Tecnología Digital V: Diseño de Algoritmos

TP1 - Servicios de movilidad on-demand en tiempo real: estrategias y algoritmos

Autores: Federico Giorgi, Gastón Loza Montaña y Tomás Curzio.

Introducción

Contexto

En los últimos tiempos, el surgimiento masivo de plataformas colaborativas que conectan demanda con oferta ha transformado la manera en que interactuamos con diversos servicios, incluyendo el transporte. En este contexto, las aplicaciones colaborativas de movilidad privada, como Uber, Cabify, Didi, entre otras, han logrado un gran éxito al actuar como intermediarias entre pasajeros y conductores, brindando una experiencia más cómoda y eficiente.

El objetivo de este trabajo es analizar un aspecto fundamental del funcionamiento de estas plataformas: la asignación en tiempo real entre pasajeros y conductores. Con el crecimiento sostenido de usuarios de estas empresas, se ha ido complejizando el desafío de lograr emparejamientos efectivos ante requerimientos de experiencia del usuario cada vez más exigentes. A través de este análisis, buscamos ofrecer recomendaciones que permitan optimizar el sistema de emparejamiento y brindar una experiencia aún más satisfactoria tanto para los pasajeros como para los conductores de la plataforma.

El problema y la decisión a tomar

El trabajo nos pone en el rol de consultores para una empresa que se dedica a dar servicios de movilidad, conectando a pasajeros con conductores. La empresa, dado un instante o intervalo pequeño de tiempo, en cierta área geográfica, tiene una cierta cantidad de pasajeros pidiendo un viaje, y otra cantidad de conductores disponibles. Nuestro objetivo es decidir, de manera inteligente, qué vehículo debe buscar a cada pasajero.

Para especificar mejor el problema, debemos tener en cuenta los datos que la empresa posee a su disposición previamente, y las simplificaciones que le realizamos al mismo.

Datos:

- Para cada pasajero podemos asumir que tenemos:
  - El instante en el que realizó el pedido.
  - El origen del viaje (donde se realiza el *pick-up*).
  - El destino del viaje o la distancia estimada del recorrido.
  - La estimación de la tarifa a cobrar por el viaje.
- Por cada conductor disponible para realizar un viaje podemos asumir que tenemos su localización en tiempo real.

Luego, combinando ambas podemos asumir entonces que tenemos la distancia que llevaría a cada vehículo llegar a un potencial pasajero para comenzar el viaje.

Simplificaciones al problema:

- Asumimos que la oferta y la demanda esta balanceada, es decir que hay misma cantidad de pasajeros que de conductores.
- Asumimos que los pasajeros se encuentran ordenados de manera creciente según el instante en el que realizan el pedido.
- Asumimos que ningun pasajero tiene prioridad sobre otro.
- Asumimos que no es necesario lograr una distribución razonable de viajes a lo largo del día entre los distintos conductores.

Estrategia FCFS

El problema puede ser resuelto desde distintos enfoques. El que actualmente utiliza la empresa que nos contrata es una idea muy natural que se basa en atender a los pasajeros por orden de llegada y para cada uno de ellos tomar la mejor decisión posible en el momento.

En otras palabras, utiliza el criterio de "First Come, First Served" y toma una decisión greedy para asignar el vehículo más cercano al pasajero que está siendo atendido en el momento. De esta forma, se toma la mejor decisión local para cada cliente pero ésta podría ser distinta a la mejor decisión global.

Podemos dividir esta estrategia en los siguientes pasos:

1. Considerar a los pasajeros por orden de llegada.
2. Asignar al pasajero que llegó primero el vehículo más cercano.
3. Remover este vehículo de los posibles a asignar (para que no haya un vehículo asignado a dos pasajeros) y remover al pasajero de la lista.
4. Repetir hasta que no haya más pasajeros.

Implementación para estrategia FCFS

Como parte del trabajo como consultores para la plataforma de intermediación de movilidad nos proponemos en primer lugar analizar la estrategia actual de la empresa para poder contar con resultados de rendimiento con los cuáles comparar las nuevas propuestas a la resolución del problema.

Para ello, siguiendo la idea de la estrategia explicada, realizamos una implementación de la misma en C++ que toma los datos de la distancia entre vehículos y pasajeros (en orden de llegada) en forma de una matriz y la recorre para asignarle a cada pasajero el vehículo disponible más cercano a su posición (*ver GreedySolver.cpp*).

Esta estrategia e implementación es totalmente válida. Sin embargo, si hay una cantidad de demanda significativa, es posible tomar otro enfoque, en el que en lugar de ir atendiendo por orden de llegada en tiempo real, se espera cierta cantidad de tiempo (algunos segundos) y se agrupa a los pasajeros que hayan pedido un viaje en ese lapso de tiempo, para luego tomar una decisión de asignación de vehículos sabiendo las localizaciones de todos. Dicha estrategia se denomina "batching".

Estrategia de Batching

Llamamos a ese grupo de clientes que pide el viaje en el lapso de tiempo en el que estamos esperando "batch", de ahí el nombre batching.

Con este batch podremos tomar una decisión global para el problema de asignación. Así, nuestro objetivo es de alguna manera realizar una asignación de vehículos a pasajeros de forma tal que la suma de las distancias

recorridas de los vehículos para recoger a los pasajeros sea mínima. De esta forma, buscaremos no sólo disminuir el tiempo de espera colectivo de todos los pasajeros del batch sino también reducir los costos para los conductores teniendo que recorrer una menor distancia colectiva.

Notar que es posible que en algun momento elijamos un vehículo que no es el más cercano a un pasajero. Sin embargo, esta decisión local no optima permite aportar a la mejor decisión global para minimizar la suma de distancias. Así, esta estrategia es totalmente distinta a la estrategia *greedy* vista anteriormente.

Se puede ver que $\sum dist_{ij}$ *en batching* $\leq \sum dist_{ij}$ *en greedy* ya que, si minimizamos la suma de distancias teniendo toda la información, efectivamente esa suma sera menor o a lo sumo igual que elegir el taxi más cercano para cada pasajero en orden mientras van llegando los pedidos de viaje (podría ser igual en el caso en el que casualmente la asignación que minimiza las distancias es la misma que la de asignarle el taxi más cercano a cada pasajero por orden de llegada).

Vamos a modelar el problema de encontrar dicha asignación optima con grafos.

Modelo para estrategia de Batching

Nuestro modelo utiliza la idea de resolver matching máximo en grafos bipartitos como un problema de flujo máximo.

Para construirlo, partimos de un grafo que contiene n nodos, representando a los taxis que forman nuestra particion V_1 y otros n nodos representando a los pasajeros que forman nuestra particion V_2, sabemos además que $n = \#V_1 = \#V_2$ por como se define el problema en las consignas del trabajo.

Posteriormente, conectamos cada nodo que representa a un taxi con los n nodos que representan a los pasajeros, representando que en principio podemos asignar cualquier taxi a cualquier pasajero. No conectamos taxis entre sí o pasajeros entre sí, por lo que nos queda un grafo bipartito.

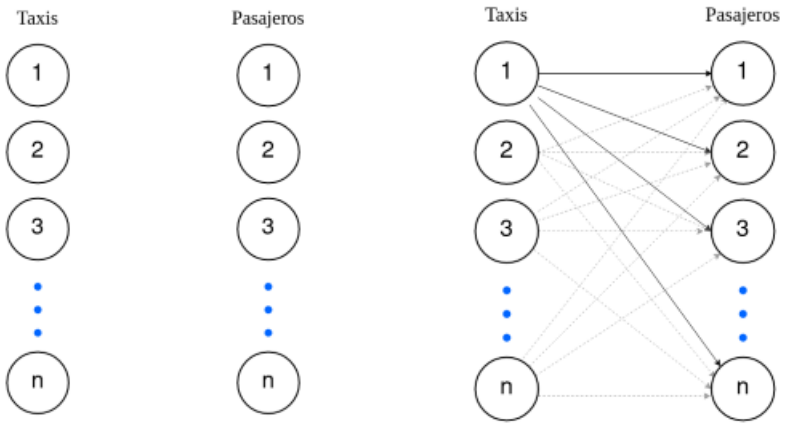

Figure 1: Grafo bipartito completo

Una vez obtenido el grafo es importante ver que no solo queremos realizar el matching máximo, si no que también queremos minimizar la distancia que recorren los taxis hasta recoger al pasajero. Para resolver esto, transformamos el problema de matching máximo en el grafo bipartito en un problema de flujo máximo y costo mínimo, siendo el costo que buscamos minimizar, la distancia mencionada. Para ello, debemos agregar al grafo que teniamos un nodo Source (S) y un nodo Sink (T) y asignar tanto capacidades como costos adecuados para representar nuestro problema.

Una vez agregados, conectamos S con todos los taxis, dándole a cada arista capacidad 1, pues queremos que cada taxi se asigne a solo un pasajero, y costo 0, pues aún no se esta asignando ningún pasajero y no hay ninguna distancia que tomar. Luego, en las aristas que conectan a los taxis con los pasajeros, les asignamos capacidad 1, por el mismo motivo que antes y es que no queremos asignarle más de un pasajero

a los taxis (aunque es verdad que al haber elegido capacidad 1 en la conexión de S con los taxis, esto ya estaba limitado y podríamos elegir cualquier capacidad ≥ 1), y el costo será dado por la distancia desde el taxi al pasajero, es decir, la arista que conecta al i-ésimo taxi con el j-ésimo pasajero tendra capacidad 1 y costo $dist_{ij}$. Finalmente, conectamos los pasajeros a T, con capacidad 1 por el mismo motivo anterior (aunque otra vez, ya estaba limitado) y costo 0, pues interpretativamente esta conexión representa que el viaje terminó y ya no hay ninguna distancia (costo) a tener en cuenta.

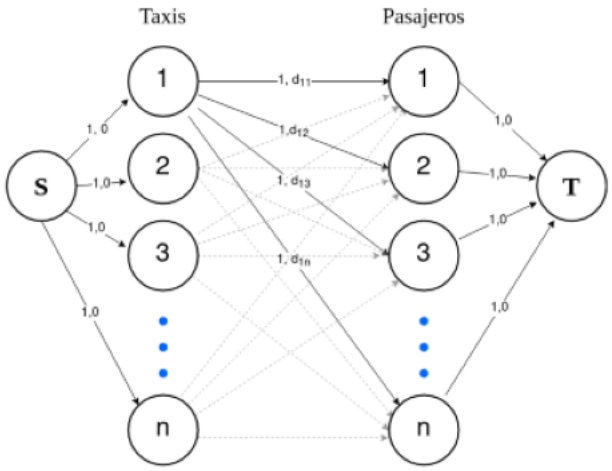

Figure 2: Modelo estrategia Batching

No se ve representado en las lineas punteadas pues saturaría la visualización pero cada linea punteada se conecta con capacidad 1 y costo $dist_{ij}$, al igual que en las que se ven marcadas.

El flujo máximo del grafo será n, enviando una unidad de flujo por cada arista desde S a los taxis, por lo tanto, cualquier combinación de taxis y pasajeros que "sacie" a todos los pasajeros, es decir, que haya un taxi asignado a cada pasajero nos dará el flujo máximo. Luego, si de estos flujos máximos buscamos el que tenga costo minimo obtendremos la combinación de taxis y pasajeros que sacia a todos los pasajeros y además minimiza la distancia recorrida por los taxis para recoger a los mismos. De esta manera, resolvemos el problema planteado en la estrategia de batching utilizando grafos de manera inteligente.

Implementación para estrategia de Batching

Utilizando el modelo explicado, realizamos una implementación de esta estrategia utilizando la librería "or-tools", que nos provee una manera de resolver el problema de flujo máximo con costo mínimo dadas las siguientes estructuras de datos:

- Un vector "start_nodes" y otro "end_nodes" donde el i-ésimo valor de start_nodes está conectado con el i-ésimo valor de end_nodes (con estos dos vectores se representan los arcos del grafo).
- Un vector "capacities" con la capacidad de cada arco.
- Un vector "cost_units" con el costo de cada arco.
- Un vector "supplies" con los desbalances/imbalances de cada nodo.

Con esto, el problema de implementación de nuestro modelo y cálculo del flujo maximo con costo minimo se reduce a simplemente representar al grafo previamente explicado con esos 5 vectores.

¿Cómo hacemos esto para una instancia cualquiera?

Debemos tener en cuenta que nuestro input tiene una matriz de $n \times n$ de taxis y pasajeros, donde la posicion i, j determina $dist_{ij}$.

Con la información de este output y numerando...

- El nodo source como 0.
- Los taxis del 1 al n.
- Los pasajeros del $n+1$ al $2n$.
- El nodo sink como $2n+1$.

Podemos armar los 5 vectores de la siguiente manera:

- start_nodes:
  1. Inicializamos el vector con n ceros, que serán todas las conexiones de nuestro nodo source (al cual numeramos como nodo 0).
  2. Luego, agregamos al vector las conexiones de cada taxi a todos los pasajeros, es decir, agregaremos n veces cada taxi (los cuales numeramos del 1 al n).
  3. Finalmente, agregamos al vector las conexiones de los pasajeros al vector sink (al cual numeramos como nodo 2n+1), es decir, agregamos la secuencia $n+1, n+2, \cdots, 2n$.
- end_nodes:
  1. Este vector de alguna manera "cierra las conexiones" que abrimos en el vector anterior, por lo que debemos inicializarlo cerrando las conexiones de nuestro nodo source a los taxis, es decir, agregamos la secuencia $1, 2, \cdots, n$.
  2. Luego, debemos cerrar las conexiones de cada taxi a todos los pasajeros, es decir, para el taxi 1, que en start_nodes se encuentra n veces lo tenemos que conectar con los n pasajeros, y así para todos los taxis, por lo que debemos agregar n veces la secuencia $n+1, n+2, \cdots, 2n$.
  3. Finalmente, debemos cerrar las conexiones de los pasajeros al nodo sink es decir, conectar el nodo del pasajero $n+1$ con el nodo sink $2n+1$, el nodo del pasajero $n+2$ con el nodo sink $2n+1$ y así hasta conectar todos los pasajeros. Para ello agregamos n veces $2n+1$.
- capacities:
  1. Como vimos en el modelo, todas las capacidades de los arcos son 1, por lo tanto simplemente debemos crear un vector con $2n+n^2$ (cantidad de arcos) unos.
- cost_units:
  1. Como mencionamos previamente, los arcos como los de source a los taxis tienen costo 0, por lo que inicializamos el vector con n ceros.
  2. Ahora, debemos agregar los costos (distancias) de cada taxi a cada pasajero, que es justamente lo que tenemos en la matriz. Por lo tanto, debemos aplanar la matriz y "concatenar" nuestro vector con n ceros y la matriz aplanada. En nuestro código a medida que vamos aplanando la matriz la vamos agregando al vector, logrando el mismo objetivo.
  3. Finalmente, como los arcos de los pasajeros al sink tambien tienen costo 0, debemos agregar n ceros más al final del vector, para obtener nuestro vector de costos.
- supplies:

1. Or-Tools nos pide un vector de inbalances, pero, si observamos nuestro modelo, no hay inbalances en los nodos, simplemente se cumple que, salvo para S y T, en cada nodo la cantidad de flujo que entra al nodo debe ser igual a la cantidad de flujo que sale del nodo. Sin embargo, si vemos la definición de inbalance, nos dice que el inbalance $b_i = \sum x_{ij} - \sum x_{ji}$, es decir, el inbalance debe ser igual a la diferencia entre "lo que sale y lo que entra". Si queremos agregar inbalances a nuestro modelo y que se siga cumpliendo la ecuación de conservación de flujo, a todos los nodos exceptuando S y T les asignamos inbalance 0 y ambos modelos son equivalentes.
2. ¿Que hacemos para S y T? Por como se define el problema de flujo con inbalances, necesitamos que estos se balanceen es decir que $\sum b_i = 0$. Como todos los inbalances de los otros nodos son 0, necesariamente $b_S = -b_T$. Podriamos elegir cualquier k y decir que $b_S = k$, $b_T = -k$ pero en este caso en particular nosotros queremos que de source salgan los n taxis y a sink ingresen los n pasajeros, por lo que para que se cumpla ello, diremos que $b_S = n$, $b_T = -n$. Recordando que el inbalance es la diferencia entre lo que sale y lo que entra podemos ver porqué tiene sentido, ya que de source salen n taxis y no entra nada $(n - 0 = n)$ y de sink no sale nada y llegan n pasajeros $(0 - n = -n)$.
3. Una vez planteado esto, es sencillo armar el vector. Inicializamos un vector de tamaño $2n + 2$ (cantidad de nodos) con todo ceros y cambiamos la posicion 0 del vector por n, y la posicion 2n+1 del vector por -n.
4. De esta manera nos queda un vector de inbalances como el planteado: $[n, 0, 0, \cdots, 0, -n]$.

Evaluación comparativa de las estrategias

Dado que una de las principales motivaciones de la formulación de un nuevo modelo (que tome una decisión global para la asignación entre vehículos y pasajeros) es la reducción de costos de los conductores y tiempos de espera de los pasajeros, nos proponemos, a través de experimentaciones, evaluar si efectivamente el nuevo modelo aporta en esa dirección.

Experimentación

Teniendo en cuenta que la implementación de la estrategia de Batching toma a las distancias de recogida como los costos a minimizar, el principal criterio a considerar será ver cómo se comparan estos costos, computados como la suma de distancias para cada instancia, entre las distintas estrategias.

En segundo lugar, además de analizar los costos, buscaremos analizar el rendimiento económico de los conductores dado un ratio de rendimiento por km recorrido. Este ratio se define como:

$$r = \frac{tarifa\ (\$)}{dist.\ recogida + dist.\ viaje\ (km)}$$

Por último, tomamos el tiempo de ejecución de cada estrategia utilizando la librería std::chrono de C++.

Para comparar el modelo propuesto con la estrategia de Batching con la versión actual de la empresa (FCFS) definimos las siguientes métricas:

- %cost_gap: diferencia porcentual relativa de la estrategia batching sobre la greedy en base a los costos.
- %time_gap: diferencia porcentual relativa de la estrategia batching sobre la greedy en base al tiempo de ejecución.
- %yield_gap: diferencia porcentual relativa de la estrategia batching sobre la greedy en base al rendimiento económico de cada kilómetro recorrido por el conductor para concretar el viaje del pasajero asignado.

Para la evaluación de nuestra estrategia, tenemos a disposición cuatro conjuntos de instancias de diferentes tamaños (10, 100, 250 y 500) que nos permiten observar los resultados de los diferentes algoritmos con una variedad de escenarios de asignación. Cada conjunto contiene diez instancias distintas para agregar diversidad a los escenarios considerados.

Las instancias fueron generadas en base a la información de los pasajeros se seleccionó al azar de los registros de viajes en taxi del área de Manhattan, Nueva York durante el mes de diciembre de 2018. Además, tanto la ubicación de los pasajeros como la de los vehículos se generaron de forma aleatoria utilizando información geoespacial sobre las zonas correspondientes.

Discusión y análisis de resultados

Los resultados obtenidos de la evaluación comparativa entre la estrategia FCFS y la estrategia de batching se presentan en la siguiente tabla. Se indica para cada tamaño de instancia, el promedio de las métricas previamente detalladas. Se pueden ver los resultados desagregados en el Anexo I.

n	%cost_gap	%time_gap	%yield_gap
10	14.74	-4005.53	-12.78
100	16.90	-6840.77	-4.00
250	17.32	-5425.12	1.04
500	14.57	-6527.45	16.67

Estos resultados basados en las métricas definidas anteriormente nos permiten evaluar el desempeño de ambas estrategias en términos de costos (tanto viaje de conductores y tiempo de espera de los pasajeros), tiempo de ejecución y rendimiento económico para los conductores.

En cuanto a los costos, observamos que para todas las instancias evaluadas se obtiene una mejora significativa en torno al 15%, lo que indica una reducción en los costos totales al asignar los vehículos con las estrategia de batching. Se puede notar que las mejoras relativas varían dependiendo de los tamaños de las instancias entre un 14.57% y un 17.32%. Notando los resultados podríamos inferir que ya cuando se experimenta con un matching entre 500 conductores y 500 pasajeros, la mejora del nuevo algoritmo con respecto al actual comienza disminuir. Sin embargo, considerando que los "batch" se encontrarán acotados la mejora es consistente con el objetivo de resolución del problema propuesto.

A pesar de la mejora en términos de costos, al analizar el tiempo de ejecución se observa que los resultados muestran un %time_gap negativo en todas las instancias evaluadas. Esto indica que el tiempo de ejecución de la estrategia de batching es mayor en comparación con la estrategia FCFS. Los porcentajes son considerablemente altos y varían entre -4005.53% y -6527.45%, lo que demuestra un empeoramiento significativo en la eficiencia del proceso de asignación al utilizar la estrategia de batching en lugar de la estrategia FCFS.

Por otro lado, en relación a la diferencia relativa en términos del rendimiento económico por kilómetro recorrido, encontramos resultados mixtos. En las instancias de menor tamaño (10 y 100), se observa una disminución en el rendimiento económico con la estrategia de batching. Sin embargo, en las instancias de mayor tamaño (250 y 500), se obtiene un aumento del rendimiento económico en comparación con la estrategia FCFS. Dado que el modelo de la estrategia batching tiene como función objetivo la minimización de la suma de distancias y no así el rendimiento económico de los conductores, en principio no podríamos explicar facilmente a que se deben estos resultados. Más adelante atacaremos este problema de estar teniendo en cuenta las distancias y no el rédito económico con nuestro modelo alternativo.

En resumen, los resultados indican que la estrategia de batching proporciona mejoras significativas en términos de la distancia colectiva de recogida de pasajeros que se traducirá también en menos tiempo de espera para los pasajeros, pues a pesar de haber un gran diferencial en el tiempo de ejecución, el mismo no es muy grande, el amplio % se explica por que greedy es muy rapido en terminos de ejecución.

Limitaciones y posibles extensiones

Como notamos, si bien la nueva estrategia propuesta aporta al objetivo reducir las distancias de búsqueda de los pasajeros y sus tiempos de espera existen ciertas limitaciones.

En particular, tal cual observamos, el rendimiento económico por km recorrido no es superior en ninguna estrategia con respecto a la otra. Muchas veces, este factor es de gran relevancia para los conductores ya que se oberva que en ocasiones se les asignan viajes que requieren recorrer distancias largas para recoger al pasajero, para luego hacer un viaje de distancia corta en comparación. Esto hace que la relación entre el beneficio (dado por la tarifa del viaje del pasajero) tenga poca relación con el costo asociado a la búsqueda de ese cliente por parte del conductor.

Además, es importante resaltar que si bien estos resultados se obtuvieron utilizando conjuntos de datos tomados de información real, la aplicabilidad de estos resultados puede variar en diferentes contextos y ubicaciones geográficas. Otro de los aspectos tal vez más relevantes que este modelo está obviando del contexto es el tráfico. Sin embargo, en este caso no contamos con información sufiente para tener este aspecto en cuenta en el modelo.

Estrategia Alternativa

Dada la información con la que cuenta la empresa la cuál nos contrata como consultores, tendríamos la posibilidad de atacar la limitación acerca de la relación entre los costos de los conductores y el beneficio económico por el viaje. Para ello, de alguna forma deberíamos buscar maximizar el ratio r de rendimiento por km mencionado en la sección de experimentación previa.

Considerando que en la estratgia de batching adaptamos nuestro problema a un problema de flujo máximo de costo mínimo, podríamos hacer lo mismo en este caso. Sin embargo, dado que aquí queremos maximizar beneficio deberíamos definir un nuevo ratio de la forma

$$r' = \frac{dist.\,recogida + dist.\,viaje\,(km)}{tarifa\,(\$)}$$

que en lugar de representar la cantidad de dinero por km recorrido, indique la cantidad de km necesarios para recibir una unidad de dinero extra. Así, cuanto más chico sea este ratio r' mayor rendimiento económico recibirán los conductores.

Modelo para estrategia alternativa

Adaptar nuestro modelo original a la nueva estrategia es sencillo, puesto a que seguimos teniendo que buscar un flujo maximo con costo minimo, antes buscabamos que se asignen n taxis a n pasajeros minimizando la suma de las distancias y ahora buscamos que se asignen n taxis a n pasajeros pero minimizando nuestro ratio r'. Teniendo en cuenta esto, nuestro modelo anterior, cambiando distancias por r' es esencialmente lo mismo.

De esta manera, llegamos al siguiente grafo, que representa nuestro modelo alternativo, donde:

$$r'_{ij} = \frac{dist_{ij} + dist.viaje_j}{tarifa.viaje_j}$$

Implementación para estrategia alternativa

Al igual que con la estrategia de batching, realizamos una implementación de esta estrategia utilizando la librería “or-tools” con las mismas estructuras de datos definidas para resolver el problema de flujo máximo

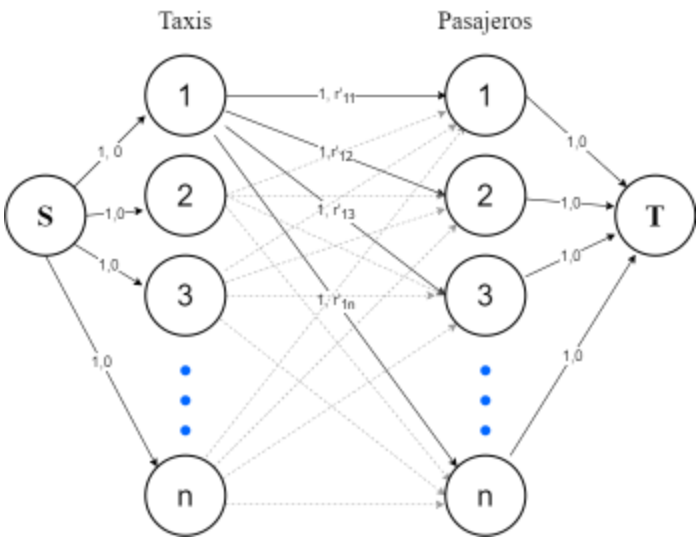

Figure 3: Modelo Alternativo

con costo mínimo. Los vectores requeridos por la librería serán construidos de la misma manera que en la estrategia batching para cada instancia con excepción del vector "cost_units". Para el mismo tendremos en cuenta:

1. Como en el anterior modelo, los arcos de source a los taxis tienen costo 0, por lo que, al igual que en la primera implementación, inicializamos el vector con n ceros.
2. Ahora, debemos agregar los costos de cada taxi a cada pasajero, esto es lo que cambia con respecto a nuestra implementación anterior, pues antes estos costos eran simplemente las distancias, que se encuentran en la matriz, y ahora deben ser los ratios r'. Utilizaremos una función similar a la anterior para aplanar la matriz, solo que esta vez, no añadiremos al vector el valor de la distancia, sino que este le sumaremos la distancia a realizar en el viaje y a esta suma la dividiremos por la tarifa a cobrar (es decir, calculamos el ratio r' para cada combinacion de taxi-pasajero). De esta manera agregamos al vector $r'_{11}, r'_{12}, \cdots, r'_{nn}$.
3. Finalmente, como en el modelo anterior, los arcos de los pasajeros al sink tambien tienen costo 0. Por lo tanto debemos hacer lo mismo que en nuestra primera implementación, agregar n ceros más al final del vector, para obtener así nuestro vector de costos.

Con esto terminamos de crear los 5 vectores requeridos por "or-tools" para resolver el problema de flujo máximo con costo mínimo y así completar nuestra implementación del modelo alternativo.

Experimentación con nuestro modelo alternativo

Métricas Alternativo vs. Greedy

n	cost_gap%	time_gap%	yield_gap%
10	11.76	-2328.29	-14.45
100	9.49	-5839.61	-8.68
250	10.46	-5187.87	-4.64
500	8.62	-5106.64	7.2

Métricas Alternativo vs. Batching

n	cost_gap%	time_gap%	yield_gap%
10	-3.45	27.91	-1.78
100	-8.99	-5.34	-5.99
250	-8.34	4.12	-6.49
500	-6.98	11.16	-11.77

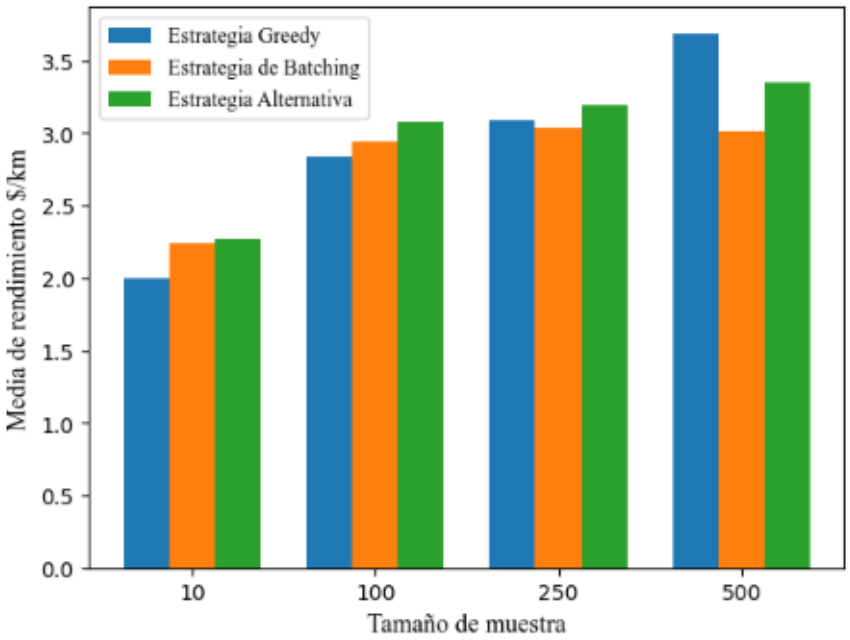

Figure 4: Comparación de medias de rendimiento entre las 3 estrategias

Como podemos ver tanto en las tablas como en el gráfico (figura 4), para nuestras instancias más grandes, el modelo greedy es el que mayor beneficio economico nos provee. Esto era algo que no esperabamos ver. Sin embargo, es importante tener en cuenta que el modelo greedy depende mucho del orden en el que se realizan los viajes (básicamente, del azar), mientras que tanto el modelo de batching como nuestro modelo alternativo son más constantes.

Por otro lado, comparando con batching, nuestro rédito económico en el modelo alternativo es siempre mejor, y no depende del azar como greedy, por lo que es un modelo que sería inteligente adoptar para la empresa. Sin embargo, no nos convence del todo, por lo que definimos otra alternativa posible que analizaremos a continuación.

Nueva estrategia alternativa

Nuestra nueva estrategia ataca otro problema que no habíamos tenido en cuenta y es que es muy probable que los conductores no quieran realizar un tramo de distancia muy larga para ir a recoger un pasajero que realizará un viaje mucho más corto que $dist_{ij}$. Para mitigar esto mismo, vamos a definir un nuevo ratio,

$$rd = \frac{dist.\,recogida\ (km)}{dist.\,viaje\ (km)}$$

Notar que, mientras más chico sea rd, mejor para el conductor pues el ratio disminuye cuando o bien la distancia para recoger al pasajero es chica o cuando la distancia del viaje a ser realizado es grande. Esto es importante a la hora de definir nuestro modelo y luego implementación.

Modelo para nueva estrategia alternativa

Nuestro modelo será prácticamente idéntico a los anteriores, con el mismo grafo, pero cambiando los costos de los arcos que conectan taxis y pasajeros por $rd_{ij} = \frac{dist_{ij}}{dist.viaje_j}$, quedando como se puede observar en la figura 5.

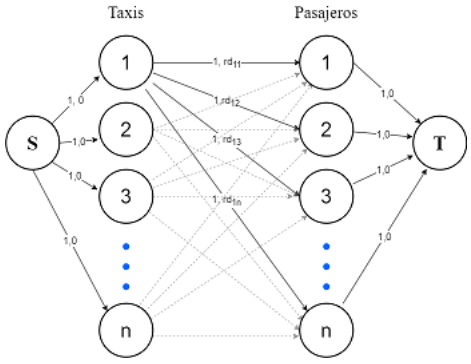

Figure 5: Nuevo Modelo Alternativo

Implementacion para nueva estrategia alternativa

Como ya vimos en los modelos anteriores, necesitamos 5 vectores para resolver el problema de flujo máximo, costo mínimo con "or-tools". El unico vector que es distinto que el de los modelos anteriores es el de "cost_units", veamos como construirlo.

1. Como en los otros modelos, los arcos de source a los taxis tienen costo 0, por lo que, al igual que en la primera implementación, inicializamos el vector con n ceros.
2. Ahora, debemos agregar los costos de cada taxi a cada pasajero, que paso de ser, en el caso de batching las distancias, y en el otro modelo alternativo r' a ser nuestro nuevo ratio rd. Haremos lo mismo de ir aplanando la matriz pero, a la hora de agregar elementos al vector, además dividiremos por la distancia del viaje, agregando así $rd_{11}, rd_{12}, \cdots, rd_{nn}$.
3. Finalmente, como en los otros modelos, los arcos de los pasajeros al sink tambien tienen costo 0. Por lo tanto debemos hacer lo mismo que en nuestra primera implementación, agregar n ceros más al final del vector, para obtener así nuestro vector de costos.

Con esto terminamos de crear los 5 vectores requeridos por "or-tools" para resolver el problema de flujo máximo con costo mínimo y así completar nuestra implementación del nuevo modelo alternativo.

Discusión y análisis de resultados

Comparamos nuestro alternativo 2 a los otros modelos como habíamos hecho antes pero agregamos una nueva métrica, relative_dist_gap%, que representa el promedio de gaps en el nuevo ratio rd.

Métricas Alternativo 2 vs. Greedy

n	cost_gap%	time_gap%	yield_gap%	relative_dist_gap%
10	10.23	-2794.35	-22.41	42.55
100	8.55	-7669.98	-25.71	60.16
250	11.12	-5423.75	-23.03	65.98
500	9.34	-4860.75	-26.39	65.14

Métricas Alternativo 2 vs. Batching

n	cost_gap%	time_gap%	yield_gap%	relative_dist_gap%
10	-5.21	21.01	-8.33	30.26
100	-10.13	-32.83	-21.36	51.73
250	-7.55	7.21	-26.29	56.55
500	-6.14	-2.46	-54.09	61.98

Métricas Alternativo 2 vs. Alternativo 1

n	cost_gap%	time_gap%	yield_gap%	relative_dist_gap%
10	-1.78	-13.74	-6.77	4.89
100	-1.05	-71.65	-15.83	24.89
250	0.72	-2.43	-19.26	21.49
500	0.79	-12.89	-37.99	32.26

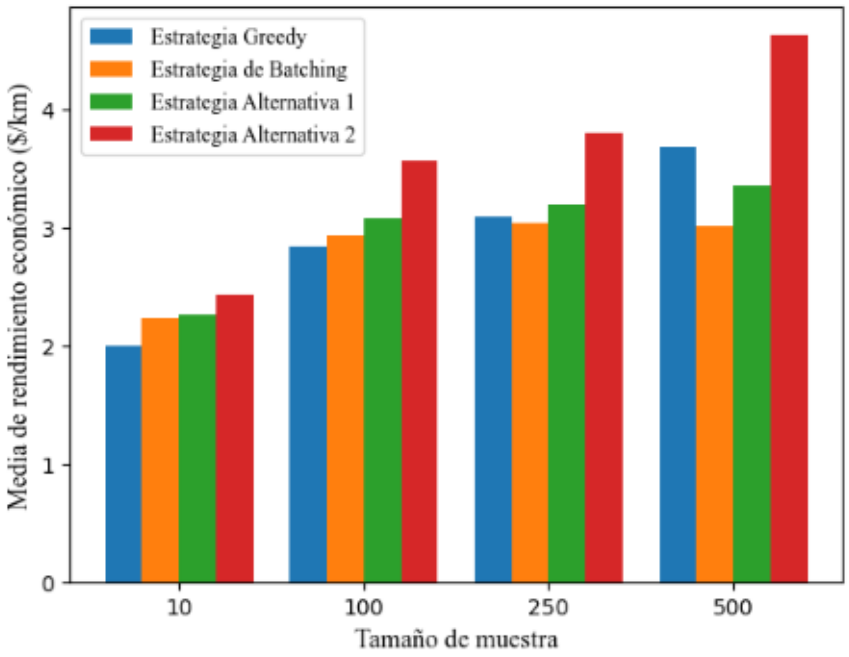

Figure 6: Comparación de medias de rendimiento $/km entre las 4 estrategias

Viendo los gráficos (figuras 6, 7 y 8) podemos observar como nuestro ratio *rd* se minimiza con este modelo, cumpliendo el objetivo de que nuestros conductores no viajen mucho para realizar un viaje corto (o al menos minimizando esos casos).

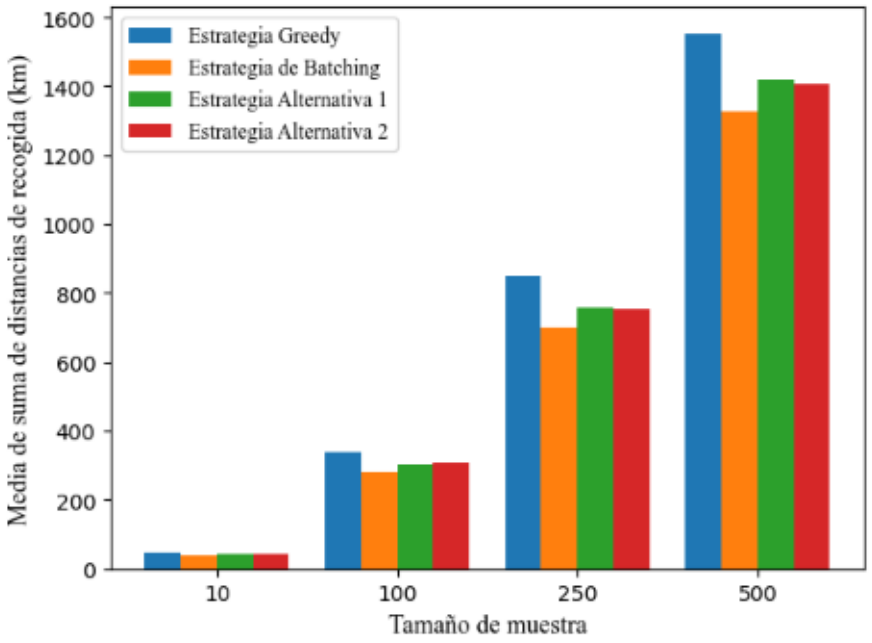

Figure 7: Comparación de medias de suma de distancias de recogida entre las 4 estrategias

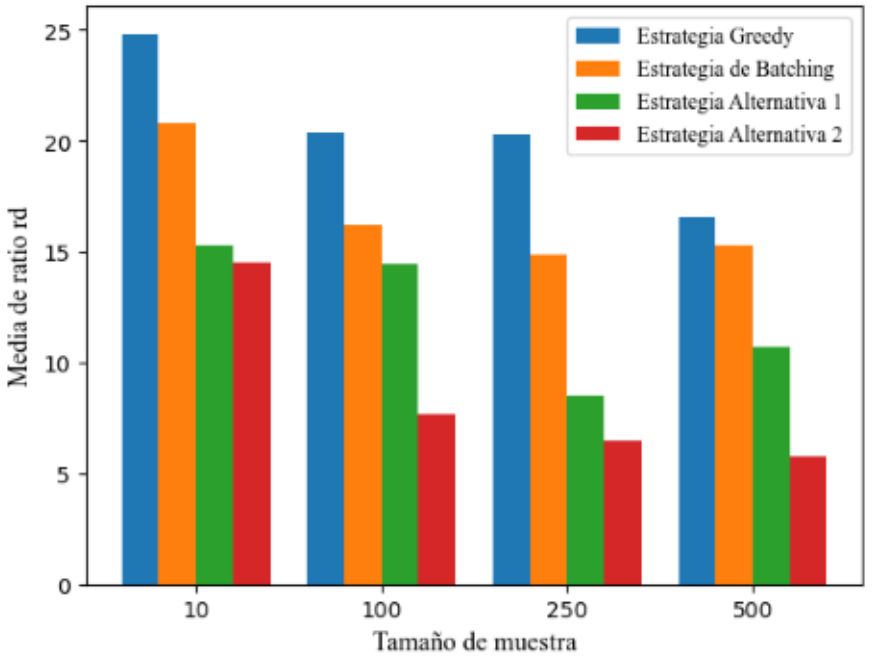

Figure 8: Comparación de medias de los ratios rd entre las 4 estrategias

Además, podemos ver como, al menos con la información con la que contamos, nuestro nuevo modelo alternativo también es el que mejores rendimientos económicos nos provee. En terminos de distancias de recogida, es un intermedio entre greedy y batching, resultados muy similares a nuestro primer modelo alternativo en este sentido.

Por último, analizando los tiempos de ejecución que podemos ver en las tablas, nuestro modelo alternativo tarda bastante más que greedy, al igual que batching y el primer modelo que habíamos propuesto, lo cual es lógico ya que todos se basan en la misma idea. Comparar entre ellos no tiene mucho sentido ya que las diferencias no son muy grandes y probablemente tengan que ver con irregularidades a la hora de correr el código (por ejemplo, si había algun navegador abierto al momento de correr un código y para el otro no), pues la implementación en los tres casos es muy similar.

En cuanto a la diferencia con *greedy*, si bien la diferencia es muy grande, el tiempo de cómputo necesario para instancias grandes como $n = 500$ es del orden de aproximadamente 90ms, despreciable en nuestro contexto pues tan poco tiempo no hará la diferencia para una persona pidiendo un taxi.

Conclusión

A lo largo de este informe, se han planteado distintas estrategias y junto a ellas modelos e implementaciones para resolver el problema de asignar vehículos a cada pasajero.

Con la información con la que contamos, nuestra experimentación sobre la misma con los distintos modelos y el análisis realizado, podemos ver como nuestro segundo modelo alternativo nos provee mejor rédito económico, asi como de cierto modo hace más felices a los conductores, minimizando los viajes indeseados para ellos, a cambio de que la suma de distancias de recogida sea un poco mayor que en *batching* y un tiempo de ejecución, aunque completamente asumible incluso para instancias grandes, bastante mayor que con la estrategia *greedy*.

Teniendo en cuenta todo esto, consideramos que el modelo que debería aplicar la empresa para la cual realizamos la consultoría debería ser este último.

Aclaraciones

En el dataset hay tarifas negativas y tarifas que son 0. Las tarifas negativas consideramos que eran un error en los datos y les aplicamos valor absoluto. Aquellas tarifas que eran 0 las cambiamos por 1 aplicando una especie de "tarifa mínima", pues no tiene sentido "regalar un viaje" y en nuestro primer intento de modelo alternativo, estaba causando que dividamos por 0.

Cuando calculamos los rendimientos economicos (con la solucion ya dada), si por algun motivo el denominador daba 0 (es decir, tanto distancia de recogida como distancia del viaje = 0) no los tomamos en cuenta para el analisis a la hora de calcular los promedios (variable casos_anomalos en el código).

Intentamos ser consistentes a la hora de agregar int64_t a los vectores creados pero es posible que en algún caso hagamos push_back de algun int a un vector de tipo int64_t. De todas formas C++ realiza el casteo, por lo que no genera problemas a la hora de compilar/correr el código.

En las imagenes del modelo y la explicación, muchas veces nos referimos tanto a los taxis como a los pasajeros del 1 al n (por ejemplo, en el modelo de batching la distancia del taxi 1 al pasajero 1 esta expresada como d_{11}). En el código necesitabamos que no se repitan los "nombres" de los nodos, por lo que los taxis estan numerados del 1 al n y los pasajeros del n+1 al 2n pero en esencia el pasajero n+1 es lo que muchas veces llamamos el pasajero 1.