UNIVERSIDAD TORCUATO DI TELLA
TD V: Diseño de Algoritmos

Trabajo Práctico 2: Gestión eficiente de recursos en sistemas ferroviarios

Integrantes:

Tomás Gallo

Nadia Molina

Jazmín Sneider

19 de Junio 2024

Indice

0 Introducción

En este trabajo práctico, simulamos ser una empresa dedicada a la resolución de problemas utilizando algoritmos avanzados. Nuestro objetivo actual es perfeccionar la planificación de recursos ferroviarios, con el fin de optimizar la cantidad de vagones necesarios para un funcionamiento eficiente del sistema. Actualmente, la tecnología nos da la posibilidad de modelar y resolver problemas con gran velocidad y precisión, proporcionando, en segundos, soluciones que a un personal humano experto le llevaría horas o incluso días calcular.

1 Ejercicio 1

Para modelar el problema, nos basamos en el paper publicado en junio de 2008 por Alexander Schrijver titulado *"Flows in Railway Optimization"*. En este artículo, se describe cómo la compañía de trenes Nederlandse SpoorWegen (NS) adaptó su antiguo cronograma de trenes de 1975 para crear uno nuevo en diciembre de 2006. Para lograrlo, NS contó con la ayuda de un equipo de matemáticos del Center for Mathematics and Computer Science (CWI) en Ámsterdam. El problema se modeló como uno de máximo flujo.

La interpretación de los horarios de los trenes como un problema de flujo no es nueva; otros investigadores ya habían explorado esta idea. De hecho, inspiró a Ford y Fulkerson en su famoso teorema de Máximo Flujo Corte Mínimo. Aunque el artículo menciona investigaciones previas, nos centraremos principalmente en el enfoque que NS adoptó en 2007 para modelar su nuevo cronograma de trenes. El objetivo que tenía la compañia y el que tenemos ahora nosotros es el mismo, responder la siguiente pregunta: **¿Cuál es la cantidad mínima de vagones necesaria para que el cronograma de trenes funcione normalmente y satisfaga todas las demandas diarias de cada tren?**

En ambos modelos, hacemos las siguientes suposiciones:

- Se considera una serie de estaciones por las que pasan los trenes. Aunque el artículo original menciona cuatro estaciones, en nuestro caso, solo tenemos dos: Retiro y Tigre. Aunque en ambos casos podría haber más estaciones intermedias, se simplifica el problema al reducir el número de estaciones.
- Todos los trenes en el cronograma tienen una demanda estimada de pasajeros que debe ser satisfecha.

En la propuesta del enunciado y en la del paper se modela el problema en un grafo de la siguiente manera:

Para representar los horarios de llegada y salida de cada cabecera, se crean nodos en un grafo. Los nodos se conectan si hay un viaje entre las cabeceras, y las aristas diagonales representan los horarios de salida y llegada. Además, los nodos de una misma cabecera también se conectan, ya que se pueden agregar o retirar unidades de trenes que hayan llegado anteriormente para los trenes que están por salir.

Finalmente, se conecta el último nodo de cada cabecera con el primero para que el proceso pueda reiniciarse por la mañana y otorgar un cronograma sostenible en el tiempo. Estas aristas tienen un peso de 1, lo que nos permite calcular el total de vagones sumando los valores de flujo de estos arcos nocturnos.

Podemos ver que el modelo que se presenta en el paper y en el enunciado son bastante similares con algunas ligeras diferencias, el paper luego propone cómo abordar el problema con vagones con distintas capacidades pero no indagaremos mucho más en el asunto. Leerlo nos ayudó a comprender cómo adaptar el problema a uno de máximo flujo con costo mínimo, más allá de las explicaciones proporcionadas en el enunciado.

2 Ejercicio 2

El modelo con el se manejaba la empresa está basado en un algoritmo que gestiona *unidades nuevas* y *unidades en stock* en cada cabecera. Las *unidades nuevas* son aquellos vagones que se deben agregar para satisfacer la demanda en ciertos horarios, mientras que las *unidades en stock* son los vagones que llegan a la estación como resultado de arribo de trenes.

Al inicio del día, antes de realizar cualquier envío, hay 0 unidades nuevas y 0 en stock en ambas cabeceras. El objetivo del algoritmo es cumplir con las demandas de los trenes utilizando primero los vagones en stock y, si estos no son suficientes, emplear unidades nuevas. Esta estrategia permite reutilizar vagones, reduciendo la cantidad de nuevos vagones necesarios y, por ende, la cantidad en circulación. Finalmente, la cantidad de vagones necesarios en cada cabecera al inicio del día está determinada por el número de unidades nuevas de cada estación.

A pesar de que el algoritmo provee soluciones factibles, éstas no son del todo óptimas ya que siempre envía la cantidad mínima de vagones necesarios para satisfacer la demanda. Esto significa que no puede enviar unidades adicionales a otra estación para ser utilizadas de manera más eficiente en el futuro. Por ejemplo:

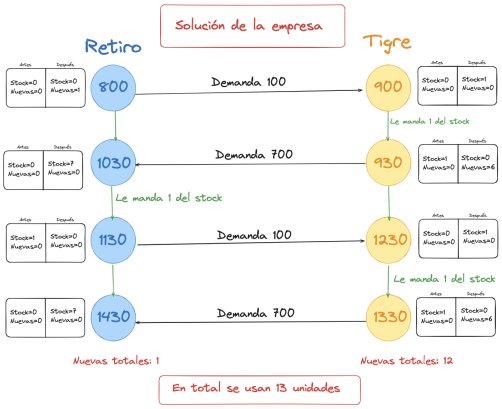

Figure 1: Algoritmo empresa

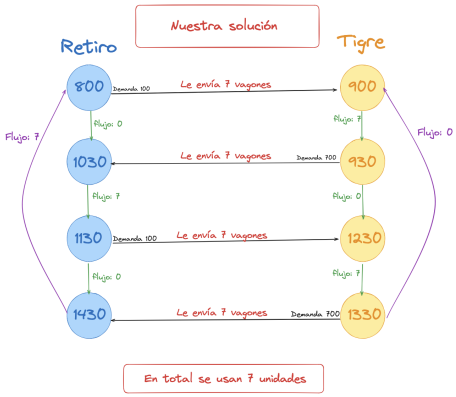

Figure 2: Algoritmo óptimo

En este ejemplo, se puede ver que una solución posible es tener 13 vagones en circulación. No obstante, esta solución no es óptima, ya que existe una opción más económica que consiste en tener solo 7 vagones en circulación. Esta solución implica enviar más vagones de los necesarios en ciertos horarios, lo cual ayuda a reducir la necesidad de agregar nuevas unidades de vagones en el futuro.

Por un lado, el algoritmo de la empresa comienza con 0 unidades en stock y nuevas en ambas cabeceras. Luego, comienza con el envío de un tren desde Retiro a las 08:00 que tiene una demanda de 1 vagón, la cual satisface enviando 1 vagón, que es lo mínimo. Esa unidad llega a Tigre a las 09:00 y se guarda como una unidad en stock para ser utilizada posteriormente. A las 09:30, sale otro tren desde Tigre hacia Retiro que demanda 7 vagones. Tigre tiene 1 vagón en stock, por lo que envía ese vagón y saca 6 unidades nuevas. Con la llegada del tren a las 10:30 a Retiro, la estación tiene 7 unidades en stock.

A las 11:30, sale un tren desde Retiro hacia Tigre con una demanda de 1 vagón, por lo que envía 1 vagón de los 7 en stock. Este vagón llega a Tigre a las 12:30 y se guarda como stock. Finalmente, a las 13:30, sale un tren de Tigre a Retiro que demanda 7 vagones. Tigre utiliza la 1 unidad en stock y pide 6 unidades nuevas.

Siguiendo este algoritmo, se necesitaron 12 unidades nuevas en Tigre (6 a las 09:30 y 6 a las 13:30) y 1 unidad nueva en Retiro (a las 08:00), sumando un total de 13 unidades en circulación.

El algoritmo óptimo, en cambio, permite enviar una mayor cantidad de vagones de los necesarios para cumplir con la demanda específica del horario, optimizando el uso futuro. Por ejemplo, a las 08:00, en lugar de enviar solo 1 vagón, se envían 7 vagones para ser reutilizados a las 09:30 en Tigre, evitando la necesidad de comprar vagones nuevos y aumentar la cantidad en circulación. Lo mismo sucede a las 11:30 desde Retiro hacia Tigre. En total, se utilizan 7 vagones en todo el día, casi la mitad que en la solución de la empresa.

3 Ejercicio 3

Siguiendo las especificaciones del enunciado y del paper referencial, el código lee los datos desde un archivo proporcionado por el usuario como parámetro. Los datos incluyen los horarios de salida y llegada de los trenes en las estaciones de interés, almacenándolos en las listas HorariosRetiro y HorariosTigre como números tipo float. Además, se implementa un diccionario que asocia cada tren (clasificado por horario, tipo y estación) con una demanda específica de usuarios, dividida por la capacidad de cada vagón. Estos datos son esenciales para la construcción del grafo. Con el objetivo de ordenar el grafo según los tiempos de menor a mayor, se itera primero sobre las listas ordenadas de floats y obtenemos, a partir de ellos, las claves del diccionario para nombrar los nodos del grafo (notar que las claves del diccionario incluyen el horario, entre otros datos). Cada nodo está asociado con una demanda específica que refleja los imbalances en el grafo: los nodos de salida de trenes tienen demanda positiva para asegurar que se envíen al menos los trenes necesarios para cumplir con el cronograma, mientras que los nodos de llegada tienen demanda negativa para indicar que retienen los trenes que llegan. Una vez que se agregaron todas las aristas requeridas según las especificaciones del enunciado (aristas de tren, de traspaso y de trasnoche), el grafo está preparado para ejecutar un algoritmo de flujo máximo con costo mínimo. Este algoritmo, utilizando el comando **nx.capacity_scaling**, devuelve el costo mínimo necesario y un diccionario de todos los flujos entre los nodos del grafo. Para determinar la cantidad mínima de vagones necesarios para que el sistema opere con normalidad, se suma el valor de flujo de las aristas de trasnoche. Estas aristas representan los vagones que permanecen en cada estación durante la noche para iniciar el servicio por la mañana, representando así la mínima cantidad necesaria. Dado que estas aristas son las únicas con un costo asociado de 1, el costo total devuelto por el algoritmo también representa el número total de vagones necesarios. Decidimos que el código devuelva tanto el costo total como, para cada estación, el valor de flujo asociado a su arista de trasnoche, especificando así los requisitos de vagones en cada estación cabecera al inicio del servicio.

3.1 ¿Cómo se representa el flujo en el grafo?

Mientras revisabamos el diccionario de flujo para verificar el correcto funcionamiento del algoritmo, observamos que varias aristas "de tren" tenían flujo 0. Inicialmente, esto nos generó confusión, ya que según los imbalances definidos, debían enviarse al menos la cantidad mínima de vagones necesarios para cumplir con la demanda del sistema. Posteriormente, comprendimos que debido a la forma en que se incorporaron los imbalances al grafo, los vagones mínimos obligatorios no se reflejan explícitamente en el flujo. Cuando el diccionario de flujo indica un valor 0, en realidad significa que se está enviando la cantidad mínima requerida según los imbalances establecidos. Por otro lado, si el flujo es mayor que 0, esto implica que se están enviando no solo

la cantidad mínima necesaria, sino también vagones adicionales que se utilizan para optimizar la solución.

Por ejemplo, si dos nodos conectados tienen un desbalance de +5 y -5 respectivamente, y el flujo que pasa por la arista que los une tiene un valor de 2, esto indica que en realidad se están enviando 7 vagones por ese tren. Los vagones adicionales más allá del mínimo requerido representan la capacidad del sistema para ajustar la asignación de recursos de manera eficiente.

3.2 Resultados con las intancias del enunciado:

Realizamos pruebas con nuestro código en ambas bases de datos proporcionadas: **toy_instance.json** y **retirotigresemana.json**, con el fin de observar los resultados obtenidos. En el primer caso, pudimos verificar que el grafo se construía correctamente, ya que la cantidad de nodos y aristas no era excesiva. Los resultados mostraron que se requerían un total de 15 vagones, coincidiendo con el enunciado. Más específicamente, 10 vagones eran necesarios en Tigre y 5 en Retiro al inicio del servicio. En el diccionario de flujos, observamos y confirmamos las diferencias entre este modelo y el que utilizaba anteriormente la empresa. A diferencia del método antiguo, este modelo no solo envía la cantidad indispensable para satisfacer la demanda, sino que también incluye vagones adicionales y realiza transferencias estratégicas para optimizar la cantidad total de vagones utilizados. Para la segunda instancia, dado que era una base de datos real, nos fue imposible verificar la correcta construcción del grafo debido a su magnitud. Sin embargo, los resultados fueron coherentes. En total, se necesitaban 48 vagones, lo cual tiene sentido dado el incremento en la cantidad de servicios y las variaciones en la demanda. Específicamente, se requerían 47 vagones en Tigre y 1 en Retiro al inicio del servicio. Al comparar los resultados con nuestros compañeros de clase, notamos que todos coincidíamos en la cantidad total de vagones necesarios (48). Algunos obtuvieron el mismo resultado que nosotros (47 en Tigre y 1 en Retiro), mientras que otros determinaron que se necesitaban 48 vagones en Tigre y 0 en Retiro. Esto nos llevó a considerar la posibilidad de que, **para una misma solución óptima, puedan existir múltiples formas de construirla.** Más abajo, en la sección de experimentación intentamos replicar un escenario en el que hubiera más de una manera de construir la solución óptima.

3.3 Precaución extra:

Aunque el modelo que explicamos anteriormente funcionaba perfectamente en todos los casos que probamos, nos dimos cuenta de que no estábamos considerando un **detalle importante**: si dos trenes llegan y salen simultáneamente en la misma estación, se generan 2 nodos con el mismo horario y estación, pero uno de tipo A y otro de tipo D. Dependiendo de cómo se organicen estos nodos en el grafo (un proceso completamente aleatorio sin condiciones específicas en el código), podrían o no enviarse los vagones que acaban de llegar mediante una arista de traspaso.

Decidimos que no tiene sentido permitir que un tren que llega transfiera vagones a otro que está saliendo en el mismo momento exacto, ya que este proceso no tendría sentido temporalmente. Para probar este escenario, creamos dos archivos CSV donde una misma estación contiene dos nodos idénticos (uno de llegada y otro de salida), pero con el orden invertido en los CSV. En uno de los archivos, el nodo de llegada se registra primero, lo que resulta en una transferencia de vagones desde el tren que llega hacia el que está saliendo. Como resultado, este escenario requiere menos vagones que el otro archivo CSV, donde se registra primero el nodo de salida y no se realiza ninguna transferencia de vagones al tren que está llegando. Estos ejemplos deberían arrojar la misma solución, pero observamos resultados diferentes debido a que en el código no especificamos lo suficiente como para no permitir esos traspasos.

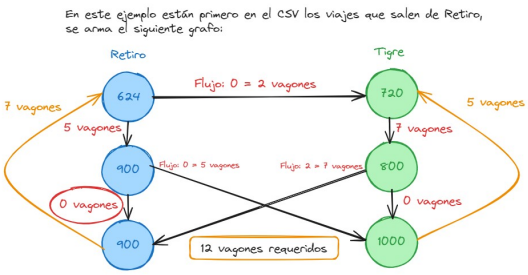

Figure 3: Se registra primero el nodo de salida (más vagones requeridos)

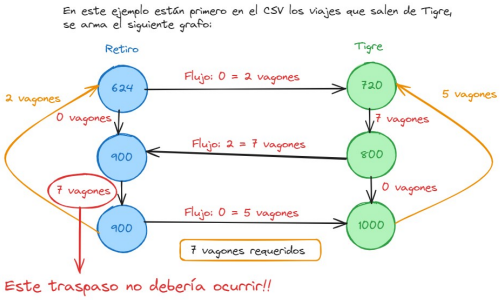

Figure 4: Se registra primero el nodo de llegada (se hace el traspaso)

Como solución, se nos ocurrió que si, en una misma estación, el horario del nodo que estamos a punto de registrar **es de salida** y ya existe ese valor numérico en la lista de horarios, le restamos un valor muy pequeño (0.1) para posicionarlo antes del nodo que está llegando. De esta manera, evitamos la transferencia de vagones entre trenes del mismo horario. Del mismo modo, si el horario del nodo que estamos registrando **es de llegada** y ya existe ese valor numérico en la lista, le sumamos 0.1 para ubicarlo después del nodo de salida, también evitando la transferencia de vagones. Con estos cambios en el código conseguimos que ambos escenarios mencionados anteriormente arrojen la misma cantidad de vagones mínimos necesarios.

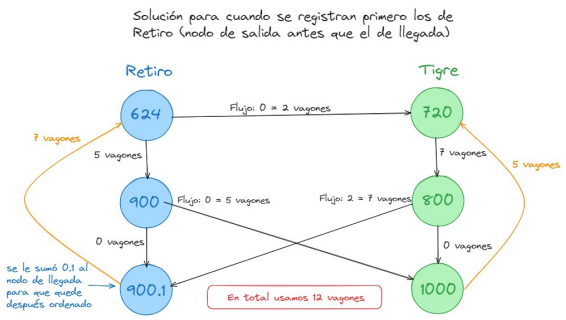

Figure 5: Se registra primero el nodo de salida, entonces se le resta a llegada

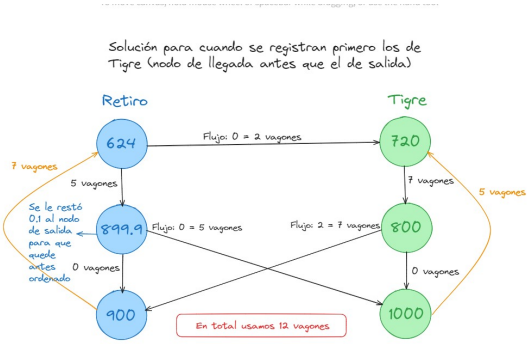

Figure 6: Se registra primero el nodo de llegada, entonces se le suma a salida

4 Ejercicio 4

Para este inciso, decidimos crear una instancia de datos basada en información real obtenida de la página oficial del gobierno argentino: https://www.argentina.gob.ar/nuevo-cronograma-de-horarios-retiro-tigre . Esta información incluye los horarios de salida y llegada de los trenes en distintos días y horarios. Sin embargo no incluian datos de la demanda por servicio, por lo que, arbitrariamente, decidimos introducir demandas de pasajeros muy variadas y sin un patrón específico. Esto nos permitió observar cómo el sistema maneja diferentes escenarios y niveles de demanda.

Una vez que probamos ejecutar nuestro algoritmo con este cronograma, observamos que la cantidad mínima de vagones necesarios para operar eficientemente según los horarios y demandas establecidos es de 94 vagones. Necesitando al inicio del servicio 87 en Tigre y 7 en Retiro.

5 Ejercicio 5

En este inciso, se nos pide modelar el mismo problema que en el Ejercicio 3 pero con una diferencia importante. Antes, en ambas cabeceras los trenes podían quedarse allí durante la noche para poder iniciar el servicio a la mañana. Esto estaba representado en el grafo por la arista de trasnoche que tenía capacidad ilimitada, es decir que no nos interesaba la cantidad de trenes que "durmieran" en esa estación mientras garantizaran una solución con costo mínimo para satisfacer las demandas. Sin embargo, ahora una de las cabeceras **posee una restricción de la cantidad de trenes que pueden quedarse allí en la noche** por lo que tuvimos que replantearnos el problema.

Reflexionando sobre esta nueva problemática tuvimos que pensar qué tanto afectaba la cota a la solución del problema. Si la cota **es mayor** que la cantidad de vagones que pasaban por la arista

en la solución del ejercicio 3 (la solución óptima), entonces la solución era la misma. Sin embargo, si la cota **era menor** que la cantidad de trenes que pasaban por esa arista normalmente, había que pensar qué hacer con el resto de los trenes que no podían "dormir" en la estación por la noche. Se nos ocurrió la alternativa de **agregar una arista extra**, enviando el resto de los trenes hacia la otra estación para que pudieran iniciar ahí el servicio a la mañana. Esto desembocó en 2 escenarios, dependiendo del número que se eligiera como cota para la estación:

- **Desemboca en una nueva solución factible**: El servicio de la estación restringida puede iniciar normalmente con la cantidad de vagones que la restricción permitió que pasaran y la otra estación inicia el servicio con vagones extra que luego se irán traspasando a medida que avance el cronograma.
- **Desemboca en una solución no factible:** Esto ocurre cuando la cota es demasiado restrictiva y la cantidad de vagones que dejó pasar no es suficiente para que el servicio pueda funcionar con normalidad. Por lo tanto, por más de que hagamos llegar los vagones restantes a la otra estación y hacer que inicien allí el servicio, organizados de esta manera no alcanzan para cumplir las demandas de todo el cronograma.

¿Cómo expresamos esta solución en el código? El último nodo de cada una de las estaciones sigue conectado al primero de esa estación con costo 1, la diferencia es que ahora una de esas dos aristas (dependiendo de la cabecera que elija el usuario) tiene una cota máxima del flujo que puede pasar por ahí. Desde el último nodo de la cabecera que tiene la arista restringida conectamos otra arista al primer nodo de la otra cabecera para que pueda enviar por allí los trenes restantes. Decidimos ponerle un costo elevado (arbitrariamente elegimos 100) para que la solución solo utilice esta ruta alternativa si no hay otra opción (es decir, si no se pueden enviar más vagones por la primera arista que tiene menor costo debido a la restricción impuesta por las obras). Si no le aumentábamos lo suficiente el costo a la arista que conecta una estación con la otra, el algoritmo podía usarla cuando no era necesario para pasarle vagones a la otra estación y así reducir la cantidad mínima de vagones totales a un costo igual o menor que el de la solución original, modificando el resultado. Notamos rápidamente que esta decisión también cambiaba lo que íbamos a devolver como solución ya que antes (Ejercicio 3), retornar el costo final implicaba devolver la cantidad de vagones minimos necesarios para arrancar el dia siguiente. Si sumábamos los flujos de las aristas de trasnoche obteníamos el mismo resultado ya que éstas tenían costo 1.

Sin embargo, en este nuevo enfoque, retornar el costo sería un error ya que este no representa más la cantidad mínima de vagones debido a las modificaciones que hicimos. Al hacer que traspasar vagones hacia la otra cabecera tenga un mayor costo que trasladándolo a la misma cabecera, alteramos el costo y devolveríamos que son necesarios más vagones que los que en realidad se requieren. Entonces decidimos que nuestro nuevo valor de retorno ahora sea el flujo total enviado durante la noche, es decir la suma de ambas aristas de trasnoche, más la nueva arista que conecta

las dos cabeceras. Todo esto se devuelve solamente si nos encontramos en el escenario de una solución factible.

6 Experimentación:

Experimento 1: *Reutilización de los vagones.*

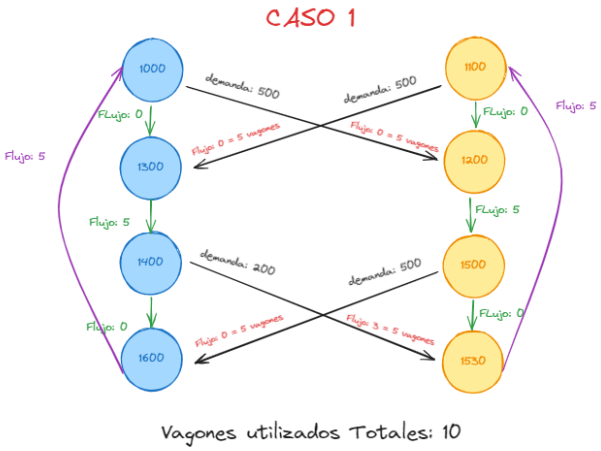

Figure 7: Se registra primero el nodo de salida, entonces se le resta a llegada

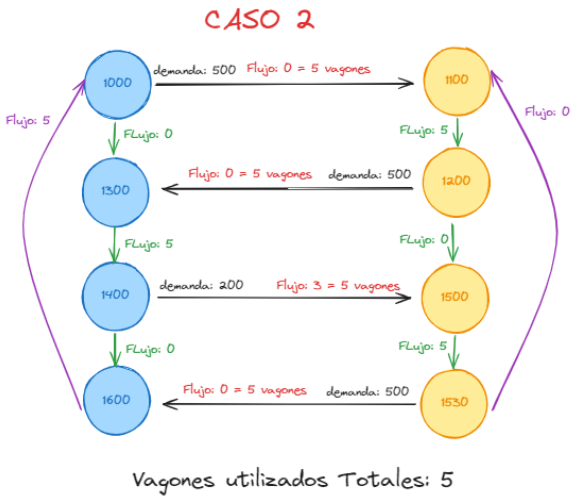

Figure 8: Se registra primero el nodo de llegada, entonces se le suma a salida

En este experimento quisimos ver cómo varían los flujos de las aristas. Para ello creamos 2 archivos csv con los mismos horarios y demandas, pero con las conexiones entre estaciones cambiadas. En el primer caso no es posible reutilizar trenes en circulación para ahorrarselos a futuro, por lo que hay una demanda total de 10 vagones, en el caso 2 las aristas están puestas para que se puedan ahorrar vagones a futuro, así teniendo que usar solo 5 vagones totales. Logramos ver que en el caso 2, en una arista se envían 3 vagones de exceso a la demanda que ayudan a optimizar el resultado final, tal y como queríamos.

Experimento 2: *Tiempo de ejecución del algoritmo con cota y sin cota.*

En este experimento miramos los tiempos de ejecución de ambos algoritmos con datos que contenían 5 vagones de demanda en todos sus servicios, por lo que la demanda no debería de haber afectado el tiempo de ejecución, únicamente la cantidad de servicios. Además, para calcular el tiempo que tarda en correr el algoritmo con cotas, lo ejecutamos con varias cotas *factibles* y realizamos un promedio de esos tiempos resultantes.

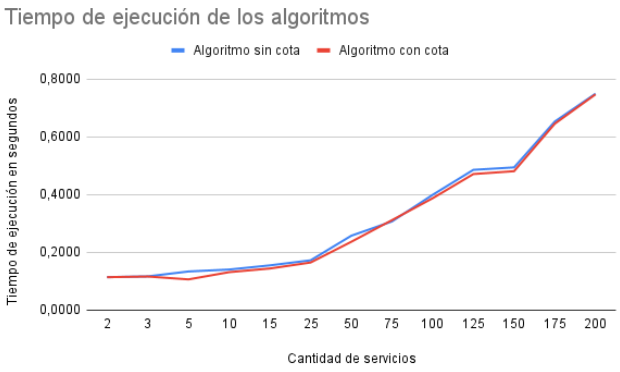

Figure 9: Enter Caption

Por un lado, pudimos observar que, claramente el tiempo de ejecución aumenta a medida que tengo más servicios. También vimos que el tiempo de ejecución del algortimo con cotas, es un poco más rápido que el normal, estuvimos investigando y esto se puede deber a que al poner una cota a una arista en un problema de flujo en redes, se reducen las soluciones que se deben analizar. El resultado es porque las cotas en las aristas limitan la cantidad máxima de flujo que puede pasar a través de ellas, lo cual puede influir en la determinación del flujo máximo en el grafo. En conclusión, el algoritmo se vuelve más eficiente al limitar las opciones disponibles y dirigir la búsqueda hacia una solución óptima más rápidamente. Sin embargo, la diferencia entre tiempos de ejecuciones está dada por milesimas de segundo, por lo que no podemos afirmar con certeza que el algortimo con cota realmente funciona más rápido que el común.

Experimento 3: *Variación de la cota máxima en aristas entre estaciones.* Corrimos un archivo creado por nosotros al cual le fuimos cambiando la variable *max_rs* entre 25 y 5 vagones (menos de 5 no tiene una solución factible).

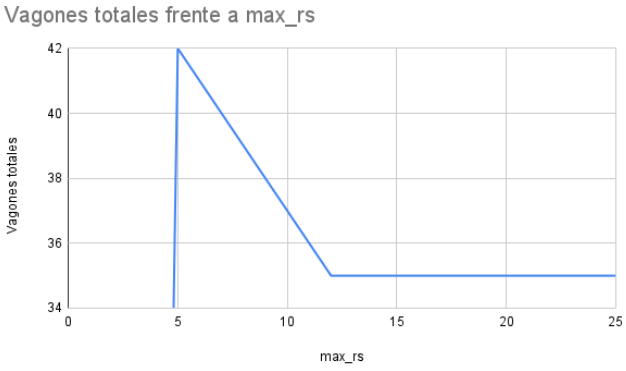

Figure 10: Vagones dependiendo de la cota máxima

Pudimos observar que, a medida que la cota se achica, se utilizan más vagones para correr

el algoritmo. Deducimos que esto se puede deber a que no se pueden enviar muchos vagones en exceso entre estaciones para reutilizarlos a futuro de manera más óptima, por lo que para satisfacer la demanda hay que pedir nuevos vagones, más de los que ya están en circulación.

Experimento 4: *Variación del tiempo de ejecución con demandas.* Corrimos el mismo archivo, pero aumentando la cantidad promedio demandada del mismo en el código sin cota:

Vagones demandados	1	5	10	100	1000	100000000
Tiempo de ejecución en segundos	0.132	0.135	0.145	0.122	0.140	0.132

Table 1: Tabla de datos de vagones demandados y tiempo de ejecución

Llegamos a la conclusión de que la demanda no afecta el tiempo de ejecución del código.

Experimento 5: *¿Es posible que una solución óptima se construya de diferentes formas?* Como mencionamos anteriormente, la diferencia que observamos respecto a otros grupos en la forma de construir la solución óptima para **retiro-tigre-semana.json** nos generó curiosidad sobre si podrían existir instancias donde las soluciones óptimas varíen. No en cuanto al valor final, sino en cómo se distribuyen los trenes en las estaciones. Realizamos pruebas con varias instancias, pero no logramos determinar qué factor definía si una instancia tendría más de una solución posible. Sin embargo, casualmente, al probar la extensión del cronograma real de otro grupo (para verificar si nuestros algoritmos coincidían), nos dimos cuenta de que la instancia siempre requería 56 vagones, pero la distribución de estos variaba según el momento de la ejecución. En algunas ocasiones, obteníamos 32 vagones en Tigre y 24 en Retiro; en otras, 40 en Tigre y 16 en Retiro. Ejecutamos la función 100 veces y los resultados fueron más o menos equilibrados: aproximadamente la mitad de las veces obteníamos una opción y la otra mitad, la otra. No logramos descubrir por qué ocurre esto ni qué tienen de especial ambas instancias que permiten múltiples soluciones, pero nos gustaría investigar más sobre esto en el futuro, ya que nos resultó interesante.

7 Conclusión

La resolución de este trabajo práctico, que requería adaptar un problema del mundo real - la optimización de material ferroviario para ahorrarle costos a una empresa - a un modelo de maximización de flujo y minimización de costo que quizás veíamos como puramente teórico, nos ayudó a comprender con mayor profundidad estos conceptos. Además, nos permitió adquirir habilidades prácticas en la resolución de problemas complejos utilizando algoritmos existentes cuya eficiencia ya es conocida.

Consideramos que esta experiencia también mejoró nuestras capacidades de programación utilizando la librería **networkx** y archivos de tipo **json**, reforzando nuestra habilidad para abordar

problemas que requieran el análisis de bases de datos de gran magnitud.
Finalmente, llegamos a la conclusión de que los algoritmos son herramientas que pueden aplicarse para resolver diversos problemas, más allá del problema para el que fueron específicamente diseñados. Imaginemos si tuviéramos que abordar cada cuestión desde cero, diseñando un algoritmo específico para resolverlo. Sería un proceso largo y complejo. Sin embargo, al adaptar las resoluciones eficientes existentes a una problemática especifica, simplificamos enormemente la búsqueda de una solución. Este enfoque es muy común en la actualidad. Una vez que se logra adaptar un problema a uno que ya cuenta con un algoritmo probado, muchos pasos se vuelven más sencillos. Además, aprovechar algoritmos existentes permite beneficiarse de décadas de investigación y desarrollo. De esta forma, podemos concentrarnos en aplicar y ajustar la solución en lugar de empezar desde cero.