## TP2: Logística centralizada de primera milla

### Introducción al problema

La problemática a abordar se trata de encontrar la manera óptima, o en su defecto efectiva, de realizar una asignación. Trabajaremos sobre un caso particular en el cual buscamos asignar un conjunto de negocios a un conjunto de depósitos. Cada negocio posee una distancia a cada depósito, denotada por , con y . También posee una demanda, que se refiere a la cantidad de stock que el negocio pedirá de dicho depósito. Por último, cada depósito posee un capacidad, denotada por . En el este caso particular, los negocios no pueden pedir stock de distintos depósitos, por lo cual serán asignados tan solo a uno de estos. Asimismo, la demanda de cada negocio será la misma para todo depósito, por lo cual la variable no depende de y podría escribirse como . Consideramos una solución óptima al problema de asignación a aquella que minimiza la suma de las distancias entre cada negocio y cada depósito asignados, teniendo en cuenta no exceder la capacidad de estos últimos.

El tipo de problema con el cual trataremos es conocido como *GAP* (Generalized Assignment Problem), y su versión de optimización se encuentra en la categoría NP-Hard. Esto quiere decir que todo problema perteneciente a la clase NP puede ser reducido polinomialmente a *GAP*. Además, un problema NP-Hard no necesariamente pertenece a NP, es decir que no necesariamente existe un algoritmo polinomial para verificar un certificado positivo de una instancia del problema. Si encontraramos un algoritmo polinomial para *GAP* entonces encontraríamos una solución polinomial para todo problema en NP y demostraríamos que P = NP. De todos modo, eso aún no ha sucedido y es por eso que se desarrollan heurísticas para resolver el problema *GAP,* buscando un balance entre la complejidad temporal y la calidad de la solución.

En el siguiente trabajo buscamos modelar distintas alternativas de heurísticas y búsquedas locales en pos de hallar una solución de cierta calidad para la problemática GAP, en tiempo polinomial. Comenzaremos diseñando heurísticas constructivas, luego operadores de búsqueda local y por último generaremos una metaheurística que los englobe.

### Diseño de heurísticas constructivas

#### Heurística 1: depósito más cercano

En primer lugar diseñamos e implementamos una heurística cuyo funcionamiento es el siguiente: por cada negocio buscamos el depósito más cercano, es decir aquel cuyo sea el menor, y asignamos el negocio a dicho depósito. Asimismo, vamos disminuyendo la capacidad del depósito en función a la demanda de la asignación. En caso de que el depósito de menor distancia a un negocio no tenga capacidad restante suficiente, este sería asignado al próximo depósito más cercano.

Esta heurística es posiblemente la más intuitiva, y tiene ciertas ventajas como no necesitar actualizar una lista de disponibilidad, ya que los depósitos pueden tener tantos negocios asignados como su capacidad permita. En otros enfoques, es necesario llevar la cuenta de qué negocios ya fueron asignados, pero en este caso al iterar por negocios y en cada iteración asignarlos a un depósito, no tenemos dicho problema.

La complejidad de este algoritmo goloso es siendo la cantidad de negocios y la cantidad de depósitos. Como vemos, es polinomial. Claramente no encontraremos la solución óptima, pero encontraremos una solución que creemos que tendrá cierto grado de efectividad.

#### Heurística 2: negocio más cercano

En segundo lugar desarollamos una heurística que funciona de manera inversa a la primera, es decir que itera por cada depósito y le asigna aquellos negocios que se encuentren cerca, mientras la capacidad del depósito lo permita. Es importante asegurarse que no se evalúen aquellos negocios que ya han sido asignados, por lo cual debemos llevar la cuenta de estos. Además, es necesario encontrar un modo de definir qué quiere decir que un negocio está cerca de un depósito. Nuestro criterio fue el siguiente: por cada depósito, ordenamos los negocios en función a cuán lejos estan de este, de menor a mayor. Así, iteramos por los depósitos asignandoles en orden los negocios mientras estos aún no esten asignados y mientras la capacidad del depóstio sea mayor a la demanda del negocio. A grandes rasgos, asignamos a cada depósito los negocios más cercanos a él.

La complejidad de este algoritmo goloso es de , polinomial pero no tan eficiente como la heurística anterior.

Analizando el funcionamiento de neustra heurística, llegamos a la conclusión de que el orden en que evaluamos los depósitos alterará el resultado. Luego, quizás podríamos definir un orden de manera que ayude a reducir aún más el valor de la función objetivo. El primer depósito tenido en cuenta tendrá prioridad sobre los otros y será probablemente el único que tenga todos los negocios más cercanos a él asignados. Imaginemos la situación en la cual existe un depósito considerablemente más alejado de los negocios que el resto. Si este depósito es considerado antes que el resto, se le asignarán múltiples negocios que a pesar de estar más cerca del depósito que otros negocios, claramente estarán más cerca de los otros depósitos que de aquel en cuestión. Así, estaríamos dando mucho peso a un depósito al cual deberíamos tratar de asignarle la menor cantidad de negocios posibles. Luego, podría ser útil ordenar los depósitos según distancia promedio a los negocios y comenzar por aquellos con menor promedio. Así, evitarímos situaciones como la descripta anteriormente. Aquí partimos de la base de que la suma de las capacidades de los depósitos sea mayor a la suma de las demandas de los negocios, por lo cual preferimos que los depósitos “más vacíos” sean aquellos más lejanos. Si la situación fuera diferente entonces priorizar por cercanía los depósitos quizás no se necesario.

#### Consideraciones

Se pueden diseñar heurísticas que tomen un enfoque reacionado a las demandas que tiene cada negocio en relación a cada depósito, buscando quizás asignar negocios a depósitos de los cuales ocupen menor capacidad si es que dichos depósitos son cercanos a otros negocios y buscamos asignarlos a la mayor cantidad posible de negocios. Pueden surgir muchos matices considerando las demandas y capacidades. De todos modos, a pesar de que en el problema general de GAP las demandas de los negocios pueden variar dependiendo del depósito, como en nuestra instancia real todo negocio demanda lo mismo de cualquier depósito, no profundizaremos en estas alternativas. A pesar de estar diseñando una heurística para GAP en general, nuestro objetivo final se refiere en específico a la instancia particular de *ThunderPack*.

### Operadores de búsqueda local

#### ¿Cómo representamos una solución?

#### Operador 1: relocate

El primer operador que definimos es un relocate. Este itera por los negocios evaluando si hay algún depósito con menor distancia al negocio que aquel al cual se encuentra asignado actualmente, y en caso afirmativo, si las capacidades lo permiten, asigna el negocio a su nuevo depósito. Definimos como vecindario a todas las soluciones alcanzables desde mi solución actual haciendo un único relocate. Utilizamos como criterio de aceptación el *first improvement*, y por lo tanto apenas hallamos una solución mejor a la actual redifinimos nuestra solución como la nueva y continuamos explorando su nuevo vecindario.

Para aceptar una nueva solución debemos evaluar 2 condiciones, con el depósito anterior, el nuevo depósito y el negocio:

* , siendo la capacidad restante del depósito

Este operador no aportará una mejora con toda heurística posible como base para la solución inicial. Por ejemplo, si tomamos nuestra primer heurística, como cada negocio está asignado al depósito más cercano, nunca encontraremos otro depósito que sea más preferible y tenga capacidad disponible, sin realizar antes algún otra modificación sobre la soluición como un swap, por ejemplo. En cambio, en nuestra segunda heurística sí ofrecería mejoras desde el incio, ya que que cada depósito tenga asignado los negocios más cercanos, nada asegura que cada negocio este asignado a su depósito más cercano. Consecuentemente, en dicho caso el operador relocate sería útil.

El operador relocate nos permite asignar los negocios que no era posible asignar a ningún depósito dadas las capacidades y las asignaciones ya realizadas. Se sufría una gran penalización sobre el valor objetivo por cada negocio no asignado, por lo cual el hecho de poder incluirlos en la solución mejora considerablemente el valor objetivo. Por supuesto, no se podrán ubicar dichos negocios sin antes haberse realizados otros relocate que generen suficiente espacio en algún depósito.

La complejidad del opeador relocate es de siendo la cantidad de negocios y la cantidad de depósitos. Notemos que la búsqueda corre hasta haber analizado todas las posibles combinaciones de negocios y depósitos, realizando las alteraciones de la solución simultánemante.

#### Operador 2: swap

El segundo operador que diseñamos es un swap. Este itera por los negocios, en busca de dos negocios que al intercambiar el depósito en el que se encuentran el valor objetivo disminuya, respetando las capacidades. Utilizamos de nuevo el criterio de *first improvement*. Este operador no servirá para añadir los negocios no asignados a la solución, ya que swapearlos simplemente añadiría un negocio para quitar otro.

Para aceptar una nueva solución debemos evaluar 2 condiciones, con el depósito A, el depósito B depósito, el negocio A (asignado al depósito A) y el negocio B (asignado al depósito B):

Con la capacidad restante del depósito .

Pareciera que este operador funcionará con ambas heurísticas diseñadas ya que siempre puede suceder que las suma de las nuevas distancias sea menor que la suma de las anteriores.

La complejidad del operador swap es de siendo la cantidad de negocios, ya que la búsqueda evalúa cada par de negocios posible en busca de una mejora al valor objetivo.

¿Qué podría hacer a una solución infactible? En primer lugar que la suma de las demandas de los negocios asignados a cierto depósito superen su capacidad. En segundo lugar, que un negocio sea asignado a más de un depósito. Ambos operadores evalúan que las capacidades de los depósitos no sean excedidos. Asimismo, no permiten que un negocio peretenezca a dos depósitos: el operador relocate quita al negocio del depósito anterior al relocalizarlo y el operador swap quita a ambos negocios de sus anteriores depósitos. Luego, ambos son correctos.

### Metaheurística

Como metaheurística diseñamos una variación de VND (variable neighbourhood descent) utilizando ambos vecindarios, swap y operate. Tiene ciertas diferencias con el concepto de VND, ya que itera dentro de cada operador hasta alcanzar un mínimo local.

Definimos a como el operador relocate y a como el operador swap. Comenzamos investigando el vecindario , buscando una solución mejor y pasando al vecindario de dicha solución hasta llegar a un mínimo local. Luego, comenzamos a investigar con el vecindario del mismo modo. Cuando llegamos a un mínimo local, retornamos a analizar el vecindario . Alteramos entre ambos vecindarios de este modo hasta que ninguno ofrece una mejora a la solución, es decir, nos encontramos en una solución que es un mínimo local para ambos. En dicho caso, retornamos la solución como el mínimo obtenido por la metaheurística.

¿Cuál es la calidad de la solución? Nada nos asegura que la solución hallada por nuestra metaheurística sea efectivamente la solución óptima. Es muy probable que no lo sea. De todos modos, a partir de experimentación que explicaremos a continuación, notamos que cuánto realmente mejora la solución inicial depende en un alto grado de la calidad de la heurística llevada a cabo.

Idealmente, podríamos resolver una vez el problema con un algoritmo exacto, obteniendo el óptimo, y así podríamos determinar con mayor seguridad la calidad de nuestras soluciones aproximadas. Por otro lado, la solución inicial, y en mayor escala la heurística, genera un gran impacto en la metaheurística. Puede suceder que una heurística que genera una “peor” solución inicial nos de un mayor espacio a mejoras. Puede suceder que nuestra heurística sea muy buena y por lo tanto no podamos mejorar. Las posibilidades son múltiples, y dependen de variados factores, como también la naturaleza de la instancia que evaluamos.

La complejidad de la metaheurística depende de cuantas iteraciones se realicen de esta, lo cual no es posible definir de antemano. Los operadores relocate y swap que utiliza son de complejidad y respectivamente por lo cual por supuesto la complejidad de la metaheurística es al menos de . La complejidad depende de la cantidad de iteraciones, es decir de la cantidad de soluciones mejores hacia las cuales nos movemos, por lo cual podríamos decir que depende de cuán cerca está nuestra solución inicial de un mínimo local para ambos operadores, lo cual depende de nuevo de la heurística utilizada para obtener dicha solución inicial.

### Experimentación y discusión

Para evaluar el funcionamiento y calidad de nuestras heurísticas, búsquedas locales y por úlitmo metaheurística, utilizamos tres conjuntos de bases de datos.