temporalmente más tripulantes para la época de cosechas o de alta actividad y, por último, a nivel operativo se deberá decidir cada día qué tripulación se le asigna a cada tren.

También estos problemas variarán significativamente de acuerdo a la empresa o país en el que se opere. Diferentes empresas pueden tener distintas políticas tanto laborales, operativas o de negocios que impliquen ciertas variaciones a estos problemas. Asimismo las reglamentaciones de cada país sobre la industria ferroviaria también afectarán a la formulación de estos problemas.

Este gran espectro de complejos problemas de decisión vuelve al transporte ferroviario de cargas un candidato ideal para la aplicación de herramientas y metodologías de la investigación operativa.

Aunque el transporte ferroviario tiene más de dos siglos de existencia, no hubo mayores cambios durante este tiempo en la forma de crear planes detallados para su operación, que organice el transporte de cargas en la red ferroviaria. En cambio, las demás industrias de transporte, en especial el transporte automotor y aéreo, han tenido una fuerte inversión en métodos computacionales para optimizar sus operaciones, a nivel estratégico, táctico u

operativo. Recién en los últimos años se ha incrementado el desarrollo de investigaciones en optimización de las operaciones de la industria ferroviaria, pero todavía se encuentra muy relegada en comparación con el resto de los métodos de transporte [1].

Ahuja et al. [1] seleccionan y desarrollan seis de los problemas más relevantes de la operación ferroviaria.

- *Blocking Problem*: Una empresa ferroviaria suele transportar millones de vagones desde su origen a su destino. Para reducir el costo de manejar estos paquetes individualmente durante su traslado, los mismos se clasifican (o agrupan) creando bloques que se trasladarán juntos, facilitando así su operación. Un bloque es un conjunto de vagones que deben ser transportados juntos desde una estación de origen a una de destino. El *blocking problem* consiste en definir cómo deben agruparse esos vagones en bloques para minimizar el costo de transporte y de operación.

- *Yard Location Problem*: Un *yard* es una estación dentro de la red ferroviaria en la cual se pueden reclasificar los paquetes de un bloque en nuevos bloques y/o intercambiarlos de tren. El problema consiste en definir la mejor ubicación de *yards* en la red ferroviaria, tanto en cantidad como en ubicación. Este problema está sumamente interrelacionado con el *blocking problem*, ya que la ubicación de los *yard* tiene un gran impacto en la generación de los bloques.

- *Train Scheduling Problem*: Una vez resuelto el *blocking problem*, el *train scheduling problem* debe definir tanto el recorrido de los trenes, como su frecuencia, días de operación y asignar trenes a los bloques definidos en la solución del *blocking problem* para trasladarlos desde su origen a su destino buscando minimizar los costos operativos de los trenes y el costo de transportar los bloques.

- *Locomotive Scheduling Problem*: Consiste en asignar diferentes tipos de locomotoras a los trenes programados en el *train scheduling problem*. Las locomotoras asignadas deben proveer el poder de tracción necesario así como satisfacer una serie de exigencias de mantenimiento, combustible, etc.

- *Train Dispatching Problem*: Este problema consiste en definir los movimientos y horarios detallados de los trenes mientras circulan por la red ferroviaria. Estos deben cumplir con una serie de restricciones operacionales (distancia mínima entre dos trenes, velocidades máximas, etc.) mientras busca minimizar la desviación del plan inicial creado en el *train scheduling problem* y el retraso total de los trenes.

- *Crew Scheduling Problem*: Este problema consiste en asignar una tripulación para cada tren en cada sección de tripulación que atraviesa durante su recorrido. Un tren puede atravesar múltiples secciones de tripulación durante su recorrido, por lo cual necesitará una tripulación diferente para cada una de estas secciones. Esta asignación debe cumplir las normas impuestas por los sindicatos, así como debe buscar minimizar los costos de tripulación y las demoras generadas por la no disponibilidad de tripulaciones.

1.1. Objetivos

En el año 2011, el *Institute for Operations Research and the Management Sciences-Railway Applications Section* (INFORMS-RAS) organizó una competencia abierta con

el fin de promover la investigación y la aplicación de la investigación operativa en el transporte ferroviario [5]. Para esta competencia, INFORMS-RAS diseñó un problema, al que denominó *Train Design Optimization* (TDO). Este problema consiste en el diseño de un plan detallado del movimiento de bloques a nivel táctico desde cierto origen hasta un destino final. Este plan debe cumplir con una serie de restricciones de la infraestructura ferroviaria, así como debe minimizar los diferentes costos.

En particular, TDO se encarga de resolver los problemas de *Train Scheduling* y *Crew Scheduling* a nivel táctico. Asume como resuelto los problemas de *Blocking* y *Yard Location* y los toma como *input*. Además, el TDO supone que en este nivel de detalle el material rodante es idéntico e ilimitado y que no es importante la sincronización de los trenes.

Para evaluar la calidad de las resoluciones propuestas, se utilizó los datos correspondientes a dos instancias presentadas por INFORMS-RAS como parte de la competencia.

El objetivo de esta tesis es proponer nuevos métodos para la resolución del TDO y compararlos con el estado del arte.

También se busca investigar y resolver nuevas extensiones al TDO teniendo en cuenta variantes del problema no consideradas en el TDO original.

1.2. Estructura de la tesis

A continuación presentaremos la estructura de esta tesis que contiene 8 capítulos. En el capítulo 2 se define el problema TDO, describiendo tanto las restricciones como los costos asociados a la operación de carga ferroviaria.

En el capítulo 3 se presenta el estado del arte y diferentes métodos usados para la resolución del TDO y se analizan las características de las instancias que se utilizan en este trabajo.

El capítulo 4 desarrolla en profundidad el método presentado por Jin et al. [6] y se presentan mejoras sobre el mismo.

Luego, el capítulo 5 desarrolla dos nuevos métodos para la resolución del TDO, mientras que el capítulo 6 exhibe los resultados de la experimentación de los métodos expuestos en los capítulos 4 y 5.

A continuación, en el capitulo 7 se presentan nuevos escenarios no tenidos en cuenta en la definición original del TDO, se desarrollan métodos para resolverlos y se exhibe la experimentación que los compara. Por último, el capítulo 8 presenta las conclusiones del trabajo y ofrece posibles direcciones para trabajos futuros.

2. DEFINICIÓN DEL PROBLEMA

INFORMARS-RAS define al *Train Design Optimization Problem* [5] de la siguiente manera:

2.1. Train Design Optimization Problem

Dada una red ferroviaria y un conjunto de bloques que deben ser transportados, el *Train Design Optimization Problem* (TDO), consiste en decidir el recorrido de los trenes (lo que llamaremos rutas) y definir para cada bloque su itinerario. Este itinerario del bloque consiste en definir por qué camino viajará el bloque entre su origen y su destino y en qué tren recorrerá cada parte de ese camino. Al conjunto de estos itinerarios de bloque lo llamaremos *Block to Train Assignent* (BTA). La solución propuesta deberá cumplir con todas las restricciones operacionales y buscar minimizar los costos de operación. Este problema se analizará a nivel táctico, por lo que no se tendrá en cuenta el horario detallado de cada tren.

El *input* del problema constará del grafo de la red física $G(V, E)$, donde los nodos serán estaciones y los ejes vías de tren. La estación será representada por un identificador numérico único. Las vías se representarán como un par (estación, estación) y tendrán diversas características: la distancia entre las dos estaciones, la cantidad de trenes que podrán circular por esta vía y el peso y longitud máxima de esos trenes. Los bloques tendrán un identificador numérico, y varias características: la estación de origen y de destino, el número de vagones que componen el bloque y el peso y la longitud total del bloque. Por último, las secciones de tripulación también serán definidas como un par (estación, estación), donde estas estaciones representarán las estaciones cabecera de esta sección. La sección de tripulación estará compuesta por todas las estaciones y vías dentro del camino mínimo entre sus dos estaciones de cabecera.

Los costos operativos que se buscan minimizar en el TDO son:

- Costo fijo por poner un tren en movimiento: Es el costo fijo en que debe incurrir una compañía ferroviaria para poner un tren en funcionamiento, que incluye el costo de mantenimiento de la locomotora, el costo del personal necesario para el mantenimiento y operación del tren, el costo de la infraestructura, tanto las vías, estaciones, como talleres mecánicos y cocheras, entre otros.

- Costo de viaje de los trenes: Este costo se refiere al gasto de combustible, el desgaste por el uso tanto del tren como de las vías, el costo del personal a cargo de la operación del tren, etc. Este costo aumenta linealmente en función de la distancia recorrida por el tren.

- Costo de viaje de los vagones: Es el costo generado por el desgaste de los vagones producto del viaje recorrido y aumenta linealmente en función de la distancia recorrida por el vagón.

- Costo de los *block swaps*: Llamaremos *block swap* a la operación de transferir un bloque entre trenes. El *block swap* está definido en el evento que ocurre sobre el bloque y solo ocurre cuando el bloque pasa de un tren a otro y no cuando un tren lo

recoge de su estación inicial o cuando lo descarga en su estación final. Al detenerse un tren en una estación intermedia para cargar o descargar bloques, incurre tanto en un costo directo (como puede ser el costo del personal necesario en la estación para realizar la carga y descarga), como en un costo indirecto, producto de la demora del tren en la estación.

- Costo de los *work events*: Llamaremos *work event* al evento que ocurre cuando un tren se detiene en una estación intermedia para incorporar o descargar un bloque. Aunque un tren se detenga en una estación y ocurran múltiples *block swaps* (se cargan o descargan múltiples bloques), solo sucederá un *work event*. Además, puede ocurrir el caso en que un tren se detenga en un estación a cargar un bloque (genera un *work event*) sin que ocurra ningún *block swap*. Esto sucede cuando un tren se detiene en una estación intermedia a levantar un bloque, pero esa estación es el origen del bloque, por ende, no se genera *block swap*, ya que estos ocurren solo cuando un bloque cambia de tren y no en sus estaciones de origen o llegada. Al igual que los *block swaps*, los *work events* requieren tanto de recursos extra como de tiempo para ser llevados a cabo, lo que conlleva un gasto para la empresa.

- Costo de desbalance de tripulación: El desbalance de tripulación es la diferencia absoluta entre la cantidad de trenes que circularon en cada dirección (ida y vuelta) en una determinada sección de tripulación. Si las secciones de tripulación están dadas entre dos estaciones A y B, el desbalance de tripulación se calcula como la diferencia absoluta entre la cantidad de trenes yendo de A a B y la cantidad de trenes yendo de B a A. Al final del día la tripulación debe volver a su punto de origen por normas sindicales, por lo que si no se lograra regresarlos tripulando un tren, se les deberá pagar el costo del transporte hasta su lugar de origen. En este trabajo, ese costo supone un monto fijo por tripulación.

- Costo de desbalance de trenes: El desbalance de trenes consiste en la diferencia absoluta entre los trenes que se inician en una estación y los trenes que terminan en esa misma estación. Por ejemplo, si de una estación A inician cinco trenes y terminan en ella solo tres trenes, el desbalance en la estación A será de dos trenes. El TDO busca construir un planificación que se repita diariamente, para esto al comenzar el día, el material rodante deberá encontrarse siempre en el mismo lugar. Si al finalizar el día no hay la misma cantidad de trenes que la que partió de esta estación al principio del día, habrá que transportar la cantidad faltante hasta allí, lo cual acarrea un costo extra tanto en combustible, como en personal y capacidad de la red. En este trabajo se considera el costo asociado a reubicar un tren como fijo.

- Multa por bloques no entregados: Al no lograr transportar el pedido de un cliente, la empresa se verá perjudicada tanto directamente (al no cobrar su servicio o al tener que abonar un resarcimiento por el no cumplimiento del servicio), así como también en forma indirecta, al obtener una consideración negativa que podría afectar la imagen de la empresa y desalentar a futuros clientes. En este trabajo se supone que esta penalización está linealmente relacionada con el tamaño de los bloques (cantidad de vagones).

Además, el problema TDO contiene las siguientes restricciones operacionales:

- Cantidad de bloques por tren: Asignar una gran cantidad de bloques a un tren puede resultar en una mayor cantidad de paradas intermedias que afectarían la eficiencia operacional. Por este motivo, se restringe la cantidad de bloques que puede llevar un tren en cada momento de su recorrido a un número máximo.

- *Blocks swaps* por bloque: Como ya se explicó, los *block swaps* conllevan asociado un aumento tanto de costo como de demora, lo que vuelve desaconsejable que un bloque incurra en múltiples *block swaps*. Por esto, se limita a que un bloque no pueda ser transferido más de una cierta cantidad de veces.

- *Work event* por tren: Al igual que los *block swaps*, los *work events* implican costos tanto a nivel de recursos como a nivel temporal, lo que genera que se restrinja a que un tren no pueda realizar más de una cierta cantidad de *work events* en su recorrido.

- Longitud y peso máximo de un tren para circular por una vía: En función de las características tanto geográficas como físicas de cada vía, ésta podrá soportar trenes de a lo sumo cierto peso y cierta longitud. Cabe aclarar que la longitud y el peso de un tren depende de los bloques que traslade, ya que cada bloque tiene un peso y una longitud diferente.

- Número máximo de trenes que circulan por una vía: Al analizar el problema a nivel táctico, no se define el horario exacto en el que un tren pasará por una vía. En vez de sincronizar los trenes para asegurarse que no haya colisiones, se estima una cantidad "segura" de trenes que pueden circular por una vía en un día. De este modo, se limita a que circulen a lo sumo ese número de trenes por la vía independientemente de la dirección.

- Secciones de tripulación: Las tripulaciones, por normas sindicales, solo pueden viajar en ciertas secciones de tripulación predeterminadas. Como cada tren tiene que tener asignada una tripulación en todo momento, todos los trenes deben originarse en el inicio de una sección de tripulación y terminar en el final de una sección de tripulación. La restricción de secciones de tripulación es una de las más cambiantes y difíciles de modelar en la práctica, ya que diferentes secciones están dirigidas por diferentes sindicatos con sus respectivas normas, generando una muy variada cantidad de normativas y restricciones. En este trabajo se simplifica esta restricción suponiendo que las secciones de tripulación están dadas entre dos estaciones A y B y que el recorrido está dado por el camino mínimo entre ambas.

3. ESTADO DEL ARTE

A la competencia organizada por INFORMS-RAS en 2011 [5] para resolver el Train Design Optimization Problem se presentaron 4 equipos. La competencia fue ganada por el equipo Koppa, seguido por OR@UNIMI y en tercer lugar NCKU. El equipo RAIL-OPT obtuvo una mención especial.

Al momento de determinar el orden de los ganadores de la competencia, el jurado de INFORMS-RAS tuvo en cuenta la calidad, el tiempo y la escalabilidad de la solución. También consideró la calidad del reporte entregado y la innovación y elegancia de la formulación y de la solución.

En la siguiente tabla exponemos el costo total de las mejores soluciones presentadas por cada equipo para cada DataSet. Como se busca minimizar el costo, a menor valor la solución es mejor.

DataSet	**Koppa**	**OR@UNIMI**	**NCKU**	**RAIL-OPT**
DataSet1	2.043.471$	2.212.163$	2.018.646$	2.069.752$
DataSet2	3.187.999$	3.538.924$	3.152.018$	3.240.177$

A continuación se analizan las instancias provistas por la competencia, y luego se resumen los trabajos presentados por los equipos Koppa, OR@UNIMI y NCKU. El trabajo presentado por el equipo RAIL-OPT se desarrollará con mayor detalle en el siguiente capítulo ya que trabajaremos y haremos nuevas propuestas sobre este método.

3.1. Instancias

INFORMS-RAS proveyó dos instancias para la competencia [5] que denominaron DataSet1 y DataSet2. Sobre estas instancias se evaluaron las metodologías presentados por cada uno de los equipos participantes.

Las características de las instancias son las siguientes:

	DataSet1	**DataSet2**
$N°$ de estaciones	94	221
$N°$ de bloques	239	369
$N°$ de vías	134	294
$N°$ de secciones de tripulación	154	154

En las siguientes secciones describiremos algunas características que encontramos al analizar estas instancias que influyen en los resultados obtenidos.

3.1.1. Secciones de tripulación ambiguas

La definición del problema puede acarrear ambigüedad al momento de decidir cuál es el camino mínimo de una sección de tripulación, ya que pueden existir múltiples caminos mínimos entre dos estaciones.

En el DataSet1 los caminos mínimos entre las cabeceras de las secciones de tripulación son siempre únicos. Por el contrario, en el DataSet2 existen secciones de tripulación con más de un camino mínimo y esto afecta fuertemente las posibles soluciones.

En particular, de las 154 secciones de tripulación del DataSet2, 20 tienen más de un camino mínimo. De estas 20 secciones de tripulación, 18 tienen dos posibles caminos mínimos, una tiene 3 y la última tiene 4.

En esta instancia hay 5 bloques (block ids: 285, 545, 548, 592, 596) que son imposibles de mandar independientemente de la decisión que se tome sobre estas secciones de tripulación y generan una penalización de $420.000.

Pero hay otros 5 bloques (block ids: 1, 8, 11, 333, 354) que pueden o no ser posibles de enviar en función de la decisión que se tome sobre algunas de estas 20 secciones de tripulación. Estos bloques conllevan un costo extra de $395.000 si no son enviados.

El conjunto de bloques {1, 8, 11} depende de que la sección de tripulación (5899,6872) tome un cierto camino entre sus dos posibles caminos.

En el caso de los bloques {333, 354} depende de que al menos una de las 4 secciones de tripulación (2031,6991), (6991,11018), (6991,5879), (2228,11018), (2031,6991) tome un camino en particular entre los dos posibles caminos de cada uno.

Para evitar esta situación, para cada sección de tripulación con más de un camino mínimo posible, seleccionamos uno de ellos garantizando que el conjunto de los bloques que pueden ser llevados o no dependiendo de esta decisión puedan ser llevados. Los caminos que elegimos están expuestos en el apéndice C. De aquí en adelante se tomarán estos como los caminos mínimos de las secciones de tripulación.

3.1.2. Vías intransitables

Otra característica particular de los DataSets presentados por la competencia es que los trenes no podrán circular por todas las vías. Esto se debe a que existen vías que no pertenecen al camino mínimo de ninguna sección de tripulación y a la limitación de que los trenes solo podrán circular por los caminos mínimos de las secciones de tripulación.

En particular esto no sucede en el DataSet1, pero sí en el DataSet2, en el cual existen múltiples vías intransitables. El número exacto de vías intransitables depende de las elecciones que se hayan tomado sobre cual será el camino mínimo de las secciones de tripulación donde existen múltiples caminos mínimos, problema expuesto en el punto anterior. Aunque el número de vías intransitables varía en función de esta decisión, siempre está en el rango de entre 62 y 72 vías intransitables. En particular para la elección de caminos mínimos de este trabajo, habrá 67 vías intransitables.

Estas vías pueden ser removidas del grafo de la red en un preprocesamiento para reducir el tamaño del grafo y así simplificar la resolución del problema.

3.1.3. Intersección entre secciones de tripulación

Otra de las particularidades que encontramos en los DataSet es que existen secciones de tripulación que comparten vías, equivalentemente existen vías que pertenecen a múltiples secciones de tripulación.

No solo esto, sino que también hay secciones de tripulación que están completamente contenidas en otra sección de tripulación.

Por ejemplo, en el DataSet1 la sección de tripulación (485,6207) está completamente contenida en la sección (485,2196), ya que el camino de la primera consiste en las estaciones (485-9228-6207) y la segunda en (485-9228-6207-2196).

	DataSet1	**DataSet2**
N^o de secciones de tripulación que se solapan	65/154	90/154
N^o de secciones de tripulación completamente incluidas en otras	31/154	30/154

Uno de los problemas que genera esta característica, es que rompe la biyección entre un camino de secciones de tripulación y un camino de vías, pudiendo existir un camino de vías que sea válido para más de un camino de secciones de tripulación. Es importante tener en cuenta esto desde un punto de vista implementativo, ya que solo guardar los caminos de vías y no los caminos de secciones de tripulación puede llevar a una pérdida de información.

3.1.4. Particularidad de los bloques

Una de las particularidades de los bloques de las instancias dadas por INFORMS-RAS [5] es que hay una correlación lineal entre la cantidad de vagones en un bloque y tanto su peso como su longitud. Esto es un dato muy valioso ya que correlaciona la penalización por no mandar un bloque (dependiente de la cantidad de vagones) con cuánta capacidad de la red consume mandar ese bloque (dependiente del peso y longitud del mismo).

En particular en los DataSets provistos vemos una media de peso por vagón de 60 toneladas con una desviación estándar de 1.28 y en el caso de la longitud, la media es 80 pies y la desviación estándar 1.7.

3.1.5. Cota inferior del costo

A continuación presentaremos una cota inferior del costo de cada uno de los DataSets presentados en la competencia. Esta cota nos ayudará a entender la calidad de las diferentes soluciones presentadas.

Para construirla tendremos en cuenta el costo de penalización por bloques no enviados y el costo de viaje de los vagones, ya que estos son los mayores costos en estas instancias y además, los más simples de acotar.

Para la penalización por bloque, calcularemos los bloques para los cuales no exista ningún camino factible entre su origen y su destino. Entendemos por camino factible, un camino donde todas sus vías son transitables. Estos bloques nunca podrán ser enviados, ya que para todo camino entre su origen y su destino, siempre existirá aunque sea una vía que, al no pertenecer a ninguna sección de tripulación, ningún tren podrá transportar al bloque por esa vía. Dados estos bloques que son imposibles de entregar, calculamos la cota del costo de bloques no entregados como el costo de no llevar un vagón multiplicado por la suma de los vagones de cada uno de estos bloques.

Para calcular la cota del costo de viaje de los vagones, calcularemos el camino mínimo de cada bloque que puede ser entregado (no tendremos en cuenta los bloques que en el calculo anterior identificamos como que no se pueden enviar), y a esto lo multiplicaremos por la cantidad de vagones de cada bloque y por el costo de viaje de un vagón.

En la siguiente tabla veremos las cotas inferiores para cada uno de los DataSets de la competencia.

	DataSet1	**DataSet2**
Cota del costo por bloques no entregados	$0	$420.000
Cota del costo de viaje de los vagones	$1.556.658,975	$2.177.285,625
Cota total	**$1.556.658,975**	**$2.597.285,625**

3.2. Equipo Koppa

El equipo Koppa separa el problema en etapas de menor complejidad, ya que, en general, resolver n problemas exponenciales de tamaño $\frac{x}{n}$ será considerablemente más simple que resolver un solo problema exponencial de tamaño x. Al TDO lo dividen en 2 subproblemas, el primero al que llaman *Train Design Problem* consiste en seleccionar las rutas de tren que serán operadas, y el segundo subproblema es resolver BTA.

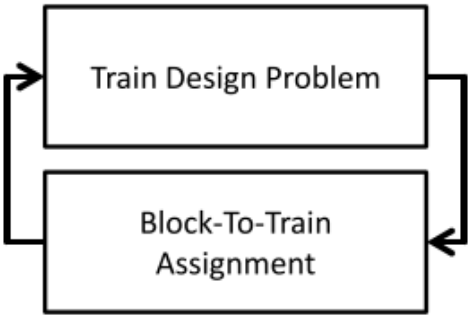

Fig. 3.1: Esquema Koppa

3.2.1. Train Design Problem

Este problema lo subdividen a su vez en tres etapas, como se puede observar en la imagen 3.2. La primera es la estimación de la demanda de tripulaciones, que consiste en estimar cuántos trenes se necesita que circulen por cada sección de tripulación para enviar cada bloque por el camino mínimo entre su origen y destino. A esta estimación agregan un valor Δ para compensar que la estimación es una cota inferior del número real de trenes necesarios, ya que asume que los bloques pueden ser divididos entre múltiples trenes, cuando en realidad deben circular todos los vagones de un bloque en el mismo tren.

Para resolver esta etapa se utiliza un ILP que minimiza solo el costo asociado a los bloques, consiguiendo así el camino mínimo de un bloque a su destino garantizando que nunca circule por una vía cuya capacidad de peso y/o longitud sea menor que el peso y/o longitud del bloque.

La siguiente etapa es la generación de rutas, que consiste en generar un gran *pool* de rutas al azar que usen las secciones de tripulación conseguidas en el cálculo de la demanda de tripulación. Para generar cada una de estas rutas, se utiliza el siguiente proceso: se selecciona una sección de tripulación inicial al azar, a continuación se selecciona una nueva sección de tripulación al azar pero cuyo inicio coincida con el final de la sección anterior. Se repite este proceso hasta que se hayan elegido un número definido de secciones de tripulación.

Por último, se resuelve la selección de rutas, que consiste en elegir entre el gran *pool* de rutas random generadas en la etapa anterior, un subconjunto de rutas de "buena calidad". Para esto, se formula un ILP de *set covering* sobre el *pool* de rutas buscando minimizar los costos que no dependen de los bloques (millas recorridas por trenes, desbalances de trenes, desbalance de tripulación, etc.) y buscando cubrir la demanda de tripulación estimada.

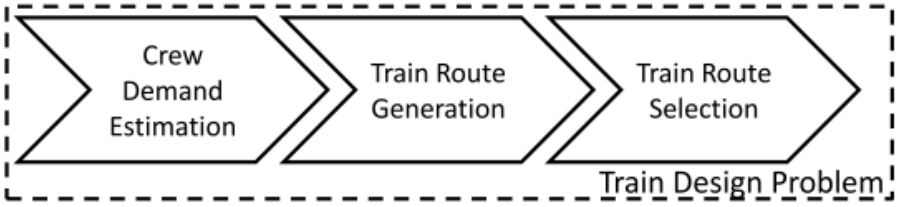

Fig. 3.2: Primera Etapa Koppa

3.2.2. Block to Train Assignment

El segundo subproblema que resuelve el equipo Koppa, consiste en conseguir una asignación de cada bloque (denominada en ingles BTA). En esta etapa decidieron crear un grafo en el cual se incluyen gran parte de las restricciones para facilitar la construcción del ILP que resuelva el problema.

Con la ayuda del grafo creado con la mayoría de las restricciones embebidas, formula un ILP que resuelve la asignación bloque-tren minimizando el costo de los bloques (millas, *swaps*, no envío) y de los *work events*, ya que el resto de los costos (desbalance de tripulación, desbalance de trenes, costo fijo y variable de trenes) ya fue resuelto en la primera etapa al seleccionar las rutas de tren.

3.2.3. Resultados

El equipo Koppa fue el ganador de la competencia. En los resultado expuestos por el grupo utilizan tanto Cplex como Xpress-MP y Gurobi para resolver los ILP. En todos los casos se determina un *gap* de optimalidad y un tiempo límite para detener la optimización. En el caso del límite de tiempo se determinan 4 horas, y experimentaron variando el *gap* entre 1 % y 0.1 %. Los resultados de los diferentes programas de optimización varían en función del *gap* de optimalidad seleccionado. Según los resultados presentados, utilizando un *gap* del 1 % Xpress-MP y Gurobi consiguen soluciones más rápido, utilizando alrededor de 50 segundos en el DataSet1 y 225 segundos en el DataSet2, en cambio Cplex requiere 120 segundos en el DataSet1 y 3350 en el DataSet2. Pero estos resultados se invierten si se utiliza un *gap* del 0.1 %, en el DataSet1 Xpress-MP sigue siendo el que requiere menos tiempo, utilizando 83 segundos, seguido por Cplex que utiliza 300 segundos y por último Gurobi requiere 350 segundos. En el caso del DataSet2, Cplex es el único que logra demostrar un *gap* de optimalidad del 0.1 %, utilizando 2.5 horas para ello, en cambio Xpress-MP y Gurobi no logran alzanzar ese *gap* luego de las 4 horas de cómputo máximas. A continuación se expone las soluciones y tiempos de cómputo utilizados para resolver el problema con un *gap* de optimalidad del 1 %.

DataSet	Costo Total	Tiempo Cplex (s)	Tiempo Gurobi (s)	Tiempo Xpress-MP (s)
DataSet1	2.043.471$	120	55	45
DataSet2	3.187.999$	3344	226	221

3.3. Equipo OR@UNIMI

Este equipo toma un enfoque heurístico que consiste en construir un solo ILP que resuelva el problema, pero al ser éste de tal magnitud, no puede ser resuelto en un tiempo razonable si se optimiza el problema completo. Por eso se aplica el método de generación de filas y columnas. Este método consiste en empezar con un número de columnas y filas reducido e ir ampliándolo sistemáticamente. Se llama generación de filas y columnas, ya que al agregar una columna es necesario también agregar cierta cantidad de filas para mantener la formulación consistente.

Primero se crea un ILP que contenga todas las restricciones del problema, a este ILP se le relaja las restricciones de integralidad de las variables de decisión. A este nuevo problema lo llamaremos *Master Problem*. En esta formulación las columnas representan tanto una ruta de tren o un camino de la red a través del cual circulará cada bloque, a los que llamaremos *block path*. En vez de considerar todos los *block paths* y rutas posibles, lo que volvería al problema casi imposible de resolver, se empieza teniendo en cuenta un pequeño conjunto inicial tanto de *block paths* como de rutas. Luego, a este conjunto inicial se le agregan primero nuevos *block paths* generados usando un ILP *net-flow Model* como *Slave / Pricing Problem* . Cuando ya no existen *block paths* que mejoran el funcional del *Master Problem*, se continua con la generación de rutas usando una *two level tabu search heuristic*. Al terminar esta segunda etapa de la generación de columnas, se vuelve a iterar, volviendo a generar primero *block paths* y luego rutas siguiendo el mismo proceso. Esto se repite hasta alcanzar algún criterio de corte (cantidad de rutas generadas o tiempo de cómputo).

Al final de la generación de columnas se impone integralidad al *Master Problem*, se resuelve el ILP y se brinda la mejor solución factible.

3.3.1. Resultados

El equipo OR@UNIMI consiguió el segundo puesto.

DataSet	Costo Total	Tiempo GC (s)	Tiempo MIP (s)	Tiempo Total (s)
DataSet1	2.212.163$	32000	4000	36000
DataSet2	3.538.924$	12600	1800	14400

Algo llamativo de los resultados presentados es que el DataSet1 tomó más tiempo para ser resuelto que el DataSet2, a diferencia de todos los demás equipos.

3.4. Equipo NCKU

Como resolver TDO con un ILP teniendo en cuenta todas las restricciones y un gran número de rutas y *block paths* es altamente costoso tanto en tiempo como en espacio (al punto de ser irrealizable en la práctica), NCKU construye un método iterativo para

ir construyendo progresivamente un conjunto de rutas y de *block paths* pequeño pero significativo para resolver TDO con una solución de buena calidad, pero en un tiempo razonable.

Dado un conjunto de *block path* P y un conjunto de rutas posibles T, el equipo NCKU define un modelo de programación lineal entera TD(P,T) para resolver TDO (tanto el BTA como la selección de rutas).

En primer lugar, construye un conjunto inicial de *block paths* P y de rutas T. Luego selecciona un subconjunto de rutas T' de T, quita estas rutas del conjunto T y resuelve TD(P,T')

Si la solución no es lo suficientemente buena, retiene las rutas usadas en esta solución en T* como un *warm start*, devuelve el resto de las rutas de T' a T y vuelve a seleccionar un conjunto T" de T para crear el nuevo conjunto T' junto con T*.

Con este nuevo conjunto T' se vuelve a iterar este proceso, volviendo a resolver TD(P,T'). Este proceso se reitera hasta que la calidad de las soluciones empieza a mejorar muy lentamente (se garantiza que la nueva iteración siempre tendrá una solución mejor o igual que la anterior, ya que se conservan las rutas usadas en la solución anterior). En ese momento se decide si terminar el proceso (ya sea porque se llegó a una solución suficientemente buena, o porque se alcanzó el tiempo de cómputo límite) o mejorar las conjuntos P y T y volver a ejecutar el proceso iterativo.

El conjunto inicial de *block path* P se genera usando K caminos mínimos y el conjunto de rutas T, se genera construyendo la ruta mínima para cada *block path* en P.

Para seleccionar el subconjunto T' de T, se crea para cada ruta t un contador q_t que cuenta la cantidad de veces que la ruta t fue descartada por el método iterativo. Luego, se selecciona T' basándose en un *roulette wheel selection algorithm* que consiste en elegir a una ruta t' con probabilidad $\frac{(q_{t'}+1)^{-1}}{\sum_{t \in T}(q_t+1)^{-1}}$, o sea cuanto menos veces haya sido descartada la ruta t', más probabilidad tendrá de ser seleccionada.

Por último, para mejorar los conjuntos P y T, se usa un conjunto de técnicas heurísticas, como unir dos rutas buenas en una sola, generar nuevos *block paths*, generar rutas asociadas a esos *block paths*, o hasta descartar rutas que demostraron no ser buenas.

3.4.1. Resultados

El equipo NCKU consiguió el tercer puesto en la competencia.

DataSet	**Costo Total**	**Tiempo (s)**[1]
DataSet1	2.018.646$	∼70.000
DataSet2	3.152.018$	∼80.000

Es llamativo que aunque consiguieron los mejores resultados con respecto a la calidad de solución, hayan conseguido el tercer puesto en la competencia. Suponemos que el necesitar significativamente más tiempo (∼19 horas vs menos de una hora el resto de los equipos), fue un factor determinante para el jurado a la hora de discernir las posiciones de la competencia.

[1] Tiempo no reportado, inferido a través de los gráficos presentados

3.4.2. Observaciones críticas

A esta solución le encontramos dos problemas. El primero, como reporta Jin et al. [6], es que el grafo creado para construir las rutas no considera la continuidad de las secciones de tripulación, permitiendo a una ruta cambiar de sección en el medio de la misma, generando así rutas que no satisfacen los requerimientos de TDO.

El segundo problema es el mayor tiempo de cómputo requerido por esta metodología. Aunque el equipo no reporta explícitamente el tiempo que se usó para llegar a las soluciones presentadas, las curvas de convergencias provistas muestran más de 70.000 segundos (>19 horas) en ambas instancias para converger. Este tiempo es significativamente mayor a los reportados por el resto de los equipos (siempre menor a 1 hora), y dificulta la comparación de los resultados.

4. FORMULACIÓN PARA TDO

En esta tesis trabajamos basados en la metodología y modelización del equipo RAIL-OPT, publicado en Jin et al. [6] para resolver el *Train Design Optimization Problem* presentado en la competencia de INFORMS-RAS de 2011 [5].

Primero explicaremos en profundidad la metodología usada por el equipo RAIL-OPT sobre la que hemos realizado algunas adaptaciones para reflejar mejor algunas particularidades del problema, y después expondremos los cambios que introdujimos sobre esta metodología.

El enfoque del equipo RAIL-OPT [6] consiste en dividir el problema en sub-problemas de menor complejidad y resolverlos secuencialmente. Esta forma de atacar al problema es particularmente efectiva dada la naturaleza altamente exponencial del TDO.

En una primera etapa, se selecciona un conjunto de *block paths* para cada bloque. Como ya definimos previamente, estos serán caminos de la red ferroviaría y se restringirá en las etapas siguientes a que los bloques se trasladaden solo por vías pertenecientes a sus *block path*.

Dados estos conjuntos de *block path* para cada bloque, se construye un conjunto de rutas candidatas. Cada una de estas rutas constituye un camino en la red con la particularidad de consistir a la vez en una secuencia de secciones de tripulación (cumpliendo con la restricción de origen y destino de tripulaciones). Los trenes circularán por algunas de estas rutas candidatas. Para construir estas rutas candidatas, se utiliza un ILP que se resuelve por generación de columnas.

En la segunda etapa, tomando como input tanto los *block paths* como el conjunto de rutas candidatas, se selecciona qué subconjunto de las rutas candidatas se van a usar en la solución final y se crea una planificación del itinerario de cada bloque. Esta planificación será el BTA, que como ya se explicó, consiste en seleccionar tanto el camino que recorre el bloque entre su origen y destino, como en qué trenes circulará por cada parte del recorrido, y en qué estaciones intermedias se intercambiará el bloque entre dos trenes.

Estas son todas las decisiones necesarias para resolver el TDO, con ellas se puede calcular el costo total de la solución.

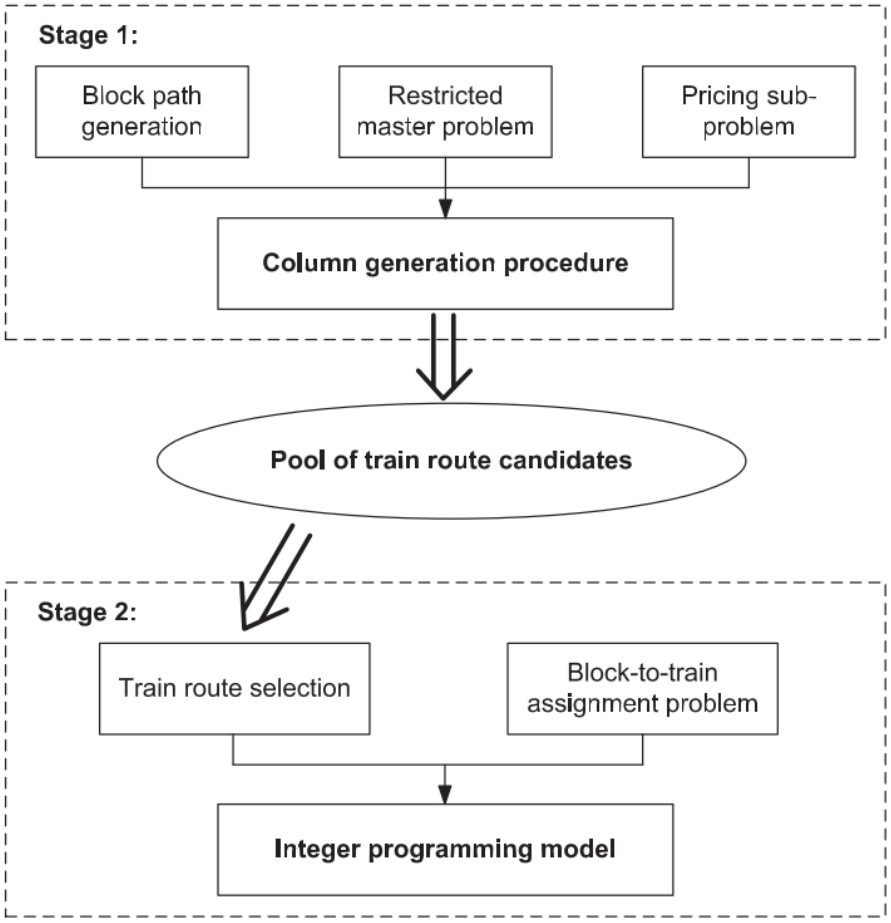

Fig. 4.1: Overview de RAIL-OPT

4.1. Generación de *Block Path*

Jin et al. [6], definen un parámetro k como la cantidad (máxima) de *block paths* a crear para cada bloque. Además, deciden que para conseguir los k *block paths* de un bloque, se tomarán los k caminos mínimos entre el origen y el destino del bloque.

Calculan los k caminos mínimos (k *min paths.*) resolviendo un ILP que encuentra el camino mínimo. Luego se agrega un corte que quita este primer camino mínimo del dominio del modelo, y se vuelve a resolver el nuevo ILP consiguiendo así el segundo camino mínimo. Este proceso se repite k veces para conseguir los k caminos mínimos.

En cambio, en nuestro trabajo utilizamos la generalización del algoritmo de Dijkstra, descripta en el apéndice B.4.1 para conseguir los k caminos mínimos por la facilidad de su implementación.

4.2. Generación de Columnas

El siguiente paso del proceso es la generación de un conjunto de rutas candidatas. Para esto se utiliza un modelo que se resuelve utilizando el método de generación de columnas explicado en B.3.

Se tomarán los *block path* generados en el paso anterior y se restringirá a cada bloque a circular por uno de los *block path* generados en el punto anterior.

En esta etapa se utilizarán dos nuevos parámetros. El primero será el número máximo de rutas a construir durante la generación de columnas al que llamaremos *MaxRoutes*. El segundo será *Lmax*, que representará la longitud máxima, medida en secciones de tripulación, que pueden tener las rutas generadas.

4.2.1. *Master Problem*

Al crear el problema maestro, se tendrá como objetivo optimizar los costos asociados a la operación de los trenes y a la asignación de tripulación a los trenes, pero no se considerarán los costos de los *work events* y de los *block swaps*. Estos costos no son considerados en esta etapa, ya que en esta instancia no se genera un BTA, lo que imposibilita calcular los mismos.

Otra restricción que se relaja es la restricción de peso y longitud máximos de un tren circulando por una vía. Esta es otra restricción que no se puede calcular con exactitud sin tener un BTA. Por esto se pedirá que la restricción se cumpla en promedio.

Para la resolución del problema se usará el grafo de la red física $G_1(V, E)$, donde V es el conjunto de estaciones y E el conjunto de vías de tren que unen estas estaciones, al ser todas las vías bidireccionales, tanto (i, j) como (j, i) pertenecerán a E. B es el conjunto de los bloques y C el conjunto de las secciones de tripulación. Q_b será el conjunto de k caminos mínimos para el bloque b calculados en la etapa anterior. Por último, P será el conjunto de rutas candidatas construido hasta el momento, particularmente en la primera iteración P será el conjunto vacío.

Las variables de decisión de este problema serán λ_p con $p \in P$ que será una variable binaria que representará si se utiliza la ruta p. δ_b^q siendo b un bloque y q un camino del bloque b, también variable de decisión binaria, representará si el bloque b circulará por el camino q.

σ_{mn} será una variable de decisión entera positiva que representará la cantidad de desbalance de la tripulación de la sección (m, n). De forma similar, ϕ_i será una variable de decisión entera positiva que representará la cantidad de desbalance de trenes en la estación i.

Por último, τ^b será una variable binaria que representará si el bloque b será entregado a destino.

Para la formulación del problema maestro, se usarán las siguientes notaciones presentadas en la tabla 4.1.

Conjuntos	
$G_1(V, E)$	Grafo que define la red física. V es el conjunto de estaciones y E es el conjunto de vías entre estaciones
B	Conjunto de bloques
P	Conjunto de rutas candidatas, inicialmente P = $\emptyset$
Q^b	Conjunto de k caminos mínimos del bloque $b \in B$
C	Conjunto de secciones de tripulación
Variables de decisión	
λ_p	1 si se usa la ruta $p \in P$; 0 en caso contrario
δ_q^b	1 si el bloque $b \in B$ usa el *block path* $q \in Q^b$; 0 en caso contrario
σ_{mn}	$\mathbb{Z}^+$, cantidad de desbalance de la tripulación $(m, n) \in C$
ϕ_i	$\mathbb{Z}^+$, cantidad de desbalance de trenes en la estación $i \in V$
τ^b	1 si el bloque $b \in B$ no es llevado a destino; 0 en caso contrario
Parámetros	
c_1	Costo de inicio de un tren
c_2	Costo por milla de tren recorrida
c_3	Costo por milla de vagón recorrida
c_6	Costo por desbalance de tripulación
c_7	Costo por desbalance de tren
l_{ij}	Longitud del eje $(i, j) \in E$
v_1^b	Número de vagones en el bloque $b \in B$
v_2^b	Longitud del bloque $b \in B$
v_3^b	Peso del bloque $b \in B$
α_{pij}	1, si la ruta $p \in P$ usa el eje $(i, j) \in E$; 0 en caso contrario
α_{qij}	1, si el *block path* $q \in Q^b$ usa el eje $(i, j) \in E$; 0 en caso contrario
βpmn	1, si el *crew segment* $(m, n) \in C$ es usado por la ruta $p \in P$; 0 en caso contrario
γ_{pi}^1	1, si la ruta $p \in P$ se origina en la estación $i \in V$; 0 en caso contrario
γ_{pi}^2	1, si la ruta $p \in P$ termina en la estación $i \in V$; 0 en caso contrario
c_p	Costo de un tren viajando por la ruta $p \in P$. $c_p = c_2 \sum_{(i,j)\in E} \alpha_{qij} l_{ij}$
c_q^b	Costo de un vagón del bloque $b \in B$ viajando por el *block path* $q \in Q^b$. $c_q^b = c_3 \sum_{(i,j)\in E} \alpha_{qij} l_{ij}$
M_{ij}^1	Número máximo de trenes que pueden pasar por el eje $(i, j) \in E$
M_{ij}^2	Longitud máxima de trenes que pasan por el eje $(i, j) \in E$
M_{ij}^3	Peso máximo de trenes que pasan por el eje $(i, j) \in E$

Tab. 4.1: Notaciones para el problema maestro

Formulación del Problema Maestro

$$\text{mín } c_1 \sum_{p\in P} \lambda_p + \sum_{p\in P} c_p\lambda_p + \sum_{b\in B}\sum_{q\in Q^b} v_1^b c_q^b \delta_q^b + \tfrac{c_6}{2} \sum_{(m,n)\in C} \sigma_{mn} + c_7 \sum_{i\in V} \phi_i + M \sum_{b\in B} \tau^b \tag{4.1}$$

sujeto a

$$\sum_{q\in Q^b} \delta_q^b + \tau^b = 1 \qquad \forall b \in B \tag{4.2}$$

$$\sum_{p\in P} \alpha_{pij}\lambda_p + \sum_{p\in P} \alpha_{pji}\lambda_p \leq M_{ij}^1 \qquad \forall (i,j) \in E | i < j \tag{4.3}$$

$$\sum_{b\in B}\sum_{q\in Q^b} \alpha_{qij} v_2^b \delta_q^b - M_{ij}^2 \sum_{p\in P} \alpha_{pij}\lambda_p \leq 0 \qquad \forall (i,j) \in E \tag{4.4}$$

$$\sum_{b\in B}\sum_{q\in Q^b} \alpha_{qij} v_3^b \delta_q^b - M_{ij}^3 \sum_{p\in P} \alpha_{pij}\lambda_p \leq 0 \qquad \forall (i,j) \in E \tag{4.5}$$

$$\sigma_{mn} + \sum_{p\in P} \beta_{pmn}\lambda_p - \sum_{p\in P} \beta_{pnm}\lambda_p \geq 0 \qquad \forall (m,n) \in C \tag{4.6}$$

$$\sigma_{mn} - \sum_{p\in P} \beta_{pmn}\lambda_p + \sum_{p\in P} \beta_{pnm}\lambda_p \geq 0 \qquad \forall (m,n) \in C \tag{4.7}$$

$$\phi_i + \sum_{p\in P} \gamma_{pi}^1\lambda_p - \sum_{p\in P} \gamma_{pi}^2\lambda_p \geq 0 \qquad \forall i \in V \tag{4.8}$$

$$\phi_i - \sum_{p\in P} \gamma_{pi}^1\lambda_p + \sum_{p\in P} \gamma_{pi}^2\lambda_p \geq 0 \qquad \forall i \in V \tag{4.9}$$

$$0 \leq \lambda_p, \delta_q^b \leq 1 \tag{4.10}$$

$$\sigma_{mn}, \phi_i \geq 0 \tag{4.11}$$

La función objetivo 4.1 minimizará el costo operativo incluyendo:

- Costo fijo de los trenes: $c_1 \sum_{p\in P} \lambda_p$
- Costo de los trenes por milla: $\sum_{p\in P} c_p\lambda_p$
- Costo de los vagones por milla: $\sum_{b\in B}\sum_{q\in Q^b} v_1^b c_q^b \delta_q^b$
- Costo del desbalance de tripulación: $\frac{c_6}{2} \sum_{(m,n)\in C} \sigma_{mn}$
- Costo del desbalance de trenes: $c_7 \sum_{i\in V} \phi_i$
- Penalización por no llevar un bloque: $M \sum_{b\in B} \tau^b$

La restricción 4.2 obliga al bloque a circular por solo uno de sus *block paths*. No siempre es posible enviar todos los bloques, por ejemplo al iniciar el *master problem* se cuenta con un conjunto de rutas vacío, por lo que no se podrá llevar ningún bloque. Por este motivo, para garantizar la factibilidad se utiliza una variable τ^b que será 1 si no se puede enviar el bloque $b \in B$ y se incluye en e funcional $M \sum_{b \in B} \tau^b$ donde M es una constante suficientemente grande para que se envíe el bloque siempre que sea posible. Esto garantiza factibilidad, ya que la solución con $\tau^b = 1, \forall b \in B$ siempre será factible. La restricción 4.3 garantiza que por ninguna vía circulen más trenes que su capacidad. Modificamos levemente la restricción original para considerar los trenes que circulan en ambas direcciones de la vía como parte de la misma capacidad máxima de trenes circulando en una vía. La restricción en la formulación original era la siguiente:

$$\sum_{p \in P} \alpha_{pij} \lambda_p \leq M^1_{ij}, \ \forall (i,j) \in E$$

.

Las restricciones 4.4 y 4.5 garantizan que por cada vía circulen en promedio trenes que no superen la longitud y el peso máximo de la vía respectivamente. Las restricciones 4.6 y 4.7 definen las variables de decisión del desbalance de tripulación y las 4.8 y 4.9 las del desbalance de trenes. Por último, las restricciones 4.10 y 4.11 definen el dominio de las variables de decisión. Al relajar las variables de decisión volviéndolas continuas, se permite que un bloque se divida en sub-bloques y tome múltiples caminos.

4.2.2. *Slave Problem*

Luego de resolver el problema maestro, se creará el problema esclavo asociado que representa el costo reducido de una ruta p. Esta formulación se utilizará para generar una nueva ruta para agregar al conjunto de rutas del problema maestro. La ruta generada satisfará las siguientes restricciones:

- La ruta generada empezará en una estación terminal de una sección de tripulación.
- La ruta generada terminará en una estación terminal de una sección de tripulación.
- Cada tripulación viajará solo por el camino correspondiente a su sección.
- La ruta será una secuencia continua de secciones de tripulación.

Para crear este problema esclavo se usarán las variables duales asociadas a las restricciones 4.2 a 4.9, $\pi^1_b, \pi^2_{ij} \leq 0, \pi^3_{ij} \leq 0, \pi^4_{ij} \leq 0, \pi^5_{mn} \geq 0, \pi^6_{mn} \geq 0, \pi^7_i \geq 0, \pi^8_i \geq 0$ respectivamente.

El costo reducido de una ruta p sera:

$c_1 + c_p - \sum_{(i,j) \in E} \alpha_{pij} \pi^2_{ij} + \alpha_{pji} \pi^2_{ji} + \sum_{(i,j) \in E} M^2_{ij} \alpha_{pij} \pi^3_{ij} + \sum_{(i,j) \in E} M^3_{ij} \alpha_{pij} \pi^4_{ij} - \sum_{(m,n) \in C} (\beta_{pmn} - \beta_{pnm}) \pi^5_{mn} + \sum_{(m,n) \in C} (\beta_{pnm} - \beta_{pmn}) \pi^6_{mn} - \sum_{i \in V} (\gamma^1_{pi} - \gamma^2_{pi}) \pi^7_i + \sum_{i \in V} (\gamma^1_{pi} - \gamma^2_{pi}) \pi^8_i$

Para garantizar que la ruta generada sea una sucesión de secciones de tripulación, se definirá la ruta en función de las secciones de tripulación que la compondrán. Se usará una variable de decisión, x_{mn}, por cada sección de tripulación, en vez de usar una variable de decisión por vía. También se agregará una estación *dummy* a la que se llamará estación 0 y secciones de tripulación *dummy* $(0,i)$ y $(i,0)$ $\forall i \in V$ para facilitar la creación de las rutas.

El objetivo de crear la estación y las secciones de tripulación *dummy* es poder identificar el inicio y el fin de la ruta, para esto toda ruta empezará y finalizará en la estación *dummy* 0. La función objetivo del esclavo será el costo reducido de una ruta p pero se cambiará la fórmula al trabajar sobre las secciones de tripulaciones en vez de las vías.

Conjuntos	
V^0	Conjunto de nodos, incluyendo nodo *dummy*. $V^0 = V \cup \{0\}$
C^0	Conjunto de secciones de tripulación incluyendo las secciones de tripulación *dummy*. $C^0 = C \cup \{(0,i),(i,0),\forall i \in V\}$
Variables de decisión	
x_{mn}	1 si la ruta usa la sección de tripulación $(m,n) \in C^0$; 0 en caso contrario
Parámetros	
θ_{mnij}	1 si la sección de tripulación $(m,n) \in C$ pasa a través del eje $(i,j) \in E$; 0 en caso contrario
$Lmax$	Cantidad máxima de secciones de tripulación que puede recorrer una ruta generada

Tab. 4.2: Notaciones para el problema esclavo

Formulación del Problema Esclavo

$$\text{mín } c_1 + \sum_{(m,n\in C)} x_{mn}\Bigg[\sum_{(i,j)\in E}\theta_{mnij}\left(c_2 l_{ij} - \pi^2_{ij} + M^2_{ij}\pi^3_{ij} + M^3_{ij}\pi^4_{ij}\right) - (\pi^5_{mn} - \pi^6_{mn}) + (\pi^5_{nm} - \pi^6_{nm})\Bigg] - \sum_{i\in V} x_{0i}(\pi^7_i - \pi^8_i) + \sum_{i\in V} x_{i0}(\pi^7_i - \pi^8_i) \tag{4.12}$$

sujeto a

$$\sum_{n\in V^0|(0,n)\in C^0} x_{0n} = 1 \tag{4.13}$$

$$\sum_{n\in V^0|(m,n)\in C^0} x_{mn} - \sum_{n\in V^0|(n,m)\in C^0} x_{nm} = 0 \qquad \forall m \in V^0 \tag{4.14}$$

$$\sum_{(m,n)\in C} x_{mn} \leq L_{max} \tag{4.15}$$

$$\sum_{n|(m,n)\in C} x_{mn} \leq 1 \qquad \forall m \in V \tag{4.16}$$

$$\sum_{(m,n)\in C} \theta_{mnij} x_{mn} \leq 1 \qquad \forall (i,j) \in E \tag{4.17}$$

$$x_{mn} \in \{0,1\} \qquad \forall (m,n) \in C^0 \tag{4.18}$$

La función objetivo 4.12 minimiza el costo reducido de una ruta. Modificamos la función

objetivo del trabajo de Jin et. al [6] ya que θ_{mnij} debía aplicar a toda la sumatoria y no solo a $c_2 l_{ij}$. La sumatoría original era:

$$\sum_{(i,j)\in E} \left(\theta_{mnij} c_2 l_{ij} - \pi^2_{ij} + M^2_{ij}\pi^3_{ij} + M^3_{ij}\pi^4_{ij}\right)$$

Las restricciones 4.13 y 4.14 son de conservación de flujo, garantizan que se utilice una vía que inicia en la estación *dummy* y que en toda estación se usen tantas vías que llegan a esa estación como vías que salen. La restricción 4.15 define como L_{max} el número máximo de secciones de tripulación que puede usar una ruta. Esta restricción no es intrínseca al problema, sino que se agrega como una heurística para generar mejores rutas, ya que una ruta muy larga puede generar mayor cantidad de *work events* y mayor costo. La restricción 4.16 evita la generación de rutas complejas que pasan por una vía múltiples veces. Esta restricción tampoco es parte del problema, sino que se agrega para facilitar el cumplimiento de las restricciones en el BTA, en particular las restricciones de cantidad de trenes circulando por una ruta, el cálculo de los *work event* y la asignación bloque-tren en sí.

Al haber vías que pertenecen a múltiples secciones de tripulación (3.1.1), se debe limitar a que cada tren pase una sola vez por cada vía en una dirección dada. Si esto no se cumpliera, todo costo o restricción asociado a si un tren pasa por cierta vía será incorrecto, ya que en el modelo se utiliza si el tren pasa o no por la vía, pero nunca se tiene en cuenta si pasa múltiples veces por la misma. Para solucionar esto, agregamos la restricción 4.17 que no es parte de la formulación original.

Se puede apreciar que el problema esclavo es un problema de camino mínimo pero con costos posiblemente negativos. Por esto, puede ocurrir que se generen múltiples sub-tours disconexos. Para evitar esto, luego de generada una solución, se comprobará si existen estos sub-tours disconexos, y en caso de existir, se agregará al problema el corte 4.19 y luego se volverá a generar una nueva solución. Esto se repetirá hasta conseguir una solución que no contenga sub-tours disconexos.

$$\sum_{(m,n)\in S} (1 - x_{mn}) \geq 1, \text{con S sub-tour disconexo.} \tag{4.19}$$

4.3. Block to Train Assignment

Una vez construido el conjunto de rutas candidatas y los *block path* en las etapas previas, el último paso consiste en seleccionar el conjunto de rutas finales y en construir una asignación bloque-tren que satisfaga todas las restricciones.

Para resolver esta etapa, se construirá un segundo grafo, $G_2(N, A)$ llamado red de servicio, asociado a la red física $G_1(V, E)$. En este grafo se embeberán gran parte de las restricciones, facilitando la posterior resolución del problema.

4.3.1. Construcción del grafo de la red de servicio

En el grafo de la red de servicio $G_2(N, A)$, los nodos representarán pares ruta-estación, formalmente $N = \{(p, n)|\ \forall p \in P, \forall n \in V\}$. Para el nodo i se llamará $p^+(i)$ y $p^-(i)$ a la ruta y estación del nodo i respectivamente.

Habrá dos conjuntos de ejes, el conjunto A_1 que corresponderá a ejes de trenes, representando el movimiento de los trenes entre diferentes estaciones y el conjunto A_2 que corresponderá a ejes *swap*, representando el movimiento de bloques entre diferentes trenes.

En un eje de tren, la componente del par correspondiente al tren de ambos nodos del eje será igual, pero las estaciones serán distintas y corresponderán a un eje en el grafo físico:

$$(i,j) \in A_1 \;\Leftrightarrow\; p^+(i) = p^+(j) \wedge (p^-(i), p^-(j)) \in E$$

En cambio, en un eje *swap*, se cumplirá lo inverso. La componente del par asociada al tren de cada nodo del eje cambiará, pero la estación será la misma:

$$(i,j) \in A_2 \;\Leftrightarrow\; p^+(i) \neq p^+(j) \wedge p^-(i) = p^-(j)$$

Ilustraremos esta construcción usando el ejemplo provisto en la competencia. El ejemplo está compuesto por una red física que consiste de cuatro estaciones A, B, C y D y de cinco ejes, $A \leftrightarrow B, B \leftrightarrow C, C \leftrightarrow D, D \leftrightarrow A$ *y* $B \leftrightarrow D$.

Sobre esa red se seleccionaron tres rutas, el Tren 1 consiste en los siguientes ejes $A \rightarrow D$ *y* $D \rightarrow A$, el Tren 2 $B \rightarrow D$ *y* $D \rightarrow C$ y por último el Tren 3 está compuesto por un solo eje $D \rightarrow B$.

En el siguiente gráfico, 4.2, podemos ver la diferencia entre la red física y la red de servicio.

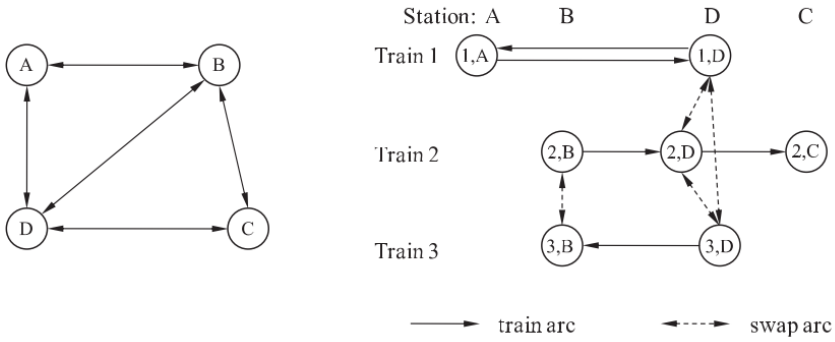

Fig. 4.2: Representación de la red física y de servicio

Además se definirán $A_1^b \subset A_1$ y $A_2^b \subset A_2$, $\forall b \in B$ representando un subconjunto de los ejes en A_1 y A_2 respectivamente, asociados al bloque b. Un eje $((t_1, v_1), (t_1, v_2)) \in A_1$ pertenecerá a A_1^b, si y solo si (v_1, v_2) es parte de alguno de los *block path* generados para b. De la misma forma, un eje $((t_1, v_1), (t_2, v_1)) \in A_2$ pertenecerá a A_2^b, si y solo si v_1 es parte de alguno de los *block path* generados para b.

También se definirán $A^b = A_1^b \cup A_2^b$ y sobre este conjunto se definirán las variables binarias de decisión y_{ij}^b. Estas variables representan si el bloque b circulará por el eje (i,j). Estas variables se definen sobre A^b ya que $|A^b| \ll |A|$, lo que reduce considerablemente la cantidad de variables a tener en cuenta.

Las variables binarias z_{ij} modelarán si un tren p realiza un *work event* después de recorrer el eje (i,j), y se definen en función de los ejes de trenes y no en función de las estaciones, esto permite modelar los casos donde un tren pasa múltiples veces por la misma estación. En cambio, esta formulación no captura el hecho de que un tren pase

múltiples veces por el mismo eje. Se agregó como corrección la restricción 4.17 al problema esclavo para no permitir la creación de este tipo de ruta, ya que no serían correctamente modeladas. Por último, al igual que en problema maestro, las variables λ_p representarán si se usará la ruta p y las variables σ_{mn} y ϕ_i representarán el desbalance de secciones de tripulación y de trenes respectivamente.

Conjuntos	
A_1^b	Conjunto de ejes de tren que pueden ser usados por el bloque $b \in B$. $A_1^b \subset A_1$
A_2^b	Conjunto de ejes *swap* que pueden ser usados por el bloque $b \in B$. $A_2^b \subset A_2$
A^b	Conjunto de todos los ejes que pueden ser usados por el bloque $b \in B$. $A^b = A_1^b \cup A_2^b$
A_p	Conjunto de todos los ejes asociados a la ruta $p \in P$. $A_p \subset A_1$
Variables de decisión	
λ_p	1 si se usa la ruta $p \in P$; 0 en caso contrario
y_{ij}^b	1 si el eje $(i,j) \in A^b$ es usado por el bloque $b \in B$; 0 en caso contrario
z_{ij}	1 si la ruta $p \in P$ tiene un *work event* justo después de recorrer el eje $(i,j) \in A_p$ (el *work event* ocurre en el nodo j); 0 en caso contrario
σ_{mn}	$\mathbb{Z}^+$, cantidad de desbalance de la tripulación $(m,n) \in C$
ϕ_i	$\mathbb{Z}^+$, cantidad de desbalance de trenes en la estación $i \in V$
Parámetros	
c_4	Costo por *work event*
c_5^i	Costo por *block swap* en la estación $i \in V$
c_0	Costo por vagón no entregado
o^b	Origen del bloque $b \in B$, con $o^b \in V$
d^b	Destino del bloque $b \in B$, con $d^b \in V$
r	Número máximo de bloques que puede llevar un tren
s^b	Número máximo de *block swaps* que puede hacer el bloque $b \in B$
w	Número máximo de *work events* que puede llevar a cabo un tren

Tab. 4.3: Notaciones para el BTA

Formulación BTA

$$\begin{aligned} \text{mín } & c_1 \sum_{p \in P} \lambda_p + \sum_{p \in P} c_p \lambda_p + c_3 \sum_{b \in B} \sum_{(i,j) \in A_1^b} v_1^b l_{ij} y_{ij}^b + c_4 \sum_{(i,j) \in A_1} z_{ij} \\ & + \sum_{b \in B} \sum_{(i,j) \in A_2^b} c_5^{p^{-(i)}} y_{ij}^b + \tfrac{c_6}{2} \sum_{(m,n) \in C} \sigma_{mn} + c_7 \sum_{i \in V} \phi_i + c_0 \sum_{b \in B} v_1^b \tau^b \end{aligned} \tag{4.20}$$

sujeto a

$$\sum_{i \in N | p^{-(i)} = o^b} \sum_{j \in N | (i,j) \in A_1^b} y_{ij}^b + \tau^b = 1 \qquad \forall b \in B \tag{4.21}$$

$$\sum_{i\in N|p^{-(i)}=o^b} \sum_{j\in N|(j,i)\in A_1^b} y_{ji}^b = 0 \qquad \forall b \in B \tag{4.22}$$

$$\sum_{j\in N|(i,j)\in A_1^b} y_{ij}^b - \sum_{j\in N|(j,i)\in A_1^b} y_{ji}^b = 0 \qquad \forall b \in B, \forall i \in N|p^{-(i)} \neq o^b, d^b \tag{4.23}$$

$$\sum_{j\in N|p^{-(j)}=d^b} \sum_{i\in N|(i,j)\in A_1^b} y_{ij}^b + \tau^b = 1 \qquad \forall b \in B \tag{4.24}$$

$$\sum_{j\in N|p^{-(j)}=d^b} \sum_{i\in N|(j,i)\in A_1^b} y_{ji}^b = 0 \qquad \forall b \in B \tag{4.25}$$

$$\sum_{b\in B} y_{ij}^b \leq r\lambda_p \qquad \forall p \in P, \forall (i,j) \in A_p \tag{4.26}$$

$$\sum_{(i,j)\in A_2^b} y_{ij}^b \leq s_b \qquad \forall b \in B \tag{4.27}$$

$$z_{ij} \geq y_{ij}^b - y_{jk}^b \qquad \begin{array}{c} \forall b \in B, \forall p \in P, \forall (i,j),(j,k) \in A_p| \\ y_{pj}^2 \neq 1 \wedge (i,j),(j,k) \textit{ consecutivos en } p \end{array} \tag{4.28}$$

$$z_{ij} \geq y_{jk}^b - y_{ij}^b \qquad \begin{array}{c} \forall b \in B, \forall p \in P, \forall (i,j),(j,k) \in A_p| \\ y_{pj}^1 \neq 1 \wedge (i,j),(j,k) \textit{ consecutivos en } p \end{array} \tag{4.29}$$

$$\sum_{(i,j)\in A^p} z_{ij} \leq w \qquad \forall p \in P \tag{4.30}$$

$$\sum_{p\in P} \alpha_{pij}\lambda_p + \sum_{p\in P} \alpha_{pji}\lambda_p \leq M_{ij}^1 \qquad \forall (i,j) \in E|i<j \tag{4.31}$$

$$\sum_{b\in B} v_2^b y_{ij} \leq M_{ij}^2 \qquad \forall (i,j) \in A_1 \tag{4.32}$$

$$\sum_{b\in B} v_3^b y_{ij} \leq M_{ij}^3 \qquad \forall (i,j) \in A_1 \tag{4.33}$$

$$\sigma_{mn} + \sum_{p\in P} \beta_{pmn}\lambda_p - \sum_{p\in P} \beta_{pnm}\lambda_p \geq 0 \qquad \forall (m,n) \in C \tag{4.34}$$

$$\sigma_{mn} - \sum_{p\in P} \beta_{pmn}\lambda_p + \sum_{p\in P} \beta_{pnm}\lambda_p \geq 0 \qquad \forall (m,n) \in C \tag{4.35}$$

$$\phi_i + \sum_{p\in P} \gamma_{pi}^1\lambda_p - \sum_{p\in P} \gamma_{pi}^2\lambda_p \geq 0 \qquad \forall i \in V \tag{4.36}$$

$$\phi_i - \sum_{p\in P} \gamma_{pi}^1\lambda_p + \sum_{p\in P} \gamma_{pi}^2\lambda_p \geq 0 \qquad \forall i \in V \tag{4.37}$$

$$\lambda_p, y_{ij}^b, z_{ij}, \tau^b \in \{0,1\} \tag{4.38}$$

$$\sigma_{ij}, \phi^i \in \mathbb{Z}^+ \tag{4.39}$$

La función objetivo 4.20 minimiza la suma de los siguientes costos:

- Costo fijo por tren: $c_1 \sum_{p \in P} \lambda_p$
- Costo por milla recorrida por tren: $\sum_{p \in P} c_p \lambda_p$
- Costo por milla recorrida por vagón: $c_3 \sum_{b \in B} \sum_{(i,j) \in A_1^b} v_1^b l_{ij} y_{ij}^b$
- Costo de los *work events*: $c_4 \sum_{(i,j) \in A_1} z_{ij}$
- Costo de los *block swaps*: $\sum_{b \in B} \sum_{(i,j) \in A_2^b} c_5^{p^{-(i)}} y_{ij}^b$
- Costo del desbalance de tripulación: $\frac{C_6}{2} \sum_{(m,n) \in C} \sigma_{mn}$
- Costo del desbalance de trenes: $c_7 \sum_{i \in V} \phi_i$
- Penalidad por bloque no entregados: $c_0 \sum_{b \in B} v_1^b \tau^b$

Las restricciones 4.21-4.25 garantizan la conservación del flujo de los bloques. Cambiamos las restricciones 4.21 y 4.24 originales del BTA:

$$\sum_{j \in N | (i,j) \in A_1^b} y_{ij}^b + \tau^b = 1, \ \forall b \in B, \forall i \in N | p^{-(i)} = o^b$$

$$\sum_{j \in N | (i,j) \in A_1^b} y_{ij}^b + \tau^b = 1, \ \forall b \in B, \forall i \in N | p^{-(i)} = d^b$$

Pasamos el "$\forall i \in N | p^{-(i)} = o^b$" a una sumatoria dentro de la restricción de la forma "$\sum_{i \in N | p^{-(i)} = o^b}$". Realizamos un cambio similar en la restricción 4.24 pero en vez de o^b, será d^b. Esto es necesario, ya que sino estas restricciones representarían que un bloque debe ser enviado por absolutamente todos los ejes posibles que nacen de su origen y que llegan a su destino, o bien, no enviado. Esto llevaba en la práctica a que siempre convenga no transportar el bloque. Con esta corrección, el bloque será enviado por uno solo de sus caminos posibles, o no enviado. También modificamos de la misma forma las restricciones 4.22 y 4.25, aunque ambas formas de escribir la restricción generan el mismo conjunto de soluciones válidas, decidimos escribirlas de este modo para reducir la cantidad de restricciones (a cambio de que cada restricción tenga mayor cantidad de variables).

La restricción 4.26 define la relación entre λ_p y y_{ij}^b, además garantiza que en ningún momento un tren lleve más bloques que los permitidos. La restricción 4.27 impone la cantidad máxima de *block swaps* por bloque. Por su parte, las restricciones 4.28 y 4.29 definen las variables asociadas a las *work events*, generando una relación entre z_{ij} y y_{ij}^b, en estas restricciones aclaramos que solo se tendrán en cuenta los pares (i,j),(j,k) cuando (i,j) y (j,k) son ejes consecutivos en la ruta p. Sino, esta formulación no sería correcta para rutas que pasen dos veces por la misma estación.

La restricción 4.30 asegura que ningún tren tenga más *work events* que los permitidos. La restricción 4.31 restringe la cantidad de trenes que pasan por una vía, al igual que en

la restricción 4.3 del problema maestro se modificó levemente la restricción para tener en cuenta los trenes que circulan en ambas direcciones de la vía. Las restricciones 4.32 y 4.33 imponen la longitud y el peso máximo de los trenes que circulan por una vía. Las variables asociadas al desbalance de tripulación y trenes se definen en las restricciones 4.34-4.37. Por último, se define el dominio de las variables en las restricciones 4.38 y 4.39.

4.4. Mejoras

Sobre las formulaciones originales presentadas por Jin et al. [6], agregamos algunas mejoras que desccribimos a continuación.

1. Las variables ϕ son usadas para calcular el desbalance de trenes, tanto el desbalance positivo como el desbalance negativo. Pero no es necesario calcular ambos, ya que siempre serán iguales porque por cada tren que aporta un desbalance positivo en una estación (estación en la que finalizó, pero que no era su estación inicial) también aportará necesariamente un desbalance negativo en su estación de origen. Por esto, podremos prescindir de la restricción 4.8 en el *Master Problem* y 4.36 en el BTA, que calculan el desbalance negativo de trenes y cambiar el funcional de las variables ϕ de c_7 a $2 \times c_7$. También habrá que ajustar el *Slave Problem* al nuevo costo reducido del *Master Problem*. Esto reducirá las restricciones en la formulación del *Master Problem* y en el BTA.

2. Algo similar sucede con las variables σ_{mn}, se genera una por cada sentido del segmento de tripulación, y por cada una de ellas se calcula el desbalance de tripulación de ese segmento. Pero al tener dos variables por cada segmento es necesario dividir el costo de penalización por dos. En cambio, proponemos solo crear una variable por segmento de tripulación y cambiar el funcional de las variables σ_{mn} de $\frac{c_6}{2}$ por c_6. Esto nos ayudará a reducir a la mitad la cantidad de restricciones asociadas al cálculo del desbalance de los segmentos de tripulación. Similar al caso anterior, esta optimización se puede aplicar tanto al *Master Problem* como al BTA, y habrá que ajustar la formulación del *Slave Problem* al nuevo costo reducido.

3. La desventaja de agregar cortes de sub-tour luego de resolver el problema esclavo es que se deberá optimizar el problema múltiples veces, una vez por cada sub-tour encontrado. Una forma de reducir la cantidad de iteraciones es agregar desde el inicio ciertos cortes de sub-tour que tienen alta probabilidad de aparecer. Por esto agregamos un conjuntos de cortes iniciales con el objetivo de quitar de las soluciones factibles cierto conjunto especial de ciclos que se encontraron como problemáticos en la experimentación. Quitamos todos los ciclos de longitud 2 que consistan en los dos ejes de un mismo segmento de tripulación pero en sentidos opuestos. Para esto agregamos la siguiente restricción:

$$x_{mn} + x_{nm} \leq 1, \forall (m,n) \in C. \tag{4.40}$$

Agregar cortes de sub-tour desde el inicio supone tener en cuenta dos implicancias. Por un lado, tiene el potencial de reducir la cantidad veces que se resolverá el problema esclavo, pero por el otro, aumenta la cantidad de restricciones, por lo que incrementa la complejidad de la resolución. Llevaremos a cabo un análisis del impacto de estos cortes en la sección 6.1.

4. Modificamos la restricción 4.16 para que la sumatoria sea sobre C_0 en vez de sobre C para evitar generar rutas que terminen en una estación por la que ya habían pasado, ya que la restricción original permite rutas del estilo $A \to B \to C \to B$.

5. Además, agregamos al *Master Problem* una restricción sobre la cantidad máxima de bloques por tren.

$$\sum_{b \in B} \sum_{q \in Q} \alpha_{qij} \delta_q^b + \sum_{p \in P} -r\alpha_{pij} \lambda_p \leq 0, \forall (i,j) \in E | i < j. \tag{4.41}$$

   Como en el caso del peso y la longitud máxima, esta restricción se aplica en promedio, ya que en esta instancia no tenemos una asignación para saber en qué tren circulará cada bloque. Esta restricción ya se aplica de forma estricta en el BTA, por lo que agregarla al *Master Problem* no cambiará la factibilidad del resultado, sin embargo, sí puede ayudar a crear mejores rutas en la generación de columnas. Analizaremos el impacto de esta nueva restricción en la sección 6.3

5. ALGORITMOS ITERATIVOS PARA TDO

Del proceso propuesto por Jin et al. [6] para resolver el TDO, el BTA es la etapa más compleja y requiere la mayor parte del tiempo de cómputo. Esto nos impone un número máximo de rutas que podemos considerar si queremos alcanzar una buena solución en una hora de cómputo. Por esto decidimos analizar distintos enfoques para mejorar el tiempo de cómputo del BTA, y así poder resolverlo teniendo en cuenta una mayor cantidad de rutas, con el objetivo de explorar un mayor espacio de soluciones y lograr conseguir una solución de mayor calidad.

Para lograr reducir esta complejidad, decidimos subdividir el BTA en subproblemas de menor tamaño. Esto reducirá considerablemente el tiempo de procesamiento, ya que en un problema de complejidad exponencial resolver n problemas de tamaño $\frac{x}{n}$ será considerablemente más rápido que resolver un problema de tamaño x.

5.1. Algoritmo Iterativo Por Bloques

Nuestro primer algoritmo iterativo consiste en subdividir los bloques en n grupos de igual tamaño. Luego, resolver el TDO sobre el primer subconjunto de bloques, usando nuestra formulación expuesta en el capítulo 4. A continuación, tomar el siguiente grupo de bloques y volver a resolver el TDO sobre este nuevo subconjunto, pero en este caso, teniendo en cuenta la solución parcial de la iteración anterior. Para esto último, se debe marcar las rutas usadas en la solución parcial anterior, como rutas que se deben utilizar en esta nueva iteración, y descontar el peso y la longitud ya utilizada por los bloques en la solución parcial anterior de la capacidad de esas rutas. Así construiremos una nueva solución parcial. Este proceso deberá repetirse para cada subconjunto de bloques restante y la solución del último subconjunto será la solución final.

Algorithm 1 Algoritmo Iterativo por Bloques

1: Particionar los bloques en n subconjuntos
2: Nuestra solución acumulada empezará siendo una solución vacía
3: **for** cada subconjunto en la partición **do**
4: Resolver TDO para ese subconjunto de bloques teniendo en cuenta la solución acumulada
5: Unir esta solución parcial con la solución acumulada
6: **return** solución acumulada

A continuación desarrollaremos las decisiones implementativas y los parámetros de nuestra implementación. La primera decisión que tomamos es particionar los bloques en conjuntos de igual cantidad de bloques. Tomaremos como parámetro la cantidad de subconjuntos en los que se particionarán los bloques y una función para ordenarlos al momento de realizar la partición. Para particionar $\#B$ bloques en n subconjuntos, ordenaremos los bloques usando la función de orden recibida como parámetro, el primer conjunto estará compuesto por los bloques en las posiciones 1 a $\frac{\#B}{n}$, el segundo grupo por los bloques en la posiciones $\frac{\#B}{n} + 1$ a $\frac{2\#B}{n}$ y así sucesivamente hasta conseguir los n subconjuntos.

Al crear una mayor cantidad de conjuntos, estos contendrán una menor cantidad de bloques. A menor cantidad de conjuntos, el algoritmo tenderá a comportarse como nuestra formulación base del TDO (en el caso extremo de usar un solo conjunto, ambos algoritmos serán iguales). Por otro lado, a mayor cantidad de conjuntos, el algoritmo requerirá menor cantidad de tiempo, a cambio de explorar un espacio de soluciones más acotado y restringido.

Tanto la cantidad de conjuntos de la partición como la función de orden con la que se particionará, serán determinantes para la calidad de la solución. Esto ocurre porque el primer conjunto de bloques tendrá toda la capacidad de la red disponible para utilizar, pero los subconjuntos subsiguientes tendrán una capacidad cada vez más limitada.

Al momento de decidir qué función de orden elegir para los bloques, existen muchas opciones. Una alternativa es ordenar en función de la penalidad por no transportar el bloque, lo que sería ordenar en función de la cantidad de vagones del bloque. Otra opción es ordenar en función de cuanta capacidad de la red consumen, pudiendo ordenarse tanto por peso o longitud del bloque o alguna combinación de ambos. También se puede armar un orden que sea un cociente entre la penalidad por no llevar un bloque y la capacidad que consume llevarlo. Por último, también se puede ordenar los bloques al azar para determinar que el orden de los mismos no es importante.

Como ya se mencionó en la sección 3.1.4, los bloques de los DataSets de la competencia tienen la particularidad de tener una relación casi lineal entre cantidad de vagones, peso y longitud. Esto es particularmente provechoso al momento de elegir el orden en el que se particionan los bloques, ya que genera que sea equivalente ordenar tanto por número de vagones como por peso, por longitud o cualquier combinación de los tres. Sin esta correlación, esos tres órdenes serían distintos y habría que experimentar con todos ellos (tanto de forma ascendente como descendente). Esto nos ayuda a reducir considerablemente los órdenes con los que experimentaremos.

Probamos las siguientes formas de particionar los bloques:

- Ordenado por cantidad de vagones de mayor a menor
- Ordenado por cantidad de vagones de menor a mayor
- Ordenado aleatoriamente

Ordenar los bloques en función de su cantidad de vagones es razonable, ya que la cantidad de vagones en un bloque está linealmente relacionada con la penalización por no llevar ese bloque y con la capacidad necesaria para llevarlo dentro de la red.

Evaluando la diferencia entre las tres formas de ordenar los bloques, llegamos a la conclusión de que la mejor es ordenarlos de mayor a menor en función del número de vagones. La experimentación en la que se basa esta conclusión está expuesta en la sección 6.2. La intuición de este orden es intentar llevar primero los bloques más “dificiles”, mayor cantidad de vagones, peso y longitud y así evitarnos las penalidades más altas por no llevar bloques, ya que estas penalidades dependen de la cantidad de vagones.

La cantidad de conjuntos en los que se particionará (n) y la función de orden de los bloques se suman a los parámetros necesarios para resolver nuestra implementación del TDO. Necesitaremos decidir la cantidad de rutas que se crearán en cada iteración, el $Lmax$ y el k. Los parámetros $Lmax$ y k, serán idénticos a los de TDO, $Lmax$ representará la longitud máxima de las rutas creadas en la generación de columnas y k la cantidad de *block paths* que se construirán para cada bloque. Para el caso de la cantidad de rutas a generar

en cada resolución del TDO, en vez de determinar el número de rutas total, se elegirá la proporción de rutas generadas en función de la cantidad de bloques de la instancia parcial que se está resolviendo. Por ejemplo, si se elige 2 como proporción entre rutas y bloques, por cada bloque en la solución parcial, se crearán 2 rutas. En una solución parcial con 50 bloques, se crearán 100 rutas. Elegimos definir este parámetro de forma relativa, en vez de absoluta, para comparar más fácilmente experimentaciones que particionen en diferentes cantidades de conjuntos.

5.2. Algoritmo Iterativo Por Rutas

Desarrollamos un segundo algoritmo iterativo, siguiendo el criterio planteado anteriormente en el cual para resolver un problema exponencial en menos tiempo, se puede resolver múltiples subproblemas de menor tamaño. En este caso optamos por reducir las instancias en el número de rutas de cada iteración, en vez de en el número de bloques.

Para esto definiremos, al igual que en el método base, una cantidad máxima de rutas a generar $MaxRutas$ y una cantidad de iteraciones n. Luego resolveremos n veces TDO generando en cada iteración $\frac{MaxRutas}{n}$ rutas y teniendo en cuenta la solución parcial de la iteración anterior (de la misma manera que hicimos en el algoritmo anterior).

Algorithm 2 Algoritmo Iterativo por Rutas

1: La solución acumulada empezará siendo una solución vacía
2: **for** n veces **do**
3: Resolver TDO usando a lo sumo $\frac{MaxRutas}{n}$ rutas y teniendo en cuenta la solución acumulada
4: Unir esta solución parcial con la solución acumulada
5: **return** solución acumulada

6. EXPERIMENTACIÓN TDO

El código necesario para resolver TDO y todos los problemas relacionados tratados en esta tesis fue implementado utilizando C++ como lenguaje de programación e ILOG CPLEX 12.6 como nuestro software de optimización. Todos los experimentos que desarrollamos a continuación fueron ejecutados en una computadora con un procesador Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz con 16GB de memoria.

Todos nuestros experimentos fueron ejecutados con un tiempo límite de cómputo de una hora por instancia, salvo que se especifique lo contrario.

6.1. Experimentación agregando cortes de subtour iniciales en el problema esclavo

Como ya mencionamos anteriormente, al experimentar reconocimos que los cortes de subtour que agregaba el *slave problem* en las primeras iteraciones eran similares en todos los experimentos. Estos cortes consisten en ciclos de dos ejes de un mismo segmento de tripulación en sus dos sentidos. Siempre existirá uno de estos ciclos por cada segmento de tripulación.

Como observamos que el *slave problem* dedicaba múltiples iteraciones en identificar y cortar estos ciclos, decidimos agregar explícitamente a la formulación del *slave problem* cortes para todos ellos.

Esta optimización podría tener dos consecuencias, o bien, reducir el tiempo de cómputo del *slave problem*, ya que no serán necesarias las múltiples iteraciones iniciales para cortar estos ciclos, o en cambio, un aumento del tiempo de cómputo, ya que resolveremos múltiples veces una formulación con una mayor cantidad de restricciones.

Para entender si esta optimización mejora el tiempo de cómputo de la generación de columnas, realizamos la siguiente experimentación. Resolvimos la generación de columnas para ambos DataSet con k igual 3, 5 y 7 y $Lmax$ igual a 4, 7 y 10. De esas corridas tomamos la cantidad total de cortes de subtour que se agregaron y el tiempo de cómputo del proceso de generación de columnas completo.

Al analizar los resultados del DataSet1 en el gráfico 6.1 vemos que la cantidad de cortes que se necesitó agregar durante la generación de columnas disminuyó drásticamente, lo cual era esperable, ya que pre-agregamos cortes al problema esclavo antes de empezar el procesamiento. Pero para asegurarnos que esta optimización sea valiosa es necesario entender si reduce el tiempo de la generación de columnas. Viendo el gráfico 6.2 se comprueba que se produjo una reducción del tiempo de cómputo significativa en los casos de mayor tamaño, que llegó a alcanzar una reducción de más del 45 % en el mejor caso.

Al comparar ambas figuras, se observa gráficamente que la reducción de la cantidad de cortes agregados al resolver la generación de columnas no garantiza una reducción semejante del tiempo de cómputo. Por ejemplo, para el caso del DataSet1 $Lmax$ 10 y k 7, se redujo más de un 73 % la cantidad de cortes agregados y eso logró reducir el tiempo en un 45 %. Pero por otro lado, en el caso del DataSet1 con $Lmax$ 4 y k 3, se redujo en un 80 % el número de cortes agregados y solo conllevó una reducción del 12 % del tiempo de cómputo.

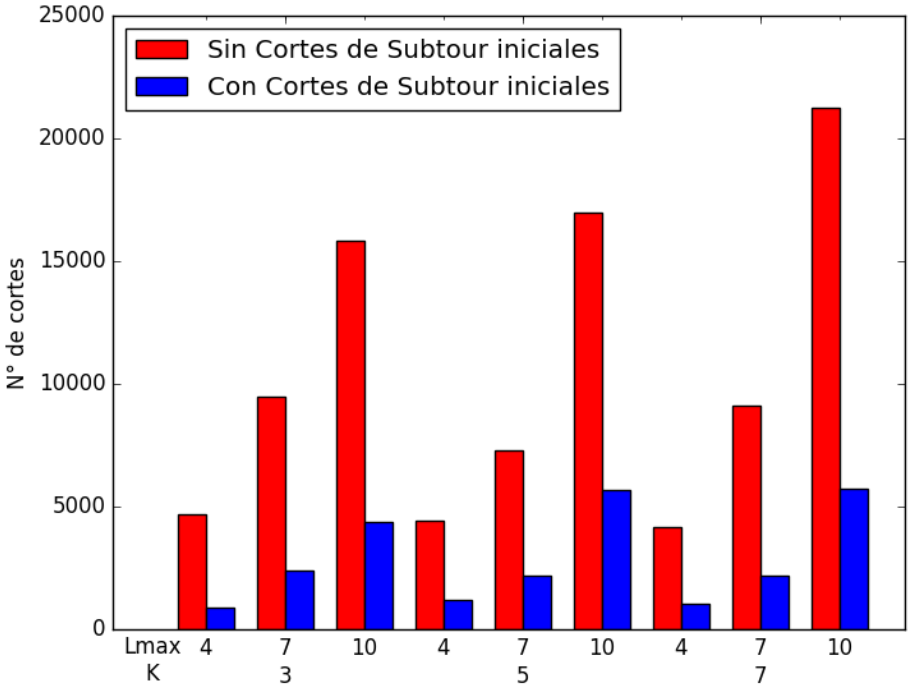

Fig. 6.1: Cantidad de cortes subtour totales en DataSet1

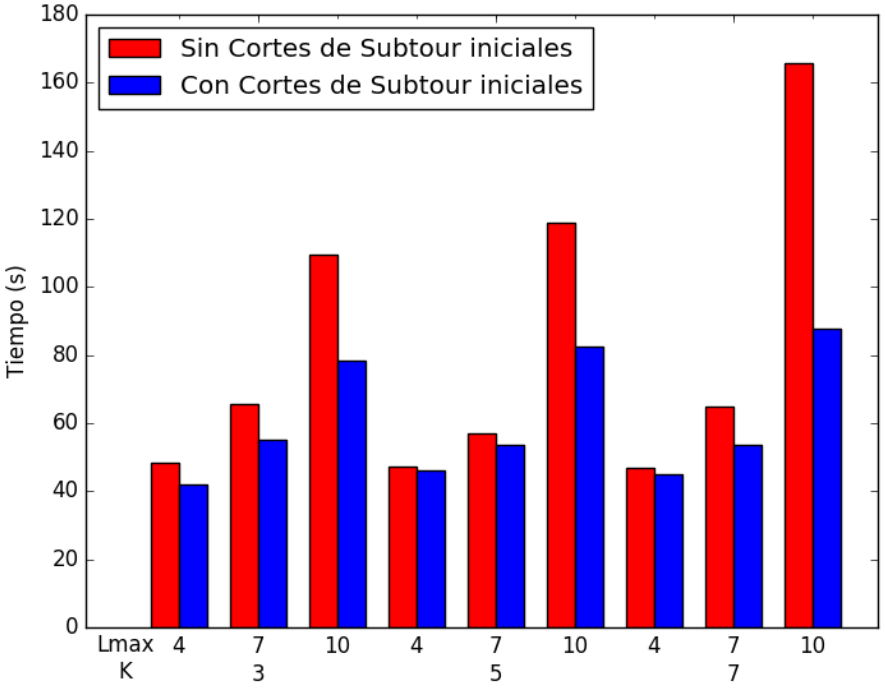

Fig. 6.2: Tiempo Total de generación de columnas en DataSet1

Al observar los gráficos 6.3 y 6.4 podemos ver que en el DataSet2 se repite el mismo comportamiento encontrado en el DataSet1, alcanzando en el mejor caso una reducción del tiempo de computo del 28 %.

En todos los experimentos subsiguientes se aplicará esta mejora.

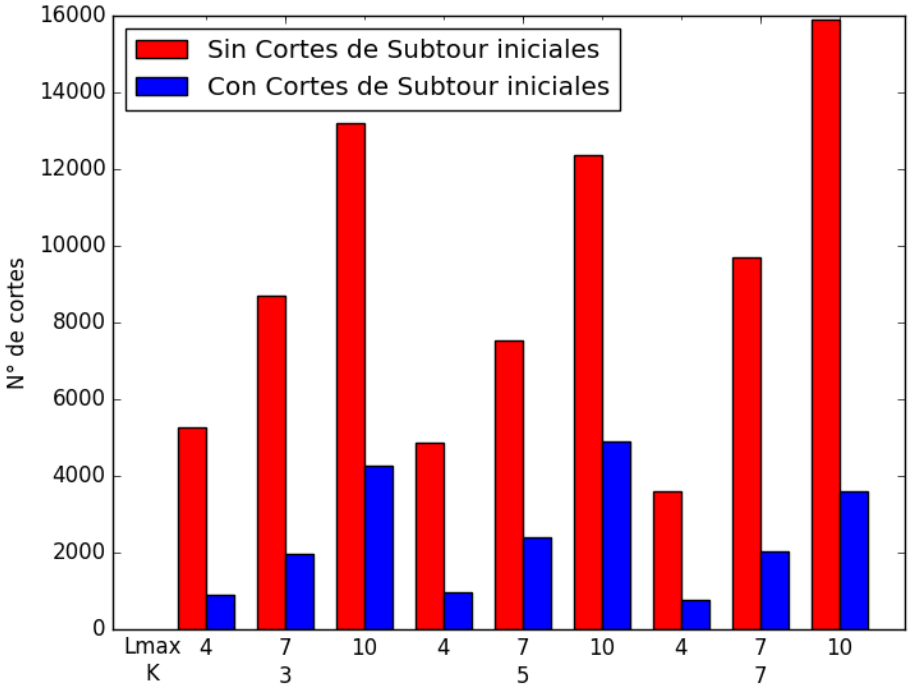

Fig. 6.3: Cantidad de cortes subtour totales en DataSet2

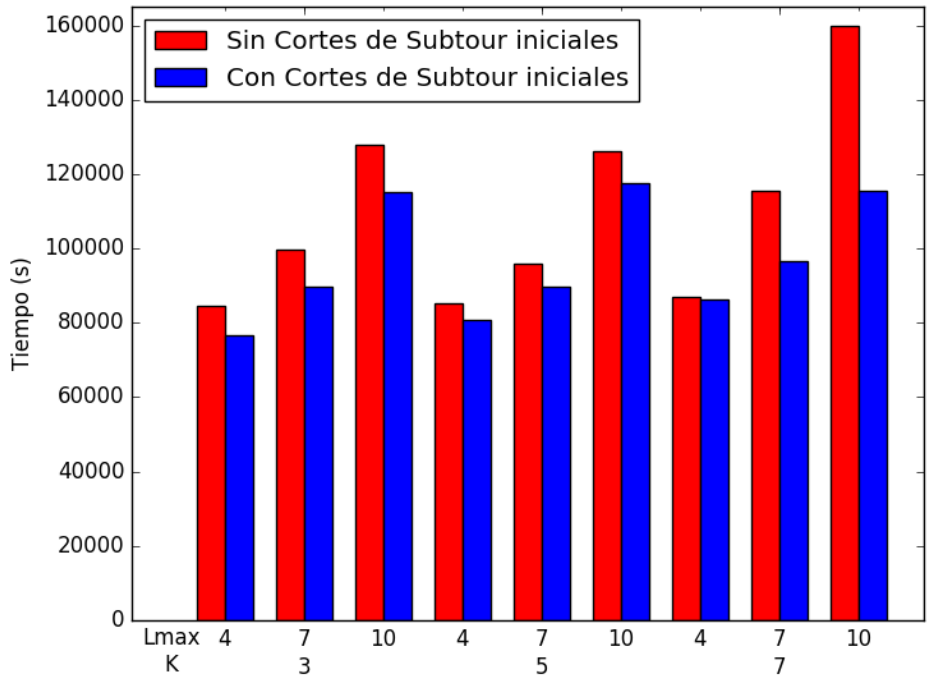

Fig. 6.4: Tiempo Total de generación de columnas en DataSet2

6.2. Comparación del impacto de la función de orden de los bloques en el algoritmo iterativo por bloques

En el algoritmo iterativo por bloques presentado en la sección 5.1, una de las decisiones implementativas más importante fue la elección de la función de orden de los bloques. Como explicamos previamente, proponemos tres órdenes distintos a tener en cuenta: ordenar los bloques en forma ascendente o descendente según su cantidad de vagones, o bien, usar un orden aleatorio.

Para comparar las diferentes funciones propuestas, se experimentó resolviendo el TDO usando el algoritmo iterativo por bloques en ambos DataSets, variando tanto la función de orden usada como el resto de los parámetros del algoritmo.

A continuación compararemos tanto la calidad de la solución como el tiempo usado para obtenerla. Para esto, en el DataSet1 utilizaremos n igual a 3, 1.2 rutas por bloques, $Lmax$ igual a 7 y k igual a 5 y 7. En el caso del DataSet2 usaremos n igual a 5, 1.2 rutas por bloques, $Lmax$ igual a 7 y k igual a 5 y 7.

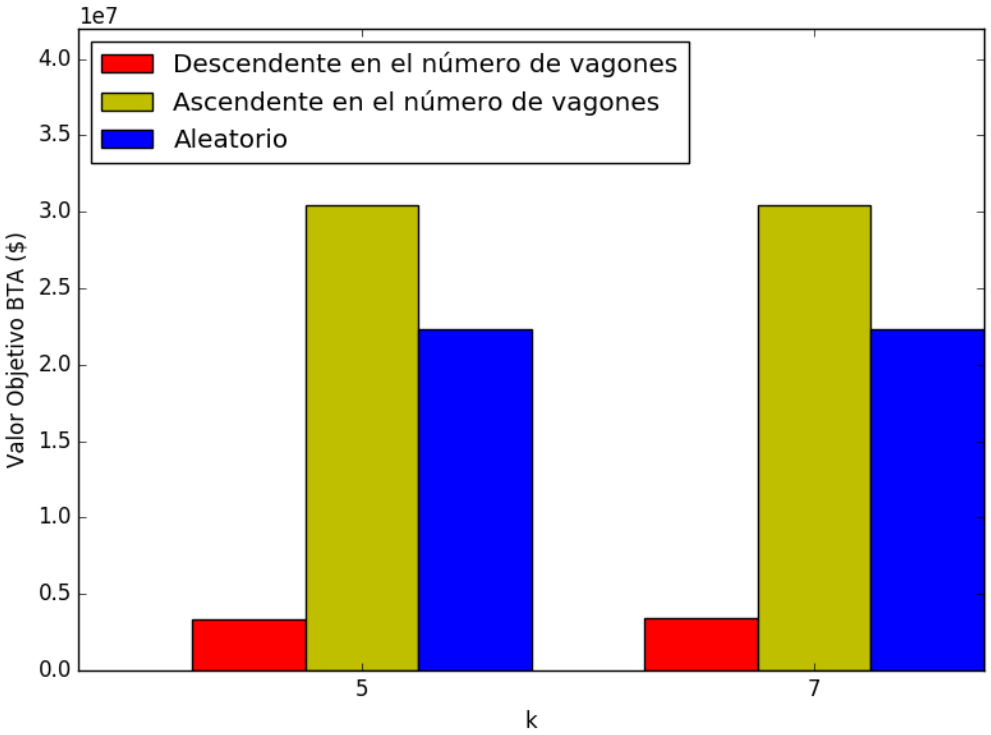

Fig. 6.5: Calidad BTA en función del orden de los bloques en DataSet1

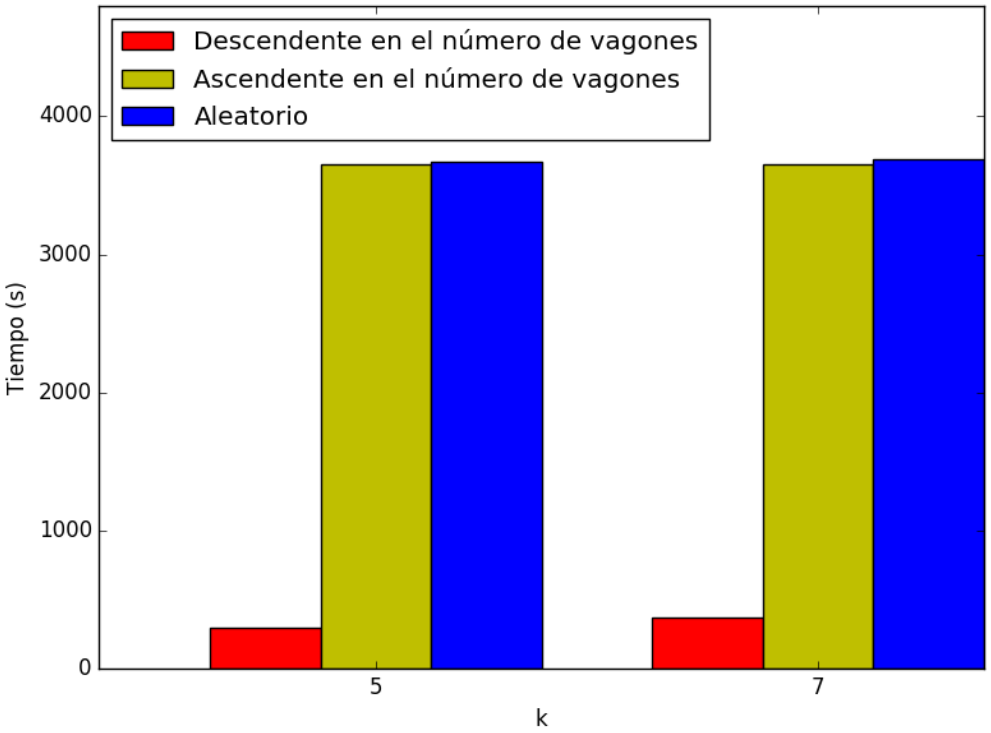

Fig. 6.6: Tiempo Total en función del orden de los bloques en DataSet1

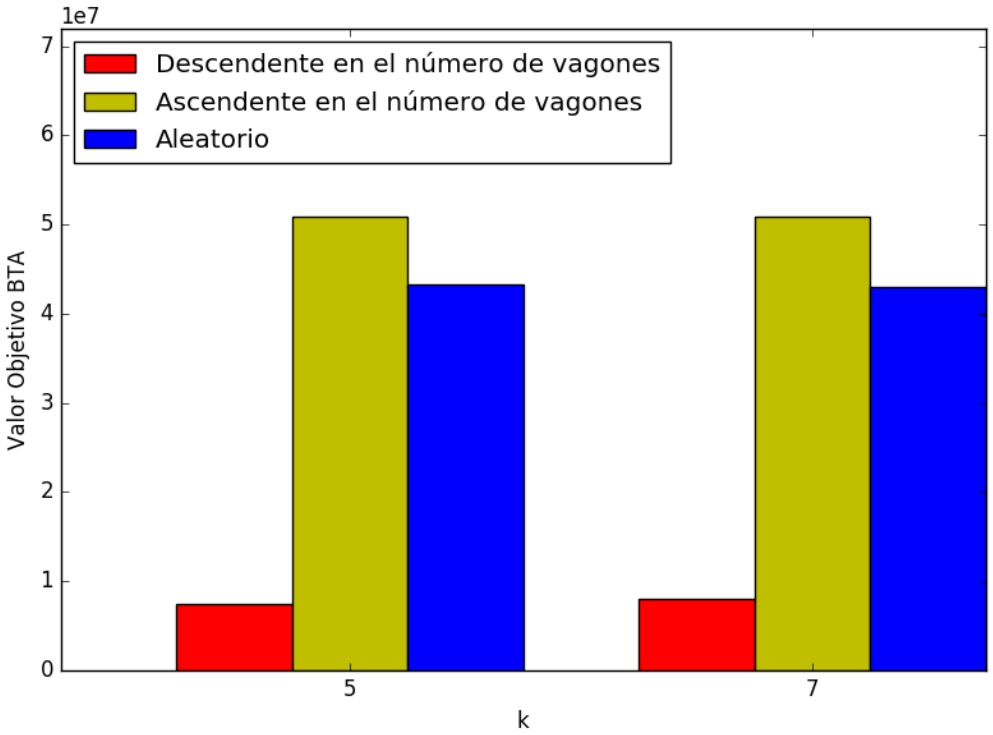

Fig. 6.7: Calidad BTA en función del orden de los bloques en DataSet2

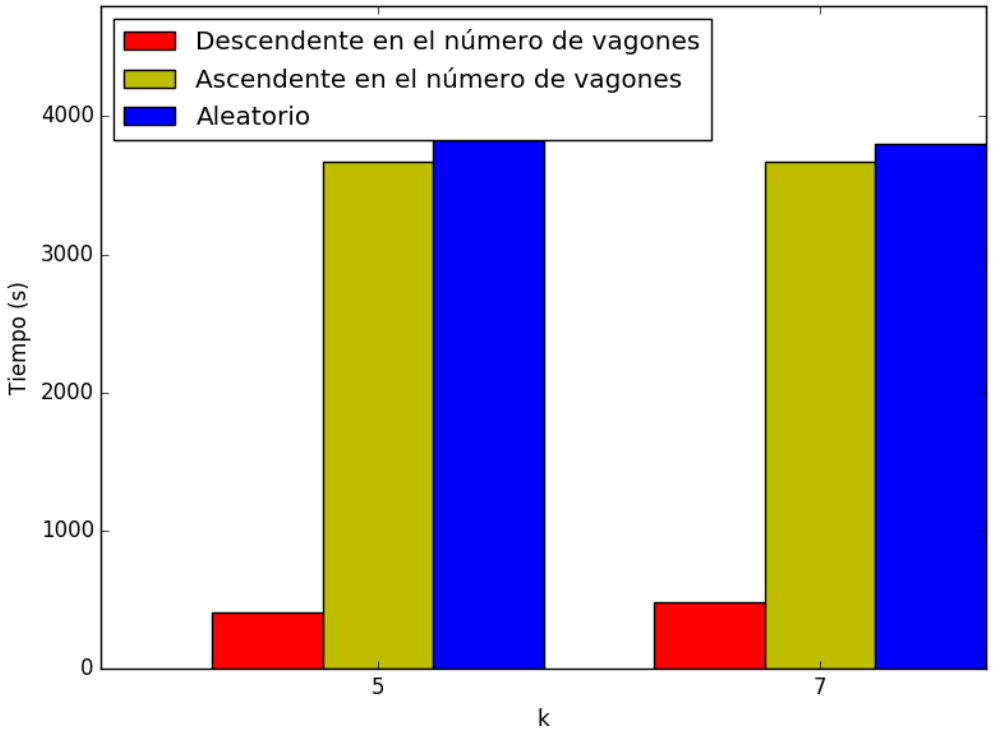

Fig. 6.8: Tiempo Total en función del orden de los bloques en DataSet2

En los gráficos 6.5-6.8 observamos que ordenar de manera descendente en el número de vagones obtuvo una solución mejor tanto en calidad como en tiempo en ambos DataSets. La mejor calidad de solución se debe a que se da prioridad a enviar los bloques con mayor cantidad de vagones, ya que estos son los primeros a los que se les asigna una ruta, cuando todavía toda la capacidad de la red está disponible. En las iteraciones subsiguientes se buscará asignar una ruta a bloques con menor cantidad de vagones, pero parte de la capacidad de la red ya estará ocupada por los primeros bloques. Así, en cada iteración, los bloques tendrán una mayor posibilidad de no ser entregados o que se le asigne una ruta sub-óptima. Los bloques de mayor cantidad de vagones son más difíciles de llevar, ya que consumen mayor capacidad de la red, y a la vez tienen una mayor penalización al no ser enviados.

En cuanto al tiempo de cómputo, vemos que tanto ordenar de manera ascendente en el número de vagones, como de forma aleatoria, llegan al límite de tiempo impuesto de una hora, en cambio el orden descendente es 10 veces más rápido tardando alrededor de 6 minutos. Suponemos que esto se debe a que las primeras iteraciones trasladarán a los bloques más pequeños, y en las iteraciones subsiguientes, los bloques grandes no tendrán disponible capacidad para ser trasladados, lo que generará que ese subproblema no logre encontrar el óptimo y requiera toda el tiempo de cómputo asignado.

6.3. Impacto de las mejoras propuestas para resolver el TDO

En la sección 4.4 presentamos una series de mejoras a la formulación original de Jin et. al [6] para resolver el TDO. Para entender el impacto de estas mejoras compararemos tres versiones de este método. La primera será la versión original propuesta por Jin et al. La segunda versión será la implementación original agregándole todas las mejoras propuestas en la sección 4.4. Por último, la tercera versión será igual que la anterior pero exceptuando la propuesta 5 (restricción del número de bloques por tren en el *Master Problem*). A estas tres versiones las llamaremos *Original*, *Mejorada* y *Mejoras Parciales* respectivamente.

Para compararlas resolvimos ambos DataSets con cada una de estas versiones usando todas las combinaciones posibles de los parámetros: número máximo de rutas 100 y 160; *Lmax* 2 y 3; y k 3 y 4. Elegimos estos parámetros para comparar nuestras mejoras con la versión original de Jin et al. [6] ya que son algunos de los mejores parámetros que seleccionaron luego de experimentar. A continuación, compararemos la mejor solución de cada versión para cada DataSet.

En todos los casos, la mejor solución utilizó 160 rutas. En el caso de la versión *Original* la mejor configuración de parámetros para ambos DataSet fue *Lmax* 3 y k 3. Los mejores parámetros para la versión *Mejorada* en ambos DataSets fue *Lmax* 3 y k 4. Para la versión *Mejoras Parciales* los mejores parámetros fueron *Lmax* 3 y k 3. Además, compararemos una segunda solución del DataSet1 usando la versión *Mejoras Parciales*, ya que aunque su costo es marginalmente superior que la mejor solución encontrada con este método, requirió significativamente menos tiempo de cómputo. Para esta solución se usó un *Lmax* de 2 y un k de 3 y la llamaremos *Mejoras Parciales Balanceada*.

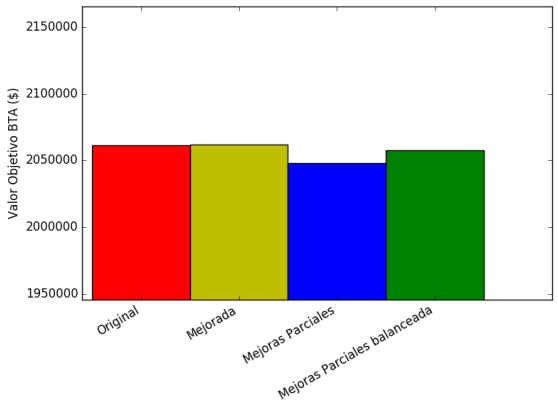

Fig. 6.9: Calidad de solución en DataSet1

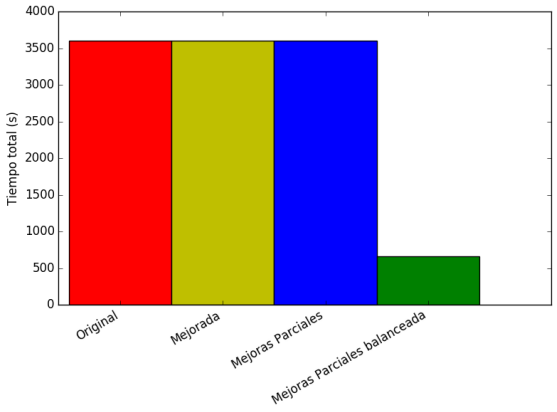

Fig. 6.10: Tiempo de ejecución en DataSet1

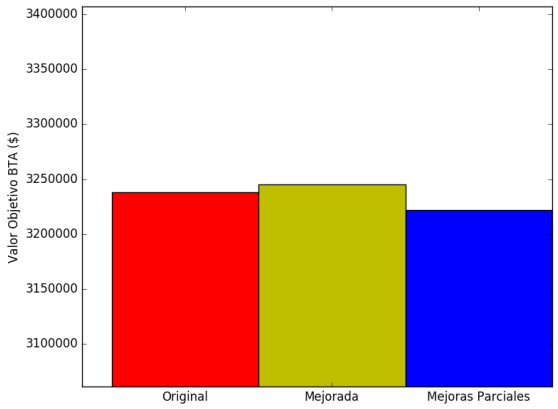

Fig. 6.11: Calidad de solución en DataSet2

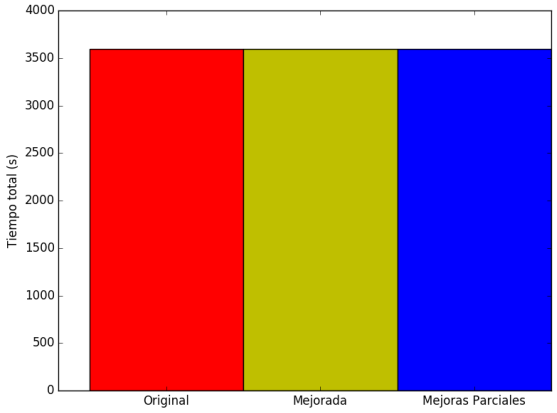

Fig. 6.12: Tiempo de ejecución en DataSet2

Como podemos ver en los gráficos 6.9-6.12, la calidad de la solución de todas las versiones son similares. Siendo la versión *Mejoras Parciales* la mejor en ambos DataSets, ya que en el DataSet1 su función objetivo es 0.6 % mejor que la versión *Original* y en el DataSet2 es 0.5 % mejor. Para poner en perspectiva esta mejora, es necesario considerar que existe un espacio de mejora limitado entre la solución propuesta por Jin et al. y la solución óptima. Al no conocer el valor de esta solución óptima, utilizaremos la cota inferior calculada en la sección 3.1.5. Calculamos el porcentaje de mejora de la solución de la versión *Mejoras Parciales* sobre la diferencia entre la función objetivo de la solución de Jin et al. y la cota inferior. Esta mejora es del 2.6 % en el DataSet1 y del 2.5 % en el DataSet2.

Otro aspecto a resaltar es que la mejor solución de cada versión utilizó el 100 % del tiempo de cómputo permitido (una hora). Además, queremos destacar otra solución encontrada usando la versión *Mejoras Parciales* en el DataSet1, a la que en los gráficos identificamos como *Mejoras Parciales Balanceada*. La función objetivo de esta solución es un 0.17 % mejor que la versión *Original* aunque es un 0.45 % inferior a la mejor solución encontrada con este método. Queremos destacarla, ya que requirió solo 11 minutos de cómputo, utilizando más de un 80 % menos del tiempo de cómputo que el resto de las soluciones.

Como la versión *Mejoras Parciales* fue marginalmente mejor en la calidad de solución, pero significativamente mejor en tiempo de cómputo, será la que utilizaremos para el resto de las experimentaciones.

6.4. Comparación con algoritmos iterativos

En el capítulo 5 presentamos dos nuevos algoritmos iterativos para resolver TDO. Para comprender las calidad de estos algoritmos los compararemos con el método *Mejoras Parciales*, ya que según lo experimentado en la sección 6.3 es el mejor de las tres formulaciones del TDO. Para comparar estos métodos usaremos las mejores soluciones encontradas usando los algoritmos iterativos y el método *Mejoras Parciales*.

Tanto para el algoritmo iterativo por bloques como para el por rutas, usamos todas las combinaciones posibles de los siguientes parámetros: 3, 5 ó 7 iteraciones; 75 %, 100 % ó 125 % como proporción entre rutas y bloques; 3 ó 4 como $Lmax$; y 5, 7 ó 9 como k. Para el algoritmo iterativo por bloques utilizamos como función de orden, el orden descendente en la cantidad de vagones del bloque, que es la mejor entre las funciones de orden propuesta según la experimentación de la sección 6.2.

En todos los casos, los mejores resultados encontrados usaron 3 iteraciones y 3 como $Lmax$. En el caso del DataSet1, las mejores soluciones de ambos algoritmos usaron una proporción de rutas del 125 %, pero el algoritmo iterativo por bloques usó un k de 5, mientras que el iterativo por rutas, un k de 7. En el caso del DataSet2, las mejores soluciones de ambos algoritmos usaron una proporción de rutas del 100 %, pero el algoritmo iterativo por bloques usó un k de 9, mientras que el iterativo por rutas, un k de 5.

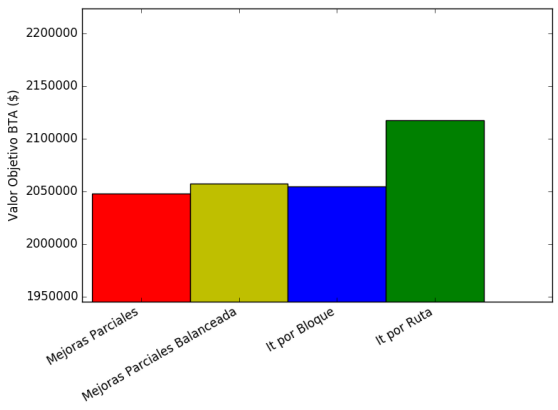

Fig. 6.13: Calidad de solución en DataSet1

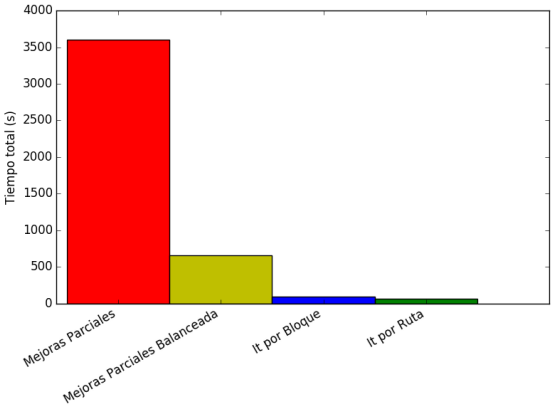

Fig. 6.14: Tiempo de ejecución en DataSet1

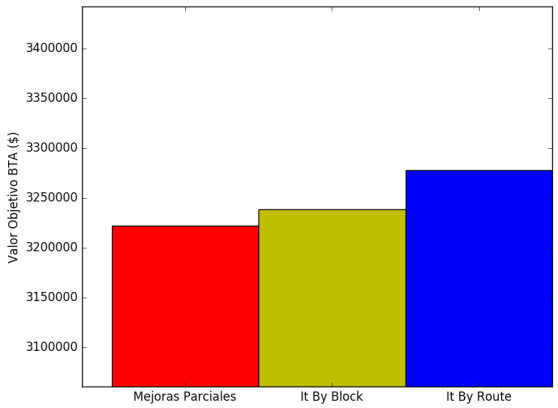

Fig. 6.15: Calidad de solución en DataSet2

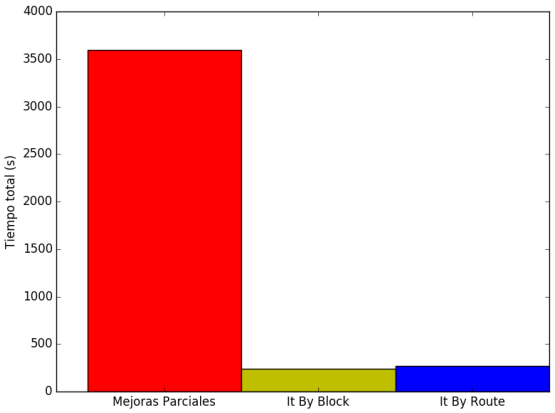

Fig. 6.16: Tiempo de ejecución en DataSet2

Como podemos ver en los gráficos 6.13-6.16, la calidad de la mejor solución del algoritmo iterativo por bloques es similar a la mejor solución encontrada con el método *Mejoras Parciales* (solo un 0.3 % peor en el DataSet1 y un 0.5 % en el DataSet2), pero consiguiendo una mejora muy significativa en el tiempo de cómputo, utilizando solamente 91 segundos para el DataSet1 y 239 en el DataSet2, requiriendo 97 % y 93 % menos de tiempo de cómputo respectivamente. Al comparar esta solución con la solución "balanceada" del método *Mejoras Parciales* vemos que la solución del algoritmo iterativo tiene tanto una solución de mejor calidad, como un menor tiempo de cómputo.

Al analizar el algoritmo iterativo por rutas vemos que requiere un tiempo de ejecución muy bajo, al igual que el algoritmo iterativo por bloques. Sin embargo, la calidad de las soluciones del algoritmo iterativo por rutas son significativamente inferiores a las encontradas tanto por el algoritmo iterativo por bloques como por el método *Mejoras Parciales*.

6.5. Análisis de soluciones Homogéneas

Para entender mejor las características de los DataSet con los que estamos trabajando, analizaremos en detalle una de las mejores soluciones que encontramos usando el método *Mejoras Parciales* y la compararemos con la solución reportada por Jin et. al [6] para cada uno de los DataSets.

6.5.1. DataSet 1

En el caso del DataSet1 compararemos la solución reportada por Jin et al. [6] con una solución conseguida con el método *Mejoras Parciales*. Jin et al. utilizó como parámetros *maxRoutes:120, Lmax:3, k:4* y nosotros *maxRoutes:160, Lmax:3, k:3*. Nuestra solución utiliza 72 rutas con un costo de $2.048.530, mientras que Jin et al. no reporta cuantas rutas usa su solución, pero tiene un costo total de $2.069.752.

	Nuestra Solución	Solución Jin et. al
Costo de inicio de tren	28.800	29.200
Costo de tren por milla	316.161	322.821
Costo de bloque por milla	1.579.280	1.590.491
Costo de *work event*	51.800	53.200
Costo de *block swap*	5.090	5.840.
Costo de desbalance de tripulación	35.400	34.200
Costo de desbalance de trenes	32.000	34.000
Costo por bloques no enviados	0	0
Total	**2.048.530**	**2.069.752**

En este caso nuestra solución reduce casi todos los costos, excepto el de desbalance de tripulación, mostrando el impacto de las optimizaciones planteadas en la sección 4.4.

Análisis de Slacks

- **Bloques por tren:** De los 72 trenes que se usan en nuestra solución, solo un tren lleva su máxima capacidad de bloques y esto ocurre en un solo momento de su recorrido, el resto del recorrido de este tren y los otros 71 trenes tienen capacidad para llevar más bloques[1].
- **Trenes por vía:** Solo 2 de las 134 vías tienen el máximo de trenes circulando por ellas.
- **Block Swaps por bloque:** 7 de los 239 bloques usan la máxima cantidad de *block swaps* permitidos.
- **Work events por tren:** 7 de las 72 rutas usan la máxima cantidad de *work events* permitidos.

Este análisis se puede utilizar para entender y estimar la capacidad de la red de una empresa ferroviaria. Podemos ver que casi todos los trenes en circulación tienen capacidad para transportar más bloques. Además, la red tiene capacidad para que circulen más trenes por lo que la capacidad máxima de la red es considerablemente mayor a la cantidad de bloques que se intenta llevar en esta instancia.

Otra aplicación de este análisis es identificar los puntos críticos de la infraestructura que limitan la capacidad de transporte para así invertir en su mejora. Si vemos que hay muchas vías saturadas, la empresa podría invertir en agregar más carriles a esas vías, en cambio sabremos que no agregará valor invertir en las vías que no están saturadas. También podremos ver si hay una limitación en los trenes, sea tanto de peso, longitud o cantidad de bloques, por lo que se podría invertir en locomotoras de mejor calidad que puedan trasladar más peso y/o longitud.

6.5.2. DataSet 2

En el caso del DataSet2 comparamos la solución reportada por Jin et. al [6] con una solución conseguida con el método *Mejoras Parciales*. Ambas utilizan los parámetros *maxRoutes:160, Lmax:3, k:4*. Nuestra solución utiliza 96 rutas con un costo de $3.228.130,

[1] No saturan la restricción de cantidad de bloques por tren, lo que no quiere decir que tengan capacidad (tanto de peso como de longitud) disponible para llevar un bloque extra

mientras que Jin et. al [6] no reporta cuántas rutas usa su solución, pero tiene un costo total de $3.240.177.

Desglose de solución por tipo de costo

	Nuestra Solución	**Solución Jin et. al**
Costo de inicio de tren	38.000	39.600
Costo de tren por milla	431.159	446.203
Costo de bloque por milla	2.192.470	2.186.604
Costo de *work event*	89.250	90.300
Costo de *block swap*	7.050	6.270
Costo de desbalance de tripulación	22.200	25.200
Costo de desbalance de trenes	28.000	26.000
Costo por bloques no enviados	420.000	420.000
Total	**3.228.130**	**3.240.177**

Se puede ver en la tabla que nuestra solución, aunque tiene un mayor costo de bloque por milla, que es el mayor costo, logra conseguir una solución cuyo costo total es inferior, reduciendo los costos de menor importancia. Esto nos muestra que minimizar el costo principal, aunque éste corresponda al 75 % de los costos, no siempre garantiza una solución cuyo costo global sea menor.

Análisis de Slacks

- **Bloques por tren:** Ningún tren lleva la cantidad máxima de bloques[2].
- **Trenes por vía:** Solo 2 de las 294 vías tienen el máximo de trenes circulando por ellas.
- **Block Swaps por bloque:** 9 de los 369 bloques usan el máximo de *block swaps* permitidos.
- **Work events por tren:** 24 de las 96 rutas usan la máxima cantidad de *work events* permitidos.

Al igual que en el DataSet1, vemos que la capacidad de la red dista de estar saturada. Todos los trenes pueden llevar más bloques y se podría agregar nuevos trenes que pasen por la gran mayoría de las vías.

6.6. Experimentos Fallidos

6.6.1. Utilizar muchas columnas por mucho tiempo

Se probó utilizar una mayor cantidad de columnas y dejar el proceso corriendo por más tiempo para entender si esto podría ayudar a mejorar la solución. En particular, se quería observar si dentro de un conjunto mayor de rutas, existía una solución considerablemente mejor a las obtenidas hasta el momento. Para esto resolvimos el DataSet2, aumentando el

[2] No saturan la restricción de cantidad de bloques por tren, lo que no quiere decir que tengan capacidad (tanto de peso como de longitud) disponible para llevar un bloque extra

tiempo límite de cómputo a 4 horas y probamos creando 200 y 240 rutas respectivamente, manteniendo la configuración previa de Lmax:3 y k:4. Al realizar esta experimentación, no se encontró un resultado significativamente mejor.

# Rutas	Mejor entera	Best Bound	Gap
200	3.253.000	3.155.000	3.02 %
240	3.667.000	3.138.137	14.41 %

Como podemos ver en la tabla 6.6.1, aunque hayamos aumentado la cantidad de rutas y el tiempo no logramos mejorar el resultado conseguido en una hora (3.228.130$). En el primer caso conseguimos un resultado similar, tanto en mejor solución como en la cota. En el segundo caso, la solución entera es significativamente peor, y la cota marginalmente mejor. Creemos que esto se debe a que al ser el problema de mayor magnitud, no alcanzan las 4 horas para llegar a una buena solución. Esto lo podemos ver en el amplio *gap* que se mantiene después de las 4 horas (14.41 %).

6.6.2. Generar muchas rutas y elegir un subconjunto para el BTA

Otro enfoque que tomamos para intentar mejorar la solución actual, es generar un número mayor de rutas (300) y de ese pool seleccionar un número menor (160) de rutas al azar y correr el BTA con esa subconjunto durante 1 hora. Todos los resultados conseguidos con este método fueron significativamente peores que los obtenidos hasta el momento. Por ejemplo, los resultados en el DataSet2 oscilaron entre ∼5.300.000$ y ∼8.600.000$ como costo total, pero siempre con un *gap* bajo.

Al compararlo con el costo total de los otros métodos, ∼3.230.000$, concluimos que el método de generación de columnas es bueno, ya que las primeras 160 rutas generadas son significativamente mejores que un subconjunto aleatorio de 160 rutas entre las primeras 300 rutas.

7. TDO CON FLOTA HETEROGÉNEA

La definición original del TDO simplifica múltiples escenarios para facilitar la resolución del problema, pero estos escenarios omitidos aparecen frecuentemente en la práctica y pueden ser de gran interés. Por este motivo, decidimos explorar algunos escenarios no considerados en el TDO. Primero desarrollaremos los nuevos escenarios que queremos considerar, luego expondremos tres métodos para resolver uno de estos nuevos escenarios y por último compararemos estos métodos empíricamente. El primer método será una heurística de asignación, el segundo será una adaptación de los ILP del TDO original y el último será la aplicación del método iterativo por bloques expuesto en 5.1 pero utilizando como ILP el ILP del método anterior.

7.1. Extensiones del problema

Una de las mayores simplificaciones del TDO corresponde al tratamiento y las características de las locomotoras. En particular expondremos tres casos donde esto ocurre. El primero es que las locomotoras tienen capacidad irrestricta, pueden transportar cualquier cantidad de peso o vagones. El segundo caso es la suposición de que todas las locomotoras son idénticas y por último se supone que hay a disposición una capacidad infinita de las mismas.

A continuación analizaremos en detalle estos tres escenarios.

7.1.1. Capacidad de las locomotoras

Como ya mencionamos, la definición original del TDO supone que las locomotoras pueden transportar cualquier cantidad de vagones y peso. Esto dista del escenario real ya que una locomotora puede remolcar una cantidad máxima de peso. Además, a mayor peso, la formación alcanzará menor velocidad pudiendo generar mayor congestión en la red. Por otro lado, trenes largos necesitan mayor infraestructura y tiempo en las terminales para colocar y quitar los vagones de la formación, generando así mayor congestión en la red. Por tal motivo, limitar la longitud de los trenes también es una restricción que surge del contexto del problema. Por lo cual agregamos la limitación de que una locomotora puede mover una cantidad limitada de peso y longitud de vagones. Estas nuevas restricciones son:

- Peso máximo del tren: independientemente del camino que recorra el tren, este no puede en ningún momento llevar más que cierto peso.
- Longitud máxima del tren: más allá del camino que recorra el tren, en ningún momento la totalidad de los bloques deberá superar cierta longitud máxima.

Para esto definimos c_{ton} y c_{lenght} como la cantidad máxima de peso y longitud que puede transportar una locomotora respectivamente.

7.1.2. Heterogeneidad de las locomotoras

Comparando el problema TDO definido en la competencia con la bibliografía de problemas de ruteo, se observa que la definición del problema supone una flota de locomotoras

homogéneas, todas con los mismos costos y las mismas restricciones. Existen empresas que poseen una flota heterogénea, en la que cada tipo de locomotora posee diferentes costos de mantenimiento y combustible, además de diferentes capacidad de carga y velocidad. Por esta razón decidimos extender TDO teniendo en cuenta una flota de características heterogéneas.

Llamaremos L al conjunto de tipos de locomotoras. La heterogeneidad de la flota radicará en que los diversos tipos de locomotoras, $l \in L$, tendrán diferentes costos asociados:

- Costo por iniciar un tren: c_1^l
- Costo por milla: c_2^l
- Costo por *work event*: c_4^l
- Costo por desbalance de trenes (además de cambiar el costo, se pide que el balance sea por cada tipo de locomotora): c_7^l

Además de diferentes capacidades:

- Máxima capacidad de bloques que pueden ser llevados : r^l
- Máxima cantidad de *work events*: w^l
- Máxima capacidad de peso: c_{ton}^l
- Máxima capacidad de longitud: c_{lenght}^l

Esta nueva versión del problema es más amplia y abarcativa, ya que el problema original es un caso particular de éste. Un problema de flota homogénea se puede pensar como un problema heterogéneo con un solo tipo de locomotoras.

7.1.3. Límite de la cantidad de locomotoras

Otra simplificación que toma el problema original es suponer que se puede utilizar una cantidad infinita de locomotoras, y que la restricción de cantidad de locomotoras vendrá implícita en la capacidad de la red. Esto es lejano a la realidad, toda empresa tiene una flota finita. Además, con esa simplificación una empresa no podría usar este modelo para estimar el impacto de ampliar su flota.

Esta restricción se vuelve todavía más importante cuando tenemos en cuenta la heterogeneidad de las locomotoras, ya que si suponemos una flota infinita, nunca podremos modelar locomotoras que sean estrictamente mejores a otras, ya que el modelo siempre usará las mejores. En cambio, un enfoque que tenga en cuenta la finitud de la flota podrá modelar estos casos. Esto sucede en la realidad, ya que las empresas cuentan con un cierto número de locomotoras más modernas (con mejor eficiencia de combustible, más tracción, etc), así como con otro número de locomotoras más antiguas, cuyas cualidades son peores que las primeras, pero éstas siguen siendo necesarias para poder operar mayor cantidad de carga.

Como ya dijimos, esta variación se puede aplicar tanto al problema base como a la variante con flota heterogénea.

Para el caso del TDO original, llamaremos $MaxLocomotoras$ a la cantidad máxima de locomotoras que pueden circular, o sea, la cantidad máxima de trenes.

En el caso de incorporar esta modificación al problema heterogéneo, existen dos formas de realizarlo. La primera es pensar el límite de trenes independientemente del tipo de locomotoras. Por ejemplo, se limita a que existan cien rutas como máximo pero pueden ser tanto todas del mismo tipo, mitad de un tipo y mitad de otro, o cualquier otra distribución posible. Esta forma sirve especialmente cuando se busca saber cuál es la composición de flota óptima. En este caso usaremos $MaxLocomotoras$ al igual que en el TDO original.

La segunda forma en la que se puede aplicar esta modificación es determinar que el límite de trenes sea por cada tipo de locomotora. Por ejemplo, solo podrá haber 30 trenes con una locomotora más moderna, 40 con un segundo tipo de locomotora más antigua, etc. Este segunda forma modela mejor la flota actual de una empresa. Para esto definiremos $MaxLocomotoras^l \ \forall l \in L$ como la cantidad máxima de locomotoras de tipo l que pueden circular.

7.2. Heurística de Asignación

A continuación proponemos un primer método para resolver el TDO en su variante con flota heterogénea, como fue descripto en la sección 7.1. Este método consistirá en tres etapas. Primero, dada la instancia heterogénea que se quiere resolver, se construirá una instancia homogénea. En la segunda etapa, se resolverá esta instancia homogénea con alguno de los métodos descriptos en los capítulos anteriores. Por último, se usará una heurística de asignación que, dada la solución homogénea conseguida en la segunda etapa, construirá una solución heterogénea factible para la instancia original.

7.2.1. Construcción de la instancia homogénea

El primer paso de nuestro algoritmo consiste en construir un instancia homogénea en base a la instancia heterogénea que se quiere resolver. La instancia homogénea creada tendrá las mismas estaciones, vías, bloques, secciones de tripulación que la instancia heterogénea que se quiere resolver.

También usaremos los mismo costos, exceptuando los costos asociados a las locomotoras. Para estos costos tomaremos el promedio entre los costos de cada tipo de locomotora. Por ejemplo el costo de iniciar un tren (c_1) será:

$$c_1 = \frac{1}{\#L} \sum_{l \in L} c_1^l$$

El mismo promedio se usará para conseguir el costo por milla (c_2), el costo por *work event* (c_4), el costo por desbalance de tren (c_7), la cantidad máxima de bloques que puede ser llevados por un tren (r), la cantidad máxima de *work events* que puede realizar un tren (w) y la máxima capacidad de peso y longitud (c_{ton} y c_{lenght}).

Por último, la cantidad máxima de trenes ($MaxLocomotoras$) en el problema homogéneo será igual al del problema heterogéneo en el caso de tener un límite global y a la suma de la cantidad máxima de trenes de cada tipo en el problema heterogéneo cuando se tenga un límite por tipo de locomotora. $MaxLocomotoras = \sum_{l \in L} MaxLocomotoras^l$.

7.2.2. Resolución de TDO

La siguiente etapa consiste en resolver el TDO sobre la instancia que construimos en la etapa anterior. Para resolver TDO podemos utilizar cualquiera de los métodos expuestos

en esta tesis, pero habrá que hacer unas ligeras modificaciones para tener en cuenta el límite de la cantidad locomotoras que pueden circular y el peso y longitud máxima que pueden transportar un tren. Para esto agregaremos algunas restricciones a la formulación expuesta en el capítulo 4 o modificaremos algunas de las ya existentes.

Para modelar el límite de locomotoras que pueden circular agregaremos la siguiente nueva restricción tanto al problema maestro como al BTA:

$$\sum_{p \in P} \lambda_p \leq MaxLocomotoras. \tag{7.1}$$

Para el caso del límite de peso y longitud que puede llevar una locomotora, reemplazaremos las restricciones 4.4 y 4.5 del problema maestro por las siguientes restricciones:

$$\sum_{b \in B} \sum_{q \in Q^b} \alpha_{qij} v_2^b \delta_q^b - min(M_{ij}^2, c_{lenght}) \sum_{p \in P} \alpha_{pij} \lambda_p \leq 0, \forall (i,j) \in E \tag{7.2}$$

$$\sum_{b \in B} \sum_{q \in Q^b} \alpha_{qij} v_3^b \delta_q^b - min(M_{ij}^3, c_{ton}) \sum_{p \in P} \alpha_{pij} \lambda_p \leq 0, \forall (i,j) \in E \tag{7.3}$$

También será necesario reemplazar las restricciones 4.32 y 4.33 del BTA por:

$$\sum_{b \in B} v_2^b y_{ij} \leq min(M_{ij}^2, c_{lenght}), \forall (i,j) \in A_1 \tag{7.4}$$

$$\sum_{b \in B} v_3^b y_{ij} \leq min(M_{ij}^3, c_{ton}), \forall (i,j) \in A_1 \tag{7.5}$$

Dada esta nueva formulación modificada se puede resolver el TDO tanto usando directamente estas formulaciones como se explicó en el capítulo 4, o aplicando los métodos iterativos expuestos en el capítulo 5 pero usando como formulación base esta nueva formulación modificada.

7.2.3. Asignación de tipo de locomotora

El último paso del algoritmo es construir una solución del problema heterogéneo original, dada una solución del problema homogéneo construido en las etapas anteriores. La diferencia entre una solución homogénea y una heterogénea es que en la primera no se específica qué tipo de locomotora se le asignará a cada uno de los trenes de la solución, ya que solo existe un tipo de locomotora. Por lo que para construir la solución heterogénea deseada lo que haremos es a cada uno de los trenes de la solución homogénea asignarle un tipo de locomotora. Esta asignación tendrá que hacerse con cuidado porque puede romper la factibilidad de la solución. Puede llegar el caso de que a un tren no se le pueda asignar ningún tipo de locomotora sin que rompa la factibilidad de la solución, en este caso quitaremos ese tren de la solución y todo bloque que era transportado en algún momento de su trayecto en ese tren pasará a ser no enviado.

Como se explicó en la sección 7.1.3, existen dos formas de limitar la cantidad de locomotoras que pueden circular, imponiendo un límite de locomotoras global (independiente del tipo de locomotora) o imponiendo un límite diferente para cada tipo de locomotora.

A continuación expondremos para cada uno de estos escenarios, un algoritmo para resolver la asignación de tipos de locomotoras.

Asignación para límite de locomotoras global

El caso más sencillo de resolver es cuando la instancia heterogénea no tiene composición específica de flota. En ese caso, para cada ruta de la solución homogénea buscaremos entre los tipos de locomotoras que vuelven factible a esa ruta y elegiremos la que genere menor costo.

Este algoritmo funcionará siempre y cuando exista al menos un tipo de ruta heterogénea, que para toda restricción de capacidad, tenga más capacidad que el promedio. En los casos en que esto no suceda, puede darse que a una ruta de la solución promedio no se le pueda asignar ningún tipo de ruta heterogénea que mantenga la factibilidad. En ese caso, dejaremos de usar esa ruta y dejaremos de enviar los bloques que usaban esa ruta.

Asignación con composición de flota fija

En este caso tenemos una composición fija de la flota. Por ejemplo, podemos usar a lo sumo x locomotoras de tipo 1, y locomotoras de tipo 2, etc. Esta restricción aumenta fuertemente la dificultad de la asignación, por lo cual construiremos una heurística de asignación solo para una familia de problemas heterogéneos. Solo tendremos en cuenta problemas que tengan dos tipos de locomotoras, la primera con una mayor capacidad de carga de bloques pero con un mayor costo por milla que llamaremos locomotoras de alta capacidad; y una segunda de menor capacidad de carga de bloques pero con un menor costo por milla que llamaremos de baja capacidad.

Vale aclarar que este subcaso posee las deseables propiedades de:

- Toda ruta factible de baja capacidad también será factible si se cambia su tipo a alta capacidad.
- Toda ruta de alta capacidad tendrá menor costo si se cambia su tipo por baja capacidad.

El algoritmo propuesto para resolver esta asignación, consiste en marcar cada tren de la solución promedio que sea factible de llevar con una locomotora de baja capacidad. El resto de los trenes serán marcadas como trenes de alta capacidad.

$MaxLocomotoras^{low}$ y $MaxLocomotoras^{high}$ será la cantidad máxima de trenes de baja y alta capacidad respectivamente.

Definiremos R^{low} y R^{high} como la cantidad de trenes marcados en el paso anterior como de baja capacidad y de alta capacidad respectivamente.

En función de esto existirán tres casos:

1. $R^{low} \leq MaxLocomotoras^{low} \wedge R^{high} \leq MaxLocomotoras^{high}$
2. $R^{low} > MaxLocomotoras^{low} \wedge R^{high} \leq MaxLocomotoras^{high}$
3. $R^{low} \leq MaxLocomotoras^{low} \wedge R^{high} > MaxLocomotoras^{high}$

Vale aclarar que el caso faltante, $R^{low} > MaxLocomotoras^{low} \wedge R^{high} > MaxLocomotoras^{high}$, es un caso que no puede darse, ya que limitamos la cantidad de rutas máxima del homogéneo a la suma de la cantidad máxima de rutas por tipo del heterogéneo. Por lo cual $R^{low} + R^{high} \leq MaxLocomotoras^{low} + MaxLocomotoras^{high}$

El caso 1 es el más sencillo. Asignaremos cada tren al tipo de locomotora marcado, es decir, a los trenes marcadas como de baja capacidad, se le asignará una locomotora de baja

capacidad y a los trenes marcadas como de alta capacidad, se le asignará una locomotora de alta capacidad. Esto nos generará una solución factible, ya que cada tren cumplirá sus restricciones de capacidad y también la restricción de cantidad de locomotoras por tipo.

Para resolver el caso 2 es necesario seleccionar un subconjunto de n trenes marcados como de baja capacidad, con $n = R^{low} - MaxLocomotoras^{low}$, y asignarles locomotoras de alta capacidad. Para esto, ordenaremos todos los trenes marcados como de baja capacidad en función de su cantidad de millas recorridas, y modificaremos las n menores marcándolos como trenes de alta capacidad. Al resto de los trenes de baja capacidad se le asignará una locomotora de baja capacidad, igual que a los trenes marcados como de alta capacidad se les asignará una locomotora de alta capacidad.

El caso 3 es el más complejo, en los casos anteriores siempre se podía respetar el BTA generado y encontrar fácilmente una asignación de tipos de locomotoras que minimice el costo final. Pero en este caso eso no será posible, ya que al no haber suficientes locomotoras de alta capacidad, no se podrá cumplir con el BTA generado por el algoritmo homogéneo. Entonces, primero se asignarán locomotoras de baja capacidad a todos los trenes marcados como de baja capacidad. Luego se elegirán t trenes marcados como de alta capacidad con $t = R^{high} - MaxLocomotoras^{high}$. A estos les asignaremos locomotoras de baja capacidad y quitaremos del BTA todo eje relacionado a los bloques que están dentro del conjunto de bloques a quitar de esos trenes. Para seleccionar estos trenes, calcularemos para cada tren el conjunto de bloques que quitaríamos en caso de cambiarlo a ser de baja capacidad. Luego, ordenaremos los trenes según el número de vagones de los bloques a quitar, y seleccionaremos los t trenes con menor cantidad de vagones a quitar. Por último, les asignaremos locomotoras de alta capacidad al resto de los trenes marcados como de alta capacidad.

Para calcular el conjunto de bloques que deberemos quitar de un tren si le asignamos una locomotora de baja capacidad, construiremos un conjunto de bloques, inicialmente vacío e iremos agregando los bloques que no se podrán llevar.

Por cada vía en el recorrido del tren, tomaremos el conjunto de bloques que el tren lleva a través de esa vía, que será el conjunto de bloques que lleva originalmente por esa vía según la solución homogénea que se utiliza, quitándole los bloques que ya fueron marcados como que no se llevarán. Si ese conjunto tiene más cantidad de bloques que los que la locomotora de baja capacidad puede transportar, habrá que quitar algunos bloques, en particular habrá que quitar f bloques, donde f será la diferencia entre la cantidad de bloques en este conjunto y la cantidad de bloques que una locomotora de baja capacidad puede transportar.

Para elegir qué bloques no se llevarán, se ordenaran todos estos bloques en función de su cantidad de vagones y no se llevarán los f bloques con menos vagones, agregándolos al conjunto de bloques que se deberán quitar. El objetivo de no transportar los bloques con menos vagones es buscar minimizar la penalización de no entregar bloques, ya que esta es linealmente dependiente de la cantidad de vagones.

Este proceso se repetirá para todas las vías del recorrido del tren, consiguiendo así el conjunto de bloques que se quitaría si a ese tren se le asigna una locomotora de baja capacidad.

7.3. Formulación TDO Heterogéneo

Nuestro segundo método para resolver el problema heterogéneo, fue adaptar la formulación del problema homogéneo para que soporte diferentes tipos de rutas.

El proceso para resolver el problema será el mismo que el expuesto en el capítulo 4, utilizando las mejoras propuestas en la sección 4.4 y cambiando ciertas variables y restricciones de las formulaciones del *Master Problem*, *Slave Problem* y BTA.

A continuación, se presenta las formulaciones modificadas, remarcando en color verde las principales diferencias con las formulaciones del problema homogéneo.

7.3.1. Master Problem

Primero se introduce el conjunto L de tipos de locomotoras, y el conjunto de rutas P se transformará en un conjunto P^l para cada tipo de locomotora en L. Por lo que las variables λ_p pasarán a ser λ_p^l representando una ruta p de tipo l. Por este cambio de variables, se reemplazarán las sumatorias de la forma $\sum_{p \in P} \dots \lambda_p \dots$ por $\sum_{l \in L} \sum_{p \in P^l} \dots \lambda_p^l \dots$

La otra variable que se modifica es ϕ_i que representaba el desbalance de trenes en la estación i, pasará a ser ϕ_i^l y representará el desbalance de trenes de tipo l en la estación i.

Conjuntos	
L	Conjunto de tipos de locomotoras
P^l	Conjunto de rutas de tipo l, $\forall l \in L$
Variables de decisión	
λ_p^l	1 si se usa la ruta $p \in P^l$ con una locomotora de tipo $l \in L$; 0 en caso contrario
ϕ_i^l	$\mathbb{Z}^+$, cantidad de desbalance de trenes de tipo $l \in L$ en la estación $i \in V$
Parámetros	
c_1^l	Costo de inicio de un tren de tipo l, $\forall l \in L$
c_2^l	Costo por milla recorrida por un tren de tipo l , $\forall l \in L$
c_7^l	Costo por desbalance de tren de tipo l, $\forall l \in L$
c_p^l	Costo de un tren de tipo $l \in L$ viajando por la ruta $p \in P^l$. $c_p^l = c_2^l \sum_{(i,j) \in E} \alpha_{qij} l_{ij}$
c_{lenght}^l	Longitud máxima de carga de trenes de tipo l, $\forall l \in L$
c_{ton}^l	Peso máximo de carga de trenes de tipo l, $\forall l \in L$
r^l	Número máximo de bloques que puede cargar un tren de tipo l, $\forall l \in L$

Tab. 7.1: Notaciones para el problema maestro heterogéneo

$$\text{mín} \sum_{l\in L} c_1^l \sum_{p\in P^l} \lambda_p^l + \sum_{l\in L}\sum_{p\in P^l} c_p^l \lambda_p^l + \sum_{b\in B}\sum_{q\in Q^b} v_1^b c_q^b \delta_q^b + c_6 \sum_{(m,n)\in C} \sigma_{mn} + \sum_{l\in L} 2c_7^l \sum_{i\in V} \phi_i^l + M \sum_{b\in B} \tau^b \tag{7.6}$$

sujeto a

$$\sum_{q\in Q^b} \delta_q^b + \tau^b = 1 \qquad \forall b \in B \tag{7.7}$$

$$\sum_{l\in L}\sum_{p\in P^l} \alpha_{pij}\lambda_p^l + \sum_{l\in L}\sum_{p\in P^l} \alpha_{pji}\lambda_p^l \leq M_{ij}^1 \qquad \forall (i,j) \in E | i < j \tag{7.8}$$

$$\sum_{b\in B}\sum_{q\in Q^b} \alpha_{qij} v_2^b \delta_q^b - \sum_{l\in L} min(M_{ij}^2, c_{lenght}^l) \sum_{p\in P^l} \alpha_{pij}\lambda_p^l \leq 0 \qquad \forall (i,j) \in E \tag{7.9}$$

$$\sum_{b\in B}\sum_{q\in Q^b} \alpha_{qij} v_3^b \delta_q^b - \sum_{l\in L} min(M_{ij}^3, c_{ton}^l) \sum_{p\in P^l} \alpha_{pij}\lambda_p^l \leq 0 \qquad \forall (i,j) \in E \tag{7.10}$$

$$\sigma_{mn} + \sum_{l\in L}\sum_{p\in P^l} \beta_{pmn}\lambda_p^l - \sum_{l\in L}\sum_{p\in P^l} \beta_{pnm}\lambda_p^l \geq 0 \qquad \forall (m,n) \in C \tag{7.11}$$

$$\sigma_{mn} - \sum_{l\in L}\sum_{p\in P^l} \beta_{pmn}\lambda_p^l + \sum_{l\in L}\sum_{p\in P^l} \beta_{pnm}\lambda_p^l \geq 0 \qquad \forall (m,n) \in C \tag{7.12}$$

$$\phi_i^l + \sum_{p\in P^l} \gamma_{pi}^1 \lambda_p^l - \sum_{p\in P^l} \gamma_{pi}^2 \lambda_p^l \geq 0 \qquad \forall l \in L, \forall i \in V \tag{7.13}$$

$$0 \leq \lambda_p^l, \delta_q^b \leq 1 \tag{7.14}$$

$$\sigma_{mn}, \phi_i \geq 0 \tag{7.15}$$

El mayor cambio en el funcional 7.6, más allá del cambio de las variables λ_p^l y ϕ_i^l, es el cambio en los costos c_1^l, c_7^l y c_p^l que pasan a depender del tipo de locomotora (l).

Las restricciones 7.9 y 7.10 pasan de representar que un tren puede cargar tanta longitud y peso como la vía soporta a representar que un tren puede cargar el mínimo entre lo que la vía le permite y lo que la locomotora en sí misma puede cargar, representado por c_{lenght}^l y c_{ton}^l respectivamente.

Por último, como el desbalance de trenes pasará a ser por cada tipo de locomotora, la restricción de desbalance de tren 7.13 pasará a ser una por cada estación y por cada tipo de locomotora, donde solo calculará el desbalance de trenes del tipo correspondiente.

7.3.2. Slave Problem

Al momento de resolver el *Slave Problem* en el caso heterogéneo, es necesario tomar dos decisiones: el recorrido de la ruta generada y el tipo de locomotora que se le asignará. Para generar estas rutas plantearemos dos métodos: *Slave(L) Problem* y *Slave Problem Normal*. El primero es un método que, dado un tipo de locomotora, generará un recorrido para una ruta de ese tipo. Utilizaremos este método para cada tipo de locomotora, consiguiendo

así un recorrido de cada tipo y elegiremos la ruta cuyo costo reducido sea el menor. En cambio, el *Slave Problem Normal* resuelve ambas decisiones al mismo tiempo, tanto el recorrido de la ruta como su tipo de locomotora.

Estos dos métodos vuelven a presentar el balance entre resolver un solo problema de mayor tamaño o múltiples problemas de menor tamaño. La comparación entre ambos métodos se presentará en la sección 7.4.1

Slave(L) Problem

Dado un tipo de locomotora l, el algoritmo *Slave(L) Problem*, es muy similar al *Slave Problem* del problema heterogéneo (4.2.2), lo único que variará es que las constantes $c_1^l, c_2^l, c_{len}^l, c_{ton}^l, r^l$ dependerán del tipo de locomotora l que se esté resolviendo en cada momento.

Más allá del diferente valor de estas constantes, el problema se resuelve de la misma manera que el *Slave Problem* homogéneo, utilizando los mismos cortes de subtour extra que propusimos en la sección 4.4.

Para elegir qué nueva ruta candidata se agregará al *Master Problem*, resolveremos $Slave(l), \forall l \in L$ y tomaremos la ruta candidata de menor costo reducido entre las l rutas generadas.

A continuación se expone la formulación del *Slave(L) Problem*, resaltando en color verde las diferencias con el *Slave Problem* homogéneo.

$$\text{mín } c_1^{\boxed{l}} + \sum_{(m,n\in C)} x_{mn}\Bigg\{ \sum_{(i,j)\in E} \theta_{mnij}\Big[c_2^{\boxed{l}} l_{ij} - \pi_{ij}^2 + min(M_{ij}^2, c_{len}^{\boxed{l}})\pi_{ij}^3 + min(M_{ij}^3, c_{ton}^{\boxed{l}})\pi_{ij}^4\Big] - (\pi_{mn}^5 - \pi_{mn}^6) + (\pi_{nm}^5 - \pi_{nm}^6)\Bigg\} - \sum_{i\in V} x_{0i}\pi_i^7 + \sum_{i\in V} x_{i0}\pi_i^7 \tag{7.16}$$

sujeto a

$$\sum_{n\in V^0|(0,n)\in C^0} x_{0n} = 1 \tag{7.17}$$

$$\sum_{n\in V^0|(m,n)\in C^0} x_{mn} - \sum_{n\in V^0|(n,m)\in C^0} x_{nm} = 0 \qquad \forall m \in V^0 \tag{7.18}$$

$$\sum_{(m,n)\in C} x_{mn} \leq L_{max} \tag{7.19}$$

$$\sum_{n|(m,n)\in C^0} x_{mn} \leq 1 \qquad \forall m \in V \tag{7.20}$$

$$\sum_{(m,n)\in C} \theta_{mnij} x_{mn} \leq 1 \qquad \forall (i,j) \in E \tag{7.21}$$

$$x_{mn} \in \{0,1\} \qquad \forall (m,n) \in C^0 \tag{7.22}$$

Slave Problem Normal

El otro método para generar rutas candidatas para el *Master Problem*, consiste en resolver un solo ILP que decida al mismo tiempo el recorrido de la ruta y su tipo.

Variables de decisión	
x_l	1 si la ruta creada es de tipo $l \in L$; 0 en caso contrario
x^l_{mn}	1 si la ruta de tipo $l \in L$ usa la sección de tripulación $(m, n) \in C^0$; 0 en caso contrario

Tab. 7.2: Notaciones para el problema esclavo heterogéneo normal

A continuación presentamos la formulación del ILP, resaltando las principales diferencias con el *Slave Problem* homogéneo.

$$\text{mín} \sum_{l \in L} x_l \Big(c_1^l + \sum_{(m,n \in C)} x_{mn}^l \Big\{ \sum_{(i,j) \in E} \theta_{mnij} \Big[c_2^l l_{ij} - \pi_{ij}^2 + min(M_{ij}^2, c_{len}^l)\pi_{ij}^3 + min(M_{ij}^3, c_{ton}^l)\pi_{ij}^4 \Big] - (\pi_{mn}^5 - \pi_{mn}^6) + (\pi_{nm}^5 - \pi_{nm}^6) \Big\} - \sum_{i \in V} x_{0i}^l (\pi_i^7 - \pi_i^8) + \sum_{i \in V} x_{i0}^l (\pi_i^7 - \pi_i^8)) \tag{7.23}$$

sujeto a

$$\sum_{l \in L} \sum_{n \in V^0 | (0,n) \in C^0} x_{0n}^l = 1 \tag{7.24}$$

$$\sum_{n \in V^0 | (m,n) \in C^0} x_{mn}^l - \sum_{n \in V^0 | (n,m) \in C^0} x_{nm}^l = 0 \qquad \forall m \in V^0, \forall l \in L \tag{7.25}$$

$$\sum_{l \in L} \sum_{(m,n) \in C} x_{mn}^l \leq L_{max} \tag{7.26}$$

$$\sum_{l \in L} \sum_{n | (m,n) \in C^0} x_{mn}^l \leq 1 \qquad \forall m \in V \tag{7.27}$$

$$\sum_{l \in L} \sum_{(m,n) \in C} \theta_{mnij} x_{mn}^l \leq 1 \qquad \forall (i,j) \in E \tag{7.28}$$

$$\sum_{l \in L} x_l = 1 \tag{7.29}$$

$$\sum_{(m,n) \in C} x_{mn}^l \geq x_l \qquad \forall l \in L \tag{7.30}$$

$$\sum_{(m,n) \in C} x_{mn}^l \leq \#(C) x_l \qquad \forall l \in L \tag{7.31}$$

$$x_{mn}^l \in \{0,1\} \qquad \forall (m,n) \in C^0, \forall l \in L \tag{7.32}$$

$$x_l \in \{0,1\} \qquad \forall l \in L \tag{7.33}$$

Las variables binarias x_l determinan el tipo de la locomotora de la ruta generada, y utilizaremos la restricción 7.29 para garantizar que solo una variable x_l sea 1.

Las variables x_{mn}^l representan si la sección de tripulación mn, se utilizan en la ruta construida de tipo l.

Las restricciones 7.24-7.28, son equivalentes a las del *Slave Problem* homogéneo, pero iterando también sobre $l \in L$.

Las restricciones 7.30 y 7.31 garantizan que las variables $x_{mn}^{l^*}$ sean 1 si y solo si, x^{l^*} es 1.

Por último, las restricciones 7.32 y 7.33 definen el dominio de las variables de decisión.

7.3.3. BTA

Como en los casos anteriores, la formulación del BTA heterogéneo es muy similar al caso homogéneo. Las principales diferencias serán que gran parte de las constantes serán dependientes del tipo de locomotora, y que para iterar todas las rutas habrá que iterar también por los tipos de locomotora ($\sum_{l\in L}\sum_{p\in P^l}\lambda_p^l$).

A continuación se presenta la formulación del BTA heterogéneo, resaltando en color verde las principales diferencias con la formulación homogénea.

Variables de decisión	
λ_p^l	1 si se usa la ruta $p \in P^l$ con una locomotora de tipo $l \in L$; 0 en caso contrario
ϕ_i^l	$\mathbb{Z}^+$, cantidad de desbalance de trenes de tipo $l \in L$ en la estación $i \in V$
Conjuntos	
A_1^l	Conjunto de todos los ejes asociados a rutas de tipo $l \in L$, $A_1^l = \bigcup_{p\in P^l} A_p$
Parámetros	
c_4^l	Costo por *work event* de rutas de tipo $l \in L$
w^l	Número máximo de *work events* que puede llevar a cabo un tren de tipo $l \in L$

Tab. 7.3: Notaciones para el problema BTA heterogéneo

$$\text{mín} \sum_{l\in L}\sum_{p\in P^l}(c_1^l + c_p^l)\lambda_p^l + c_3\sum_{b\in B}\sum_{(i,j)\in A_1^b} v_1^b l_{ij} y_{ij}^b + \sum_{l\in L} c_4^l \sum_{(i,j)\in A_1^l} z_{ij}$$
$$+ \sum_{b\in B}\sum_{(i,j)\in A_2^b} c_5^{p^{-(i)}} y_{ij}^b + c_6\sum_{(m,n)\in C}\sigma_{mn} + \sum_{l\in L} 2c_7^l \sum_{i\in V}\phi_i^l + c_0\sum_{b\in B} v_1^b\tau^b \tag{7.34}$$

sujeto a

$$\sum_{i\in N|p^{-(i)}=o^b}\sum_{j\in N|(i,j)\in A_1^b} y_{ij}^b + \tau^b = 1 \qquad \forall b \in B \tag{7.35}$$

$$\sum_{i\in N|p^{-(i)}=o^b}\sum_{j\in N|(j,i)\in A_1^b} y_{ji}^b = 0 \qquad \forall b \in B \tag{7.36}$$

$$\sum_{j\in N|(i,j)\in A_1^b} y_{ij}^b - \sum_{j\in N|(j,i)\in A_1^b} y_{ji}^b = 0 \qquad \forall b \in B, \forall i \in N|p^{-(i)} \neq o^b, d^b \tag{7.37}$$

$$\sum_{j\in N|p^{-(j)}=d^b}\sum_{i\in N|(i,j)\in A_1^b} y_{ij}^b + \tau^b = 1 \qquad \forall b \in B \tag{7.38}$$

$$\sum_{j\in N|p^{-(j)}=d^b}\sum_{i\in N|(j,i)\in A_1^b} y_{ji}^b = 0 \qquad \forall b \in B \tag{7.39}$$

$$\sum_{b\in B} y_{ij}^b \leq r^l\lambda_p^l \qquad \forall l \in L, \forall p \in P^l, \forall (i,j) \in A_p \tag{7.40}$$

$$\sum_{(i,j)\in A_2^b} y_{ij}^b \leq s_b \qquad \forall b \in B \tag{7.41}$$

$$z_{ij} \geq y_{ij}^b - y_{jo}^b \qquad \forall b \in B, \forall l \in L, \forall p \in P^l, \forall (i,j),(j,o) \in A_p | \; y_{pj}^1 \neq 1 \wedge (i,j),(j,o) \textit{ consecutivos en } p \tag{7.42}$$

$$z_{ij} \geq y_{jo}^b - y_{ij}^b \qquad \forall b \in B, \forall l \in L, \forall p \in P^l, \forall (i,j),(j,o) \in A_p | \; y_{pj}^2 \neq 1 \wedge (i,j),(j,o) \textit{ consecutivos en } p \tag{7.43}$$

$$\sum_{(i,j)\in A^p} z_{ij} \leq w^l \qquad \forall l \in L, \forall p \in P^l \tag{7.44}$$

$$\sum_{l\in L}\sum_{p\in P^l} \alpha_{pij}\lambda_p^l + \sum_{l\in L}\sum_{p\in P^l} \alpha_{pji}\lambda_p^l \leq M_{ij}^1 \qquad \forall (i,j) \in E | i < j \tag{7.45}$$

$$\sum_{b\in B} v_2^b y_{ij} \leq min(M_{ij}^2, c_{len}^l) \qquad \forall l \in L, \forall p \in P^l, \forall (i,j) \in A_p \tag{7.46}$$

$$\sum_{b\in B} v_3^b y_{ij} \leq min(M_{ij}^3, c_{ton}^l) \qquad \forall l \in L, \forall p \in P^l, \forall (i,j) \in A_p \tag{7.47}$$

$$\sigma_{mn} + \sum_{l\in L}\sum_{p\in P^l} \beta_{pmn}\lambda_p^l - \sum_{l\in L}\sum_{p\in P^l} \beta_{pnm}\lambda_p^l \geq 0 \qquad \forall (m,n) \in C \tag{7.48}$$

$$\sigma_{mn} - \sum_{l\in L}\sum_{p\in P^l} \beta_{pmn}\lambda_p^l + \sum_{l\in L}\sum_{p\in P^l} \beta_{pnm}\lambda_p^l \geq 0 \qquad \forall (m,n) \in C \tag{7.49}$$

$$\phi_i^l + \sum_{p\in P^l} \gamma_{pi}^1 \lambda_p^l - \sum_{p\in P^l} \gamma_{pi}^2 \lambda_p^l \geq 0 \qquad \forall i \in V, \forall l \in L \tag{7.50}$$

$$\lambda_p^l, y_{ij}^b, z_{ij}, \tau^b \in \{0,1\} \tag{7.51}$$

$$\sigma_{ij}, \phi^i \in \mathbb{Z}^+ \tag{7.52}$$

En la función objetivo, las principales diferencias son que las constantes $c_1^l, c_p^l, c_4^l, c_7^l$ y las variables λ_p^l *y* ϕ_i^l pasan a depender de l.

Las restricciones 7.35-7.39, 7.41-7.43, 7.51 y 7.52 son idénticas a las correspondientes restricciones del problema homogéneo.

Las restricciones 7.40 y 7.44-7.49 son iguales pero utilizando l para iterar todas las rutas y para las constantes correspondientes.

Por último, el desbalance de trenes en las estaciones del problema heterogéneo pasa de ser único a ser dependiente del tipo de locomotora, por lo que la restricción 7.50 pasa de ser una sola restricción, a ser una para cada tipo de locomotora.

7.4. Experimentación TDO Heterogéneo

7.4.1. Comparación de los algoritmos del problema esclavo

A fin de evaluar cuál de los dos algoritmos del problema esclavo es más conveniente usar, los comparamos empíricamente. Para esto, resolvimos la generación de columnas

usando ambos algoritmos y una variedad de parámetros e instancias. Como parámetros usamos una cantidad de rutas de 100, 160 y 200; como $Lmax$ 2, 3 y 4; y como k 3, 4 y 5. Aunque usamos siempre los DataSets presentados en la competencia, tuvimos que definir las nuevas variables del problema heterogéneo: la cantidad de tipos de locomotoras y los parámetros de cada una de ellas. Para este experimento utilizamos 4 configuraciones diferentes de estos parámetros heterogéneos:

- 3 Igual: consiste en 3 tipos iguales de locomotoras.
- 10 Igual: 10 tipos iguales de locomotoras. Utilizamos la comparación entre este escenario y el anterior para comparar la degradación de *performance* cuando la cantidad de tipos de locomotoras aumenta aunque sean todas iguales.
- 10 Peor: 10 tipos de locomotoras donde cada una es estrictamente peor que la anterior (menor capacidad y mayores costos). Es decir, para cada variable de costo c, $c^1 < c^2 < c^3 < \ldots < c^{10}$ y para cada variable de capacidad d, $d^1 > d^2 > d^3 > \ldots > d^{10}$.
- 10 Balanceadas: 10 tipos de locomotoras donde cada locomotora provee mayor capacidad pero a cambio de un mayor costo. Esto es, para cada variable de costo c, $c^1 < c^2 < c^3 < \ldots < c^{10}$ y para cada variable de capacidad d, $d^1 < d^2 < d^3 < \ldots < d^{10}$.

Para cada escenario de tipos de locomotoras y parámetros, calculamos el porcentaje de la diferencia de tiempo entre el *Slave Problem Normal* y *l Slaves(L) Problem*, donde un porcentaje positivo representará un menor tiempo de cómputo del *Slave Problem Normal* y un porcentaje negativo representará un menor tiempo de cómputo del *Slaves(L) Problem*.

$$\frac{Tiempo(SlaveProblemNormal) - Tiempo(Slaves(L)Problem)}{Tiempo(SlaveProblemNormal)}$$

Para cada DataSet y cada configuración de rutas, graficamos los siguiente casos: el caso con menor diferencia, la diferencia promedio y el caso con mayor diferencia.

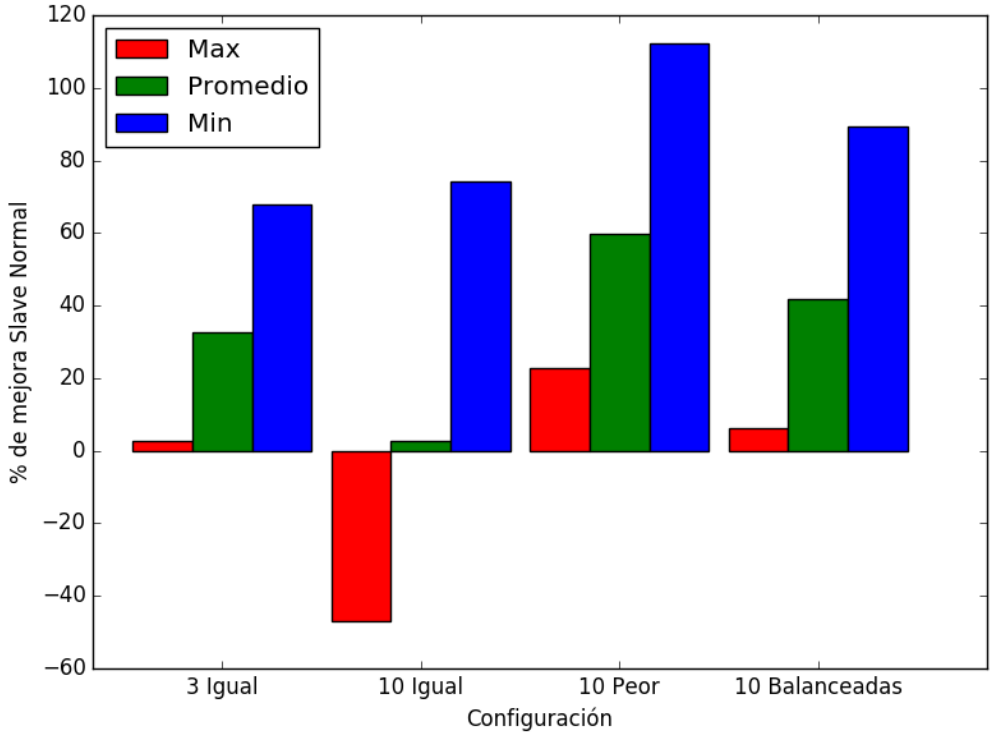

Fig. 7.1: Diferencia de tiempo entre Slave Problem Normal y L SlavesL Problems en el DataSet1

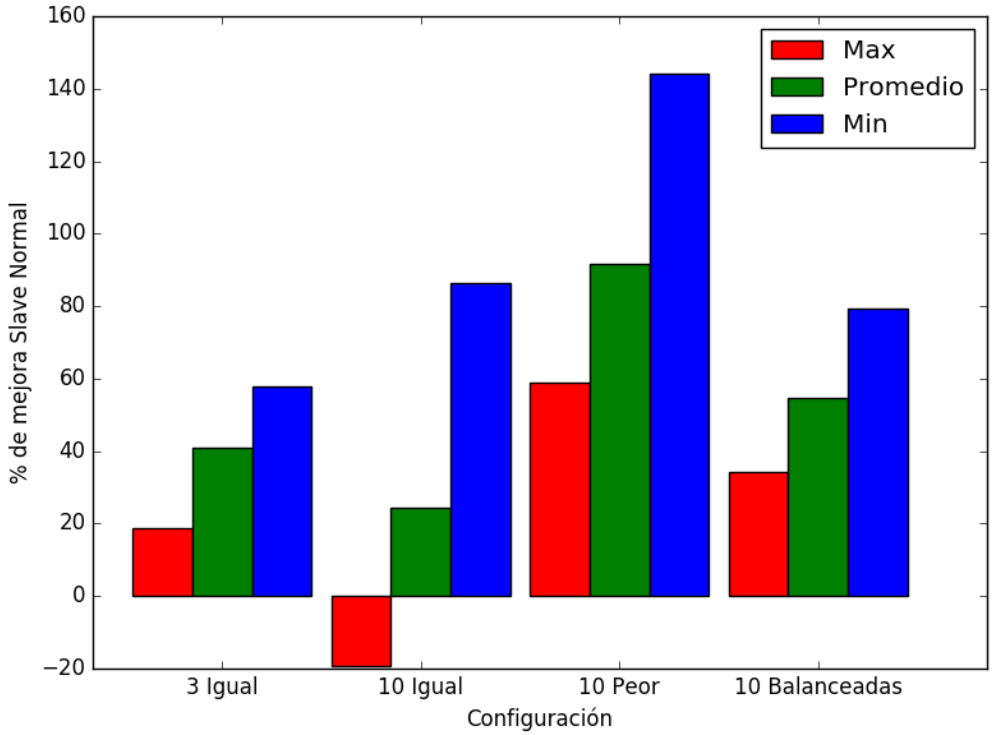

Fig. 7.2: Diferencia de tiempo entre Slave Problem Normal y L SlavesL Problems en el DataSet2

Como se puede observar en los gráficos, tanto en el DataSet1 como en el DataSet2, usar en la generación de columnas un solo *Slave Problem Normal* que considere al mismo tiempo todos los tipos de locomotoras, provee una mejora de tiempo de cómputo en la gran mayoría de los casos. Podemos ver que en promedio esta estrategia es siempre la mejor, y solo es superada por usar l *Slave(L) Problems* en el peor caso del escenario con 10 tipos de locomotoras iguales. Vale aclarar que este caso es completamente artificial, ya que un problema heterogéneo con todos los tipos de locomotoras iguales, es un problema homogéneo, por lo que para resolverlo no haría falta utilizar estos métodos, sino que podremos recurrir a los métodos homogéneos.

Como conclusión de estos experimentos, determinamos que el mejor algoritmo para resolver el problema esclavo para el caso heterogéneo es el *Slave Problem Normal*, por lo que utilizaremos éste al momento de resolver los experimentos siguientes.

7.4.2. Comparación Algoritmos para el TDO Heterogéneo

A continuación comparamos los tres métodos propuestos para resolver el problema heterogéneo con límite de locomotoras global. El primer método, como se explica en el capítulo 7.2, consiste en construir un problema homogéneo, resolverlo y usar un algoritmo de asignación para construir la solución heterogénea. Para resolver el problema homogéneo construido usaremos el ILP propuesto en el capítulo 4 y el método iterativo por bloques propuesto en la sección 5.1, al primero lo llamaremos "Asignación" y al segundo "Asignación con Iterativo". El segundo método propuesto es resolver el ILP heterogéneo presentado en el capítulo 7.3 que llamaremos "ILP Het". El último método consiste en aplicar el método iterativo por bloques presentado en la sección 5.1 usando como ILP base el ILP presentado en el capítulo 7.3 y lo llamaremos "Iterativo".

Para comparar estos algoritmos construimos dos escenarios con diferentes conjuntos de tipos de locomotoras, donde solo variamos la cantidad de bloques que pueden llevar y el costo por milla recorrida, mientras que el resto de las variables asociadas a las locomotoras serán idénticas a las del problema homogéneo. Ambos escenarios cuentan con dos tipos de locomotoras, un primer tipo de locomotora con mayor capacidad de carga de bloques pero también mayor costo por milla, y un segundo tipo con menor carga y costo por milla: $r^1 > r^2 \wedge c_2^1 > c_2^2$.

Al primer escenario lo llamaremos *Locomotoras Costosas* y consistirá de un tipo de locomotora que será idéntico a las rutas del problema homogéneo, y podrá transportar 8 bloques por tren a un costo de 10$ por milla ($r^1 = 8 \wedge c_2^1 = 10$), y la segunda podrá transportar 6 bloques por tren a un costo de 8$ por milla ($r^2 = 6 \wedge c_2^2 = 8$). Al segundo conjunto lo llamaremos *Locomotoras Baratas* y el primer tipo de ruta será idéntico al primer tipo de ruta del conjunto anterior, cargará hasta 8 bloques a un costo de 10$ por milla ($r^1 = 8 \wedge c_2^1 = 10$), pero la segunda ruta solo podrá transportar 4 bloques a un costo de 6$ por milla ($r^2 = 4 \wedge c_2^2 = 6$).

A continuación, presentamos los gráficos donde comparamos los cuatro métodos para cada DataSet y cada escenario de tipos de locomotoras, teniendo en cuenta la calidad de la solución y el tiempo de cómputo utilizado.

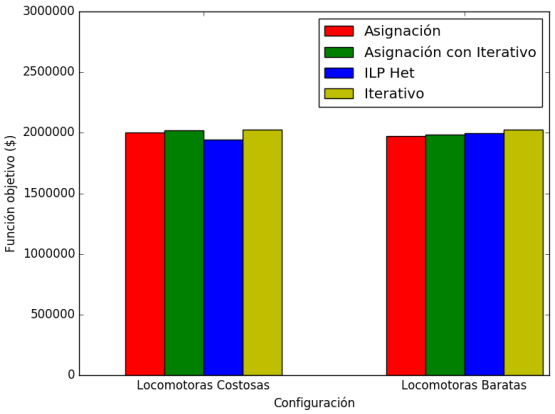

Fig. 7.3: Calidad de solución en DataSet1

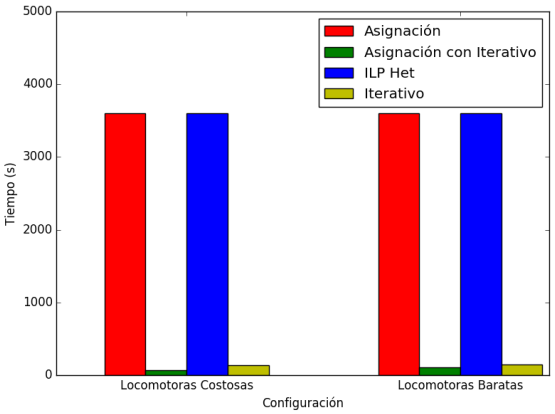

Fig. 7.4: Tiempo de ejecución en DataSet1

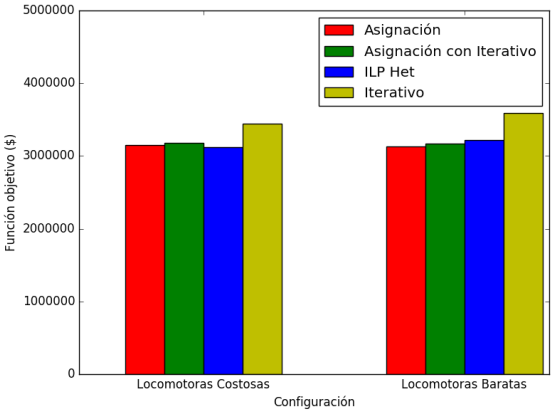

Fig. 7.5: Calidad de solución en DataSet2

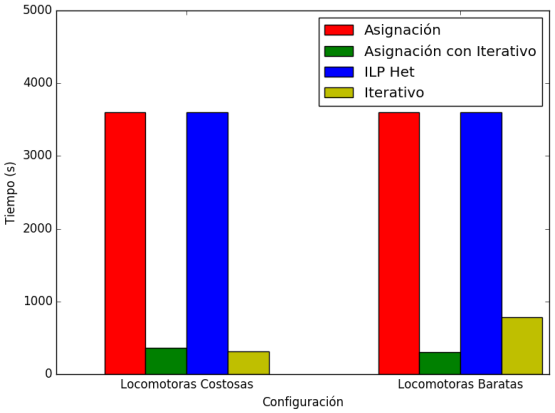

Fig. 7.6: Tiempo de ejecución en DataSet2

Al analizar la calidad de las soluciones, observamos que el ILP heterogéneo consigue la solución de mejor calidad en el escenario de *Locomotoras Costosas* y el algoritmo de asignación la mejor calidad de solución en el escenario de *Locomotoras Baratas*. En todos los casos la versión iterativa de cada uno de estos métodos tiene una peor calidad de solución.

Las diferencias entre la calidad de las soluciones del ILP y del algoritmo de asignación son bajas, 2.9 % y 1.17 % en el DataSet1 y 0.9 % y 0.3 % en el DataSet2.

Las diferencias entre la calidad de las soluciones del ILP heterogéneo y las del algoritmo iterativo, son de 4.4 % y 1.6 % en el DataSet1, pero ascienden a 10 % y 14 % en el DataSet2.

Sin embargo, si se considera también el tiempo de cómputo, el panorama cambia rotundamente, ya que tanto el ILP como el algoritmo de asignación utilizan el tiempo máximo de cómputo (una hora), pero las versiones iterativas requiere entre 1.5 y 2.5 minutos para el DataSet1 y 5 y 13 minutos respectivamente en el DataSet2. Esto representa una mejora en el tiempo de cómputo del 95 % en el DataSet1 y de entre 91 % y 78 % en el DataSet2.

Esto vuelve a demostrar la gran ganancia en tiempo de cómputo que generan los métodos iterativos a cambio de una reducción de la calidad de solución.

8. CONCLUSIONES Y TRABAJO A FUTURO

En esta tesis trabajamos sobre el *Train Design Optimization Problem* [5]. Tomamos como punto de partida la formulación propuesta por Jin et al. [6] y propusimos un conjunto de mejoras a esta formulación y dos métodos iterativos basados en la misma. La experimentación muestra que las mejoras sobre la formulación original logran aumentar la calidad de la solución.

Al evaluar los métodos iterativos presentados, se destaca el método iterativo por bloques. Este método logra conseguir una solución de calidad similar a la formulación original pero en una fracción del tiempo. Esta mejora temporal puede ser muy provechosa al momento de resolver instancias más grandes, ya que al ser un problema exponencial, el tiempo de cómputo crecerá rápidamente generando que la resolución con la formulación original se vuelva inviable.

En la formulación original del TDO se simplifican múltiples escenarios(locomotoras homogéneas e infinitas, secciones de tripulación determinadas por el camino mínimo entre sus estaciones de cabecera), estas simplificaciones son una buena estrategia para abordar inicialmente un problema tan complejo. Pero reducir estas simplifaciones y acercar la definición del problema a las restricciones reales que enfrentan las compañías de carga ferroviaria, genera herramientas más precisas al momento de la toma de decisiones. Es por esto que en esta tesis también presentamos extensiones del TDO que consideran nuevos escenarios no tenidos en cuenta en el TDO original. Estos escenarios contemplan la heterogeneidad de la flota de locomotoras permitiendo representar mejor las características de la operatoria de una empresa ferroviaria de carga.

Para resolver estos nuevos escenarios, presentamos una heurística de asignación que resuelve un subconjunto particular de instancias y una extensión del ILP propuesto por Jin et al. [6] que logra resolver cualquier instancia presentada. Además aplicamos el método iterativo por bloques sobre ambos métodos.

Los resultados que obtuvimos muestran que los dos primeros métodos tienen un desempeño similar tanto en calidad de solución como en tiempo de cómputo, y sus versiones iterativas logran conseguir una solución de calidad marginalmente inferior pero utilizando significativamente menos tiempo de cómputo.

Vale destacar que aunque ambos métodos hayan tenido un desempaño similar, la heurística de asignación solo logra resolver un conjunto particular de instancias, mientras que el ILP puede resolver cualquier instancia (sin tener en cuenta la restricción de tiempo de cómputo).

Como conclusión general, para resolver problemas exponenciales donde el tiempo de cómputo es una preocupación, es una buena estrategia buscar subdivir el problema en subproblemas de menor tamaño y complejidad aún a cambio de teóricamente explorar un espacio de soluciones menor. Ejemplos es esto es el método original de Jin et al. [6] para resolver el TDO que genera tres etapas sucesivas (generación de *block path*, generación de rutas candidatas y BTA) y también los algoritmos iterativos presentados en esta tesis que vuelven a aplicar la misma estrategia sobre el método de Jin et al.

Por último esta tesis deja abiertas ciertas líneas de investigación para futuros trabajos:

- Evaluar la calidad de los métodos en instancias más variadas: las instancias brindadas por INFORMS-RAS para la competencia [5] tienen la característica de que el costo

de traslado de los bloques por milla es abrumadoramente mayor al resto de los costos, sesgando ciertas decisiones sobre los métodos. Variar las características de los costos podría impactar en la calidad de los métodos y llevar a que otros métodos cobren relevancia.

- Generalizar la heurística de asignación: la heurística de asignación presentada demostró muy buenos resultados más allá de su simpleza, pero solo aplica a un conjunto acotado de instancias. Un posible trabajo futuro es lograr una heurística de asignación general, que no tenga ninguna restricción.

Bibliografía

[1] Ravindra K. Ahuja, Claudio B. Cunha, and Güvenç Şahin. Network models in railroad planning and scheduling. *In INFORMS Tutorials in Operations Research.*, 54-101, Published online: 14 Oct 2014. `http://dx.doi.org/10.1287/educ.1053.0013`.

[2] Vasek Chvátal. *Linear Programming.* W. H. Freeman, 1983.

[3] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[4] IBM. User's manual for CPLEXR. *IBM*, 2010.

[5] INFORMS-RAS. 2011 RAS Problem Solving Competition. 2011. `www.informs.org/Community/RAS/Problem-Solving-Competition/2011-RAS-Problem-Solving-Competition`.

[6] Jian Gang Jin, Jun Zhao, and Der-Horng Lee. A column generation based approach for the Train Network Design Optimization problem. *Transportation Research Part E: Logistics and Transportation Review*, 50:1–17, 2013.

[7] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[8] Ricardo A. Marchese and Marcos A Golato. El consumo de combustible y energía en el transporte. *Revista CET*, 33, 2011.

[9] Christos H. Papadimitrou. On the complexity of integer programming. *Journal of the ACM (JACM)*, 28(4):765–768, 1981.

[10] Laurence A. Wolsey. Integer programming. *John Wiley Sons, Inc.*, 1998.

Apéndice

A. DEFINICIONES

A continuación definiremos algunos términos comunes en la industria del transporte ferroviario que se usarán en el desarrollo de esta tesis.

- Vagón: Es donde se guarda la carga a ser transportada. Puede haber diferentes tipos de vagones, pero en este trabajo se supone que los vagones son todos idénticos.
- Bloque: Un conjunto de vagones que deben ser transportados juntos desde una estación de origen a una de destino.
- Tren: Un tren está conformado por una locomotora (que tendrá ciertas cualidades particulares como poder de tracción, velocidad, etc.) y un conjunto de bloques que serán tanto cargados como descargados del tren en el transcurso del viaje. Además en todo momento el tren deberá contar con una tripulación a bordo.
- *Block Swap*: La operación de transferir un bloque entre trenes. El *block swap* está definido en el evento que ocurre sobre el bloque y solo ocurre cuando el bloque pasa de un tren a otro y no cuando un tren lo recoge de su estación inicial o cuando lo descarga en su estación final.
- *Work event*: Es el evento que ocurre cuando un tren se detiene en una estación intermedia para incorporar o descargar un bloque. Aunque un tren se detenga en una estación y ocurran múltiples *block swaps* (se cargan o descargan múltiples bloques), solo sucederá un *work event*. Además, puede ocurrir el caso en que un tren se detenga en un estación a cargar un bloque (genera un *work event*) sin que ocurra ningún *block swap*. Esto sucede cuando un tren se detiene en una estación intermedia a levantar un bloque, pero esa estación es el origen del bloque, por ende, no se genera *block swap*, ya que estos ocurren solo cuando un bloque cambia de tren y no en sus estaciones de origen o llegada.
- Sección de tripulación (*Crew Segment*): Cada tripulación ferroviaria opera trenes entre dos estaciones prefijadas, llamadas sección de tripulación. Un tren puede atravesar múltiples secciones de tripulación durante su recorrido, por lo cual necesitará una tripulación diferente para cada una de estas secciones.
- Desbalance de tripulación: Es la diferencia absoluta entre la cantidad de trenes que circularon en cada dirección (ida y vuelta) en una determinada sección de tripulación. Si las secciones de tripulación están dadas entre dos estaciones A y B, el desbalance de tripulación se calcula como la diferencia absoluta entre la cantidad de trenes yendo de A a B y la cantidad de trenes yendo de B a A.
- Red ferroviaria: Es el conjunto de estaciones y vías por las que opera un empresa ferroviaria. Se puede representar abstractamente como un grafo, donde los nodos son las estaciones y los ejes corresponden a las vías de tren.
- Desbalance de trenes: Consiste en la diferencia absoluta entre los trenes que se inician en una estación y los trenes que terminan en esa misma estación. Por ejemplo, si de

una estación A inician cinco trenes y terminan en ella solo tres trenes, el desbalance en la estación A será de dos trenes.

- *Block to Train Assignment* (BTA): Es el plan en el que se detalla el camino por el que circulará cada bloque, y para cada tramo del trayecto de cada bloque se asigna un correspondiente tren.
- *Block Path*: Camino en la red ferroviario por el que se restringirá que circule un bloque

B. PRELIMINARES

B.1. Programación Lineal

La programación lineal permite expresar un problema de decisión en términos matemáticos para determinar cuál es la mejor solución dentro de un conjunto, generalmente grande, de soluciones factibles.

La programación lineal es una técnica para optimizar una función lineal, conocida como *función objetivo*, sujeto a que la solución de dicha optimización satisfaga un conjunto de igualdades y desigualdades lineales, conocidas como *restricciones lineales*.

Un *problema de programación lineal (PPL)* se expresa en forma canónica como:

$$\begin{array}{ll} \textit{maximizar} & c^t x \\ \textit{sujeto a} & Ax \leq b \\ & x \geq 0 \end{array}$$

Donde:

- $x \in \mathbb{R}^n$ representa el vector de variables positivas cuyo valor se debe determinar.
- $c \in \mathbb{R}^n$ es el vector de coeficientes de la función objetivo.
- $b \in \mathbb{R}^m$ es el vector de términos independientes.
- $A \in \mathbb{R}^{m \times n}$ es la matriz de coeficientes de las restricciones.

Otra forma de expresar un PPL es $z^* = max\{cx : Ax \leq b, x \geq 0\}$, donde z^* es el valor óptimo de la función objetivo.

Todo $x^* \in \mathbb{R}^n | Ax^* \leq b \wedge x^* \geq 0$ es una solución factible. A la región definida por todas las soluciones factibles se la conoce como *poliedro*.

Dados $A \in \mathbb{R}^{m \times n}$ y $b \in \mathbb{R}^m$, un *poliedro* es un subconjunto de $\mathbb{R}^n$ descripto por un conjunto finito de restricciones lineales $P = \{x \in \mathbb{R}^n : Ax \leq b\}$.

Si bien ya definimos un PPL en su forma estándar, existen múltiples maneras diferentes de plantearlo. Por ejemplo, se puede plantear como un problema de minimización, o con igualdades como restricciones, entre otras. Pero, sin importar cuál de estas variantes usemos, siempre existirá una formulación canónica equivalente. El beneficio de utilizar estas formas alternativas de plantear un PPL es lograr una formulación más comprensible.

Estos problemas pueden ser resueltos con el método *Simplex*, método de resolución general para problemas de programación lineal desarrollado por George Dantzig [2]. Este método se basa en el hecho de que la solución óptima de un PPL es alguno de los vértices del poliedro, conocidos como *puntos extremos*. De esta manera, *Simplex* explora los puntos extremos del poliedro en forma *inteligente*, eligiendo en cada iteración un punto que aumente el valor de la función objetivo (en el caso de la maximización). Este proceso se realiza en forma iterativa hasta que el valor de la función objetivo no pueda aumentar más: esto significa que se alcanzó el óptimo.

Está demostrado que el método *Simplex* tiene una complejidad algorítmica exponencial en el peor caso. Sin embargo, y a pesar de existir algoritmos polinomiales para resolver PPLs (por ejemplo, el presentado por Karmarkar [7]), *Simplex* demostró tener un muy

buen desempeño en la práctica. Esto lo convirtió en uno de los métodos de resolución de PPLs más utilizados en la actualidad.

B.2. Programación Lineal Entera

Existen problemas en la práctica que podrían ser planteados como PPL pero el dominio de algunas de sus variables es discreto y no continuo. Por ejemplo, se debe decidir cuántos autos conviene producir (variable entera), o se debe decidir si realizar una acción o no (variable binaria). Cuando todas las variables de un problema tienen un dominio discreto, llamaremos al problema un *Problema de Programación Lineal Entera* (PPLE). Cuando el problema sea una combinación de variables discretas y continuas, estaremos ante un *Problema de Programación Lineal Entera Mixto* (PPLEM).

La diferencia fundamental entre los problemas donde el dominio de las variables es continuas y aquellos con variables discretas es que estos últimos pertenecen a la clase *NP-hard* [9], por lo que no se conoce un algoritmo polinomial que los resuelva en forma general. Al no contar con un método general que pueda tratar con estos problemas en forma eficiente, se suele usar la idea de *relajación* del problema. Esto consiste en modificar el problema original para que siga contemplando todas las soluciones pero que, al mismo tiempo, pueda resolverse más fácilmente.

Llamamos a PR una relajación de un PPLE PE si:

- $PR \equiv max\{f(x) : x \in T \subseteq \mathbb{R}^n\}$
- $PE \equiv max\{c(x) : x \in X \subseteq \mathbb{R}^n\}$
- $X \subseteq T$, y
- $f(x) \geq c(x)$ para todo $x \in X$.

La forma más sencilla de relajar un PPLE es modificar el dominio de las variables enteros y permitir que tomen valores reales. Esta relajación es conocida como *Relajación Lineal* y el problema relajado termina siendo un PPL. Un método sencillo para obtener una solución de un PPLE podría ser resolver la relajación lineal y luego redondear la solución obtenida de manera que termine siendo una solución factible del problema original. Si bien esto parece una buena heurística que se podría aplicar en forma general, existen muchos ejemplos en los cuales una solución redondeada dista mucho de la solución óptima del problema original. Además, no hay garantías de que al redondear la solución del problema relajado, la solución obtenida sea factible en el problema original.

La diferencia entre el valor óptimo del PPLE y el de su relajación lineal se conoce como *gap* y suele usarse como medida de la calidad de la relajación. Como no es posible conocer a priori el valor del *gap* se utiliza una cota superior del mismo dada por la diferencia entre el valor de la mejor solución factible conocida del PPLE y el valor óptimo de la relajación. Esto permite conocer la calidad de una solución y, en algunos casos, demostrar la optimalidad de la misma (cuando ambos valores coinciden).

B.3. Generación de Columnas

En la práctica es común resolver PPLs que cuentan con un gran número de variables. Este número es tan grande que no es prácticamente posible considerar todas las variables en forma explícita.

La estrategia, en estos casos, es resolver la relajación lineal restringiéndose sólo a un subconjunto de variables. Esta idea se basa en el hecho de que la mayoría de las variables no serán parte de la solución óptima, por lo que en teoría sólo es necesario un subconjunto chico de todas ellas para resolver el problema de forma óptima. Al no considerar todas las variables, la solución obtenida será factible pero no hay garantía de que sea óptima. Para garantizar la optimalidad de la solución es necesario buscar entre las variables no consideradas si existe alguna candidata que pueda mejorar el valor de la función objetivo. Este proceso se denomina *generación de columnas*.

Esta técnica de generación de columnas separa al problema original en dos: un problema *maestro* y un problema *esclavo*. El problema maestro es el problema original pero sólo considerando un subconjunto de todas las variables. El problema esclavo consiste en identificar una nueva variable (o columna) para agregar al subconjunto mencionado antes. La nueva columna a ingresar debe ser generada de tal manera que, de ser incluida en la base, mejore el valor de la función objetivo. En el caso de una minimización, *Simplex* establece que para mejorar la función objetivo en una iteración dada, la nueva columna debe tener un *costo reducido negativo*. En términos matemáticos esto significa que:

$$c_i - y^* A_i < 0$$

Donde:

- A_i es la columna i de la matriz A.
- c_i es el coeficiente de costo de la variable correspondiente a la columna i.
- y^* es el vector solución del problema *dual* asociado al problema original restringido.

Los valores de y^* se conocen como *prices* de las columnas de A y, de cierta manera, ponderan la decisión de qué variable debe ingresar a la base en la correspondiente iteración.

El problema esclavo se encargará entonces, en cada iteración del problema maestro, de buscar una columna A_i que verifique la anterior inecuación. Si dicha columna existe, se agregará al subconjunto de variables y se volverá a optimizar el problema maestro para luego repetir el proceso. Si por el contrario, la columna buscada no existe, no será posible mejorar el valor de la función objetivo. En este caso, se puede concluir que se alcanzó el óptimo del problema maestro. Existe la posibilidad de que en cada iteración se encuentre una nueva variable que verifique la inecuación y se termine agregando todas las variables al problema maestro. Por esto, en la práctica se utilizan, además de la optimalidad, otros métodos para detener la resolución de la generación de columnas. Las formas más usadas son un límite de tiempo o un límite de las columnas generadas. Cuando se alcanza uno de estos límites antes de demostrar optimalidad, la solución obtenida será factible pero no podremos garantizar si es la solución óptima.

En las formulaciones tradicionales, donde se cuenta con todas las variables en forma explícita, es fácil obtener una solución factible inicial. En cambio, en un modelo que utiliza generación de columnas, es necesario proveer un subconjunto inicial de variables que conformen una solución factible para el problema maestro. Dependiendo del problema, esto puede resultar trivial o puede requerir de un algoritmo complejo de generación de columnas iniciales.

B.4. K Caminos Mínimos

El problema de K Caminos Mínimos es una extensión del problema de Camino Mínimo en un grafo. Este problema consiste en, como su nombre lo indica, encontrar entre todos los caminos existentes entre dos nodos de un grafo, los k caminos de menor distancia.

B.4.1. Generalización de Dijkstra

Una forma de resolver este problema es aplicar una generalización del algoritmo de Dijkstra de camino mínimo. Sea $G(V, E)$ un grafo dirigido donde V es el conjunto de nodos y E el conjunto de ejes. Sea $c(u, v)$ la distancia entre los nodos u y v. Sea o el origen y d el destino de los caminos que se buscan encontrar y k la cantidad de caminos mínimos. Llamaremos P_u a un camino desde o hasta u, B a un *heap* que contendrá caminos y estará ordenado en función de la longitud de los caminos de menor a mayor. Sea P un conjunto de caminos entre o y d y $cantidad_u$ el número de caminos encontrados hasta el nodo u. La generalización de Dijkstra consiste en:

Algorithm 3 Generalización de Dijkstra para K caminos mínimos

1: $\mathbf{P} \leftarrow \emptyset$
2: $\mathbf{count_u} \leftarrow 0, \forall u \in V$
3: $\mathbf{B} \leftarrow \{\{o\}\}$
4: **while** **B** no este vació $\wedge$ $\mathbf{count_u} < \mathbf{k}$ **do**
5: Sea $\mathbf{P_u}$ el camino de menor longitud en **B** con costo **C**
6: $\mathbf{B} = \mathbf{B} - \{\mathbf{P_u}\}$
7: $\mathbf{count_u} = \mathbf{count_u} + 1$
8: **if** $\mathbf{u} = \mathbf{d}$ **then**
9: $\mathbf{P} = \mathbf{P} \cup \mathbf{P_u}$
10: **if** $\mathbf{count_u} < \mathbf{k}$ **then**
11: **for** **v** adyacente a **u** **do**
12: **if** **v** no pertenece a $\mathbf{P_u}$ **then**
13: Sea $\mathbf{P_v}$ un nuevo camino con costo $\mathbf{C} + \mathbf{c(u, v)}$ y formado al concatenar el eje $(\mathbf{u}, \mathbf{v})$ al camino $\mathbf{P_u}$
14: $\mathbf{B} = \mathbf{B} \cup \{\mathbf{P_v}\}$
15: **return** **P**

C. CAMINOS MÍNIMOS DE SECCIONES DE TRIPULACIÓN

A continuación expondremos todas las secciones de tripulación que en el DataSet2 tienen más de un posible camino mínimo y el camino que utilizamos en el desarrollo de esta tesis, descripto como una sucesión de estaciones.

- $(410, 6647) : 410 \to 2176 \to 6541 \to 6647$
- $(485, 2196) : 485 \to 4962 \to 9355 \to 9228 \to 8469 \to 6207 \to 2196$
- $(599, 8429) : 599 \to 825 \to 9024 \to 3777 \to 8429$
- $(1107, 9057) : 1107 \to 3753 \to 10826 \to 11181 \to 10819 \to 9057$
- $(1112, 4376) : 1112 \to 10032 \to 1792 \to 6632 \to 4170 \to 4376$
- $(1339, 8429) : 1339 \to 825 \to 9024 \to 3777 \to 8429$
- $(1699, 4227) : 1699 \to 8198 \to 6965 \to 5160 \to 4227$
- $(1699, 4376) : 1699 \to 8078 \to 5941 \to 4376$
- $(1954, 2276) : 1954 \to 7344 \to 7697 \to 2276$
- $(1954, 5899) : 1954 \to 7344 \to 11186 \to 5899$
- $(1954, 8765) : 1954 \to 899 \to 8765$
- $(2031, 6991) : 2031 \to 5686 \to 9305 \to 7005 \to 5879 \to 7424 \to 6991$
- $(2228, 11018) : 2228 \to 2701 \to 6991 \to 7424 \to 5879 \to 7005 \to 9305 \to 11018$
- $(3753, 9057) : 3753 \to 10826 \to 11181 \to 10819 \to 9057$
- $(5899, 6872) : 5899 \to 1264 \to 6383 \to 3864 \to 6872$
- $(6069, 6647) : 6069 \to 6541 \to 6647$
- $(6425, 9917) : 6425 \to 2544 \to 3683 \to 4968 \to 6494 \to 445 \to 9917$
- $(6991, 5879) : 6991 \to 7424 \to 5879$
- $(6991, 11018) : 6991 \to 5879 \to 7005 \to 9305 \to 11018$
- $(7883, 8429) : 7883 \to 11039 \to 599 \to 825 \to 9024 \to 8429$