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

Bernardez Camila, Giménez Costa María Agustina, Oliva Micaela

2023-06-28

Introducción al problema

En este trabajo nos centramos en resolver el problema de asignación de vehículos en una agencia de taxis, es decir, buscamos encontrar un modelo de decisión que nos permita saber que conductor debe pasar a buscar a cada pasajero.

En principio, tomamos como métrica de éxito de una asignación la minimización de la distancia recorrida por los conductores hasta la ubicación de su pasajero asignado.

Inicialmente contamos con un modelo intuitivo y fácil de implementar llamado `GreedySolver`. Es una heurística basada en FCFS, que asigna el taxi más cercano a los pasajeros según su orden de llegada a la cola de espera. Sin embargo, al ser una heurística, no tenemos garantía de que encuentre una solución óptima al problema planteado. Por eso, a continuación buscaremos e implementaremos un modelo que tome una decisión a nivel global, lo que nos garantizará que la distancia recorrida por los conductores hasta sus pasajeros sea mínima.

Descripción del modelo y la decisión

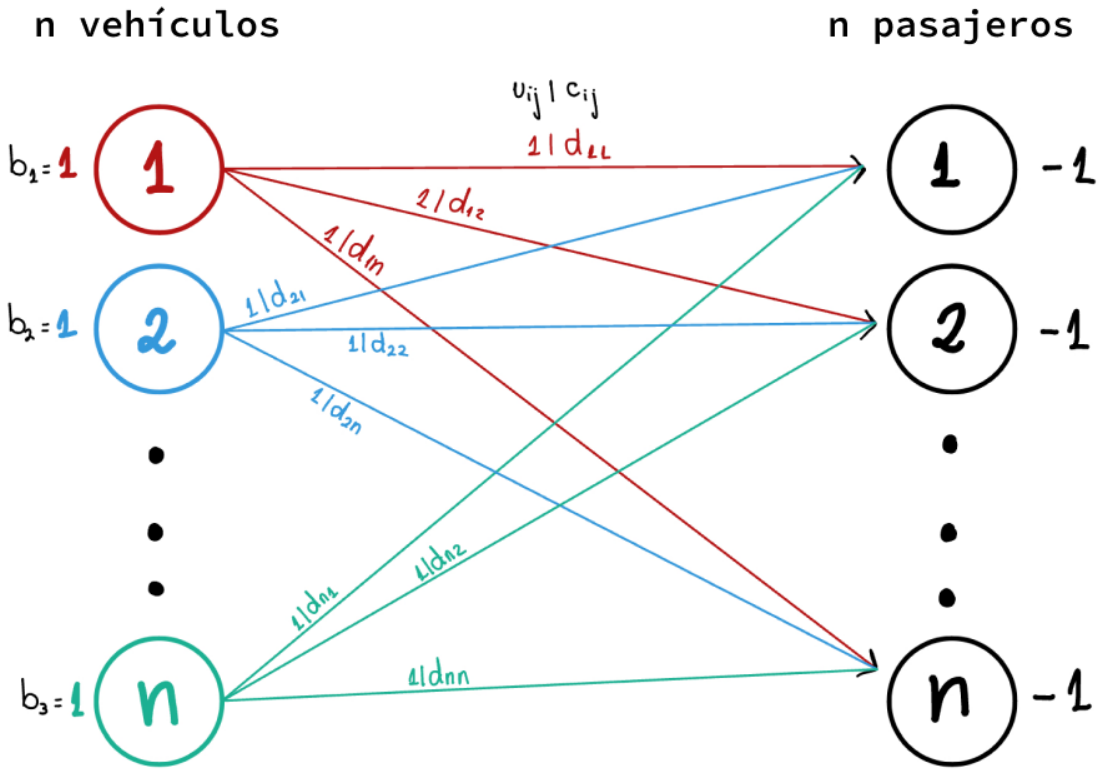
Modelamos el problema de asignación con un grafo $G = (N, A)$ de n^2 nodos, donde n es la cantidad de conductores disponibles (y de pasajeros que buscan viajar, ya que asumimos que la cantidad de ambos es simétrica).

Planteamos un grafo bipartito dirigido, donde los nodos de la partición V_1 representan a los taxis, y los de la partición V_2, a los pasajeros. Para todo taxi i existe un arco $ij \in A$ con un peso asignado según la función $d : A \to \mathbb{R}$. Los pesos o costos de los arcos son la distancia (medida en kilómetros) que debe recorrer el taxi i hasta su pasajero asignado j.

Con este modelo, queremos encontrar el flujo desde los taxis hacia los pasajeros que minimice la suma total de los costos (es decir, de la distancia total a recorrer). En otras palabras, estamos planteando un problema de flujo de costo mínimo. Para este planteo nos falta considerar los imbalances b_i y las capacidades u_{ij}:

- Los imbalances b_i de cada nodo $i \in N$ representan la cantidad de flujo que genera o absorbe cada nodo. Consideramos $b_i = 1$ para cada taxi porque "generan" una unidad de flujo cada uno, y $b_i = -1$ para los pasajeros porque reciben el flujo generado por los taxis. Al haber n taxis y n pasajeros nos aseguramos que la suma de imbalances de 0.
- En cuanto a la capacidad u_{ij} de cada arco ij, cualquier capacidad mayor a uno nos sirve, ya que debe permitir que pase todo el flujo generado por los taxis. Por conveniencia y simplicidad, elegimos utilizar $u_{ij} = 1$ para todos los arcos.

El modelo de flujo de costo mínimo no toma decisiones *golosas* como el modelo anterior, sino que explora las distintas soluciones posibles hasta encontrar aquella que minimiza el costo. En este caso, el costo es la distancia recorrida desde los taxis hasta los pasajeros, por lo que esta minimización nos devuelve nuestra función objetivo.



Consideraciones: podríamos también incluir un nodo s fuente, y un nodo de destino t. Con este cambio deberíamos fijar el imbalance de s en n, el de t en −n, y el del resto de los nodos en 0, ya que ahora no son los taxis los que generan el flujo, ni los pasajeros los que lo reciben. Los costos de los arcos si y jt serían 0 (siendo i los nodos de los taxis y j los de los pasajeros), mientras que los costos del resto de los arcos se mantendrían. La capacidad de si y jt deben ser 1, para que no se envíe más de una unidad de flujo a cada taxi, ni más de una desde los pasajeros. Decidimos no tomar esta implementación porque con los imbalances de los taxis y pasajeros no nos pareció necesario para representar apropiadamente el problema.

Consideraciones generales respecto a la implementación del modelo, incluyendo dificultades que encontramos

Para implementar el modelo utilizamos la librería `graph` del software OR-Tools, específicamente el archivo de `min cost flow.h`.

Para ello debimos computar nuestros nodos junto a sus imbalances, y los arcos junto a sus costos y capacidades.

A pesar de tener n taxis numerados de la misma manera que los n pasajeros, para que la función anduviera correctamente fue necesario cambiar el número de los pasajeros, para que no se confundieran con los taxis (podríamos haber cambiado los taxis, era indistinto). Esto lo hicimos sumándole n a todos los nodos para crear el vector de nodos pasajeros `end nodes`. Luego era importante recordar restarle ese n para poder realizar la asignación.

También debimos multiplicar los costos por 10, ya que el vector que los almacena `unit cost` es de tipo `vector<int64 t>`, y nuestras distancias estaban en formato `double` redondeados a un decimal. Luego, una vez computada la función objetivo, volvimos a dividirla por 10 para obtener el valor real. Con este cambio, nuestra función de costos pasa a ser $d : A \to \mathbb{Z}$.

Como resultado inicial observamos una reducción en el valor de la función objetivo con el archivo `small 1`, de 42.4 a 32.4. También observamos un aumento en el tiempo de ejecución (como este es variable no incluimos los valores exactos en este informe)

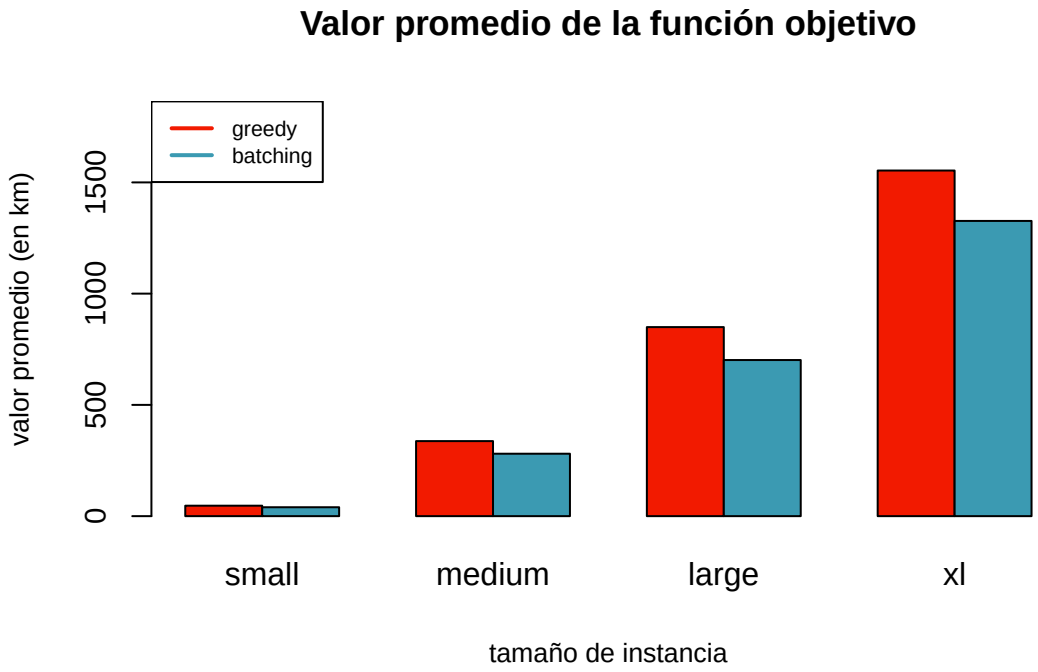
Resumen de resultados obtenidos en la experimentación

Durante la experimentación consideramos tres métricas de éxito:

- La minimización de la función objetivo (distancia recorrida hasta los pasajeros)
- La minimización del tiempo que se tarda en obtener la función objetivo
- La minimización de la distancia máxima a recorrer por un taxi de la asignación

Minimización de la función objetivo

Como esperábamos, en promedio el método propuesto logra valores menores para la función objetivo: tiene una mejora relativa del 15% con respecto al *greedy*, y una mejora absoluta (representada en el gráfico) de 109.5 km de diferencia para la función objetivo (es decir que en promedio, si realizamos la asignación con este método necesitamos recorrer 109.5 km menos para poder recorrer a todos los pasajeros). Esto se debe a que, como mencionamos antes, el método FCFS es una heurística que elige la mejor opción posible a cada paso, pero no tiene una visión global de las decisiones posibles, cómo si lo tiene el modelo de flujo de costo mínimo.



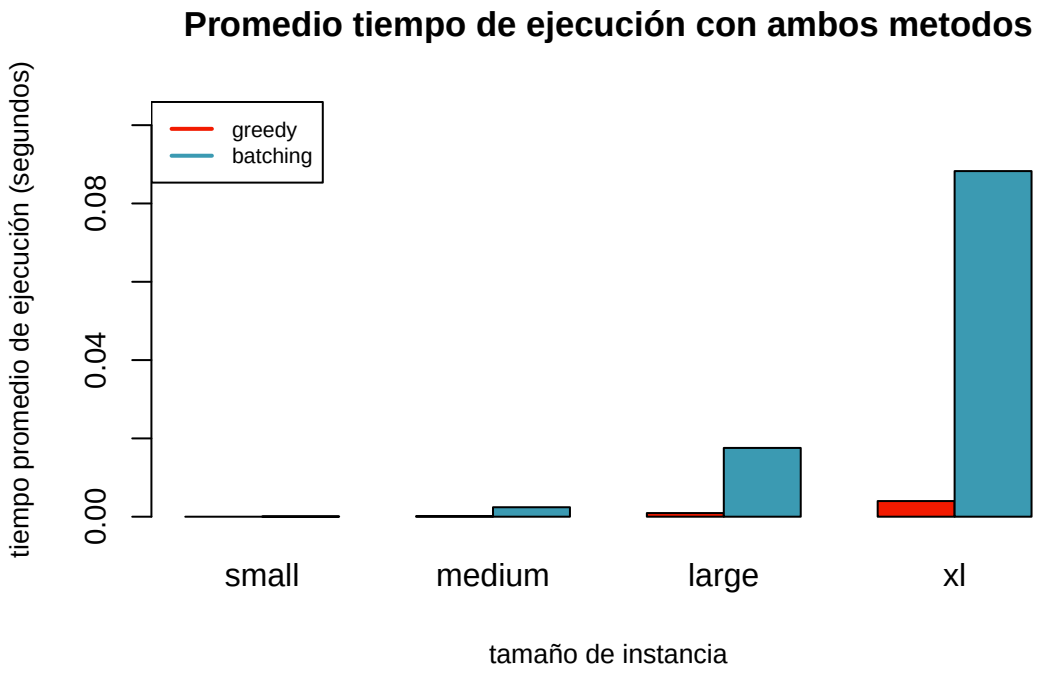
Sin embargo, vemos que cuando se tienen pocos vehículos/pasajeros (como sucede en los archivos *small* que tienen $n = 10$) la diferencia no es muy significativa. Es decir, es una mejora relativa notoria y similar a otros tamaños de instancia, pero en términos prácticos la distancia a recorrer con el segundo modelo es apenas unos kilómetros menor que con el modelo FCFS cuando hay pocos taxis y pasajeros.

A continuación se puede comparar la evolución de la mejora relativa y absoluta para el valor de la función objetivo.

	mejora_relativa	mejora_absoluta
small	14.7	6.9
medium	16.9	56.7
large	17.3	148.0
xl	14.6	226.5
promedio	15.9	109.5

Minimización del tiempo de cálculo

El `GreedySolver` es notoriamente más rápido que el `BatchingSolver` para computar el valor de la función objetivo, lo cuál es lógico si recordamos que el primero solo considera la mejor solución posible *en el momento*, mientras que el segundo explora todas antes de seleccionar la mejor solución *global*.



En promedio, el método basado en el flujo de costo mínimo es 2235% peor (o más lento) que el método FCFS (su mejora relativa es negativa).

Sin embargo, incluso con los *batchings* más grandes que buscan asignar 1500 taxis a 1500 pasajeros, el tiempo promedio en nuestra experimentación fue de 120000 microsegundos, o 0.12 segundos. A efectos prácticos, esta diferencia es irrelevante para los usuarios (siempre asumiento que la construcción de la instancia no demore demasiado, es decir, que no haya que esperar a que se conecten más conductores o pasajeros)

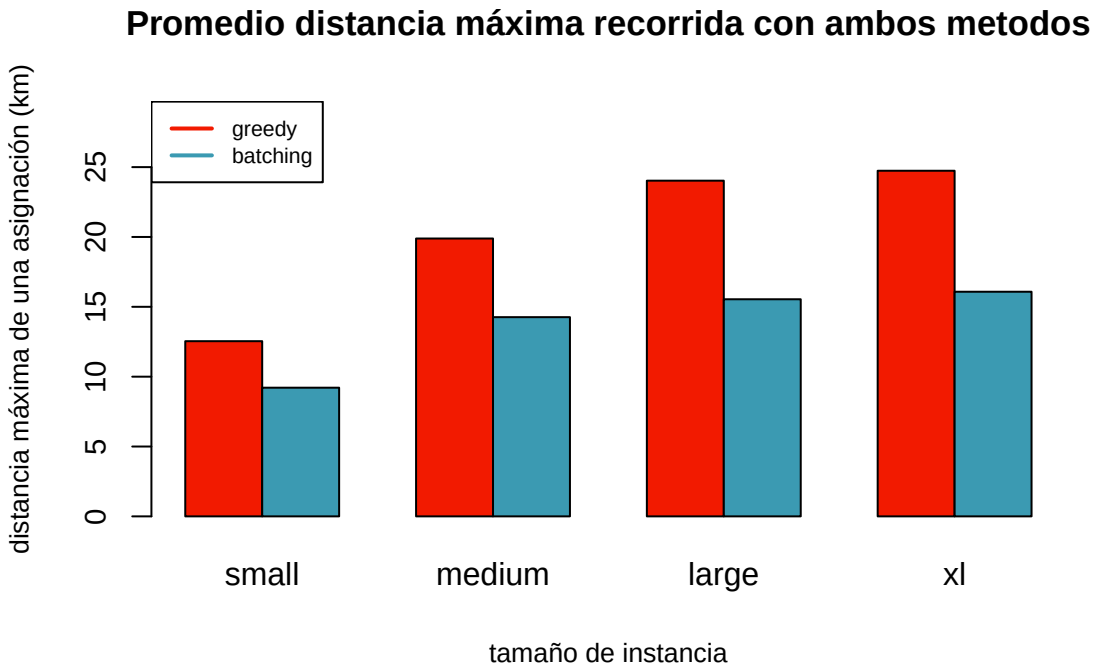
Por ende, concluimos que a pesar de que el modelo de flujo de costo mínimo empeora mucho (absoluta y relativamente) en comparación al modelo FCFS, no es una diferencia relevante en la vida real.

	mejora_relativa	mejora_absoluta
small	-3318.3	-71.2
medium	-1714.8	-2280.9
large	-1793.0	-16629.9
xl	-2114.4	-84274.5
promedio	-2235.1	-25814.1

Minimización de la distancia máxima

Teníamos la hipótesis de que el modelo propuesto, a pesar de minimizar la distancia total a recorrer por los conductores hasta sus pasajeros asignados, podría derivar en que para uno o algunos taxis, la distancia fuera significativamente mayor que con el modelo actualmente en uso.

Por eso computamos como tercer métrica a la mayor distancia recorrrida por un taxi de la asignación hasta su pasajero con ambos métodos. Sin embargo, como se ve en el gráfico, ocurrió contrario.



Como se ve en el gráfico, en promedio para todos los tamaños de instancia el *greedy* resultó en asignaciones con una mayor distancia máxima, aunque resulta difícil decir que la diferencia promedio de 6.5 km es muy

relevante a efectos prácticos.

Veamos la mejora relativa y absoluta del batching en relación a esta métrica:

	mejora_relativa	mejora_absoluta
small	22.4	3.3
medium	26.8	5.6
large	35.2	8.5
xl	34.9	8.7
promedio	29.9	6.5

Conclusiones, posibles mejoras y observaciones adicionales que consideren pertinentes

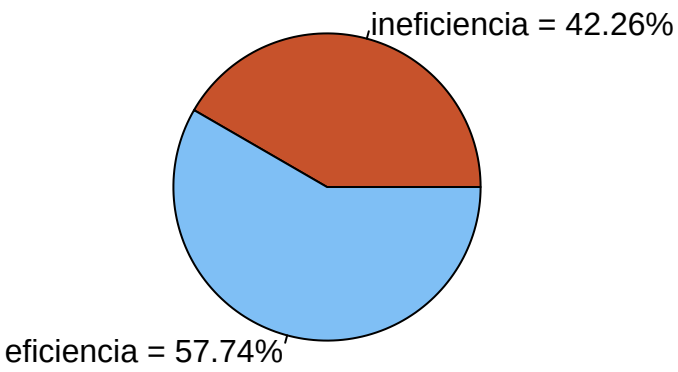
En base a las métricas evaluadas, concluimos que si buscamos minimizar la distancia total, el segundo método es decididamente mejor, en especial si tenemos una cantidad de vehículos y pasajeros más grande. Por extensión, también es mejor opción si se quiere minimizar la distancia máxima recorrida por un conductor de la asignación.

En cuanto al tiempo, si bien dijimos que la diferencia es notable en términos puramente numéricos, no creemos que sea relevante a la hora de elegir qué método emplear, ya que no es una diferencia que vaya a notar el usuario.

Otras consideraciones: 'pérdida de tiempo' por viajes cortos

Nuestro modelo propuesto cuenta con ciertas limitaciones. En particular, que muchas veces los conductores deben recorrer distancias muy largas hasta sus pasajeros asignados para luego realizar un viaje muy corto. La sensación es que estos viajes son muy costos (en tiempo o en costo específico). Para analizar si esta queja era válida, analizamos cuántos viajes ineficientes se realizan en promedio con nuestro modelo de *batching*.

Consideramos que un viaje es 'eficiente' si la distancia del viaje es *al menos* tan larga como la distancia que el conductor tiene que recorrer para recoger a su pasajero, es decir, si $d_{ij} \leq d_j$. O lo que es lo mismo, si el ratio $\frac{d_{ij}}{d_j} \leq 1$. ($d_{ij}$ es la distancia desde el taxi i hasta el pasajero j, y d_j es la distancia del viaje en sí mismo)



Como evidencia el gráfico, con el modelo actual hay una cantidad muy elevada de viajes que no resultan eficientes. También notamos que el tamaño del *batching* no es tan relevante, ya que para cada tamaño de instancia n evaluadas el porcentaje de viajes ineficientes (ratios mayores a 1) son $\%_{10} = 57$, $\%_{100} = 37.5$, $\%_{250} = 37.37$ y $\%_{1500} = 34.74$. El cambio más notorio se da cuando agrandamos la entrada de 10 a 100 vehículos/pasajeros, pero de ahí en más el porcentaje sigue siendo casi igual de alto.

Planteamos entonces un nuevo modelo que pueda resolver estas limitaciones, tratando de mantener la minimización de la función objetivo. Como seguimos pensando dentro de un modelo de flujo de costo mínimo, queremos que los viajes más eficientes tengan menor costo, es decir que tengan un menor ratio. Para esto proponemos un nuevo modelo que tiene en cuenta el costo que significa para un taxista recorrer muchos kilómetros hasta su pasajero, para luego hacer un viaje excesivamente corto en comparación.

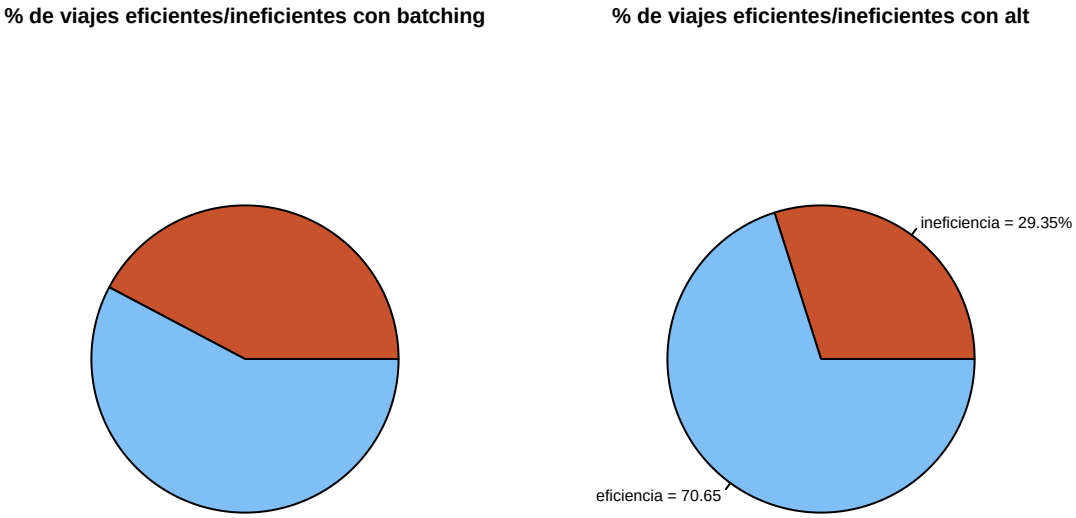
Definimos un nuevo costo(*) $c_{ij} = \frac{d_{ij}}{d_j}$. De esta manera tratamos de minimizar d_{ij} para que los pasajeros no tengan que esperar tanto por sus taxis (a nivel global), y a la vez intentar que los taxistas realicen viajes más eficientes.

Creamos un nuevo método de la clase `BatchingSolver`, que lo único diferente que tiene son los nuevos costos (los ratios), que debemos multiplicar por un múltiplo de 10 elevado, para perder la menor cantidad de decimales posibles en su redondeo a `int64 t` (elegimos 1000).

Luego, para calcular el valor de la función objetivo podemos encontrar el d_{ij} de cada par de taxi, pasajero asignado como solución (la solución la encontramos con la misma lógica del flujo de costo mínimo, con la misma herramienta de OR-Tools).

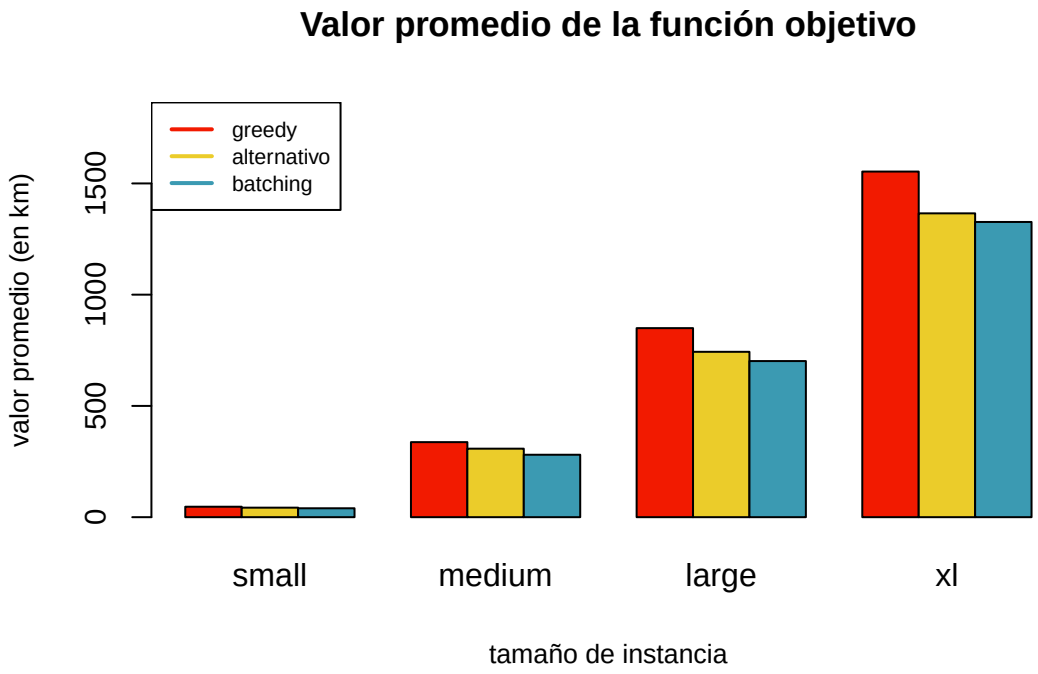
Para el archivo `small 1` el nuevo modelo arrojó un valor para la función objetivo de 33, que se encuentra entre el valor del método *greedy* y *batching*. Esto era esperable ya que el modelo nuevo sigue tomando una decisión global, por lo que debería dar mejor resultado que el *greedy*. Pero también al cambiar la función de costos, reemplazando $\frac{d_{ij}}{d_j}$, ya no se minimiza la función objetivo.

Luego de la experimentación encontramos que el nuevo modelo alternativo logra reducir la cantidad de viajes ineficientes de manera significativa en la mayoría de las instancias, cuando el n es lo suficientemente grande. Los nuevos porcentajes de ineficiencia son $\%_{10} = 56$, $\%_{100} = 26.73$, $\%_{250} = 20.04$ y $\%_{1500} = 16.84$. El nuevo porcentaje de viajes eficientes e ineficientes queda así:



A cambio de esta mejor en la eficiencia obtenemos un valor de la función objetivo (una distnacia total) más grande que para el *batching*, pero aún así es una alternativa decididamente mejor que el método *greedy*. Además tiene el plus agregado de que no solo logra minimizar la distancia total, sino que también puede lograr que los taxistas realicen viajes más eficientes.

Era de esperarse que el método alternativo propuesto fuera peor que el *batching* porque ya no busca minimizar d_{ij}, sino d_j, es decir que lo que minimizamos difiere de nuestra función objetivo. Sin embargo, también es lógico que no resulte peor que el *greedy* porque sigue siendo un algoritmo exacto de minimización en vez de una heurística, y el ratio minimizado guarda una relación con lo que se minimiza en la función objetivo.



Consideraciones: otra posibilidad que barajamos fue tener en cuenta el precio de cada viaje, y cuál era la relación de beneficio para los viajes según cuanto demandaba llegar hasta el pasajero. Al final decidimos no hacerlo porque nos pareció que también se podía medir de la forma propuesta, considerando más la 'pérdida de tiempo' que de dinero por parte del conductor. Además, no sería tan sencillo definir que constituye un viaje 'eficiente', ya que deberíamos definir primero cuál es un precio aceptable por kilómetro.

(*) Muchos archivos tenían valores de d_j en 0 o muy cercanos, lo que resultaba en un ratio infinito. De esta manera nos era imposible calcular la ineficiencia del modelo, por lo que decidimos que cuando eso sucediera, el valor de d_j pasaría a ser igual a d_{ij}, de manera que el ratio fuera 1, es decir, no lo consideramos ni eficiente ni ineficiente, y buscamos alterar los resultados lo menos posible. Esta modificación también tiene sentido si pensamos que lo que era ilógico es que d_j, la distancia del viaje, sea cero.