Inteligencia Artificial

Informe Final: Manejo de Trenes en Estaciones Ferroviarias

Martín Villanueva A.

28 de Junio 2016

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

Los problemas de planificación horaria para sistemas ferroviarios, han sido estudiados desde distintas perspectivas en las últimas dos décadas. El problema que se estudio en este artículo es el *Rolling Stock Unit Management on Railway Sites Problem*, propuesto en el desafío ROADEF/EURO 2014. Este es un enfoque innovador, pues modela la complejidad existente en las estaciones ferroviarias, de un modo integrado. En este documento se realiza un estudio detallado de la descripción y formulación del problema, así como un estado del arte, haciendo una revisión de los distintos modelos, técnicas y variantes conocidas al día de hoy.

Palabras Clave: *Train Timetable Problem, Departure Matching Problem, Rolling Stock Unit Managment on Railway Sites Problem.*

1. Introducción

El presente documento tiene como motivación entender y realizar una revisión concisa al *estado del arte* para el problema propuesto en ROADEF/EURO Challenge 2014 [1]. Aquí se propone modelar la totalidad de operaciones en una estación de ferrocarriles; Asignación de trenes a las correspondientes salidas, manejando las rutas y uso de recursos que hacen estos en la estación, de una manera integrada. La motivación de este problema, viene desde la SNFC (*Société Nationale des Chemins de Fer Français*), quienes exponen las dificultades existentes hoy en día en los sistemas ferroviarios, en donde hay una alta tasa de flujo de trenes, así como

una alta congestión de tráfico, para lo cual se requieren métodos automatizado, que ayuden a generar las planificaciones de horario y rutas, de la mejor forma posible.

En la sección 2 se define el problema y sus alcances. En la sección 3, se procede con el estudio del *estado del arte*, considerando los trabajos directamente relacionados con el problema, como aquellos trabajos sobre subproblemas o problemas relacionados. En la sección 4 se muestra el modelo matemático propuesto a los participantes del concurso. Se finaliza en la sección 5 con las conclusiones obtenidas.

2. Definición del Problema

La definición del problema que se expone en esta sección, esta basada en *ROADEF/Euro Challenge 2014* [1], con ciertas modificaciones y simplificaciones que se detallan en lo que sigue.

Como se mencionó anteriormente, el principal propósito del problema, consiste en encontrar la mejor manera de manejar trenes en estaciones ferroviarias, entre las llegadas, las salidas y el uso de recursos que hacen en la estación. Todo este proceso puede ser visto como una secuencia de subproblemas, sin embargo aquí se trata y modela de una manera integrada. En vista de que el problema tiene en cuenta una gran cantidad de *dimensiones*, a continuación se describen cada una de ellas de manera simple.

Horizonte de Planificación. El horizonte de planificación corresponde al conjunto de instantes de tiempo (*discretizados*), sobre el que se espera realizar toda la planificación del sistema. El formato de cada instante de tiempo es (d hh:mm:ss), con d $\in \{1, nbDays\}$ el día respectivo, hh $\in \{0, 23\}$ la hora, mm $\in \{0, 59\}$ el minuto, y ss $\in \{0, 59\}$ el segundo. El conjunto de todos los instantes se denominará $\mathcal{H}$. Con esta representación, el menor intervalo de tiempo permitido es un segundo, lo que permite representar el tiempo con una alta precisión. Sin embargo dependiendo de la instancia, pueden tomarse duraciones más largas (múltiplos de segundos, o minutos), reduciendo el tamaño del *espacio de búsqueda*.

Llegadas. Las llegadas corresponden al ingreso de trenes en el sistema. Los tiempos de llegada (a la plataforma), son datos de entrada y por lo tanto no modificables. Hay que tomar en cuenta que los trenes entran al sistema/estación un tiempo antes que su hora de llegada. Durante este periodo intermedio realizan una *secuencia de llegada*, que representan la secuencia de recursos (y los respectivos tiempos) de los cuales hace uso el tren, antes de llegar a la plataforma para que los pasajeros bajen. Los aspectos y restricciones a considerar son los siguientes:

1. Cada llegada $a \in \mathcal{A}$ tiene un conjunto de plataformas preferidas a usar prefPlat_a. Esta es una restricción blanda, pero penalizada en la función objetivo.
2. Existen tiempos ideal y máximo de permanencia en la plataforma: idealDwell_a y maxDwell_a. Tiempos inferiores y superiores al idealDwell_a son penalizados, pues el primero disminuye la satisfacción de los pasajeros que deben bajar, y el segundo constituye un mal uso de los recursos del sistema. En cualquier caso, el tiempo de permanencia no puede superar el máximo maxDwell_a.
3. Es posible no cubrir una llegada, esto es, que el tren asociado a la llegada no entre a la estación. Como es de esperar, esto incurre una penalización.

Salidas. Las salidas corresponden a la ida de trenes del sistema. Análogo a las entradas, los tiempos y secuencias de salida (ruteo entre los recursos hasta salir de la estación) son datos de entrada fijos. La principal tarea a cumplir aquí, es asignar un tren a cada salida. Los aspectos y restricciones a considerar son los siguientes:

1. No asignar un tren a un salida es penalizado gravemente en la función objetivo. Adicionalmente, no puede asignarse más de un tren a un salida.

2. Cada salida $d \in \mathcal{D}$ tiene un conjunto de plataformas preferidas a usar prefPlat_d. Esta es un restricción blanda, pero penalizada en la función objetivo.
3. Existen tiempos ideal y máximo de permanencia en la plataforma: idealDwell_d y maxDwell_d. Las penalizaciones siguen la misma lógica que en las llegadas.
4. Una salida tiene asociado un conjunto de categorías de trenes compatibles compCatDep_d. Esto va asociado a las características que debe tener el tren para hacer el viaje, y constituye una restricción fuerte (no se puede violar).
5. Cada tren tiene una *distancia por recorrer antes de mantención* (remDBM) y *tiempo de viaje antes de mantención* (remTBM). Cada salida tiene asociada un distancia y tiempo necesario para realizar el viaje. Luego el tren asignado debe tener remDBM y remTBM mayor o iguales a los que requiere la salida.

Trenes. Un tren se define como una unidad móvil de *visita* en el sistema. Esto quiere decir que no se considera como tren a la unidad física; un mismo tren que visita más de una vez el sistema, se toma como trenes distintos. El conjunto de trenes $\mathcal{T}$ se compone de dos tipos de trenes: Los trenes presentes en la estación desde el inicio del horizonte de tiempo $\mathcal{T}_I$, y los trenes asociados a llegadas $\mathcal{T}_A$. Las principales consideraciones a tener con los trenes son las siguientes:

1. Cada tren es una unidad atómica; No se pueden separar en vagones, ni recombinarlos con los de otros trenes. Por lo tanto un tren es la unidad de asignación más pequeñas. Como nota, esta restricción es acorde a la naturaleza de los trenes modernos.
2. Un tren $t \in \mathcal{T}$ esta asociado a una categoría cat_t, que define las características técnicas que comparten. Se puede pensar en los trenes de una misma categoría como el mismo tren (fabricado en serie por alguna compañía), pero con diferentes condiciones de mantención iniciales.
3. No ocupar los trenes inicialmente en el sistema es una opción posible, pero será penalizada. Tales trenes no usarán recursos durante el horizonte de planificación.
4. Asociado una llegada y salida, puede haber un *reuso*. Esto consiste en definir previamente (dato de entrada) cuál tren que viene llegando, será asignado a una salida. Existe un costo asociado a no cumplir un reuso dado. Los reusos no serán respetados, siempre y cuando esto beneficie en mayor medida a otros objetivos. Por último, cada salida $d \in \mathcal{D}$ puede estar asociada a lo más en un reuso con la entrada $a \in \mathcal{A}$, y de igual modo al revés.

Mantenciones. Los trenes deben ser sometidos a mantenciones regularmente, de modo que cumplan las condiciones de seguridad y comodidad necesarias que requiere cada salida. Cómo se mencionó anteriormente, hay dos parámetros a considerar en las mantenciones:

1. **TBM**, el tiempo de viaje antes de realizar mantención (*Time Before Maintenance*). Este va asociado al comodidad de los pasajeros (que el tren esté limpio, asientos en buenas condiciones, etc).
2. **DBM**, la distancia de viaje antes de realizar mantención (*Distance Before Maintenance*). Se relaciona con condiciones de seguridad (combustible, estado de frenos, etc).

Las operaciones de mantención básicamente reajustan los valores remTBM y remDBM a los valores máximos de cada tren. Las consideraciones a tener en cuenta son:

1. El movimiento de un tren dentro de la estación de resta a remTBM y remDBM (despreciables en comparación al de los viajes).

2. Las mantenciones se realizan en las *Instalaciones de Mantenimiento* descritas en la siguiente sección.

Recursos de Infraestructura. Los trenes en el sistema, están siempre haciendo uso de algún recurso (ya sea moviéndose o estacionados). Estos recursos son 5 tipos: pistas simples $\mathcal{S}$, plataformas de llegada o salida $\mathcal{P}$, instalaciones de mantenimiento $\mathcal{F}$, grupos de pistas $\mathcal{K}$ y estacionamientos $\mathcal{Y}$. Las pistas simples, plataformas, y instalaciones de mantenimiento son consideradas como porciones individuales de una pista. Por otro lado los grupos de pistas se constituyen de varias pistas de forma agregada. Una configuración típica para los recursos del sistema pueden verse en Figura 1.

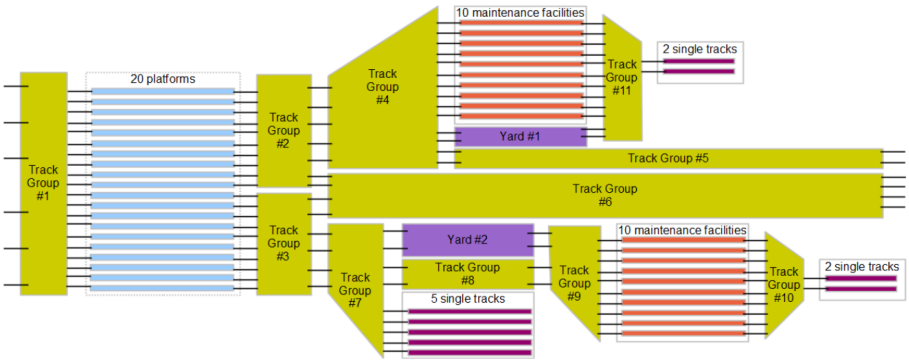

Figura 1: Ejemplo de recursos de infraestructura (Fuente [1])

Es muy importante considerar la forma en que los trenes se mueven entre los recursos y sus restricciones. Los principales aspectos se listan como sigue:

1. Un mismo tren no puede estar ocupando dos recursos en el mismo instante de tiempo (algo que sí sucede en la realidad). Por lo tanto cada tren se considera como un objeto puntual que puede moverse de un recurso a otro de modo instantáneo.
2. Cada recurso $r \in \mathcal{R}$ tiene un conjunto de recursos vecinos neighSet_r, que definen las posibles transiciones de un tren, esto es, un tren en un recurso r, sólo puede moverse a un recurso vecino en neighSet_r.
3. Usualmente (hay excepciones) cada recurso puede ser accedido por dos lados, que se denotan A y B. Entonces el conjunto de recursos vecinos de r puede ser descompuesto en $\text{neighSet}_r = \text{neighSet}_r^A \cup \text{neighSet}_r^B$. Adicionalmente se tiene como restricción $\text{neighSet}_r^A \cap \text{neighSet}_r^B = \emptyset$, esto es, que no se generen ciclos entre los recursos.
4. Las excepciones al punto anterior las constituyen los recursos al extremo del sistema, por donde los trenes entran o salen.
5. Los recursos tienen en ambos extremos *puertas* por donde se realiza la transición hacia otro recurso. Cada puerta $g \in \mathcal{G}_r$ de un recurso r, está asociada a una única puerta de un recurso vecino (si la puerta está en el fin del sistema, puede no tener puerta vecina asociada).
6. Las pistas simples, plataformas y instalaciones de mantención tienen como máximo una puerta en cada lado.
7. Los grupos de pistas y estacionamientos pueden tener mas de una puerta en cada lado.
8. Algunos recursos pueden ser utilizados sólo por un tipo específico de tren (por lo general en las estaciones de mantenimiento). Por ello cada recurso r tiene su conjunto de categorías de trenes compatibles compCatRes_r.

9. Pueden haber recursos ocupados con antelación en el sistema. Estos factores son externos al problema y fijos, siendo algunos ejemplos: trenes de mantenimiento, trenes que no terminan en la estación, trenes externos de otras compañías, entre otros.

Teniendo una idea de la mecánica de uso de recursos del sistema, a continuación se entregan detalles específicos de cada tipo de recurso.

1. **Pistas Simples.** Corresponden a pistas unitarias, usada para el movimiento de los trenes dentro de la estación. Estos son (usualmente) los recursos que utilizan los trenes para entrar y salir del sistema. Cada pista simple $s \in \mathcal{S}$ tiene asociado un largo total lenght_s y cantidad de trenes capa_s, que debe ser respetado en cada instante de tiempo.
2. **Plataformas.** Estas representan las pistas en donde los pasajeros pueden salir o abordar un tren. Por lo tanto son estas, las que deben ser asignadas a cada llegada o salida respectivo. A diferencia de las pistas simples, una plataforma $p \in \mathcal{P}$ sólo está restringida en capacidad por su largo lenght_p.
3. **Instalaciones de Mantenimiento.** Estas instalaciones son pistas donde los trenes se detienen a realizar su respectiva mantención (reajustar su TBM y DBM). Es muy importante notar que cada instalación de mantenimiento $f \in \mathcal{F}$ está equipada para hacer sólo un tipo de operación ("D" o "T"). Por lo tanto si un tren requiere hacer ambos mantenimientos, deberá pasar por dos instalaciones distintas. Por último, al igual que las plataformas, su capacidad está restringida por su largo lenght_f.
4. **Grupos de Pistas.** Son un conjunto agregado de pistas, utilizados para mover a los trenes entre los distintos recursos del sistema. Todas las puertas de un lado son alcanzables desde el otro (y viceversa), por lo tanto si en un lado hay m puertas y en el otro lado hay n, existen m n posibles rutas para atravesar el grupo de pistas. Un grupo de pistas $k \in \mathcal{K}$ tiene dos parámetros importantes: el tiempo necesario para atravesar k trTime_k (constante), y el tiempo mínimo que debe haber entre dos trenes sucesivos para que no estén tan cerca hwTime_k. Técnicamente los grupos de pistas son una estructura compleja, que debe manejar normas de seguridad y conflictos que suelen ocurrir. Sin embargo el problema se abstrae de esta complejidad, viendo a tal estructura como una *caja negra*.
5. **Estacionamientos.** Al igual que el anterior, son también un conjunto de pistas pero utilizadas para estacionar trenes. Por lo tanto un tren puede permanecer en un estacionamiento sin limites de tiempo. Un estacionamiento $y \in \mathcal{Y}$ está limitado en capacidad por la cantidad de trenes permitidos capa_y.

Objetivo. Tal como se plantea el problema, la objetivo general se descompone en múltiples objetivos, que corresponden a minimizar las penalizaciones descritas anteriormente. Luego la función a minimizar es:

$$f = f^{\text{uncov}} + f^{\text{over}} + f^{\text{plat}} + f^{\text{pref}} + f^{\text{reuse}}, \tag{1}$$

donde cada una representa:

1. f^{uncov}: Costo por no cubrir una salida.
2. f^{over}: Costo asociado al sobre-mantenimiento.
3. f^{plat}: Costo por uso de las plataformas.
4. f^{pref}: Costo por no respetar la preferencia de uso de plataforma.
5. f^{reuse}: Costo por no respetar un reuso dado inicialmente.

un análisis más detallado de (1) será visto en la sección de Modelo Matemático.

Solución Esperada. Como solución al problema, se espera una conjunto de programación de eventos para cada tren, esto es, una secuencia de eventos junto con los recursos usados, y los tiempos asociados. Los diferentes tipos de eventos pueden ser: `EnterSystem`, `ExitSystem`, `Arrival`, `Departure`, `EnterResource`, `ExitResource`, `BeginMaintenance` y `EndMaintenance`. Un ejemplo de posible solución para un tren dado se da en Figura 2. Teniendo esta información para todos los trenes, será posible inferir el estado del sistema

Train	Time	Event type	Resource	Gate	Complement
...	...	...	...	...	...
Train12	d_2 07:35:00	BegJunction	Yard5		Train9+Train12
Train12	d_2 07:38:00	EndJunction	Yard5		Train1+Train9+Train12
Train12	d_2 09:02:00	ExitResource	Yard5	B3	
Train12	d_2 09:02:00	EnterResource	TrGroup7	B1	
Train12	d_2 09:04:00	ExitResource	TrGroup7	A2	
Train12	d_2 09:04:00	EnterResource	TrGroup8	A1	
Train12	d_2 09:09:00	ExitResource	TrGroup8	B2	
Train12	d_2 09:09:00	EnterResource	Facility1	A1	
Train12	d_2 09:09:00	BegMaintenance	Facility1		D
Train12	d_2 11:09:00	EndMaintenance	Facility1		D
Train12	d_2 11:45:00	ExitResource	Facility1	A1	
Train12	d_2 11:45:00	EnterResource	TrGroup8	B3	
Train12	d_2 11:50:00	ExitResource	TrGroup8	A8	
Train12	d_2 11:50:00	EnterResource	Platform14	A1	
Train12	d_2 12:20:00	Departure	Platform14		Departure36
Train12	d_2 12:20:00	ExitResource	Platform14	A1	
Train12	d_2 12:20:00	EnterResource	TrGroup9	A2	
Train12	d_2 12:25:00	ExitResource	TrGroup9	B2	
Train12	d_2 12:25:00	ExitSystem	TrGroup9		

Figura 2: Extracto de programación de eventos para un tren dado (Fuente [1])

y sus recursos, en cada instante del horizonte de planificación.

3. Estado del Arte

En la presente sección se hace un estudio acerca de cómo nace este problema y la forma en que ha sido abordado. Pese a que hay gran cantidad de trabajo realizado (y en desarrollo) al respecto [2], hay pocos que han sido publicados. Por lo tanto también se expondrán las variantes o sub-problemas más conocidos, así como los distintos enfoques con los que han sido abordados.

Inicios del Problema. En el problema de estudio en el presente artículo, corresponde a una versión *simplificada* del problema propuesto en el ROADEF/EURO Challenge 2014 [1], competencia organizada por la *SFrench Operational Research (OR) and Decision Support Society* (ROADEF) y la *European Operational Research Society* (EURO) en conjunto con la *Société Nationale des Chemins de Fer Français* (SNFC). El objetivo de esta, fue encontrar la mejor forma realizar el manejo, ruteo y asignación de trenes en una estación ferroviaria, tal como se explicó en la sección 2. La simplificación nombrada, consiste en que se da por alto una de las dimensiones del problema, que es la unión y separación de trenes. Como se explicó en la sección 2, los trenes son las unidades de asignación más pequeñas a utilizar, esto es, no pueden ser separados ni recombinados sus vagones. Sin embargo, en la medida de poder satisfacer salidas con gran cantidad de pasajeros, es posible *unir* dos trenes en el sistema para cumplir con la capacidad. El proceso de *unir* y *separar* trenes en las estaciones es llevado en los *grupos de pistas*, y agrega una complejidad considerable a la formulación y a su solución. Una buena cantidad de equipos lograron dar con soluciones satisfactorias [2]. Algunas de estas se detallan en lo que sigue.

Propuestas de Solución. El primer trabajo publicado corresponde a *A Math-Heuristic Framework for the ROADEF/EURO Challenge 2014* [3], cuyos autores participaron en la categoría junior del concurso obteniendo el segundo lugar. El nombre *math-heuristic* se debe a que se combinan métodos exactos con heurísticas para hallar una solución. El problema original se aborda como cuatros sub-problemas, tal como se describe en la figura 3. Cada etapa se resume a continuación:

1. Arrival&Departure Matching. En esta primera etapa, se intenta hacer un *match* para cada salida, con un tren compatible (tren inicial o de llegadas). El problema es planteado como MIP (*Mixed Integer Programming*), con básicamente dos objetivos: Minimizar el número de cancelaciones de salidas (salidas sin cubrir) y maximizar la cantidad de reusos entre las llegadas y salidas. Debido a la gran cantidad de variables involucradas, este es resuelto usando el método de generación de columnas. Para generar los *matches* se generan los conjuntos Comp(d) y Comp(t) que contienen los trenes compatibles para cada salida d y las salidas compatibles a cada tren t, respectivamente. Estos se computan filtrando las combinaciones inviables, y utilizando heurísticas (eg. eliminando *matches* en espacios de tiempo muy grandes).
2. Platform Assignment. En segundo lugar es necesario asignar las plataformas a utilizar por cada llegada y salida a cubrir. Se intenta aquí asignar de manera que se minimicen las cancelaciones de llegada y salidas, y que cada una sea asignada a su plataforma preferida, en la medida de lo posible. Este también se plantea como MIP, pero se resuelve buscando una solución exacta.
3. Arrival&Departure Sequence Assigner. Luego para cada una de las llegadas y salidas a cubrir, se generan patrones de uso de los grupos de pistas, de modo que no existan conflictos entre en las secuencias respectivas, ni el uso de recursos impuestos inicialmente.
4. Simulated Annealing / Train Routing. En última instancia, *Simulated Annealing* (SA) es utilizado para generar rutas (secuencia de recursos) que los trenes realizan en la estación. Para ello en cada iteración se *rutea* a un tren, seleccionando una ruta aleatoria basada en los *matches* y asignación de recursos anteriores. Cada nueva solución es aceptada dependiendo del costo y temperatura.

Cabe mencionar el este artículo no trabaja las condiciones de uniones de trenes en llegadas y salidas. En definitiva este enfoque se basa en ocupa SA para generar rutas aleatorias, apoyándose en resultados precomputados para la utilización de recursos y evitar conflictos.

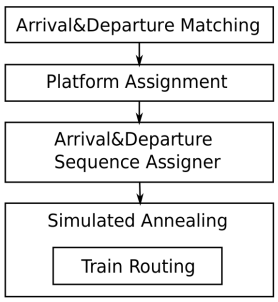

Figura 3: Esquema de solución en Math-Heuristic Framework (Fuente [3])

El segundo trabajo *Roadef Challenge 2014: A Modeling Approach, Rolling stock unit management on railway sites*[4], presenta un acabado estudio con un modelo similar al anterior;

Descompone el problema original en dos etapas de decisión: asignación y enrutamiento. La idea de generar estos dos subproblemas, es reducir la complejidad inicial, y ocupar la técnica de resolución indicada para cada uno. Por un lado, la primera etapa se compone básicamente de asignar trenes (de las llegadas o inicialmente en el sistema) a las salidas, asignar plataformas, y el resto de recursos del sistema. Este tipo de problemas son fácilmente formulables en MIP, y pueden ser resueltos de manera eficiente con dicha representación (por generación de columnas). Por otro lado, la fase de enrutamiento de trenes en la estación se desacopla totalmente de la fase de asignación, y se propone utilizar CP (*Constraint Programming*) para su solución, pues es la opción más viable y eficiente. Algunos detalles se listan a continuación para cada fase (Ver Figura 4).

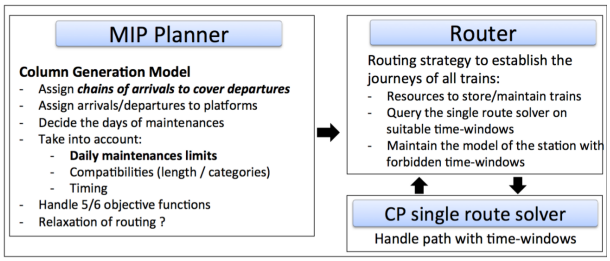

Figura 4: Esquema de solución de Modeling Approach (Fuente [4])

1. Asignación. El problema de asignación abarca asignar trenes a las salidas, asignar plataformas a las llegadas y salidas y asignar las mantenciones. La función objetivo que se plantea planteada considera: *llegadas/salidas no cubiertas*, *trenes iniciales no usados*, *plataformas preferidas no satisfechas*, *reusos no satisfechos* y *sobre mantenciones*. Todo lo anterior se modela por medio de MIP. Se hace luego una extensión a esta formulación para considerar el caso de llegadas *enlazadas* con salidas previas, verificando que tal problema se reduce a *Dependent Matching Problem* (DPM) que es NP-completo.
2. Enrutamiento. Para esta se parte de la premisa de que es posible computar las rutas de los trenes en la estación, sin cuestionar los resultados previos de la fase de asignación. Aquí modela la estación y sus recursos como un grafo, y muestra que el problema de enrutar un sólo tren se reduce a *Resource Constrained Shortest Path Problem* (RCSPP) que es NP-hard. En base a esto, un modelo de CP es inferido para definir las rutas, y luego se computan una por una las rutas de todas las llegadas/salidas, de modo que si el *solver* de CP es incapaz de encontrar una solución, se cancela la llegada/salida asociada.

Los autores reconocen que uno de los puntos débiles del modelo propuesto está en la estrategia de enrutamiento, sobre la cual es posible usar distintas heurísticas para tratar el problema de formas más integrada, y así minimizar las cancelaciones.

Un estudio interesante es el que realiza (nuevamente) Haahr et al. [5], en donde se enfocan en la tarea de analizar, modelar y solucionar uno de los subproblemas de *Rolling Stock Unit Management on Railway Sites Problem* (RSUM) más importantes; El *train Departure Matching Problem* (DMP), que consiste en realizar las asignaciones de trenes (entre los que están inicialmente, y los de las llegadas) a las correspondientes salidas, satisfaciendo las restricciones de compatibilidad y mantenciones. La importancia del DPM subyace en que cualquier enfoque utilizado para solucionar el RSUM, se ve afectado en gran parte por cómo los trenes son asociados a las salidas. En primer lugar se muestra que el DMP es NP-hard por reducción al *0-1 Multiple Knapsack Problem*. Luego se propone resolver el problema

con métodos exactos, formulando el problema como MIP (*Mixed Integer Programming*) de dos formas equivalentes; La primera de ellas es resulta con el MIP *solver* IBM ILOG CPLEX, y la segunda con métodos de generación de columnas. En las pruebas ambos métodos demoran una gran tiempo en computar sus soluciones. Para ello propone se proponen heurísticas para generar columnas, eliminando las columnas innecesarias en el modelo anteriores.

La tesis de Mulders y Scholliers [6] constituye un análisis en detalle respecto al problema. A modo de estudio preliminar, se modela inicialmente el problema con tres enfoques distintos. El primero de ellos es plantear las asignaciones de salidas a trenes como un *Matching* en un grafo bipartito, la segunda ocupa *constraint programming* restringiendo las posibles configuraciones, y la tercera se plantea resolver mediante greedy. El cómputo de las rutas que siguen los trenes en el sistema, se hace a traves de MIP con LPsolve. Pese a que los tres enfoques sufren de serios problemas en cuanto a tiempo de computación (no son factibles de utilizar), su análisis ayudó a desarrollar la solución final. Esta última posee cuatro componentes principales:

1. **Objects**, que corresponden a una representación conveniente (para esta formulación) de los datos y variables del sistema.
2. **Oracle**, es un componente encargado de verificar que distintas acciones puedan ser llevada a cabo sin violar restricciones, así como determinar donde y cuando pueden ser llevadas a cabo (mantenciones por ejemplo).
3. **WaysPC**, que es capaz de computar rutas compatibles para todos los trenes presentes en el sistema en un instante dado.
4. **Solver**, componente principal que hace uso de los anteriores, para encontrar soluciones usando una metodología greedy.

El objetivo principal del solver propuesto es dejar la menor cantidad de partida sin asignar, esto pues por ejemplo, es mucho más importante cubrir una partida que asignarla a una plataforma no preferida. En la Figura 5 se muestra el porcentaje de partidas cubiertas en un horizonte de 7 días, para 6 distintas instancias del desafío. Se puede notar que el desempeño de las asignaciones va empeorando en el tiempo. Los autores concluyen que el

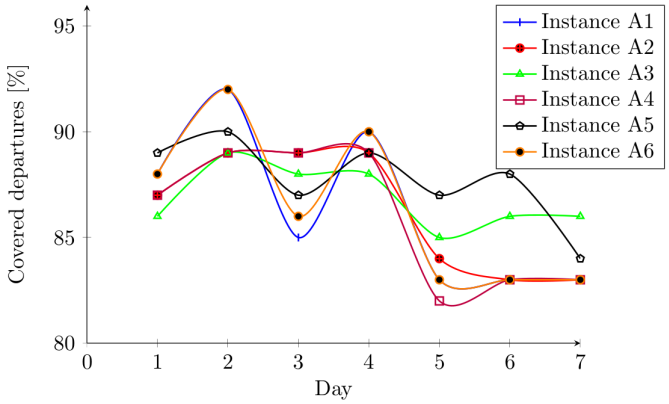

Figura 5: Porcentaje de salidas asignadas por Solver (Fuente [6])

algoritmo desarrollado, es de hecho competitivo, pues es capaz de entregar soluciones a instancias del ROADEF/EURO Challenge 2014 en menos de un minuto (el tiempo límite para estas instancias era de 10 minutos). Por otro lado obtiene desempeños similares a las mejores soluciones del desafío [2]. Por último se propone como trabajo futuro, determinar

heurísticas que permitan mejorar aún más la calidad de las soluciones, así como mejorar la definición de los objetivos en el modelo.

Problemas Relacionados. El problema en estudio ha sido estudiado y abordado múltiples maneras; Distintos objetivos, restricciones y metodologías de solución. En el presente estudio, se presentan las más notables entre estas.

Uno de los principales sub-problemas es conocido como *Train Timetabling Problem*, el cual consiste en generar una planificación horaria para los trenes en una estación, para cubrir las salidas satisfaciendo restricciones particulares del sistema. Este problema ha sido modelado y resuelto mediante múltiples técnicas. Uno de los primeros estudio formales lo entrega Caprara [7], quien modela el problema y demuestra que es NP-hard, siendo una generalización del MSSP (*Maximum Stable Set Problem*).

Una solución al TTP en con una sola pista es propuesta por Cacchiani et al. [8], basado en método de generación de columnas. Aquí se propone un modelo de ILP (*Integer Linear Programming*) en donde cada variable, corresponde a la secuencia de programaciones horarias de un tren, denominadas *path variables* (variables de ruta). Esto genera una gran cantidad de combinaciones y posibles soluciones, por lo tanto es adecuado de resolver con técnicas generación de columnas. Adicionalmente un método basado en heurísticas y *local search* es propuesto e implementado. Ambas propuestas fueron probadas con instancias reales de *Rete Ferroviaria Italian* (RFI). Los resultados mostraron que pese a que el número de variables de la primera formulación crece exponencialmente, estos son resueltos en tiempos razonables. Por otro el modelo basado en heurísticas entrega mejores resultados, sin embargo los tiempos de computación son considerablemente mayores.

Una variante interesante propone Mohammad et al. [9] en la resolución del TTP con una sola vía. Se formula el modelo como MIP, con la capacidad de manejar pequeñas perturbaciones en las llegadas y salidas de trenes, lo cual ocurre regularmente en la realidad. Para eso introduce variables que manejar el tiempo de *buffer*, los cuales se modelan de dos formas: Con una distribución conocida de las perturbaciones y con distribución desconocida. Para la resolución se utiliza en *Branch and Bound* (BB), y se propone una heurística *Beam Search* (BS) para encontrar soluciones factibles en tiempos en tiempos razonables. Para verificar la validez del método basado en BB, se comparan con los resultados que entrega el software `Lingo`. Los resultados muestran que para todas las instancias pequeñas los resultados coinciden, sin embargo ninguno es capaz de resolve instancias muy grandes. Por otro lado el método basado en BS entrega buenos resultados para las instancias pequeñas (y en menor tiempo), y es capaz de determinar soluciones factibles para las instancias en que los otros dos no pueden.

Por otro lado Tormos et al. [10] también propone resolver el TTP en una sóla pista simple, pero mediante algoritmos genéticos (GA por *Genetic Algorithm*). La elección de este método es que el problema posee un espacio de búsqueda complejo, y precisamente los GA son capaces de llegar a buenas soluciones y en poco tiempo en tales casos. El problema se limita al estudio de pistas simples, con secciones dobles. Para cada nuevo tren se define su *traversal time*, que consiste en el menor tiempo en que puede atravesar la estación sin violar restricciones del sistema. Luego la función objetivo se define como la suma de tales tiempos. El algoritmo genético que se propone se basa en modelo utilizado en *Job-Shop Scheduling Problem*, en donde se genera una programación de trabajos, los cuales satisfacen restricciones de tiempo y recursos, con el menor esfuerzo posible y en el menor espacio de tiempo. Para el proceso se utilizan y definen operaciones de *crossover* (cruzamiento), *mutation* (mutación) y *selection*, usando heurísticas para definir la población inicial. La propuesta se prueba en instancias reales obtenidas del ADIF (Administrador de Infraestructura Ferroviaria) de España. Los resultados muestran que el método propuesto permite obtener resultados factibles en tiempos relativamente bajos (minutos), lo cual es

bueno considerando el gran tamaño de las instancias.

Un buen sumario acerca del TTP en sola vía lo provee Higgins et al. [11]. En este se explica la importancia de utilizar heurísticas para este tipo de problemas, en donde encontrar óptimos globales no es una posibilidad. Para ellos se prueba con *Local Search Heuristics* (heurísticas de búsqueda local, LSH), Algoritmos Genéticos (GA), Tabu Search (TS) y dos algoritmo híbridos (HA1 y HA2). Una consideración importante del modelo a utilizar, es que supone que las llegadas y salidas no son conocidas de un inicio (no son datos iniciales), y por lo tanto tienen la libertad de ser planificadas. Las algoritmos híbridos recién nombrados, tienen como propósito combinar las ventajas de dos o más heurísticas. En la primera variante (HA1) combina GA con LSH, de modo que en cada iteración de GA (antes de la operación de *crossover*) se realiza LSH sobre el 5 % mejor de la población, ayudando de este modo a que GA alcance convergencia más rápido. La segunda variante (HA2) combina GA con TS para mejorar el proceso de *crossover*, permitiendo elegir padres y posiciones de cruzamiento que no produzcan descendencia en la próxima generación. En la Figura 6 puede apreciarse una comparación de los resultados obtenidos para cada algoritmo, donde los valores que se muestran corresponden al tiempo de viaje (en estación) con pesos promediados sobre 20 ejecuciones en cada instancia (valores más bajos son mejores). En la tabla se aprecia que los algoritmos híbridos obtienen por lejos mejores resultados, siendo H2 el mejor entre ambos. La heurística con peor desempeño fue LSH, debido a su poca capacidad de escapar de óptimos locales. Por otro lado en la Figura

No. trains (No. conflicts)	Heuristics tested					
	LSH					
	(A1)	(A2)	GA	TS	HA1	HA2
15 (13–23)	0.115	0.075	0.010 (0.002)	0.035 (0.007)	0.003 (0.003)	0.003 (0.003)
20 (20–28)	0.050	0.047	0.005 (0.002)	0.035 (0.007)	0.001 (0.001)	0.000 (0.000)
25 (25–35)	0.190	0.180	0.002 (0.000)	0.112 (0.015)	0.001 (0.000)	0.000 (0.000)
30 (49–52)	0.130	0.205	0.120 (0.034)	0.080 (0.009)	0.092 (0.012)	0.025 (0.008)
35 (55–67)	0.150	0.135	0.010 (0.012)	0.080 (0.015)	0.002 (0.001)	0.001 (0.000)
40 (79–81)	0.135	0.120	0.025 (0.014)	0.110 (0.006)	0.003 (0.001)	0.003 (0.001)
45 (91–95)	0.280	0.145	0.040 (0.015)	0.115 (0.012)	0.001 (0.000)	0.000 (0.000)
50 (103–113)	0.060	0.055	0.020 (0.010)	0.010 (0.005)	0.003 (0.001)	0.002 (0.001)

Figura 6: Comparación de resultados de Heurísticas (Fuente [11])

7 se hace una comparación de la capacidad de mejora de la solución, versus los tiempos de procesamiento para cada heurística (se agrega *Branch and Bound* que fué el método para computar la solución óptima). Se puede apreciar que GA, HA1 y HA2 no son capaces de determinar soluciones antes de los 35 segundos, esto pues primero deben computar la población inicial. Sin embargo luego de computada esta, la razón de convergencia de HA1 y HA2 es superior a la del resto. Por otro lado, se nota que TS y LSH convergen rápido en un inicio, pero luego se estancan. Se termina concluyendo que los algoritmos híbridos son los más viables, y con el mejor desempeño general.

Otra variante del problema, la constituyen el *Train Rescheduling Problem* en el cual se deben resolver conflictos en tiempo real. Esta es una situación común, en la cual debido a imprevistos (accidentes, retrasos, etc) la planificación prevista debe ser modificada, para poder satisfacer las llegadas/salidas con la menor variación posible. La dificultad se encuentra en realizar una re-planificación aceptable, en el menor tiempo posible. En el trabajo realizado por Norio et al. [12], se propone un modelo de *rescheduling* que intenta minimizar la insatisfacción de la gente. El algoritmo propuesto combina *Program Evaluation and Preview Technique* con el método de *Simulated Annealing*. De modo similar D'Ariano et al. [13] afrontan el mismo problema, pero utilizando el método de *Branch and*

Bound (BB). Debido a las restricciones de tiempo, el proceso de BB es acelerado ocupando *reglas de implicación estáticas*, que explotan las propiedades de las soluciones factibles. Las pruebas muestran que el método propuesto es capaz de computar soluciones óptimas (o cercanas) en tiempos pequeños.

Un problema con mayor parecido al que se trata aquí, es el *Multi Objective Train Scheduling Problem*. El problema tratado por Ghoseiri et al [14] se asemeja bastante al de este estudio, en cuanto a las estructura que considera: Una red de pistas compuestas por pistas simples, múltiples pistas, múltiples plataformas (con capacidad de movilidad entre ellas), y múltiples plataformas con distinta capacidad. Sin embargo los objetivos que se plantea son: 1) Minimizar la utilización de combustible (medida de eficiencia que aumenta satisfacción de compañía) y 2) minimizar el tiempo de los pasajeros (medida de eficacia que aumenta la satisfacción de pasajeros). Una formulación diferente la plantea Li et al. [15], considerando en sus objetivos: 1) Minimizar la cantidad de energía utilizada, 2) Minimizar los costos de emisiones de carbón, y 3) Minimizar el tiempo de los pasajeros. Debido a sus objetivos, sus autores llaman al modelo *green train scheduling problem*.

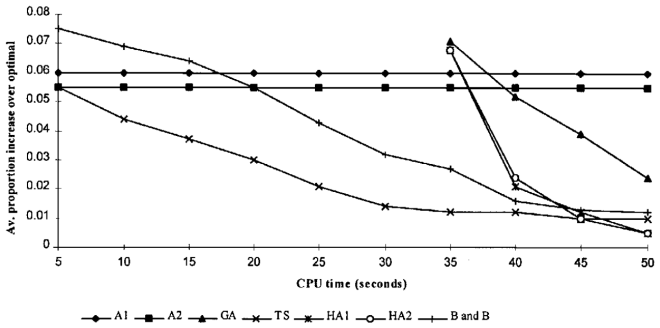

Figura 7: Comparación de convergencia versus tiempo de ejecución de Heurísticas (Fuente [11])

4. Modelo Matemático

A fin de mantener la definición del problema simple en la Sección 2, muchas notación y aspectos técnicos fueron omitidos. En lo que sigue se pretende dar una descripción formal más acabada al respecto. La formulación propuesta sigue la descripción del problema en [1], con pequeñas modificaciones equivalente para facilitar la comprensión. Por lo tanto se ocupa la misma notación para mantener la coherencia. Para no abrumar al lector, la notación se explica de modo progresivo (en la medida que se requiera).

4.1. Restricciones

Propiedades de las Programaciones.

1. **Entradas en el sistema.**

$$\begin{aligned}
\mathcal{A} &:= \text{Conjunto de llegadas.} \\
\mathcal{T}_I &:= \text{Conjunto de trenes inicialmente en el sistema.} \\
\mathcal{T}_A &:= \text{Conjunto de trenes que ingresan por una llegada } a \in \mathcal{A} \\
\mathcal{T} &:= \text{Conjunto de todos los tres: } \mathcal{T}_I \cup \mathcal{T}_A \\
\mathcal{T}^+ &:= \text{Conjunto de trenes usados en al solución.} \\
\text{arrSeq}_a &:= \text{Secuencia de llegada para } a \in \mathcal{A}. \\
\text{arrTime}_a &:= \text{Tiempo de llegada a una plataforma para } a \in \mathcal{A}. \\
\text{res}_t &:= \text{Recurso ocupado por un tren } t \text{ inicialmente en el sistema}
\end{aligned}$$

   Para todos los trenes $t \in \mathcal{T}_I$, los eventos `EnterSystem` y `EnterResource`, deben ocurrir en el instante inicial del horizonte $\text{d}_1\ 00:00:00$, en el recurso inicial res_t. Para los trenes asociados a llegadas $t \in \mathcal{T}_A$, el evento `EnterSystem` de ser procedido inmediatamente (mismo instante) por un `EnterResource` en un conjunto de pistas, seguido por un `ExitResource` de acuerdo a su secuencia de llegada arrSeq_a. El evento `EnterResource` en la plataforma debe ocurrir al instante arrTime_a.

2. **Salidas del sistema.**

$$\begin{aligned}
\mathcal{D} &:= \text{Conjunto de salidas.} \\
\text{depSeq}_d &:= \text{Secuencia de salida de } d \in \mathcal{D}. \\
\text{depTime}_d &:= \text{Hora de salida de la plataforma de } d \in \mathcal{D}. \\
\text{d}_{\text{nbDays}} &:= \text{Número de días considerados en el horizonte.}
\end{aligned}$$

   La programación para todo $t \in \mathcal{T}^+$ debe terminar en un evento `ExitSystem`. Si el tren t no se asigna a una salida, tal evento debe ocurrir al instante final del horizonte $\text{d}_{\text{nbDays}}\ 23:59:59$ (precedido por el `ExitResource` respectivo). Un tren asociado a una salida $d \in \mathcal{D}$ debe realizar un `ExitResource` al instante depTime_d, seguido por una serie de `EnterResource` y `ExitResource` en los conjuntos de pistas impuestos por la secuencia de salida depSeq_d.

3. **Uso de un recurso.**

$$\begin{aligned}
\mathcal{E} &:= \text{Conjunto de posibles eventos.} \\
r_e &:= \text{Recurso utilizado por el evento } e \in \mathcal{E}. \\
\text{sched}_t &:= \text{Secuencia de eventos asociadas al tren } t \in \mathcal{T}^+.
\end{aligned}$$

   Cada evento `EnterResource` e en sched_t, debe ser seguido por un evento e' del tipo `ExitResource`, tal que $r_e = r_{e'}$ (asociados al mismo recurso). Entre e y e' sólo puede ocurrir eventos de tipo `Arrival`, `Departure` y `Beg/EndMaintenance`, con recurso asociado r_e.

4. **Transición a un recurso vecino.**

$$\begin{aligned}
g_e &:= \text{Puerta de entrada/salida al recurso } r_e \text{ para el evento } e \in \mathcal{E}. \\
\text{neigh}_g &:= \text{Puerta vecina de g (da paso a otro recurso).} \\
h_e &:= \text{Instante en el horizonte } \mathcal{H} \text{ donde ocurre el evento.}
\end{aligned}$$

   Cada evento `ExitResource` e, debe ser *seguido inmediatamente* por un evento e' `ExitSystem` (si se encontraba en el recurso al borde del sistema), o un `Enter Resource` con $r_e \neq r_{e'}$ (recurso diferente). También debe cumplirse que $\text{neigh}_{g_e} = g_{e'}$. Seguido inmediatamente significa $h_e = h_{e'}$.

5. **Operaciones de mantención.**

$$\begin{aligned}\mathcal{F} &:= \text{Conjunto de instalaciones de mantenimiento.}\\ \text{maintTimeD}_t &:= \text{tiempo que demora el tren } t \text{ en realizar mantención tipo D.}\\ \text{maintTimeT}_t &:= \text{tiempo que demora el tren } t \text{ en realizar mantención tipo T.}\\ c_e &:= \text{Cuando } e \text{ involucra un recurso en } \mathcal{F}, \text{ esta indica el tipo de mantención.}\end{aligned}$$

   Si el sched_t de un tren $t \in \mathcal{T}^+$ contiene un evento `BegMaintenance`, entonces debe ser seguidos por eventos `EnterResource` y `ExitRosource` en alguna $f \in \mathcal{F}$, tal que $\text{type}_f = c_e$, y debe finalizar con el evento `EndMaintenance` e' cumpliendo:

$$\begin{aligned}c_e = D &\Rightarrow h_{e'} = h_e + \text{maintTimeD}_t\\ c_e = T &\Rightarrow h_{e'} = h_e + \text{maintTimeT}_t.\end{aligned}$$

   Notar que cada tren puede realizar a lo más una mantención de cada tipo (puede no realizar ninguna).

6. **Uso de un grupo de pistas.**

$$\begin{aligned}\mathcal{K} &:= \text{ Conjunto de grupos de pistas del sistema.}\\ \text{trTime}_k &:= \text{ Tiempo que toma recorrer } k \in \mathcal{K}.\\ \text{side}_g &:= \text{Lado (A o B) asociado a la puerta g.}\end{aligned}$$

   Un tren $t \in \mathcal{T}^+$ que realiza un evento e `EnterResource` en un grupo de pistas k, debe ser seguido por un evento e' `ExitResource`, cumpliendo:

$$\begin{aligned}h_{e'} &= h_e + \text{trTime}_k\\ \text{side}_{g_{e'}} &\neq \text{side}g_e.\end{aligned}$$

7. **Mínimo tiempo de reversa.** Un tren $t \in \mathcal{T}^+$ que realiza un evento `EnterResource` sobre un recurso $r \in \mathcal{S}, \mathcal{P}, \mathcal{F}$ o $\mathcal{Y}$, debe ser seguido por un evento e' `ExitResource` satisfaciendo:

$$\text{side}_{g_{e'}} = \text{side}_{g_e} \Rightarrow h_{e'} \geq h_e + \text{revTime},$$

   dónde revTime es el tiempo que le toma a la tripulación, cambiar el sentido del viaje (caminar hacia el otro extremo del tren).

Asignaciones.

1. **Restricciones de trenes.** Para cada salida $d \in \mathcal{D}$, esta debe ser asociada con a lo más un tren $t \in \mathcal{T}$.
2. **Restricciones de DBM.**

$$\begin{aligned}\text{depTrain}_d &:= \text{ Tren asociado a las salida } d.\\ \text{reqDBM}_d &:= \text{ DBM requerido para poder efectuar la salida}d.\\ \text{remDBM}_t &:= \text{ DBM restante para el tren t.}\\ \text{maxDBM}_t &:= \text{ DBM que alcanza un tren } t \text{ luego de operación de tipo D.}\\ \text{linkedDep}_a &:= \text{ La llegada } a \text{ esta asociada a la salida previa } d.\end{aligned}$$

   La restricción para que una salida sea factible es:

$$\text{Si } \ \text{depTrain}_d = t \Rightarrow \text{remDBM}_t \geq \text{reqDBM}_d$$

En caso de que una llegada $a \in \mathcal{A}$, este enlazada $\text{linkedDep}_a = d$ con $\text{depTrain}_d = t$ y $\text{arrTrain}_a = t'$, entonces para actualizar el DBM y TBM se computa:

$$\text{remDBM}_{t'} = \text{remDBM}_t - \text{reqDBM}_d$$
$$\text{remTBM}_{t'} = \text{remTBM}_t - \text{reqTBM}_d$$

3. **Restricciones de TBM.**

$$\begin{aligned} \text{reqTBM}_d &:= \text{ TBM requerido para poder efectuar la salida}d. \\ \text{remTBM}_t &:= \text{ TBM restante para el tren t.} \\ \text{maxTBM}_t &:= \text{ DBM que alcanza un tren } t \text{ luego de operación de tipo D.} \end{aligned}$$

Análogamente, para que una salida d sea factible debe cumplir:

$$\text{Si } \ \text{depTrain}_d = t \Rightarrow \text{remTBM}_t \geq \text{reqTBM}_d.$$

4. **Compatibilidad de categoría.**

$$\begin{aligned} \text{cat}_t &:= \text{ Categoría a la que pertenece el tren t.} \\ \text{CompCatDep}_d &:= \text{ Categoría de trenes compatibles para realizar la salida } d. \end{aligned}$$

La restricción es $\forall d \in \mathcal{D}, \forall t \in \mathcal{T}^+, \text{ si } \text{depTrain}_d = t \Rightarrow \text{cat}_t \in \text{compCatDep}_d$.

Utilización de Recursos.

$$\begin{aligned} \text{use}_{t,h} &:= \text{Indica el recurso } r \text{ del que hace uso el tren } t \text{ en el instante } h. \\ \text{nbTrain}_{r,h} &:= \text{Para } r \in \mathcal{S} \cup \mathcal{Y}, \text{ indica la cantidad de trenes haciendo uso de él.} \\ \mathcal{I} &:= \text{Conjunto de utilizaciones impuestas con anticipación.} \\ \text{res}_i &:= \text{Recurso (estacionamiento) utilizado por } i \in \mathcal{I}. \\ \text{nb}_i &:= \text{Número de slots/trenes utilizados por } i \in \mathcal{I}. \\ (\text{beg}_i, \text{end}_i) &:= \text{Instantes de inicio y término de } i \in \mathcal{I}. \end{aligned}$$

Notar que para las pistas simples se cumple:

$$\forall h \in \mathcal{H}, \forall r \in \mathcal{S}, \ \text{nbTrain}_{r,h} = \left| t \in \mathcal{T}^+ : \text{use}_{t,h} = r \right|,$$

mientras que las pistas de estacionamiento pueden tener utilizaciones impuestas:

$$\forall h \in \mathcal{H}, \forall r \in \mathcal{Y}, \ \text{nbTrain}_{r,h} = \left| t \in \mathcal{T}^+ : \text{use}_{t,h} = r \right| + \sum_{\substack{i \in \mathcal{I} \\ \text{res}_i = r \\ h \in [\text{beg}_i, \text{end}_i]}} \text{nb}_i$$

1. **Compatibilidad de categoría.** Un tren puede t sólo puede hacer uso de un recurso r, si tal recurso es compatible con el tren:

$$\forall t \in \mathcal{T}^+, \forall r \in \mathcal{R}, \forall h \in \mathcal{H}, \ \text{use}_{t,h} = r \Leftrightarrow \text{cat}_t \in \text{compCatRes}_r.$$

2. **Capacidad de pistas simples.**

$$\text{capa}_s := \text{ Capacidad en trenes de pista simple } s \in \mathcal{S}.$$

La capacidad de las pistas simples debe ser respetada a cada instante:

$$\forall s \in \mathcal{S}, \forall h \in \mathcal{H}, \ \text{nbTrain}_{s,h} \leq \text{capa}_s.$$

3. **Uso máximo de plataforma.**

   $\text{maxDwellTime}_a :=$ Máximo tiempo que el tren asociado a a puede permanecer en plataforma.
   $\text{idealDwellTime}_a :=$ Tiempo ideal que el tren asociado a a puede permanecer en plataforma.
   $\text{maxDwellTime}_d :=$ Máximo tiempo que el tren asociado a d puede permanecer en plataforma.
   $\text{idealDwellTime}_a :=$ Tiempo ideal que el tren asociado a a puede permanecer en plataforma.

   En caso de que un tren t entre a una plataforma, pero no realice los eventos `Arrival` o `Departure` (va de paso), entonces puede permanecer un tiempo máximo maxDwellTime. Trenes que llegan ($a \in \mathcal{A}$) o salen ($d \in \mathcal{D}$) no puede permanecer un tiempo mayor a su máximo, respectivamente. Si un tren realiza una llegada, seguida inmediatamente por una salida, el tiempo máximo de permanencia es máx($\text{maxDwellTime}_a, \text{maxDwellTime}_d$).

4. **Mínima utilización de un recurso.** Para todo tren t que utiliza un recurso r (excepto los grupos de pistas), el tiempo entre los eventos `EnterResource` y `ExitResource` debe ser al menos minResTime.

5. **Uso impuesto de recursos.** No se pueden realizar a asignaciones a recursos con utilizaciones impuestas durante los tiempos respectivos:

$$\forall i \in \mathcal{I}, res_i \in \mathcal{S} \cup \mathcal{P} \cup \mathcal{F} \Rightarrow \forall h \in [\text{beg}_i, \text{end}_i], \forall t \in \mathcal{T}^+, \text{use}_{t,h} \neq \text{res}_i.$$

6. **Largo de una pista.**

   $\text{lenght}_t :=$ Largo del tren t.
   $\text{lenght}_r :=$ Capacidad (en largo) del recurso r.

   Las reestricciones de capacidad (en largo) de los recursos de cumplirse:

$$\forall r \in \mathcal{S}, \mathcal{F}, \mathcal{P}, \forall h \in \mathcal{H}, \sum_{\substack{t \in \mathcal{T}^+ \\ \text{use}_{t,h}=r}} \text{lenght}_t \leq \text{lenght}_r.$$

7. **Capacidad de instalaciones de mantenimiento.**

   $\text{maintD}_{t,i} :=$ Igual a 1 si el tren t hace mantención tipo D el díai.
   $\text{maintT}_{t,i} :=$ Igual a 1 si el tren t hace mantención tipo D el díai.
   $\text{maxMaint} :=$ Máxima cantidad de mantenciones totales por día.

   El número total de ambas mantenciones (tipo D y T) no puede superar la capacidad diaria:

$$\forall i \in [1, \text{nbDays}], \sum_{t \in \mathcal{T}^+} \text{maintD}_{t,i} + \text{maintT}_{t,i} \leq \text{maxMaint}.$$

8. **Orden de trenes en pistas individuales.**

   $\text{beg}_i :=$ Instante en que el tren t_i ingresa al recurso r.
   $\text{end}_i :=$ Instante en que el tren t_i sale del recurso r.
   $\text{from}_i :=$ De que lado (A o B) el tren t_i entra al recurso r.
   $\text{to}_i :=$ De que lado (A o B) el tren t_i sale del recurso r.

   Dados dos trenes $t_1, t_2 \in \mathcal{T}^+$, para todo recurso $r \in \mathcal{S}, \mathcal{P}, \mathcal{F}$ las restricciones en la Figura 8 aplican.

$from_1$	to_1	$from_2$	to_2	Constraint
A	A	A	A	$end_2 < end_1$ or $beg_2 > end_1$
A	A	A	B	$beg_2 > end_1$
A	A	B	A	$end_2 > end_1$
A	A	B	B	$\varnothing$ (no constraint)
A	B	A	A	$\varnothing$ (no constraint)
A	B	A	B	$end_2 > end_1$
A	B	B	A	$beg_2 > end_1$
A	B	B	B	$beg_2 > end_1$ or $end_2 < end_1$
B	A	A	A	$beg_2 > end_1$ or $end_2 < end_1$
B	A	A	B	$beg_2 > end_1$
B	A	B	A	$end_2 > end_1$
B	A	B	B	$\varnothing$ (no constraint)
B	B	A	A	$\varnothing$ (no constraint)
B	B	A	B	$end_2 > end_1$
B	B	B	A	$beg_2 > end_1$
B	B	B	B	$end_2 < end_1$ or $beg_2 > end_1$

Figura 8: Restricciones de orden en pistas simples (Fuente [1])

9. **Conflictos en grupos de pistas.**

$$\begin{aligned} \mathcal{M} &:= \text{Conjunto de movimientos posibles en grupos de pistas.} \\ \text{hwTime}_k &:= \text{ Tiempo de seguridad mínimo entre trenes en } k \in \mathcal{K}. \\ \text{ind}_g &:= \text{ Indice de puerta para un lado de } k \in \mathcal{K}. \end{aligned}$$

Existen dos posibles casos de conflictos. Primero, cuando dos trenes van en la misma dirección:

$$\begin{aligned} \forall (m_1, m_2, k) \in \mathcal{M} \times \mathcal{K}, \text{ con side}_{o_1} = \text{side}_{o_2}, \\ (\text{ind}_{o_1} - \text{ind}_{o_2}) \times (\text{ind}_{d_1} - \text{ind}_{d_2}) \leq 0 \Rightarrow |h_1 - h_2| \geq \text{hwTime}_k, \end{aligned}$$

y cuando los trenes van en direcciones opuestas:

$$\begin{aligned} \forall (m_1, m_2, k) \in \mathcal{M} \times \mathcal{K}, \text{ con side}_{o_1} \neq \text{side}_{o_2}, \\ (\text{ind}_{o_1} - \text{ind}_{d_2}) \times (\text{ind}_{d_1} - \text{ind}_{o_2}) \leq 0 \Rightarrow |h_1 - h_2| \geq \text{trTime}_k \text{hwTime}_k. \end{aligned}$$

10. **Capacidad de estacionamientos.** La capacidad de las vías de estacionamiento debe ser respetada:

$$\forall y \in \mathcal{Y}, \forall h \in \mathcal{H}, \text{nbTrain}_{y,h} \leq \text{capa}_y.$$

4.2. Función Objetivo

La función objetivo corresponde a múltiples objetivos, asociados a penalizaciones por violación de situaciones insatisfactorias, cuyo objetivo es minimizar. Esta se modela como una suma con pesos del siguiente modo:

$$f = w_1 f^{\text{uncov}} + w_2 f^{\text{maint}} + w_3 f^{\text{plat}} + w_4 f^{\text{pref}} + w_5 f^{\text{reuse}} \tag{2}$$

donde cada término de (2) se detalla a continuación. Los pesos w_i de la suma se definen en razón de la importancia que se le quiere dar a cada término. Por lo general peso dominante es w_1, pues se quiere evitar en gran medida que llegadas/salidas queden sin cubrir.

1. **Llegadas/Salidas sin cubrir.** También incluye el costo por no usar trenes inicialmente en el sistema. Si para cada una de estas situaciones (LLegadas o salidas sin cubrir, o trenes iniciales no utilizados) se define un costo unitario uncovCost, entonces el costo puede expresarse como:

$$f^{\text{uncov}} = \text{uncovCost} \times \left(\left| \{ t \in \mathcal{T} \setminus \mathcal{T}^{+} \} \right| + |\{ d \in \mathcal{D} : \text{depTrain}_d = \emptyset \}| \right)$$

2. **Costos por sobre mantención.** Este costo va asociado a la realización de mantenciones (Tipo D y T), sobre trenes que aún tiene capacidad para seguir viajando remTBM > 0 y remDBM > 0. Para ellos se definen los costos unitarios remDCost y remTCost, por cada unidad de TBM y DBM que no se utiliza al realizar la mantención respectiva. Luego el costo total queda:

$$f^{\text{maint}} = \sum_{\substack{e \in \mathcal{E} \\ c_e = \text{D} \\ y_e = \texttt{BegMaintenance}}} \text{remDCost} \times \text{remDBM}_{t_e} + \sum_{\substack{e \in \mathcal{E} \\ c_e = \text{T} \\ y_e = \texttt{BegMaintenance}}} \text{remTCost} \times \text{remTBM}_{t_e}.$$

3. **Costo por uso de plataformas.** Como se mencionó anteriormente, aquí hay tres posibilidades. Que una llegada ocupe la plataforma (llamamos a este conjunto $\mathcal{A}' \subset \mathcal{A}$), que una salida ocupe la plataforma $\mathcal{D}' \subset \mathcal{D}$ o que una llegada seguida inmediatamente por una salida ocupe esta plataforma (llamaremos a este conjunto $\mathcal{Z} \subset \mathcal{A} \times \mathcal{D}$). Adicionalmente se define las variables dwell_a, dwell_d y dwell_z, el tiempo de la plataforma que se asigna a cada una de estas posibilidades. Naturalmente se define $\text{idealDwell}_z = \text{idealDwell}_a + \text{idealDwell}_d$ con $z = (a, d)$. Luego, una penalización dwellCost es aplicada por cada unidad de tiempo que se realice sobre el tiempo ideal, en cada caso. Esto se expresa para cada caso como sigue:

$$\begin{aligned} f^{\text{plat}_1} &= \sum_{a \in \mathcal{A}^*} \text{dwellCost} \times |\text{dwell}_a - \text{idealDwell}_a| \\ f^{\text{plat}_2} &= \sum_{d \in \mathcal{D}^*} \text{dwellCost} \times |\text{dwell}_d - \text{idealDwell}_d| \\ f^{\text{plat}_3} &= \sum_{z \in \mathcal{Z}} \text{dwellCost} \times |\text{dwell}_z - \text{idealDwell}_z|, \end{aligned}$$

   siendo la función de costo general: $f^{\text{plat}} = f^{\text{plat}_1} + f^{\text{plat}_2} + f^{\text{plat}_3}$.

4. **Costo por no satisfacer preferencia de plataformas.** Si tanto para las llegadas $a \in \mathcal{A}$, como las salidas $d \in \mathcal{D}$ se aplica un costo por no asignar una de las plataformas preferidas: platAsgCost. Entonces el costo general queda:

$$f^{\text{pref}} = \sum_{\substack{a \in \mathcal{A} \\ \text{plat}_a \notin \text{prefPlat}_a}} \text{platAsgCost} + \sum_{\substack{d \in \mathcal{D} \\ \text{plat}_d \notin \text{prefPlat}_d}} \text{platAsgCost}$$

5. **Costo por no satisfacer reusos.** Sea reuseCost el costo asociado a no satisfacer cualquier reuso $u \in \mathcal{U}$, luego el costo general es:

$$f^{\text{reuse}} = \sum_{\substack{u \in \mathcal{U} \\ \text{depTrain}_{\text{dep}_u} \neq \text{arrTrain}_{\text{arr}_u}}} \text{reuseCost}.$$

5. Representación

Tal como lo propuso Haarh et al. [3], el problema se abordó subdividiendo en tres subproblemas más simples de resolver. El primero de ellos corresponde al *Matching* entre las salidas y los trenes en el sistema (trenes asociados a llegadas o inicialmente en el sistema). Una vez conocidos que trenes cubriran qué salidas, es posible abordar el problema de *Platform Assignment*, el que consiste en asignar una plataforma a todas las llegadas y salidas por cubrir, esto es, si en la fase anterior se determinó no cubrir una salida entonces no debe ser considerada para una plataforma. Con las secuencias de entrada y salida *fijas* y las asignaciones anteriores cubiertas, queda por resolver el problema de *Routing* de cada tren dentro de la estación, es decir, la ruta de recursos por los que pasa y utiliza entre su llegada y su próxima salida, más las secuencias de puertas de los *Track Groups* que utiliza en sus secuencias de entrada y salida. A continuación se detalla la representación utilizada para cada caso, más la representación utilizada para manejar todos los datos de cada instancia.

Instancias. Para manejar cada tabla correspondiente a una instancia, se decidió utilizar diccionarios del siguiente tipo:

$$\texttt{TableName[keyId]} = \texttt{valueStruct}$$

donde `keyId` es el ID primario de la tabla, y `valueStruct` es una estructura para contener el resto de campos de cada fila. Para la implementación de cada diccionario se utiliza *ordered map* [16], que internamente mantiene los datos en una estructura de árbol que preserva el orden de los elementos, lo cual es útil pues se requiere iterar en orden sobre estas estructuras. Se elige esta representación por el fácil manejo de datos que permite, muy similar al acceso en tablas.

Matching. Un matching consiste básicamente en un par *Departure-Train*, con el tren asociado a cada uno de los *departures*. Por lo mismo, se propone utilizar también una estructura de diccionario del siguiente tipo:

$$\texttt{Matches[departureId]} = \texttt{trainId}$$

donde `departureId` y `trainId` son los IDs de la salida y del tren respectivamente. Debido a que sobre estas estructuras se requiere realizar gran cantidad de consultas y modificaciones (operaciones *insert*), se utiliza un `unordered map` [17], que internamente accede a los datos por una *tabla hash* cargada en memoria, lo cual produce tiempos de inserción y búsqueda constantes, siendo por tanto una solución eficiente.

Platform Assignment. Una asignación de plataforma consiste en un par *Arrival-Platform* o *Departure-Platform*. Por lo tanto, al igual que en el caso anterior se utiliza un diccionario del siguiente tipo:

$$\texttt{Assignment[arrivalId || departureId]} = \texttt{platformId}$$

donde la llave del diccionario pueden ser `arrivalId` (ID de llegada) o `departureId` (ID de salida), y `platformId` el ID de la plataforma asignada. Para la implementació se ocupa un *unsorted map* al igual que en el anterior.

Routing. Cada pasada que realiza un tren por un recurso de la estación se caracteriza por lo siguiente:

$$\texttt{passStruct} = [\texttt{indexGate}_a, \texttt{ResourceId}, \texttt{indexGate}_b, \texttt{direction}, \texttt{entryTime}, \texttt{exitTime}]$$

donde $\texttt{indexGate}_a$ e $\texttt{indexGate}_b$ son los indices de las puertas de los lados A y B respectivamente, `ResourceId` es el ID del recurso utilizado, `direction` es un `flag` para indicar en

qué dirección se cruza el recurso: `A` $\rightarrow$ `B` ó `A` $\leftarrow$ `B` y `entryTime` y `exitTime` son los tiempos de entrada y salida del recurso respectivamente.

Luego para representar la ruta de cada tren se usa una **lista dinámica** de tales estructuras, pues el largo y tipo de recursos de cada ruta puede ir cambiando en la solución. Notar que esta representación es fácilmente transformable a la representación de solución propuesta en el problema original [1].

6. Descripción del Algoritmo

Siguiendo las mismas ideas anteriores, se procede a explicar el algoritmo utilizado para resolver cada uno de los subproblemas. Para la resolución de tales subproblemas (dada su conocida y estudiada complejidad), se propone ocupar el método *heurístico* Tabu Search, con las adaptaciones para los movimientos que generan la vecindad correspondientes.

Matching. La idea es mantener (en la medida de lo posible) las soluciones dentro de la zona de factibilidad, por ello dos *departures* distintos no pueden tener asociado el mismo tren en el diccionario `Matches[departureId]` = `trainId`. Otra situación infactible se da cuando el tren asociado llega a la estación después de tal salida, sin embargo cuando existan tales asignaciones se considerará esa salida como no cubierta, y se penalizará con un costo `uncovCost`.

Para generar la vecindad, se itera sobre cada uno de los `departureId` en `Matches` siguiendo el siguiente procedimiento:

1. Se genera un `trainId` aleatorio entre los IDs de todos los trenes de la estación (trenes iniciales y de llegadas).
2. Si tal `trainId` ya está asignado a otro *departure* se aplica un *swap* entre las asignaciones.
3. De otro modo se aplica un *insert*, asignando el `trainId` generado al *departure*.

Tales movimientos permiten mantener soluciones factibles, y al mismo tiempo ir incorporando distintas combinaciones. Entre los vecinos generados, se evalúan los costos por medio de las asignaciones insatisfechas descritas recién (`uncovCost`) y los reusos no satisfechos (`reuseCost`). El vecino con menor costo es seleccionado como la solución actual, y se agregan las asignaciones anteriores a un diccionario `Tabu[departureIdPrev]` = `trainIdPrev`. En Algorithm 1 se resume el movimiento generador de la vecindad.

Platform Assignment. Para las asignaciones de plataforma hay 3 principales restricciones: (1) Preferencias de plataforma (blanda), (2) Capacidad de plataformas (dura), (3) Recursos usados con anterioridad (dura). Adicionalmente cada llegada y salida no puede tener asignada más de una plataforma, pero dada la representación de la solución esto no se puede dar. La restricción (1) se agrega como costo de penalización (`platAsgCost`), del mismo modo que (2) a pesar de ser restricción dura, pues no es trivial producir movimientos que la mantengan satisfecha. Sin embargo por cada plataforma que viole su capacidad, se aplica un costo de $N \cdot$ `platAsgCost`, con N definido por el usuario. Si un movimiento genera una solución que viole (3), entonces se descarta.

El movimiento en cuestión es una simplificación del anterior. Sobre cada uno de los `arrivalId` o `departureId` en `Assignment`:

1. Se genera un `plataformId` aleatorio entre los IDs de todas las plataformas.
2. Se aplica un *insert*, asociando tal plataforma al *arrival* o *departure* en cuestión.

Algorithm 1 Generador de vecindad para *Matching problem*

```
1: procedure NEIGHBORHOOD(Matches)
2:     Matches = Matches.copy()
3:     bestNeigh = Matches.copy()
4:     for depId,trId in Matches.items() do
5:         ntrId = generateRandomTrainId()
6:         if ntrId in Matches.values() then
7:             ndepId = Matches.findkey(ntr)
8:             Matches[depId] = ntrId
9:             Matches[ndepId] = trId
10:        else
11:            Matches[depId] = ntrId
12:        if cost(Matches) < cost(bestNeigh) then
13:            bestNeigh = Matches
14:        undoModification(Matches)
```

En algorithm 2 se resume el movimiento generador de la vecindad.

Observación: Para determinar que dos eventos (*arrival* o *departure*) estan haciendo uso de la misma plataforma al mismo tiempo, se verifica si hay intersección de los intervalos [`arrivalTime`, `arrivalTime` + `maxDwellTime`] o [`departureTime`, `departureTime` + `maxDwellTime`] respectivamente. Considerar tales asignaciones con el máximo tiempo que se puede estar en plataforma da más holgura a la fase de routing.

Algorithm 2 Generador de vecindad para *Platform Assignment Problem*

```
1: procedure NEIGHBORHOOD(Assignment)
2:     Assignment = Assignment.copy()
3:     bestNeigh = Assignment.copy()
4:     for eventId,platId in Assignment.items() do
5:         nplatId = generateRandomPlatformId()
6:         Assignment[eventId] = nplatId
7:         if cost(Assignment) < cost(bestNeigh) then
8:             bestNeigh = Assignment
9:         undoModification(Assignment)
```

Routing. Una vez conocidas las asignaciones anteriores, basta por conocer el itinerario o ruta que sigue cada tren dentro de la estación. Como se dijo anteriormente, la ruta está compuesta por la secuencia de entrada (fija), la *secuencia intermedia* (no determinada) y la secuencia de salida (fija). Se le llama secuencia intermedia a la ruta que sigue el tren entre su llegada y su próxima salida. De las secuencias de llegada/salida lo único que se puede decidir, son la combinaciones de *gates* utilizadas para atravesar los trackgroups, y los tiempos respectivos. En cambio para las secuencias intermedia existe mayor flexibilidad.

El movimiento propuesto para generar una vecindad de soluciones, consiste en seleccionar aleatoriamente (para un tren particular) que secuencia modificar (entrada, intermedia o salida). Luego se sigue del siguiente modo:

1. (*Secuencia de entrada.*) Para cada *Track Group* en la secuencia de entrada, genera aleatoriamente una combinación de puertas ($\texttt{indexGate}_a, \texttt{indexGate}_b$).
2. (*Secuencia Intermedia.*) Dada un secuencia intermedia de eventos `seqInter` de largo M, en primer lugar se determinan una subsección (aleatoria) de tal secuencia

`seqInter`$[i : j]$, entre los eventos $1 \leq i$ y $j \leq M$. Sean `Resource`$[i]$ y `Resource`$[j]$ los recursos al extremo de tal subsección, se elige (aleatoriamente) una subsección distinta con los mismo recursos extremos, que calce con los tiempos `entryTime` y `exitTime` respectivos.

3. (*Secuencia de Salida.*) Del mismo modo que en la entrada, para cada *Track Group* en la secuencia de salida, genera aleatoriamente una combinación de puertas (`indexGate`$_a$, `indexGate`$_b$).

Observaciones: (1) Para los movimientos de secuencias de entrada/salida no se modifican los tiempos, dado que el tiempo para atravesar cada *track group* es fijo e igual a `trTime`. (2) Los movimientos de secuencias intermedias suponen que se tienen *precomputadas* todas las posibles rutas entre dos recursos de la estación, lo que en la práctica puede ser un número muy elevado. En tal caso, siempre es posible trabajar con un subconjunto *más razonable* de tales rutas.

7. Experimentos

En lo que sigue se refiere sólo a los primeros dos subproblemas, dado que la implementación del algoritmo propuesto para resolver el problema de *Routing* mostró no ser efectivo, debido a la gran cantidad de combinaciones de subsecciones intermedias posibles entre dos recursos dados, y que la mayoría de estos se encuentran en la zona de no factibilidad.

Para generar las pruebas 10 instancias fueron ocupadas [1], ordenadas por orden de dificultad y correspondientes a versiones simplificadas de las utilizadas en el concurso *ROADEF/EURO Challenge 2014*. En particular, sólo se considera un día de horizonte de planificación, todos los recursos y salidas son compatibles con todos los trenes, el `remDBM` $>$ `reqDBM` (lo mismo para el tiempo) y no existen llegadas ni salidas conjuntas.

A continuación se detallan configuraciones propias para cada subproblema tratado.

Matching. Se tienen aquí dos principales parámetros: `numIterTabu` (número de iteraciones de Tabu Search) y `tabuLength` (largo máximo de la lista Tabú). Para obtener los mejores resultados se realiza un sintonización *manual* de parámetros. Debido a la naturaleza aleatoria de los movimientos que genera la vecindad, una misma configuración de parámetros no siempre entrega el mismo resultado, por lo que en algunos casos fue necesario repetir las pruebas varias veces con la misma configuración.

Platform Assignment. Los parámetros principales son los mismos: `numIterTabu` (número de iteraciones de Tabu Search) y `tabuLength` (largo máximo de la lista Tabú), más el peso asociado para castigar las violaciones de capacidad en las plataformas, determinadas por N : $N \cdot$ `platAsgCost`. Igual que para el caso anterior, una sintonización manual fue realizada con repeticiones para contrarrestar los efectos de la aleatoriedad. Después de pruebas en varias instancias, se determinó que un valor razonable para N es entre $[5, 10]$ (el valor ocupado en los experimentos siguientes es 5).

8. Resultados

Matching. Como se indicó anteriormente, un proceso de sintonización de parámetros fue realizado para tener buenos resultados en cada instancia. En general el parámetro más determinante en este caso es el número de iteraciones a realizar, de modo que a medida que aumenta la complejidad de las instancias, más iteraciones son requeridas. Para el largo

[1] `http://csrg.cl/~mavillan/instances/`

de la lista Tabú fue utilizado el siguiente criterio: `tabuLength` = `numIterTabu`/10 obteniendo buenos resultados. En Cuadro 1 se muestran los resultados para la instancia 4 con `numIterTabu` = 1000 y `tabuLength` = 10, obteniendo una solución óptima.

DepartureID	TrainID
Dep1	Train12
Dep2	Train2
Dep3	Train13
Dep4	Train4
Dep5	Train5
Dep6	Train6
Dep7	Train7
Dep8	Train8
Dep9	Train9
Dep10	Train10

Cuadro 1: Ejemplo de Matching (instancia 4) - `MinObjFunc` = 0

Todas las instancias pudieron ser resueltas con `MinObjFunc` = 0. Para las instancias pequeñas con 100 iteraciones fue suficiente, mientras que para las más grandes fueron necesarias $\sim$ 5000 para alcanzar una *match* óptimo. En Figura 9 se muestran los resultados sobre la instancia 10 para distintos números de iteraciones, y ocupando `tabuLength` = `numIterTabu`/10.

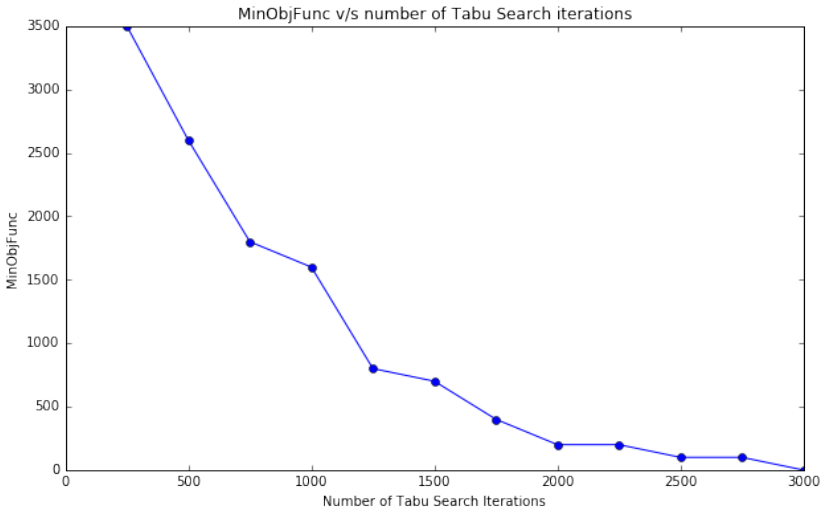

Figura 9: `MinObjFunc` para instancia 10 a distinto número de iteraciones (Fuente [11])

Platform Assignment. Al igual que el caso anterior, en primer lugar se realizó un proceso de sintonización de parámetros. Sin embargo este problema resultó más complejo de resolver que el anterior, no obteniendo soluciones perfectas para cada instancia. En 8 se muestran los resultados sobre la instancia 4 con `numIterTabu` = 1000 y `tabuLength` = 100, para lo cual no se obtiene una asignación perfecta, debido a dos *reusos* no satisfechos. El comportamiento que sigue este problema es similar al anterior; Para obtener buenos resultados en las instancias más complicadas es necesario iterar una mayor cantidad de veces.

[Arrival/Departure]ID	PlaformID
Arr10	Platform4
Arr12	Platform2
Arr13	Platform8
Arr2	Platform6
Arr4	Platform5
Arr5	Platform3
Arr6	Platform5
Arr7	Platform7
Arr8	Platform8
Arr9	Platform2
Dep1	Platform6
Dep10	Platform5
Dep2	Platform5
Dep3	Platform7
Dep4	Platform9
Dep5	Platform9
Dep6	Platform6
Dep7	Platform6
Dep8	Platform8
Dep9	Platform5

Cuadro 2: Ejemplo de Platform Assignment - `MinObjFunc` = 200

En la figura 10 se muestran los valores mínimos de la función objetivo `MinObjFunc` para cada una de las 10 instancias, con `numIterTabu`= 10000 y `tabuLength`= 100. claramente

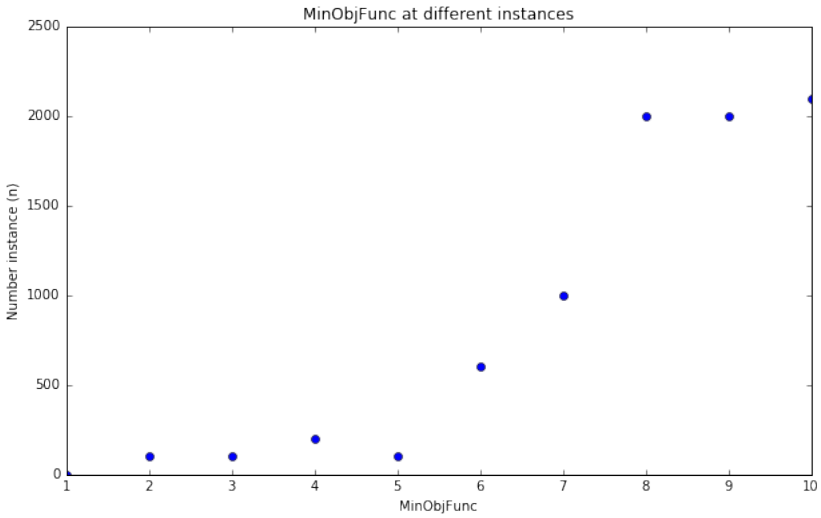

Figura 10: `MinObjFunc` para `numIterTabu`=10000 y `tabuLength`=100 (Fuente [11])

el problema es mucho más complicado de resolver en instancias mayores, habiendo una gran cantidad preferencias de plataforma insatisfechas.

9. Conclusiones

Conclusiones acerca del Estado del Arte

El estudio realizado en el presente documento, ha analizado a cabalidad el problema propuesto en ROADEF/EURO Challenge 2014 desde distintos enfoques. En base a esto se puede observar que la innovación que propone este problema, es modelar todas las operaciones que se realizan en una estación de una manera holística, y no cómo varios subproblemas más pequeños, que es lo que se ha venido haciendo desde antes. Sin embargo tal formulación, trae como consecuencia un problema extremadamente complejo. Esto se verifica en que para instancias reales, es totalmente infactible obtener una solución exacta, y es más, al día de hoy existen instancias para las cuales no se sabe (no se ha encontrado) ninguna asignación que deje sin cubrir alguna llegada o salida.

La verdadera dificultad del problema se puede verificar, viendo que tan sólo una versión simplificada de uno de sus subproblemas, el *Timetable Scheduling Problem* (TSP) en un sóla vía, ha demostrado ser NP-*hard* (en el sentido estricto) [7].

En virtud de tal dificultad, la mayoría de los trabajos relacionados que se estudiaron, tratan de desacoplar de alguna forma el problema original, para resolver subproblemas más simples. En general el enfoque más seguido es tratar el problema original en dos partes: En primer lugar determinan las posibles **asignaciones** de trenes para las salidas, y en segundo lugar se intentan computar las **rutas** que deben seguir los trenes entre los recursos de la estación, sin violar las restricciones. Es decir, el desacople consiste en tratar un problema de asignación y otro de enrutamiento.

Debido al inviabilidad de utilizar métodos exactos para computar la solución, la mayoría de enfoques que han tenido éxito han sido aquellos que han utilizado heurísticas, para así al menos generar soluciones aproximadas. En particular, tal como reporta [11], la combinación de distintas heurísticas ha probado ser muy efectiva. Debido a la gran cantidad de variables que posee el problema, y al tamaño del espacio de búsqueda, otro método que ha resultado útil ha sido el de generación de columnas. Es importante indicar, que pese a que los métodos exactos no son útiles para el problema general, estos sí han sido utilizados en subproblemas específicos. Tal es el caso de *Branch and Bound* en [13].

Para finalizar este estudio del *estado del arte*, se hace un énfasis en que la mayoría de los autores estudiados, indican que sus métodos y algoritmos tienen muchas variantes por probar y mejorar, por lo tanto las posibilidades de resolver instancias cada vez más grandes, y obtener buenas soluciones están abiertas.

Conclusiones acerca de la Implementación

La Heurística *Tabu* search mostró un comportamiento bastante bueno en la resolución de los dos primeros subproblemas tratados: *Matching Problem* y *Platform Assignment Problem*. Para el primero los resultados son notablemente buenos, pu-

diendo resolver todas las instancias de forma óptima en una cantidad razonable de iteraciones del algoritmo. Como se hizo notar, con ~ 5000 iteraciones fue posible resolver hasta las instancias más complejas del problema de forma óptima. Sin embargo para número inferiores de iteraciones, pese a no tener una solución óptima se generan soluciones subóptimas bastante aceptables. Esto es un gran aporte para casos de instancias reales, en donde aplicar técnicas completas resulta totalmente inviable, y por lo tanto la única alternativa es ocupar técnicas basadas en Heurísticas para acercarse de algún modo al óptimo.

Otro aspecto importante a notar, es que dado que la lista Tabú (que en verdad es un diccionario) posee una estructura simple, y sólo almacena los pares de asignaciones utilizadas anteriormente, es que entonces es posible manejar listas de gran tamaño sin requerir de un costo computacional elevado, ni de gran uso de memoria.

Para el segundo subproblema no se obtienen tan buenos resultados como en el caso anterior, pero de todos modos son aceptables. Los costos reportados en la función objetivo, son mayormente debido las preferencias de plataformas insatisfechas. Esto se debe a que, cómo se formuló el problema, la penalización por exceder la capacidad de una plataforma es mucho mayor que no satisfacer preferencias. Por otro lado, los recursos consumidos con anterioridad no aportan a la función objetivo, pues la soluciones que violan tales restricciones son simplemente descartadas. Probablemente un movimiento más elaborado que el *insert* propuesto permitiese generar mejores vecindarios, y sería una propuesta interesante de estudiar. Adicionalmente, para verificar el exceso de capacidad de una plataforma se toma una ventana de tiempo *pesimista*, en el sentido que se supone que cada tren estará el máximo de tiempo posible en la plataforma. Este criterio puede ser muy duro, y probablemente es una de las causales de que no se puedan llegar a mejores soluciones.

El tercer subproblema *Routing problem*, resultó ser el más complicado de formular y resolver. La mayor complicación se encontró en la determinación de soluciones vecinas (que fuesen cercanas de algún modo) para la aplicación de Tabu Search. El algoritmo propuesto para la generación de secuencias intermedias es interesante, pero muy poco práctico, pues requiere de manejar una gran cantidad de combinaciones entre rutas que unen dos recursos dados, siendo algunas de estas rutas *poco coherentes*. Una posible mejora a tal proceso, es un generación manual de rutas *coherentes* y *viables* entre dos recursos dados, ahorrandole así al algoritmo probar soluciones que no tienen mucho sentido, o que son del todo infactibles. Tal idea no es tan *descabellada* si se tiene en cuenta que se debe realizar una sola vez, puesto que raramente la estructura de los recursos de una estación de trenes se ve modificada.

10. Bibliografía

Referencias

[1] Françoise Ramond and Nicolas Marcos. ROADEF/EURO 2014 Challenge, Trains don't vanish! - Final phase, Rolling stock unit management on railway sites, 2014. SNCF - Innovation & Research Department.

[2] Challenge ROADEF/EURO 2014 Final results. `http://challenge.roadef.org/2014/en/finalResults.php`. Visitado: 23-05-2016.

[3] Haahr, Jørgen Thorlund and Simon Henry Bull. A Math-Heuristic Framework for the ROADEF/EURO Challenge 2014. Technical report, DTU Management Engineering, 2014.

[4] Nicolas Catusse and Hadrien Cambazard. Roadef Challenge 2014: A Modeling Approach, Rolling stock unit management on railway sites. Technical report, OSP, 2014.

[5] Jørgen Thorlund Haahr and Simon Henry Bull. Exact Methods for Solving the Train Departure Matching Problem. Technical report, Technical University of Denmark, 2015.

[6] Salomé Mulders and Yu-Lan Scholliers. Train Scheduling and Resource Managment in a Railway Station. Master's thesis, Unversité Catholique de Louvain, 2014.

[7] Caprara A, Fischetti M, Toth P. Modeling and solving the train timetabling problem. *Oper Res*, 50:851–861, 2002.

[8] Valentina Cacchiani, Alberto Caprara and Paolo Toth. A column generation approach to train timetabling on a corridor. *4OR*, 6(2):125–142, 2008.

[9] Mohammad Ali Shafia, Mohsen Pourseyed Aghaee, Seyed Jafar Sadjadi, and Amin Jamili. Robust Train Timetabling Problem: Mathematical Model and Branch and Bound Algorithm. *IEEE Transactions on Intelligent Transportation Systems*, 13(1):307–317, 2012.

[10] P. Tormos, A. Lova, F. Barber, L. Ingolotti, M. Abril, M. A. Salido. A Genetic Algorithm for Railway Scheduling Problems. *Studies in Computational Intelligence*, 128:255–276, 2008.

[11] A. Higgins and E. Kozan. Heuristic Techniques for Single Line Train Scheduling. *Journal of Heuristics*, page 43–62, 1997.

[12] Tomii Norio, Tashiro Yoshiaki, Tanabe Noriyuki, Hirai Chikara 1 and Muraki Kunimitsu. Train Rescheduling Algorithm Which Minimizes Passengers' Dissatisfaction. *Lecture Notes in Computer Science*, 3533:829–838, 2005.

[13] Andrea D'Ariano, Dario Pacciarelli and Marco Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183:643–657, 2006.

[14] Keivan Ghoseiri, Ferenc Szidarovszky and Mohammad Jawad Asgharpour. A multi-objective train scheduling model and solution. *Transportation Research Part B*, 38:927–952, 2004.

[15] Xiang Li, Dechun Wang, Keping Li and Ziyou Gao. A green train scheduling model and fuzzy multi-objective optimization algorithm. *Applied Mathematical Modelling*, 37:2063–2073, 2013.

[16] Ordered map C++ Reference. `http://www.cplusplus.com/reference/map/map/`. Visitado: 28-06-2016.

[17] Unordered map C++ Reference. `http://www.cplusplus.com/reference/unordered_map/unordered_map/`. Visitado: 28-06-2016.