Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Mejoras a un algoritmo de Branch and Price para el problema del viajante de comercio con un dron

Tesis de Licenciatura en Ciencias de la Computación

Marcos Blufstein

Director: Dr. Francisco Soulignac
Codirector: Lic. Gonzalo Lera Romero
Buenos Aires, 2023

MEJORAS A UN ALGORITMO DE BRANCH AND PRICE PARA EL PROBLEMA DEL VIAJANTE DE COMERCIO CON UN DRON

En este trabajo estudiamos el problema del viajante de comercio con un dron. En este problema, un camión y un dron se mueven en simultáneo para visitar a todos los clientes de un conjunto una única vez, ya sea únicamente por el camión, únicamente por el dron, o por ambos vehículos al mismo tiempo. El dron tiene una capacidad limitada; solo puede llevar de a un paquete a la vez, después de lo cual debe regresar al camión para buscar otro paquete. Por otro lado, el dron tiene la ventaja de poder evitar la red de tráfico, lo que le permite moverse de un cliente a otro en línea recta.

Analizamos un algoritmo de Branch and Price para este problema propuesto por Roberti and Ruthmair en el trabajo "Exact Methods for the Traveling Salesman Problem with Drone" (Roberti and Ruthmair, 2021), el cual reimplementamos en su totalidad dado que el código fuente no está disponible. Además, proponemos e implementamos mejoras al mismo, entre las cuales se encuentran una nueva relajación para el TSPD, nuevas reglas de dominación parcial, y una versión bidireccional del algoritmo. Estos aportes prueban ser eficaces en los experimentos computacionales realizados, en donde observamos que nuestro algoritmo obtiene resultados mejores que los encontrados en la literatura.

El algoritmo fue testeado sobre las mismas instancias que utilizan en el trabajo Roberti and Ruthmair (2021), introducidas anteriormente por Poikonen et al. (2019) (ver Capítulo 6). Las mismas llegan a un tamaño máximo de 39 clientes, y mientras que el algoritmo de Roberti y Ruthmair llega a resolver únicamente 11/75 instancias de dicho tamaño, nuestro algoritmo resuelve 71/75.

Palabras claves: Problema del viajante de comercio con un dron, generación de columnas, algoritmos de labeling, programación dinámica, branch and price.

IMRPOVEMENTS FOR A BRANCH AN PRICE ALGORITHM FOR THE TRAVELING SALESMAN PROBLEM WITH A DRONE

In this work we study the traveling salesman problem with a drone. In this problem, a truck and a drone move simultaneously to visit each client once, either by the truck alone, by the drone alone, or by both vehicles at the same time. The drone has a limited capacity; it can only deliver one package at a time, after which it must return to the truck to pick up another package. On the other hand, the drone has the advantage of being able to avoid traffic, which allows it to move from one client to another in a straight line.

We analyze a Branch and Price algorithm proposed by Roberti and Ruthmair in their work "Exact Methods for the Traveling Salesman Problem with Drone" (Roberti and Ruthmair, 2021), which we reimplement entirely since the source code is not available. Moreover, we propose and implement improvements to this algorithm, which include a new relaxation for the TSPD, new partial dominance rules, and a bidirectional version of the algorithm. This improvements prove to be beneficial in the computational experiments we conducted, where we observe that our algorithm gets better results than thouse found in the literature.

We tested the algorithm with the same test cases that where used in Roberti and Ruthmair (2021), introduced previously by Poikonen et al. (2019) (see Chapter 6). This test cases reach a maximum of 39 clients, and Roberti and Ruthmair's algorithm manages to solve 11/75 test cases of said size, while our algorithm solves 71/75.

Keywords: Traveling salesman problem with a drone, column generation, labeling algorithms, dynamic programming, branch and price.

AGRADECIMIENTOS

Gracias mamá, gracias papá, gracias Martín.
Gracias Francisco por toda tu ayuda en este trabajo.
Gracias de nuevo Francisco, Gonzalo, Juan José por toda su ayuda dentro y fuera de la tesis.
Gracias Javier, gracias Brian por ser jurados de esta tesis.
Gracias a todos mis amigos de cursada y compañeros de TPs.
Gracias Prim Floyd.

Índice general

1. Introducción 1
   1.1 Problemas de ruteo 1
   1.2 El problema del viajante de comercio con dron 1
   1.3 Estructura de la tesis 3

2. Preliminares 4
   2.1 Programación Lineal (LP) 4
   2.2 Programación Lineal Entera Mixta (MILP) 5
   2.3 Branch & Bound 7
   2.4 Generación de Columnas 9
   2.5 Branch & Price 10

3. Revisión de la literatura 11

4. Algoritmo de Roberti y Ruthmair 14
   4.1 Algoritmos de Labeling para el TSPD 14
   4.2 Relajación de rutas ng 16
   4.3 Reglas de dominación 20
   4.4 Formulación de set partitioning 21
   4.5 Reglas de branching 22

5. Mejoras y aportes 24
   5.1 Nueva definición de ng-factibilidad 24
   5.2 Reglas de dominación 26
   5.3 Labeling bidireccional 32

6. Experimentos computacionales 42
   6.1 Casos de test 42
   6.2 Resultados para el algoritmo propuesto 43
   6.3 Comparación con algoritmo de Roberti y Ruthmair 45

7. conclusiones y trabajo a futuro 48

Anexo 51
   7.1 Resultados completos para instancias de tamaño 10 51
   7.2 Resultados completos para instancias de tamaño 20 53
   7.3 Resultados completos para instancias de tamaño 30 54
   7.4 Resultados completos para instancias de tamaño 40 56
   7.5 Rutas óptimas por instancia 58

1. INTRODUCCIÓN

1.1 Problemas de ruteo

Los problemas de ruteo son problemas de optimización combinatoria cuyo objetivo es encontrar un conjunto de rutas óptimo para que una flota de vehículos visite a todos los clientes. Por lo general, las rutas deben comenzar y terminar en un depósito (puede haber más de uno) y deben satisfacer ciertas restricciones. Por ejemplo, cada vehículo puede tener una capacidad limitada, o puede ser necesario que cada cliente sea visitado en un cierto horario. El problema de ruteo más famoso es el problema del viajante de comercio, conocido como TSP (*traveling salesman problem*). En este problema, un único vehículo debe visitar a todos los clientes, empezando y terminando su recorrido en el mismo depósito. Hay distintas métricas para determinar qué tan buena es una ruta, donde algunas de las más usuales son elegir la ruta que minimice la distancia o el tiempo total de la ruta.

En general, los problemas de ruteo pertenecen a la clase de complejidad NP-hard, por lo que el tamaño de las intancias que pueden ser resueltas de manera exacta es limitado. Sin embargo, en el caso del TSP se han desarrollado numerosas técnicas y heurísticas que permiten resolver el problema en casos reales. A modo de ejemplo, el algoritmo Concorde (Applegate et al., 2006) es capaz de resolver instancias del TSP de más de mil clientes en cuestión de segundos, aunque se han encontrado casos de tan solo 200 clientes que Concorde no puede resolver (Hougardy and Zhong, 2021). Hoy en día, estos problemas de ruteo son utilizados, entre otras tantas cuestiones, para modelar los problemas de distribución de mercadería.

En los últimos años, los problemas de logística y distribución de mercadería se volvieron de gran importancia para muchas compañías, por lo que han surgido una gran cantidad de variantes de este problema. En particular, se han buscado distintos medios de transporte que permitan facilitar el proceso de envíos, y entre ellos se encuentran los drones. La principal ventaja de los drones se encuentra en su velocidad y el hecho de no estar limitados por la red de tráfico. Además, los drones proponen una alternativa más ecológica por reducir el uso de combustibles y la polución sonora. Sin embargo, entre las desventajas se encuentran su baja capacidad y su limitado rango de vuelo. Estas desventajas pueden ser reducidas si agrupamos a los drones con algún medio de transporte con mayor volumen, como puede ser un camión. De esta manera es que surge el problema del viajante de comercio con dron.

1.2 El problema del viajante de comercio con dron

En esta tesis nos vamos a concentrar en el problema del viajante de comercio con un dron (TSPD) que especificamos formalmente en esta sección. Definimos el problema sobre un digrafo completo $G = (V, A)$. El conjunto de vértices V se define como $V = \{0, 0'\} \cup N$, donde tanto 0 como $0'$ representan al mismo depósito al inicio y al final de la ruta respectivamente, y $N = \{1, \ldots, n\}$ es el conjunto de los n clientes que deben ser visitados.

Usamos también la notación $N_0 = N \cup \{0\}$. Por otro lado, el conjunto de aristas es definido como $A = \{(0, j) \mid j \in N\} \cup \{(i, j) \mid i, j \in N : i \neq j\} \cup \{(i, 0') \mid i \in N\}$.

Un camión equipado con un dron comienza su recorrido en el depósito 0. Entre ambos deben visitar a todos los clientes en N y volver al depósito $0'$. Para cada arco $(i, j) \in A$ tenemos dos costos t_{ij}^T y t_{ij}^D que representan lo que le cuesta al camión y al dron recorrer esa arista respectivamente. Cada cliente puede ser visitado solamente por el camión, solamente por el dron, o por ambos al mismo tiempo. El dron tiene una capacidad limitada; solamente puede visitar a un cliente antes de volver al camión para poder recoger otro paquete. Además, el dron puede desprenderse y unirse del camión únicamente en los vértices del grafo (es decir, en las ubicaciones de los clientes y del depósito), y no puede desprenderse y luego unirse en el mismo vértice. El objetivo del TSPD es encontrar una ruta de costo mínimo que visite a todos los clientes (ya sea con el camión o con el dron).

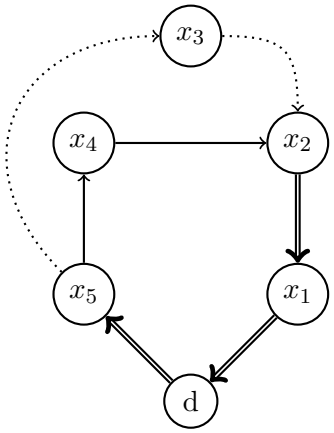

Figura. 1.1: Una solución factible de una instancia con 5 clientes. Aquí, d representa al depósito y los x_i a los clientes. Las flechas dobles representan arcos combinados (ambos vehículos se mueven juntos), las flechas punteadas representan que el dron se mueve por su cuenta, y las flechas comunes que el camión se mueve por su cuenta.

En la Figura 1.1 podemos ver una solución factible de un TSPD con 5 clientes y el depósito. En el ejemplo, el camión y el dron se mueven juntos desde el depósito hasta el cliente x_5. Después, desde el cliente x_5, el camión avanza al cliente x_4 mientras que el dron visita al cliente x_3. Luego cada vehiculo avanza al cliente x_2, donde vuelven a juntarse. Por úlitmo, ambos vehículos se mueven juntos al cliente x_1 y luego vuelven al depósito.

Es importante notar que el dron y el camión no necesariamente llegan al mismo tiempo al cliente x_2, por lo que deben sincronizarse. En este trabajo vamos a considerar que el vehículo que llega primero al cliente de unión debe quedarse esperando al otro. En este caso, dependiendo de t_{54}^T, t_{42}^T, t_{53}^D, t_{32}^D o bien el camión se queda esperando al dron ($t_{54}^T + t_{42}^T \leq t_{53}^D + t_{32}^D$) o bien el dron espera al camión ($t_{54}^T + t_{42}^T > t_{53}^D + t_{32}^D$). Por lo tanto, el costo total de esta ruta es:

$$t_{d5}^T + \max\{t_{54}^T + t_{42}^T, t_{53}^D + t_{32}^D\} + t_{21}^T + t_{1d}^T$$

El objetivo de este trabajo es mejorar el algoritmo propuesto por Roberti and Ruthmair (2021) para el TSPD. En pocas palabras, este algoritmo modela el TSPD como un problema de programación lineal entera a través de una formulación de tipo set-partitioning

en el que el conjunto de rutas está relajado para incluir algunas rutas no factibles. Este modelo se resuelve con un algoritmo de tipo Branch and Price, donde el árbol de branching garantiza la factibilidad de la ruta final. El problema de pricing resultante consiste en la resolución de un TSPD relajado, en el sentido de que algunos clientes se pueden visitar más de una vez, otorgando un beneficio en cada visita. Para resolver el problema de pricing, Roberti y Ruthmair aplican un algoritmo de programación dinámica adaptado a este problema, el cual reimplementamos completamente dado que el código fuente no está disponible. En este trabajo nos enfocamos principalmente en este algoritmo de programación dinamica. Nuestra mejora consiste en una nueva relajación del conjunto de rutas no factibles que refleja apropiadamente la sincronización de ambos vehículos (Sección 5.1) y que brinda mejores cotas inferiores para el modelo de set-partitioning. Además, proponemos e implementamos un nuevo algoritmo para resolver el problema de pricing resultante, donde introducimos reglas de dominación parcial para reducir el espacio de búsqueda, reduciendo considerablemente el costo temporal del mismo (Sección 5.2). Por otra parte, implementamos una versión bidireccional del algoritmo de pricing (Sección 5.3). Finalmente, incorporamos nuestro algoritmo de pricing en una implementación propia del algoritmo de Branch and Price propuesto por Roberti y Ruthmair.

1.3 Estructura de la tesis

En el Capítulo 2 presentamos las técnicas y conocimientos básicos utilizados en esta tesis, entre los cuales se encuentran los algoritmos de Programación Lineal (LP) y Programación Lineal Entera Mixta (MILP), y las técnicas de Branch and Bound, Branch and Price y generación de columnas. En el Capítulo 3 revisamos la literatura del TSPD, donde analizamos las distintas variantes que surgieron del problema y los resultados obtenidos. En el Capítulo 4 desarrollamos en detalle el algoritmo de Branch and Price en el que se basa esta tesis, propuesto por Roberti and Ruthmair (2021). En el Capítulo 5 presentamos las mejoras y aportes propuestas en este trabajo. Por último, en el Capítulo 6 presentamos la experimentación computacional realizada, las instancias utilizadas y los resultados obtenidos.

2. PRELIMINARES

2.1 Programación Lineal (LP)

La programación lineal es una rama de estudio de la programación matemática dedicada a encontrar una solución óptima de un modelo matemático lineal. Más especificamente, un modelo de programación lineal consiste en minimizar o maximizar una función lineal, llamada *función objetivo*, cuyas variables tienen que satisfacer ciertas desigualdades lineales, llamadas *restricciones*. Notar que la función objetivo y todas las restricciones tienen las mismas variables ya que el coeficiente que acompaña a la misma en la función objetivo o en alguna restricción puede ser cero. En general, a las variables de la función objetivo las llamamos *variables del modelo*. Ciertamente, cada combinación de valores a variables define un punto del espacio de $\mathbb{R}^n$, donde n es la cantidad de variables. Por otra parte, el conjunto de restricciones forma un poliedro convexo en $\mathbb{R}^n$, conocido como la *región factible del modelo*. Con esta visión, un problema de programación lineal consiste en encontrar un punto de la región factible que optimice la función objetivo. Un algoritmo de programación lineal debe encontrar uno de dichos puntos o informar que el poliedro está vacío. En resumen, teniendo en cuenta que las funciones y restricciones son lineales, podemos representar un problema de programación lineal utilizando una matríz $A \in \mathbb{R}^{m \times n}$, un vector $b \in \mathbb{R}^m$ y un vector $c \in \mathbb{R}^n$ donde el objetivo es encontrar un punto $x^* \in \mathbb{R}^n$ tal que $c^t x^* = \min_{x \in \mathbb{R}^n} \{c^t x | Ax \geq b\}$ si el problema es de minimización, o $c^t x^* = \max_{x \in \mathbb{R}^n} \{c^t x | Ax \leq b\}$ si el problema es de maximización.

Como mencionamos antes, las restricciones forman el poliedro que contiene a todas las soluciones factibles. Podemos definir ahora al poliedro como $P = \{x \in \mathbb{R}^n | Ax \leq b\}$; los puntos de P se llaman *soluciones factibles del modelo*.

Consideremos el siguiente ejemplo en $\mathbb{R}^2$, donde claramente tenemos un modelo de programación lineal pues tanto la función objetivo como las restricciones son lineales:

$$\begin{aligned} \max \quad & x_1 + 2x_2 \\ \textbf{s.a:} \quad & \\ & -x_1 + 2x_2 \leq 3 \\ & x_1 + x_2 \geq 3 \\ & 2x_1 - x_2 \leq 5. \end{aligned} \tag{2.1}$$

Podemos ver que cada desigualdad efectivamente se corresponde con una recta en el plano. Cada recta divide al plano en dos semiplanos, uno de los cuales cumple con la desigualdad. En este ejemplo, al quedarnos con el semiplano factible de cada restricción obtenemos el poliedro de la Figura 2.1.

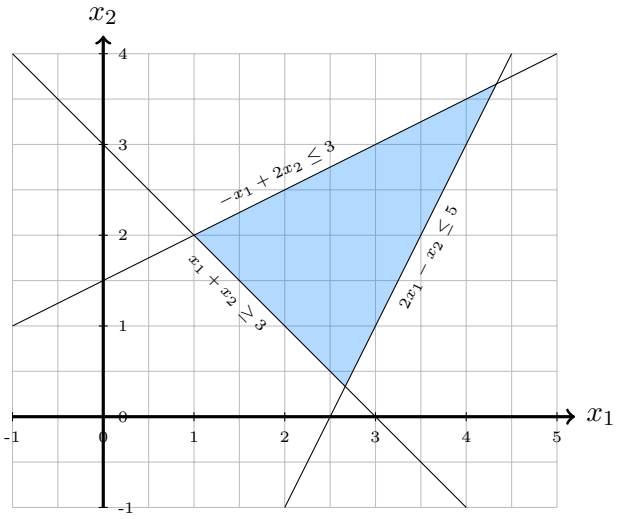

Figura. 2.1: poliedro generado por las restricciones del modelo 2.1

En este caso, el óptimo se encuentra en el punto $(\frac{13}{3}, \frac{11}{3})$, el cuál se corresponde con un vértice del poliedro. Esto no es casualidad, si no que el hecho que que siempre podamos encontrar una solución óptima en algún vértice del poliedro es una propiedad importante de los modelos de programación lineal. Intuitivamente, esto se debe a que en todo punto que no sea un vértice existe una dirección en la cual podemos trasladar la recta que define la función objetivo sin decrementar su valor.

Es importante notar que existen algoritmos polinomiales para resolver problemas de programación lineal. Sin embargo, estos problemas suelen resolverse mediante alguna variante del algoritmo simplex, el cuál tiene un costo exponencial en peor caso, pero suele funcionar muy bien en la práctica. El algoritmo simplex se basa en la propiedad de que, si el modelo tiene una solución óptima, siempre podemos encontrarla en alguno de los vértices del poliedro. Así, el algoritmo recorre estos vértices hasta encontrar un óptimo.

2.2 Programación Lineal Entera Mixta (MILP)

Como vimos, las variables en los problemas de programación lineal pueden tomar valores reales en la solución óptima. Sin embargo, podría suceder que en nuestro modelo haya variables que deban tomar un valor entero. Cuando estamos en un modelo con distintas variables que pueden tomar valores enteros y reales, tenemos un problema de Programación Lineal Entera Mixta.

Al igual que en los problemas de Programación Lineal, estos problemas se representan con una matríz $A \in \mathbb{R}^{m \times n}$, un vector $b \in \mathbb{R}^m$ y un vector $c \in \mathbb{R}^n$. Sin embargo, en este caso el conjunto de variables se divide en dos subconjuntos no vacíos E y R, de forma que:

- $x_i \in \mathbb{Z} \quad \forall\, i \in E.$
- $x_i \in \mathbb{R} \quad \forall\, i \in R.$

Analogamante, el caso en que $R = \emptyset$, se dice que el problema es de Programación Lineal Entera (ILP)

En la Figura 2.2 podemos ver la representación gráfica del sistema de la sección anterior, donde pedimos que tanto x_1 como x_2 sean enteros. Así, la solución se puede encontrar únicamente en los puntos marcados dentro del poliedro.

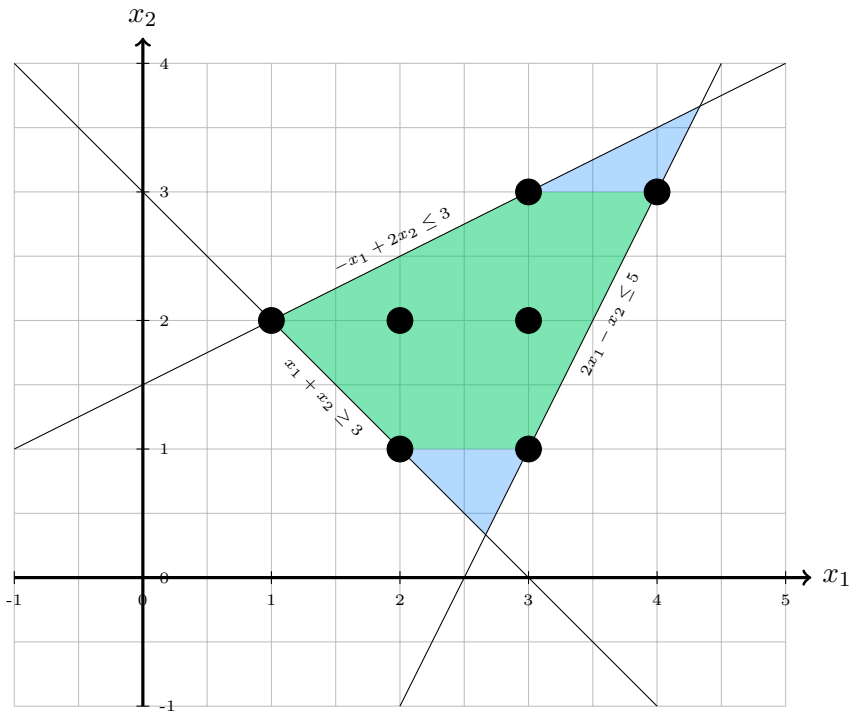

Figura. 2.2

Los problemas de Programación Lineal Entera Mixta son probleman NP-hard, por lo que no se conocen algoritmos polinomiales para resolverlos. Más aún, el algoritmo simplex no se puede aplicar directamente ya que no hay garantías de que el óptimo se encuentre en un extremo. Por este motivo se buscan distintas formas de atacar este tipo de problemas. Una de ellas es utilizar la relajación lineal del problema. Es decir, ignoramos las restricciones de integralidad obteniendo un problema de Programación Lineal, lo que nos permite aprovechar el algoritmo simplex.

Una de las formas más sencillas de utilizar relajaciones lineales para resolver estos problemas es buscando la cápsula convexa de los puntos que se encuentran dentro del poliedro. Si representamos esta cápsula mediante desigualdades, entonces podemos encontrar el óptimo simplemente resolviendo la relajación lineal de la cápsula. Esto se dá porque, como ya mencionamos, el resultado de resolver la relajación lineal va a corresponderse con uno de los vértices de la cápsula, que a su vez es una solución factible del MILP. Sin embargo, es importante notar que la cantidad de restricciones necesarias para representar a la cápsula convexa de los puntos puede ser exponencial en función del tamaño del modelo original en peor caso, por lo que este método no suele ser viable. Para estos casos, se utilizan algunos algoritmos que describiremos a continuación.

2.3 Branch & Bound

La técnica de Branch & Bound es una de las más usadas para resolver problemas de Programación Lineal Entera Mixta. La idea de este método es encontrar una solución óptima del problema explorando eficientemente el conjunto de todas las soluciones factibles. Para lograr esto los algoritmos de Branch & Bound constuyen un árbol cuya raíz representa al conjunto S de todas las soluciones factibles, y los hijos de cada nodo particionan al espacio de soluciones de dicho nodo. En otras palabras, cada nodo representa el problema de encontrar una solución óptima dentro de un subconjunto de S. Al proceso de particionar el conjunto factible de un nodo en los subconjuntos correspondientes a los hijos se lo conoce como *branching*. El proceso de *bounding* consiste en decidir si vale la pena resolver una rama del arbol; para ello se utilizan distintos chequeos que se conocen con el nombre de *podas*. Sin pérdida de generalidad, ejemplificaremos la técnica considerando un problema MILP de minimización.

Vamos a dividir a las podas en dos clases. En primer lugar tenemos a las podas por optimalidad. Cuando estamos analizando un cierto nodo, calcularemos una cota inferior para la solución óptima del problema restringido al subconjunto de soluciones factibles corresopndientes a dicho nodo. Si esa cota inferior es mayor o igual a una cota superior del problema (esta cota superior suele ser la mejor solución factible encontrada hasta el momento), podemos descartar al nodo. Dado que cada nodo se corresponde a un MILP, una forma simple de determinar una cota inferior es resolviendo su relajación lineal, pero existen otras alternativas que pueden ser más eficientes en algunos casos.

En segundo lugar tenemos las podas por factibilidad. Si el poliedro correspondiente al nodo que estamos analizando es vacío o no factible, entonces no hay ninguna solución factible posible en el nodo, por lo que podemos descartarlo.

En cuanto al branching, existen varias maneras de particionar el espacio de búsqueda. En primer lugar, si la solución de la relajación lineal es entera, se cierra el nodo y se actualiza la cota superior. Por otro lado, dado que un nodo se corresponde a un MILP, una forma sencilla de implementar un branching es agregando restricciones lineales que particionen la región factible en dicho nodo. Por ejemplo, si x es una variable entera, entonces podemos crear dos hijos agregando las restricciones $x \leq \lfloor v \rfloor$ y $x \geq \lceil v \rceil$ para algun valor v entero. La idea es que dichas restricciones aprovechen el hecho de que estamos resolviendo un MILP, por lo que probablemente ya resolvimos la relajación lineal del mismo. Luego, una forma de garantizar que se excluya a la solución óptima del MILP definido por el nodo es eligiendo x como una variable que toma un valor fraccionario en el óptimo de la relajación lineal.

Veamos un ejemplo completo de un algoritmo de Branch & Bound para un modelo

sencillo:

$$\begin{aligned} \min \quad & x_2 \\ \textbf{s.a:} \quad & \\ & x_2 \leq 3 \\ & \frac{2}{3}x_1 - x_2 \leq \frac{2}{3} \\ & \frac{-2}{3}x_1 - x_2 \leq \frac{-8}{3} \\ & x_1, x_2 \in \mathbb{Z} \end{aligned} \tag{2.2}$$

Podemos ver el poliedro correspondiente en la Figura 2.3

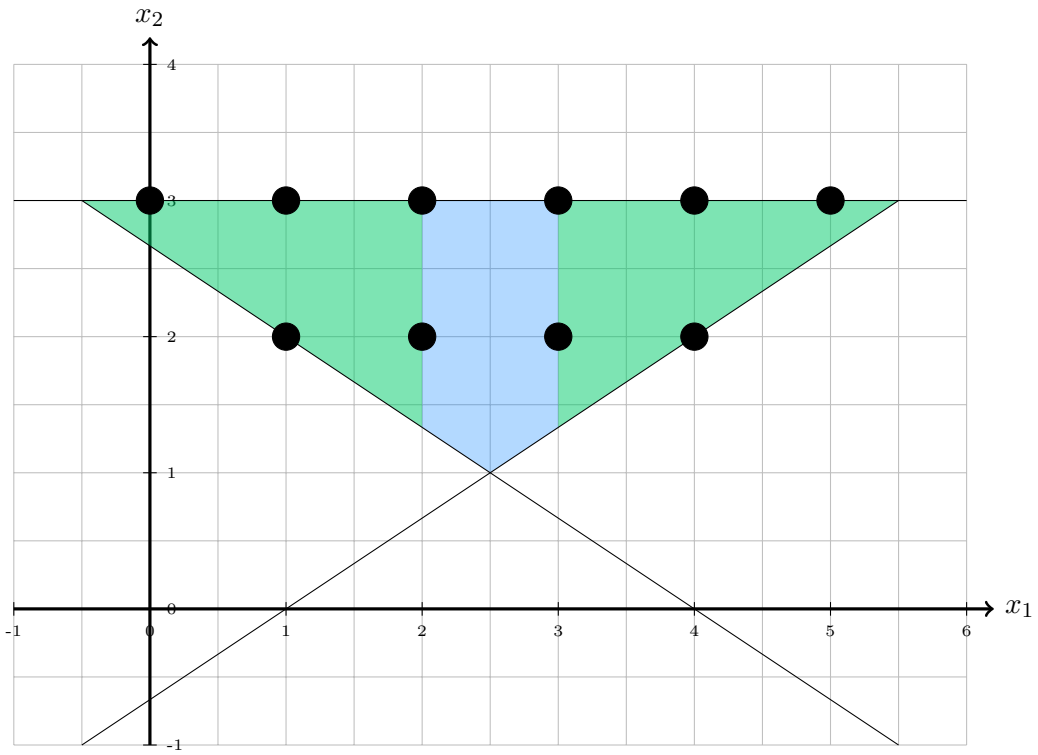

Figura. 2.3: Luego de resolver la relajación del modelo 2.2, realizamos un branch obteniendo dos nuevos nodos, cuyas regiones factibles son representadas por los poliedros verdes.

Es fácil ver que la solución óptima para la relajación lineal del problema se encuentra en el punto (2.5, 1). Sin embargo, esta solución no cumple con que $x_1 \in \mathbb{Z}$. Por lo tanto, una opción es generar dos nuevos nodos. En uno agregamos la restricción $x_1 \leq 2$ y en el otro agregamos $x_1 \geq 3$. De esta manera dividimos al poliedro en dos partes más pequeñas, representadas en la Figura 2.3 con el sombreado verde, obteniendo dos nodos hijo.

Luego, el algoritmo resuelve cada hijo por separado. Notemos que en ambos casos vamos a obtener una solución en donde x_1 toma un valor entero y x_2 toma un valor fraccionario ($(2, \frac{4}{3})$ y $(3, \frac{4}{3})$ respectivamente). Al agregar las restricciones correspondientes, cada nodo generará otros dos hijos, uno de los cuales no tendrá solución entera y otro en donde se econtrará la solución óptima.

2.4 Generación de Columnas

Hay casos en donde incluso la relajación lineal de un problema puede ser demasiado compleja de resolver. Esto sucede, por ejemplo, cuando un modelo tiene una cantidad exponencial de variables. Para estos casos, suele ser útil utilizar la técnica de generación de columnas. La idea principal de esta técnica es comenzar a resolver el problema teniendo en cuenta solamente un subconjunto pequeño de las variables del modelo. Luego, se agregan iterativamente las variables que pueden mejorar la función objetivo del programa. Este proceso es el que da el nombre a la técnica, pués cada variable se corresponde con una columna del sistema de ecuaciones, y agregar una variable consiste también en agregar la columna correspondiente a la matriz de restricciones. Una vez que podemos demostrar que agregar más variables no puede mejorar a la función objetivo, se detiene el proceso.

De esta descripción surgen dos preguntas principales, ¿Cómo sabemos cuándo ya alcanzamos el óptimo y no es necesario agregar variables? ¿Cómo decidimos que variables agregar?

Teorema 1. *(Dualidad fuerte) Sea E un problema LP con función objetivo $f(x)$, y D su problema dual con función objetivo $g(x)$. Si existe una solución óptima x para E, entonces también existe una solución óptima y para D. Además se cumple que $f(x) = g(y)$.*

El teorema de la dualidad fuerte nos es particularmente útil, ya que nos dice que el óptimo es alcanzado cuando las soluciones del problema y de su dual son iguales. Sea E' una versión restringida de E en donde consideramos solamente un subconjunto de las columnas. Por lo general, a E se lo conoce como *problema maestro*, mientras que a E' se lo conoce como *problema maestro reducido* (RMP). Luego de resolver a E', obtenemos su solución óptima x', de la cual podemos obtener la solución óptima y' correspondiente a su dual D'.

Llamemos x a la extensión de x' donde le asignamos 0 a todas las variables que no estaban en E'. Es fácil ver que x es factible en E, pero no necesariamente es óptima. Por otro lado, como D' no tenía a todas las restricciones, es posible que y' viole a alguna de ellas por lo que no necesariamente es factible en D. Sin embargo, es importante notar que si y es factible en D, entonces también es la solución óptima y, por el teorema de dualidad fuerte, x es la solución óptima de E.

Tenemos entonces el problema de determinar si y es factible en D. Para lograr esto, habría que ver si y viola a alguna restricción de D o, desde el punto de vista E', si existe alguna columna de costo reducido negativo. Recordemos que D tiene una cantidad exponencial de restricciones, por lo que esto puede ser un problema costoso. A este problema se lo suele conocer como *problema de pricing*. Una vez que se encuentran restricciones violadas, se pueden agregar a E' y se repite el proceso con los nuevos problemas.

Un dato importante es que en ocasiones el problema de pricing se corresponde con problemas bien conocidos. En esta tesis estaremos trabjando con una relajación del TSPD que incluye rutas infactibles, y cuyo problema de pricing es la misma relajación del TSPD pero con beneficios que se cobran al recorrer cada vértice (ver Sección 4.4).

2.5 Branch & Price

Branch & Price es una técnica que combina Branch & Bound con generación de columnas para resolver problemas de Programación Lineal Entera Mixta con una cantidad exponencial de columnas. Resumidamente, Branch & Price es un Branch & Bound en donde para cada nodo se resuelve la relajación lineal via generación de columnas, tanto para branchear como para acotar.

Presentamos un esquema general para los algoritmos de Branch & Price:

Algorithm 1 Esquema de un Branch & Price

Input: descripción D del problema maestro MP y conjunto E de columnas que definen el RMP

1: **while** $True$ **do**
2: Resolver relajación lineal de RMP para obtener una solución x' y su dual y'
3: Resolver el problema de pricing sobre y' obteniendo un conjunto X de columnas de costo reducido negativo
4: Agregar X a E
5: **if** x' es entero **then**
6: retornar x' y terminar
7: **else if** $X = \emptyset$ **then**
8: Para cada descripción D y cada conjunto E de columnas factible que resultan de ejecutar el Branch(RMP), resolver el Branch & Price con input D y E para obtener el conjunto de soluciones enterar X^*
9: retornar el minimo de X^*.
10:

3. REVISIÓN DE LA LITERATURA

En los últimos años, las empresas que deben realizar envíos se estuvieron concentrando en reducir el tiempo de transporte de sus productos. Un caso común es aquel en que en las redes de transporte pueden enviarse paquetes de muchos clientes a un depósito central de forma eficiente, ya sea por barco, tren, avión, etc. Una vez que los paquetes se encuentran en éste depósito, cada paquete debe ser enviado en vehículos más pequeños a los clientes. Estos *envíos de última milla* generan uno de los principales cuellos de botella en la gestión de las cadenas de suministros.

Hay muchas alternativas de vehículos para los envíos de última milla, como pueden ser los camiones, autos y bicicletas. Un nuevo vehículo que comenzó a usarse para estos propósitos es el dron (Amazon, Zipline), cuya principal ventaja es la de no tener que moverse por las redes de tráfico, evitando congestiones y pudiendo moverse en linea recta. En este trabajo vamos a concentrarnos en los distintos algoritmos exactos que fueron propuestos para el TSPD con un único camión y un único dron descripto en la Sección 1.2.

Murray and Chu (2015) fueron los primeros en proponer un problema de ruteo que involucra un camión y un dron. El problema estudiado, llamado *Flying Sidekick TSP* (FSTSP), es una de las dos variantes más estudiadas del TSPD y se obtiene agregando las siguientes restricciones sobre TSPD:

- Algunos clientes no pueden ser visitados por el dron.
- Se tiene en cuenta el tiempo que toma lanzar y aterrizar el dron.
- El dron tiene una batería que limita el tiempo de vuelo.
- En caso de que el dron llegue antes a un punto de encuentro, el mismo debe permanecer en un vuelo estático y no pude ahorrar batería aterrizando.

La otra variante más estudiada del TSPD es el TSPD con loops y revisitas (TSPD+lr) propuesto por Agatz et al. (2018). Al igual que en el FSTSP, el TSPD+lr no permite que ciertos clientes sean visitados por el dron, y además agrega las siguientes dos restricciones:

- El camión puede visitar a algunos clientes más de una vez.
- El camión no necesariamente tiene que visitar a un cliente mientras el dron está en vuelo.

Otras variantes menos populares del TSPD consideran funciones objetivo alternativas (Ha et al., 2018) y rutas de dron no rectas para poder evitar zonas donde no está permitido el vuelo (Jeong et al., 2019), entre otras nuevas restricciones.

El TSPD estudiado en este trabajo fue introducido por Roberti and Ruthmair (2021). La justificación dada en su trabajo para estudiar esta versión simple del problema es que la sincronización de ambos vehículos resulta ser una parte particularmente compleja de

resolver, por lo que las mejoras y aportes a la versión basica del TSPD pueden ayudar a resolver los casos más generales. Más aún, Roberti and Ruthmair (2021) muestran que algunas restricciones pueden incorporarse de manera sencilla a su algoritmo sin tener un gran impacto en su eficiencia.

La tabla 3.1 resume los trabajos existentes sobre las distintas variantes del TSPD estudiadas hasta la fecha. La columna *All* indica el máximo tamaño de instancia para el cual el algoritmo pudo resolver todas las instancias, la columna *50%* indica el tamaño para el cual se resolvieron al menos la mitad de las instancias, y la columna *Max* indica el máximo tamaño de instancia resuelto. Lo que puede verse en esta tabla es que, sin importar la variante del TSPD con la que se trabaje, los algoritmos propuestos tienen problemas resolviendo instancias de 30 o más clientes. Los primeros trabajos se concentraron en el modelado y en algoritmos de B&B y DP sencillos, que resolvían instancias de hasta 15 clientes. Para lograr resolver instancias más grandes se utilizaron distintas técnicas, entre las cuales se encuentran: branch & cut con planos de cortes adaptados del TSP (van Dijck and Bouman, 2018), branch & price con relajaciones para el problema de pricing resuelto con programación dinámica (Roberti and Ruthmair, 2021), y descomposición de Benders (Vásquez et al., 2021).

Publicación	Técnica	Tiempo límite	All	50%	Max
Murray and Chu (2015)	MILP	0.5hs	0	0	0
Ponza (2016)	MILP	24hs?	—	9	9
Agatz et al. (2018)	MILP	≈2hs?	—	11?	11
Bouman et al. (2018)	DP	12hs	—	15	15
Es Yurek and Ozmutlu (2018)	Iter(MILP)	1h	12	12	12
Ha et al. (2018)	MILP	1h?	—	9?	9
van Dijck and Bouman (2018)	B&C	1h?	—	19?	19?
Jeong et al. (2019)	MILP	12hs?	—	9?	9
Poikonen et al. (2019)	B&B	1h	—	9	9
Tang et al. (2019)	CP	1h?	—	17?	17
Schermer et al. (2020)	B&C	1h	10	19	20
Boccia et al. (2021b)	B&C(C&RG)	1h	10	20	20
Boccia et al. (2021a)	B&C	2hs	15	20	20
Dell'Amico et al. (2021)	B&C	1h	—	9	13
Dell'Amico et al. (2021)	B&B	1h	14	14	19
Roberti and Ruthmair (2021)	B&P(DP)	1h	19	29	39
Vásquez et al. (2021)	Benders	3hs	19	22	25
Dell'Amico et al. (2022)	MILP,B&C	1h	10	20	20
Este trabajo	B&P(DP)	1h	29	39	39[1]

Tab. 3.1: Vista general de los diferentes algoritmos exactos para distintas variantes del TSPD. Las distintas técnicas son: programación lineal entera mixta (MILP), procedimientos iterativos que resuelven un MILP (Iter(MILP)), programación dinámica (DP), Branch and Bound (B&B), Branch and Cut (B&C), constraint programming (CP), B&C con generación de columnas y filas (B&C(C&RG)), Branch and Price utilizando una DP para el problema de pricing (B&P(DP)) y descomposición de Bender (Benders).

[1] Es imporante notar que en nuestro trabajo experimentamos únicamente con las instancias utilizadas

Al día de la fecha, el algoritmo de Roberti y Ruthmair es uno de los más eficientes para el TSPD y para muchas de sus variantes. Para la versión básica, es el primero en llegar a resolver instancias de 39 clientes. En este trabajo mejoramos este algoritmo para la versión básica, llegando a resolver instancias que el algoritmo de Roberti y Ruthmair no pudo resolver. Consideramos además que las adaptaciones hechas para los otros problemas se pueden aplicar dentro de nuestro algoritmo, lo que permitiría escalar las soluciones a los otros problemas considerados.

por Roberti and Ruthmair (2021), por lo que las columnas 50% y *Max* no son del todo indicativas

4. ALGORITMO DE ROBERTI Y RUTHMAIR

Como mencionamos en el capítulo anterior, uno de los trabajos más recientes sobre el TSPD es el de Roberti y Ruthmair (Roberti and Ruthmair, 2021), al cuál llamaremos RR de ahora en adelante. Esta tesis está fuertemente basada en esa solución, y construye sobre ese algoritmo. En esta sección vamos a repasar detalladamente la solución planteada por Roberti y Ruthmair. Recordemos que RR es un algoritmo de tipo Branch and Price en donde el problema de pricing se resuelve con un algoritmo de programación dinámica. Éste algoritmo de programación dinámica es un algoritmo de labeling, como explicamos a continuación.

4.1 Algoritmos de Labeling para el TSPD

Los algoritmos de labeling para los problemas camino mínimo se basan en ir generando todos los caminos posibles desde un punto de partida, donde cada ruta es representada por una estructura llamada label (Ahuja et al., 1993). El algoritmo de RR es un caso particular de un algoritmo de labeling para TSPD. Intuitivamente, RR comienza con un label que representa a la ruta vacía (es decir, aquella en donde todavía no se visitó a ningún cliente). En cada paso del algoritmo se tomarán los labels generados en el paso anterior, y se los extenderá agregandoles una visita a un único cliente. Es decir, en cada paso aumentamos en uno la máxima cantidad de clientes visitados por un label. El algoritmo seguirá de esta manera hasta que genere a todas las rutas de longitud $|N|$. Notemos que, dado que ambos vehículos se mueven en paralelo, debemos asignar algún orden a las extensiones de los labels. En este algoritmo vamos a hacer que, cuando los vehículos se separan, el camión realiza todas sus extensiones primero, y luego el dron hace las suyas. Por ejemplo, en el caso de la Figura 1.1, primero ambos vehículos se moverán al cliente x_5, luego el camión visitará a x_4 y x_2, y luego el dron se extenderá a x_3 y x_4.

En el caso del TSPD, RR plantea que un label debe tener la siguiente información:

- S: Conjunto de clientes que ya fueron visitados por cualquiera de los vehículos.
- it: Posición del camión.
- id: Posición del dron.
- τ: Distancia que recorrió el camión desde que se separó del dron. Se usa para decidir el costo del tramo que recorrieron los vehículos cuando estaban separados.
- c: Costo del camino recorrido por el label.

Para poder generar todos los labels posibles desde un label inicial RR define tres reglas de extensión, una por cada movimiento posible (mover solamente al camión, solamente al dron, o a ambos vehículos juntos):

- *agregar un arco de camión*: Un label (S, it, id, τ, c) se extiende a todos los clientes $j \in N \backslash S$, generando los labels ($S \cup \{j\}$, j, id, $\tau + t^T_{itj}$, c).

- *agregar un arco combinado*: Si $it = id$, un label (S, it, id, τ, c) se extiende a todos los clientes $j \in N \backslash S$, generando los labels ($S \cup \{j\}$, j, j, 0, $c + t^T_{itj}$).

- *agregar un arco de dron*: Si $it \neq id$, un label (S, it, id, τ, c) se extiende a todos los clientes $j \in N \backslash S$, generando los labels ($S \cup \{j\}$, it, it, 0, $c + \max(\tau, t^D_{idj} + t^D_{jit})$).

De ahora en adelante, vamos a usar los pares (t, v), (c, v), (d, v) para referirnos a las extensiones mediante un arco de camión, combinado y de dron respectivamente. A modo de ejemplo, (t, v) se corresponde con un arco de camión que visita al cliente v.

Luego, cada ruta factible se define como una secuencia de extensiones, cada una de las cuales visita a un único cliente. Dicha secuencia debe comenzar en el depósito, terminar en el depósito, y el nodo final de cada extensión debe coincidir con el nodo inicial de la siguiente extensión. Además, es necesario que se visite a todos los clientes. Introducimos también el concepto de *ruta parcial*. Las rutas parciales son rutas que aún no han terminado su recorrido (no regresan al depósito). Es decir, son secuencias de extensiones que comienzan en el depósito, no visitan a ningún cliente más de una vez, y el nodo final de cada extensión debe coincidir con el nodo inicial de la siguiente extensión. A modo de ejemplo, la ruta de la Figura 1.1 se corresponde con la secuencia de extensiones [(c, x_5), (t, x_4), (t, x_2), (d, x_3), (c, x_1), (c, $0'$)], mientras que [(c, x_5), (t, x_4)] es una ruta parcial en la que se visitan solamente a x_5 y x_4. Llamamos *label factible* a todo label que se corresponda con una ruta factible o una ruta parcial factible. Esta estrategia de modelado nos permite dar un orden secuencial a las acciones de ambos vehículos, a pesar de que operan en simultáneo cuando se separan. En particular, estas reglas de extensión determinan que luego de separarse del dron el camión debe realizar todo su recorrido, y una vez que termine agregamos el arco de dron para volverlos a unir.

Conceptualmente, el algoritmo de labeling comienza con el label que representa a la ruta vacía, $L_0 = (\emptyset, 0, 0, 0, 0)$, el cual se agrega a una cola de labels sin procesar. En cada paso del algoritmo se saca un label de la cola y se lo extiende hacia todos sus labels sucesores factibles. A modo de ejemplo, veamos como se generaría el camino de la figura 1.1:

- Agregamos un arco combinado desde el depósito hasta x_5 para obtener el label ($\{x_5\}$, x_5, x_5, 0, t^T_{d5}).

- Agregamos un arco de camión a x_4 y obtenemos ($\{x_5, x_4\}$, x_4, x_5, t^T_{54}, t^T_{d5}).

- Agregamos otro arco de camión a x_2 y obtenemos ($\{x_5, x_4, x_2\}$, x_2, x_5, $t^T_{54} + t^T_{42}$, t^T_{d5}).

- Agregamos un arco de dron visitando x_3 y unimos a los vehículos en x_2, obteniendo el label ($\{x_5, x_4, x_2, x_3\}$, x_2, x_2, 0, $t^T_{d5} + \max\{t^T_{54} + t^T_{42}, t^D_{53} + t^D_{32}\}$).

- Agregamos un arco combinado hasta x_1. El label pasaría a ser ($\{x_5, x_4, x_2, x_3, x_1\}$, x_1, x_1, 0, $t^T_{d5} + \max\{t^T_{54} + t^T_{42}, t^D_{53} + t^D_{32}\} + t^T_{21}$).

- Agregamos un último arco combinado volviendo al depósito.

Para poder referirnos a los distintos parámetros de un label, agregamos la notación $\bullet()$, donde $\bullet$ se puede reemplazar por cualquier parámetro de los labels (τ, S,etc). Por ejemplo, dado un label L, podemos usar $\tau(L)$ y $S(L)$ para referirnos a los valores τ y S de L.

4.2 Relajación de rutas ng

Como adelantamos en la Sección 1.2, Roberti y Ruthmair resuelven el TSPD utilizando un modelo de set-partitioning sobre un conjunto extendido de rutas que define una relajación del TSPD. La razón para considerar esta relajación es que la misma permite reducir considerablemente el tiempo de ejecución del algoritmo de pricing. El objetivo de esta sección es describir la relajación que utiliza RR, que está basada en la relajación ng propuesta por Baldacci et al. (2011) para problemas de ruteo sin drones. A falta de un mejor nombre, a la relajación del TSPD propuesta por Roberti and Ruthmair (2021) también se la conoce con el nombre de relajación ng.

Las rutas *elementales* son aquellas en donde cada cliente es visitado una única vez. Notemos que las reglas de extensión propuestas en la Sección 4.1 permiten generar únicamente rutas elementales, pues ningún vehículo puede visitar a un cliente que pertenezca al conjunto S, y S contiene a todos les vértices visitados. La idea de la relajación ng es modificar dichas reglas de extensión, de manera que el algoritmo pueda generar algunas rutas no elementals descriptas a continuación.

Como motivación para la definición de relajación ng para el TSPD, veamos como se define la relajación ng para un problema más simple, el TSP. Para cada cliente $i \in N$, definimos un conjunto de clientes $N_i \subseteq N$ llamado *conjunto-ng* o *vecindario* de i. En el TSP, una ruta ng o ng-factible es una ruta no necesariamente elemental donde un cliente $i \in N$ es visitado más de una vez solo si existe un cliente j visitado entre las dos visitas consecutivas del cliente i tal que $i \notin N_j$. Notemos que esto es efectivamente una relajación del TSPD, porque toda ruta elemental es ng-factible. Por lo tanto, la solución al problema relajado provee una cota inferior para el problema original.

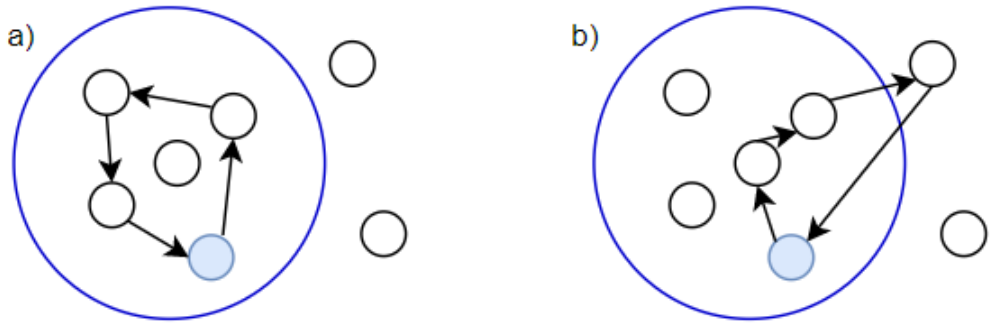

Figura. 4.1: Ejemplo de ruta ng inválida (a) y válida (b) para el TSP

En la Figura 4.1 podemos ver un caso de ruta ng válida y un caso inválido. En el caso a) vemos que la ruta parte del nodo sombreado, cuyo vecindario está marcado por el círculo azul. La ruta pasa por algunos clientes dentro del vecindario y luego regresa al nodo sombreado, por lo que este ciclo rompe con la definición de ruta ng. Por otro lado, en el caso b) tenemos que la ruta parte del nodo sombreado, sale del vecindario, y luego vuelve al nodo sombreado. Este tipo de ciclos es permitido por las rutas ng.

Para resolver el TSP con un algoritmo de labeling, alcanza con mantener un label (S, i, c) que indique que una ruta parcial pasó por los vertices de S, terminando en el vértice i con un costo c. Luego, la regla de extensión indica que un label (S, i, c) se extiende a todos los vértices $j \notin S$ para generar los labels $(S \cup \{j\}, j, c + costo(i \to j))$. Para implementar un algoritmo de labeling para el TSP con relajación ng, hay que modificar la noción de label y extensión. El nuevo label tiene por lo menos tres campos (ng, i, c) que indican que la ruta parcial termina en i, tiene costo c y se genera un ciclo prohibido en la próxima extensión si y sólo si se visita un vértice $w \in ng$. En consecuencia, la nueva extensión indica que el label se puede extender únicamente a vértices $w \notin ng$ para generar un nuevo label que contenga un conjunto ng' de extensiones prohibidas en el siguiente paso. Claramente, el nuevo conjunto ng' debe ser tal que satisfaga la noción de ng-factibilidad. Esto se obtiene cuando $ng' = (ng \cup \{i\}) \cap N_w$. Luego, vimos que la ng-factibilidad puede ser definida en términos de las reglas de extensión.

El tamaño de los conjuntos-ng determina los subtours permitidos en las rutas ng, la calidad de la cota inferior de la relajación, y la dificultad del problema de pricing. Esto se debe a que cuanto más grande sean los N_i, menos van a ser los ciclos permitidos y por tanto mejor será la cota obtenida. Sin embargo, conjuntos-ng grandes hacen que el problema sea más difícil de resolver. En el caso extremo en que $N_i = S$, la ng-factibilidad se coincide con la factibilidad.

Al tener dos vehículos, la definición de las rutas ng para el TSPD requiere de algunos cuidados. Roberti and Ruthmair (2021) proponen una noción de ng-factibilidad que se define directamente sobre las reglas de extensión de labels. En este contexto, las rutas parciales ng-factibles quedan definidas como las secuencias correspondientes a los labels, siendo las rutas ng-factibles aquellas que se corresponden con las extensiones completas. Remarcamos que es posible hacer una definición conceptual de rutas ng-factibles que, en analogía con el TSP, tenga en cuenta la salida de los vecindarios. Sin embargo, evitamos dicha definición porque es compleja y no aporta mayores detalles sobre las rutas ng-factibles.

Recordemos que como adelantamos en la Sección 2.4, el problema de pricing que estamos resolviendo se corresponde con la relajación del TSPD con pesos en los vértices. Sean $u_0 \in \mathbb{R}$ y $u_i \in \mathbb{R}$ $(i \in N)$ los pesos asociados al depósito y a los clientes respectivamente. Roberti y Ruthmair proponen la siguiente adaptación del algoritmo de labeling para generar únicamente rutas ng-factibles. Ahora, en lugar de tener labels de la forma (S, it, id, τ, c), vamos a tener labels de la forma (ng, k, it, id, τ, c). Aquí, $ng \subseteq N_{it}$ es el subconjunto de los clientes ya visitados que no pueden ser visitados en la próxima extensión. Por otro lado k es la cantidad de clientes visitados hasta el momento, y el algoritmo seguirá extendiendo los labels hasta que se cumpla que $k = |N|$.

Ahora comenzamos con el label ($\emptyset$, 0, 0, 0, 0, $-u_0$) y las extensiones son definidas de la siguiente manera:

- *agregar un arco de camión*: Un label (ng, k, it, id, τ, c) se extiende a todos los clientes $j \in N \backslash (ng \cup \{it, id\})$, generando los labels $((ng \cup \{it\}) \cap N_j, k+1, j, id, \tau + t^T_{itj}, c - u_j)$.
- *agregar un arco combinado*: Si it = id, un label (ng, k, it, id, τ, c) se extiende a todos los clientes $j \in N \backslash (ng \cup \{it\})$, generando los labels $((ng \cup \{it\}) \cap N_j, k+1, j, j, 0, c +$

$t^T_{itj} - u_j$).

- *agregar un arco de dron*: Si it $\neq$ id, un label (ng, k, it, id, τ, c) se extiende a todos los clientes $j \in N \backslash (ng \cup \{it, id\})$, generando los labels $((ng \cup \{j\}) \cap N_{it}, k+1, it, it, 0, c + \max(\tau, t^D_{idj} + t^D_{jit}) - u_j)$.

Veamos cómo se usan estas extensiones para crear una ruta ng-factible en un ejemplo. Supongamos que tenemos una instancia con 5 clientes, x_1, x_2, x_3, x_4 y x_5, cuyos vecindarios son $N_{x_1} = \{x_2, x_5\}$, $N_{x_2} = \{x_1, x_3\}$, $N_{x_3} = \{x_1, x_2\}$, $N_{x_4} = \{x_2, x_5\}$ y $N_{x_5} = \{x_4, x_1\}$. Luego, podemos generar la ruta de la figura 4.2 de la siguiente manera:

- Agregamos un arco combinado desde el depósito hasta x_5. El label pasaría a ser ($\emptyset$, 1, x_5, x_5,0, t^T_{d5}).
- Agregamos un arco de camión a x_4 y obtenemos ($\{x_5\}$, 2, x_4, x_5, t^T_{54}, t^T_{d5}).
- Agregamos un arco de camión a x_3 y obtenemos ($\emptyset$, 3, x_3, x_5, $t^T_{54} + t^T_{43}$, t^T_{d5}).
- Agregamos un arco de camión a x_4 y obtenemos ($\emptyset$, 4, x_4, x_5, $2 * t^T_{54} + t^T_{43}$, t^T_{d5}).
- Agregamos un arco de dron a x_2 y obtenemos ($\{x_2\}$, 5, x_4, x_4, 0, $t^T_{d5} + \max(2{*}t^T_{54}{+}t^T_{43}, t^D_{52} + t^D_{24})$).
- Agregamos un último arco combinado volviendo al depósito.

Notemos que en esta ruta hay una revisita al cliente x_4, pero esta revisita está permitida pues luego de visitar a x_4 por primera vez, el camión abandona su vecindario al extenderse hacia x_3. Algo más para notar es que no se están visitando a todos los clientes, ya que como mencionamos antes las rutas ng-factibles deben visitar exactamente $|N|$ clientes.

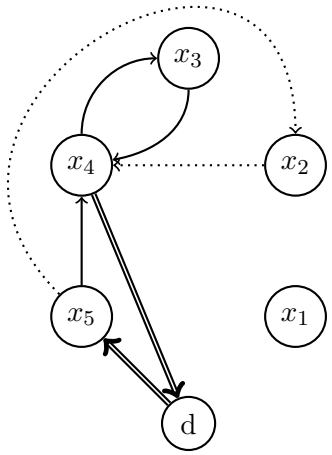

Figura. 4.2: Ejemplo de ruta ng-factible. Las flechas dobles representan arcos combinados, las flechas punteadas arcos de dron, y las flechas comunes arcos de camión. Esta ruta es ng-factible a pesar de tener una revisita y dejar un cliente sin visitar

Estas nuevas reglas de extensión siguen cumpliendo con que incrementan la cantidad de clientes visitados por un label en uno. Sin embargo, podemos dar una definición que tenga en cuenta más de un paso de extensión, ya que esto va a ser útil más adelante. Por lo tanto, podemos extender el concepto de extension a extensiónes entre labels de la siguiente manera:

Definición 1. *Sean L y L' dos labels para rutas ng-factibles que cumplen que L' se puede obtener aplicando una secuencia s de una o más reglas de extensión sobre L. Decimos que s es una extensión de L, e incorporamos la notación $L' = L + s$. En particular, si $L + s$ es una ruta completa, decimos que s es una extensión completa de L.*

Notemos que esta generalización es correcta, pues incluye a las extensiones de un solo paso. a modo de ejemplo, la ruta de la figura 4.2 puede ser escrita como $L_\emptyset$ + [(c,x_5), (t,x_4), (t,x_3), (t,x_4), (d,x_2), (c,d)] donde $L_\emptyset$ representa al label vacío.

De ahora en adelante, vamos a decir que el *dron está fuera del camión* si la última extensión fue un arco de camión, y que el *dron está dentro del camión* si la última extensión fue un arco combinado o un arco de dron.

4.3 Reglas de dominación

Notemos que el algoritmo de labeling explicado hasta ahora genera a todas las rutas ng-factibles. Nos gustaría de alguna forma podar el árbol de búsqueda de manera de generar la menor cantidad de rutas posible. Para lograr esto, vamos a introducir el concepto de *dominación* entre lables. A medida que extendemos los labels, puede pasar que nos demos cuenta de que algún label nunca pueda llevarnos a una ruta que tenga menor costo que las rutas que se generan con otro label. Los reglas que usamos para determinar cuando no debemos seguir extendiendo un label son las *reglas de dominación*.

Definición 2. *(dominación entre labels)*
Sean L_1 y L_2 dos labels de rutas ng-factibles. Decimos que L_1 domina a L_2 si toda extensión completa s de L_2 también es una extensión completa de L_1, y el costo de $L_1 + s$ es menor o igual al costo de $L_2 + s$.

En la Figura 4.3 vemos un ejemplo para el TSP clásico dode se recorren únicamente rutas elementales.

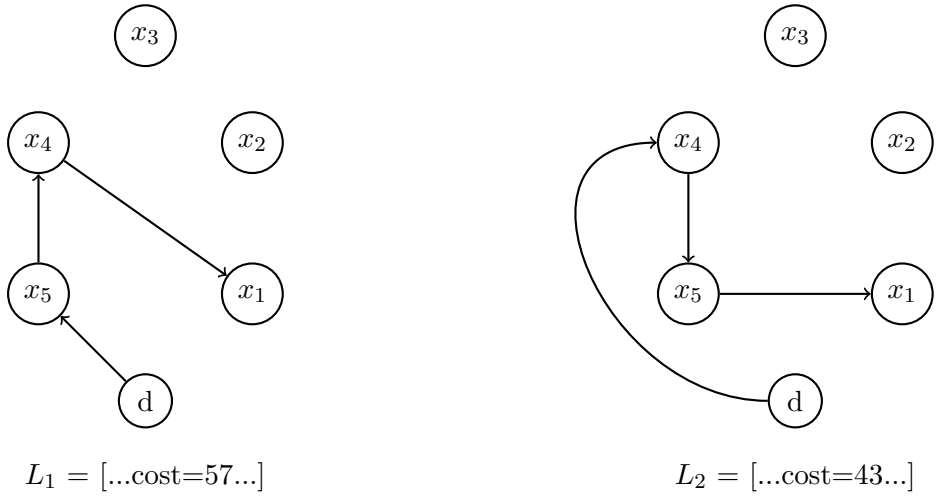

Figura. 4.3: Ejemplo de dominación. L_1 visita a los mismos clientes que L_2 y ambos terminan su recorrido en x_1, pero el costo de L_2 es menor

Como podemos ver en este ejemplo, tanto el label L_1 como el label L_2 visitan a los mismos clientes. Además, ambos labels terminan su recorrido en el mismo cliente, x_1. Sin embargo, podemos observar que el costo de L_2 es menor al del L_1. Por lo tanto, cualquier extensión que hagamos en L_1, podemos copiarla en L_2 obteniendo un camino de costo menor. Estas mismas ideas aplican al TSPD, aunque determinar cuándo un label domina a otro puede ser un poco más complejo. Dados dos labels cualesquiera, saber si uno domina al otro puede ser computacionalmente complejo. Es para esto que RR introduce tres reglas de dominación, que nos permiten determinar de manera eficiente si un label domina a otro. Notemos que estas reglas no son completas, en el sentido de que puede haber dominaciones entre labels que las reglas no pueden detectar. Sin embargo, la cantidad de labels que son descartados gracias a estas reglas es suficiente como para que el algoritmo mejore notablemente su rendimiento. Las tres reglas planteadas por RR son las siguientes:

Regla de dominación 1: Si $L_1 = \{ng_1, k, i_t, i_d, \tau_1, c_1\}$ y $L_2 = \{ng_2, k, i_t, i_d, \tau_2, c_2\}$

son dos labels tales que (a) $c_1 \leq c_2$, (b) $i_t = i_d$, (c) $ng_1 \subseteq ng_2$, y (d) $\tau_1 \leq \tau_2$, entonces, L_1 domina a L_2.

Regla de dominación 2: Sean $L_1 = \{ng_1, k, i_t, i_d, \tau_1, c_1\}$ y $L_2 = \{ng_2, k, i_t, i_d, \tau_2, c_2\}$ dos labels tales que $i_t \neq i_d$. Si se cumplen:

$$\begin{aligned} & ng_1 \subseteq ng_2 \\ & c_1 \leq c_2 \\ & c_1 + \tau_1 \leq c_2 + \tau_2 \end{aligned} \tag{4.1}$$

entonces L_1 domina a L_2.

Regla de dominación 3: Sean $L_1 = \{ng_1, k, i_t, i_d, \tau_1, c_1\}$ y $L_2 = \{ng_2, k, i_t, i_d, \tau_2, c_2\}$ tal que $i_t \neq i_d$. Si se cumplen:

$$\begin{aligned} & ng_1 \subseteq ng_2 \\ & c_1 > c_2 \\ & c_1 + \tau_1 \leq c_2 + \tau_2 \\ & c_1 + \max_{j \in N, s \in N'_0, j \neq s} \left\{ t^D_{i_d j} + t^D_{js} \right\} \leq c_2 + \tau_2 \end{aligned} \tag{4.2}$$

entonces L_1 domina a L_2.

Estas reglas de dominación logran reducir ampliamente el espacio de búsqueda del algoritmo, ayudando a que tengamos que generar menos caminos. Un aspecto importante de estas reglas es que el tamaño de los vecindarios de cada cliente influye fuertemente en la cantidad de labels que son dominados, dado que en las tres reglas se pide que $ng_1 \subseteq ng_2$. Recordemos que, como mencionamos en la Sección 4.2, cuánto más grandes son estos vecindarios más disminuye la cantidad de rutas ng-factibles. Sin embargo, la cantidad de dominaciones también disminuye ya que es menos probable que se cumpla la condición de las reglas de dominación sobre la contención de los conjuntos ng. El balance entre la cantidad de rutas que generamos y la cantidad de labels dominados es un factor importante a tener en cuenta al momento de elegir los vecindarios.

4.4 Formulación de set partitioning

El algoritmo de labeling planteado en la sección 4.1 tiene un espacio de búsqueda demasiado grande, por lo que no es posible ejecutarlo en un tiempo razonable. Por otro lado, el labeling sobre la relajación de rutas ng es más eficiente, pero no brinda rutas elementales. Por ese motivo, RR introduce un método de branch and price basado en una formulación de *set partitioning* donde el problema de pricing es una relajación del TSPD basado en rutas ng. Resolviendo este problema de pricing se obtienen mejoras desde el punto de vista computacional, ya que como vimos en la sección anterior, se reduce de forma significativa la cantidad de labels no dominados. La factibilidad de las soluciones se fuerza más tarde, en el problema maestro. El problema de pricing va a ser resuelto con programación dinámica basada en el algoritmo de la Sección 4.2.

Veamos entonces la formulación de set partitioning planteada por RR. Recordemos que una *ruta* en un grafo G se define como una secuencia de extensiones que comienzan en el depósito, terminan en el depósito, y tal que el nodo final de cada extensión coincide con el nodo inicial de la siguiente extensión. Notemos que con esta definición, se permite que una ruta tenga clientes repetidos.

Sea R el conjunto de todas las rutas ng-factibles de G. Cada ruta $r \in R$ tiene un conjunto de coeficientes asociados a_{ir} que indican la cantidad de veces que el cliente $i \in N$ es visitado por la ruta r, y d_r que indican la duración de la ruta $r \in R$. Además, sean $\xi_r \in \{0, 1\}$ variables binarias que son iguales a 1 si la ruta $r \in R$ es seleccionada, y 0 si no. El TSPD puede ser formulado con el siguiente modelo:

$$\begin{aligned} t^* = \min & \sum_{r \in R} d_r \xi_r \\ s.t. & \sum_{r \in R} \xi_r = 1 \\ & \sum_{r \in R} a_{ir} \xi_r = 1 \quad i \in N \\ & \xi_r \in \{0, 1\} \qquad r \in R \end{aligned} \tag{4.3}$$

La función objetivo minimiza la duración de la ruta seleccionada. La primera restricción se asegura de que una única ruta sea seleccionada, mientras que la segunda restricción pide que todos los clientes sean visitados. Por último, tenemos las restricciones de intergalidad.

Es fácil ver que el conjunto R contiene una cantidad exponencial de columnas, por lo que esta formulación solo puede ser usada para resolver el TSPD si se aplica generación de columnas para generar dinámicamente las variables ξ. Cualquier solución de esta formulación se corresponde a una ruta de R que visita a todos los clientes, por lo que un algoritmo de generación de columnas podría buscar únicamente esas rutas.

Para ver que el problema de pricing se corresponde con el TSPD con pesos en los vértices recordemos la técnica de generación de columnas presentada en la Sección 2.4. Dada una solución x' para una restricción E' de nuestro modelo, podemos obtener una solución y' del dual D' correspondiente. Luego, si queremos ver si la solución y' es factible en D, debemos ver si y' viola alguna restricción del problema dual o, desde el punto de vista de E, si la columna correspondiente a la restricción violada tiene costo reducido negativo. En nuestro problema, esto es equivalente a encontrar una ruta r tal que:

$$d_r - \sum_{i \in N} a_{ir} y_i < 0$$

Es decir, la duración de la ruta r debe ser menor a la suma de los valores duales asociados a los clientes que visita. Luego, ejecutando un TSPD en donde cada vez que visitamos a un cliente restamos el valor de su dual asociado, podemos encontrar columnas de costo reducido negativo.

4.5 Reglas de branching

Recordemos que el esquema de Branch & Price funciona de manera similar al Branch & Bound, pero en donde en cada nodo del árbol de búsqueda se resuelve la relajación lineal

de la Sección 4.2 utilizando un algoritmo de generación de columnas. Cuando la solución óptima de la relajación lineal no es entera, RR propone realizar un branching basándose en tres tipos de decisiones. Dada una solución óptima S^*, calculamos:

- Cuántas veces cada cliente es atendido únicamente por el dron (y_i).
- Cuántas veces cada arco es transitado únicamente por el camión (x_{ij}).
- Cuántas veces cada arco es transitado únicamente por el dron (w_{ij}).

Con estas tres variables, RR realiza un branching jerárquico. Primero, nos fijamos si y_i no es 0 o 1 para algún cliente, y brancheamos en el cliente i cuyo y_i sea más cercano a 0.5. Luego, se generan dos nodos en donde se fuerza que i sea visitado por el dron o por el camión.

Si todos los y_i son 0 o 1, entonces hacemos lo mismo para las variables x_{ij}. En este caso, se generan dos nodos en donde se fuerza que el arco ij sea transitado por el camión o no. Finalmente, si todos los x_{ij} son 0 o 1, chequeamos los w_{ij}, y brancheamos generando dos nodos en donde o bien el arco ij debe ser visitado por el dron, o bien no puede ser visitado por el dron.

La primera partición por las variables y_i mejora las particiones siguientes por x_{ij} y w_{ij} porque toma una decisión inicial que restringe considerablemente cada subespacio, ya que asigna cada cliente a un vehículo.

5. MEJORAS Y APORTES

En esta sección revisaremos las distintas partes del algoritmo de programación dinámica planteado por Roberti and Ruthmair (2021), construyendo sobre su trabajo para lograr un nuevo algoritmo más eficiente, el cual es capáz de resolver instancias de mayor tamaño. En particular, los principales aportes presentados son:

- Nueva definición de ng-factibilidad, la cual resulta en mejores cotas inferiores sin tener mucho impacto en el tiempo requerido para resolver el problema de pricing.
- Nuevas reglas de dominación parcial, las cuales permiten descartar únicamente algunas extensiones de una ruta en lugar de toda la ruta.
- Una versión bidireccional del algoritmo de programación dinámica.

Además, reimplentamos completamente el algoritmo de Branch and Price de Roberti y Ruthmair dado que el código no está disponible.

5.1 Nueva definición de ng-factibilidad

Si observamos las reglas ng definidas en la sección 4.2, podemos ver que para que una ruta sea ng-factible, se toma en cuenta principalmente el camino recorrido por el camión sin dar tanta importancia a los movimientos del dron. Por lo tanto, a pesar de traer buenos resultados, está definición falla en capturar la naturaleza sincrónica del problema, en donde ambos vehículos se mueven simultaneamente por vértices diferentes.

Uno de los principales problemas que surge es que el dron puede visitar a un cliente dos veces sin haber abandonado su vecindario, como muestra la Figura 5.1.

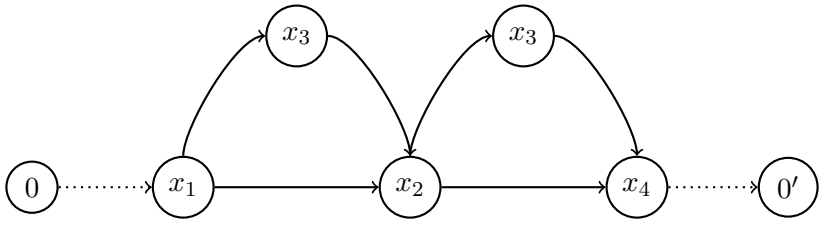

Figura. 5.1: Asumiendo que $x_3 \in N(x_2)$ y $x_3 \notin N(x_4)$, el gráfico representa un ejemplo de una ruta ng-factible que visita dos veces al mismo cliente (x_3) sin antes haber salido de su vecindario

En la ruta de la Figura 5.1, sucede que cuando ambos vehículos se encuentran en x_2, x_3 se encuentra en el conjunto de clientes prohibídos del label correspondiente. Cuando el label se extiende por un arco de camión a x_4, x_3 se quita del conjunto de clientes prohibídos del label obtenido pues $x_3 \notin N_{x_4}$. Por lo tanto, al momento de hacer la extensión del arco del dron para el próximo label, el algoritmo se va a encontrar con que el dron puede avanzar por x_3 nuevamente, ya que dejó de estar prohíbido.

Mirando el problema de forma intuitiva, esta es una situación que no debería darse ya que la ruta del dron por su cuenta no está cumpliendo con el principio de rutas ng. Al moverse de forma sincrónica, no es cierto el camión haya salido del vecindario de x_3 cuando el dron sale de x_2.

Para poder solucionar este problema vamos a agregar un conjunto a los labels, llamado *conjunto de extensiones*. Para un label L, nos vamos a referir a este conjunto como $E(L)$. Este conjunto representa a los clientes a los que el dron se puede mover en la próxima extensión. Utilizando este conjunto, podemos modificar las reglas ng de la siguiente manera: sea L un label tal que el dron está dentro del camión, y $L' = L + [(\mathrm{t}, v)]$ o $L' = L + [(\mathrm{c}, v)]$. Luego, vamos a hacer que $E(L') = E(L) \setminus ng(\mathrm{L})$. De esta forma, estamos prohibiendo que el dron se extienda a clientes que estaban prohibídos al momento en el que ambos vehículos se separaron.

Este cambio logra que las reglas ng comiencen a tener en cuenta al dron en lugar de solamente al camión. Sin embargo, otro problema se da cuando el camión hace recorridos largos por su cuenta: sea L un label en donde el dron está dentro del camión. Es posible encontrar una secuencia de propagaciones que agreguen únicamente arcos de camión a L, de manera que se visite a un cliente w y luego se salga de su vecindario. Por lo tanto, cuando más tarde se agregue un arco de dron, la propagación va a tener permitido visitar a w. Es decir, este caso permite que ambos vehículos visiten al mismo cliente mientras están separados. La Figura 5.2 ejemplifica esta situación.

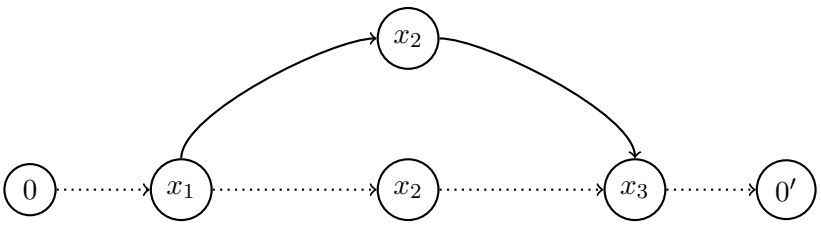

Figura. 5.2: tanto el dron como el camión visitan a un mismo cliente x_2 después de separarse

Como observamos en la figura 5.2, tanto el dron como el camión visitan a un mismo cliente desupués de separarse. Lo que está sucediendo es que en el recorrido de x_2 a x_3 el camión visita a algún cliente que no tiene a x_2 en su vecindario, por lo que al agregar el arco del dron, la extensión a x_2 está permitida.

Tanto este caso como el anterior son ejemplos de rutas que de antemano sabemos que están lejos de ser elementales, por lo que podemos evitar que se generen durante el algoritmo de labeling. Esto hace que mejore mucho el lower bound obtenido sin impactar tanto en la eficiencia del algoritmo.

Nuevamente, este problema puede solucionarse usando el conjunto E de extensiones permitidas. Sea L un label tal que el dron está fuera del camión y $L' = L + [(\mathrm{t}, v)]$. Luego, vamos a hacer que $E(L') = E(L) \setminus v$. Así logramos evitar el segundo problema, en donde el dron visita a un cliente que ya fue visitado por el camión después de que se separaron.

En resumen, nuestras nuevas reglas ng van a considerar dos conjuntos distintos: el conjunto ng definido por RR y el nuevo conjunto E. Para considerar estos cambios debemos modificar las reglas de extension. Sea u_i el peso asociado al cliente i. Luego, comenzamos con el label L tal que $ng(L) = \emptyset$, $k(L) = it(L) = id(L) = \tau(L) = 0$, $c(L) = -u_0$ y $E(L) = N$, y las extensiones son definidas de la siguiente manera:

- *agregar un arco de camión*: Dado un label L, podemos extenderlo a todos los clientes $j \in N \backslash (ng(L) \cup \{it(L)\})$, generando los labels $((ng(L) \cup \{it(L)\}) \cap N_j, k(L) + 1, j, id(L), \tau(L) + t^T_{itj}, c(L) - u_j, E(L) \setminus j)$.
- *agregar un arco combinado*: Dado un label L donde $it(L) = id(L)$, podemos extenderlo a todos los clientes $j \in N \backslash (ng(L) \cup \{it(L)\})$, generando los labels $((ng(L) \cup \{it(L)\}) \cap N_j, k(L) + 1, j, j, 0, c(L) + t^T_{ij} - u_j, N \setminus ((ng(L) \cup \{it(L)\}) \cap N_j))$.
- *agregar un arco de dron*: Dado un label L donde $it(L) \neq id(L)$, podemos extenderlo a todos los clientes $j \in E(L) \backslash (ng(L) \cup \{it(L), id(L)\})$, generando los labels $((ng(L) \cup \{j\}) \cap N_{it}, k(L) + 1, it(L), it(L), 0, c(L) + max(\tau(L), t^D_{idj} + t^D_{jit}) - u_j, N \setminus ((ng(L) \cup \{j\}) \cap N_{it}))$.

Además de la inclusión del nuevo conjunto, una diferencia importante radica en que estamos permitiendo que el camión visite a id cuando ambos vehículos están separados (notar que en la segunda extensión estamos permitiendo que $j = id$). Este cambio nos permite definir nuevas reglas de dominación y una versión bidireccional del algoritmo.

5.2 Reglas de dominación

Como vimos en la Sección 5.1, una forma de reducir el espacio de búsqueda de los algoritmos de labeling es mediante el uso de reglas de dominación. Recordemos que las reglas de dominación son comparaciones mediante las cuales podemos descartar rutas y así dejar de extenderlas.

Notemos sin embargo, dos aspectos de las reglas propuestas por Roberti y Ruthmair. Por un lado, todas las reglas requieren que los camiones de ambos labels estén en el mismo vértice, y que los drones también lo estén. Esto hace que haya muchos pares de labels que no sean comparables entre sí (por ejemplo, cuando los drones de ambos labels están en vértices distintos), aunque podamos probar que toda extensión de un label tiene costo menor a las extensiones del otro. Para mejorar esto y aumentar la cantidad de labels comparables entre sí, vamos a introducir nuevas reglas que no requieren condiciones sobre las posiciones de los drones.

Por otro lado, las reglas de dominación de RR son totales. Es decir, si un label L_1 domina a otro L_2, entonces se están dominando todas las posibles extensiones de L_2, por lo que podemos descartar ese label totalmente. De esta manera, o bien estamos dominando todas las extensiones de un label, o bien no estamos dominando ninguna. En este trabajo utilizamos una generalización de las reglas de dominación, llamada *dominación parcial* (e.g. Lera-Romero et al. (2020)), mediante la cual podemos dominar solamente algunas extensiones de un label.

Por último, las reglas de dominación de RR ya no funcionan bajo nuestra nueva definición de rutas ng. Sin embargo, las mismas pueden adaptarse fácilmente, y en este trabajo

utilizamos una modificación de la primera regla de dominación de RR compatible con la nueva noción de ng-factibilidad.

Definición 3. *Sean $w \in V$ y L_1, L_2 dos labels ng-factibles en donde el dron está fuera del camión. Decimos que L_1 domina a L_2 via w si toda extensión completa s de L_2 en donde el próximo cliente visitado por el dron es w también es una extensión completa de L_1, y además $c(L_1 + s) \leq c(L_2 + s)$*

Definición 4. *Sean L_1 y L_2 dos labels ng-factibles en donde el dron está fuera del camión, y $v, w \in V$. Decimos que L_1 domina a L_2 via w a través de v si para toda extensión de L_2 de la forma $s_2 = [(t, x_1), ..., (t, x_k), (d, w), ...]$ sucede que $s_1 = [(d, v), (c, x_1), ..., (c, x_k), ...]$ es una extensión de L_1, y además $c(L_1 + s_1) \leq c(L_2 + s_2)$*

En la Figura 5.3 podemos ver un ejemplo de la Definición 4. Como menciona la definición, podemos observar que L_2 continúa extendiendo al camión, mientras que L_1 realiza un arco de dron hacia v y luego copia el camino realizado por el camión en L_2 pero mediante arcos combinados.

Notemos que según estas definiciones podemos descartar todas las extensiones que se obtienen desde L_2 cuando el próximo movimiento del dron es un arco hacia w. Es decir, si un label L_1 domina a otro L_2 via w, no estamos descartando al label L_2 por completo, si no que únicamente evitamos que el dron se extienda hacia w. Para poder implementar este tipo de reglas podemos utilizar el conjunto E de extensiones permitidas definido anteriormente. Así, cuando dominamos a un label L via w, simplemente debemos quitar a w de $E(L)$. De esta manera, el conjunto E pasa a cumplir un doble propósito, ya que es parte de la definición de ng-factibilidad, y ahora es utilizado también como estructura de datos para la dominación. Este doble propósito va a traer algunas complicaciones al momento de implementar el algoritmo bidireccional, que ignoramos por el momento (ver Sección 5.3).

Las reglas de dominación parcial propuestas son las siguientes:

Regla de dominación 1: Sean v,$w \in V$, L_p y L_q dos labels ng-factibles en donde el dron está fuera del camión, y sea $\sigma(v)$ el peso asociado a v. Si se cumple que:

1. $ng(L_p) \subseteq ng(L_q)$,
2. $v \notin ng(L_p) \cup N_{it(L_p)}$,
3. $v \in E(L_p)$,
4. $w \in E(L_q)$,
5. $c(L_p) + \tau(L_p) \leq c(L_q) + \tau(L_q)$,
6. $c(L_p) + t^D_{id(L_p),v} + t^D_{v,it(L_p)} \leq c(L_q) + \tau(L_q)$,
7. $\sigma(v) \geq \sigma(w)$.

Entonces L_p domina a L_q via w a través de v.

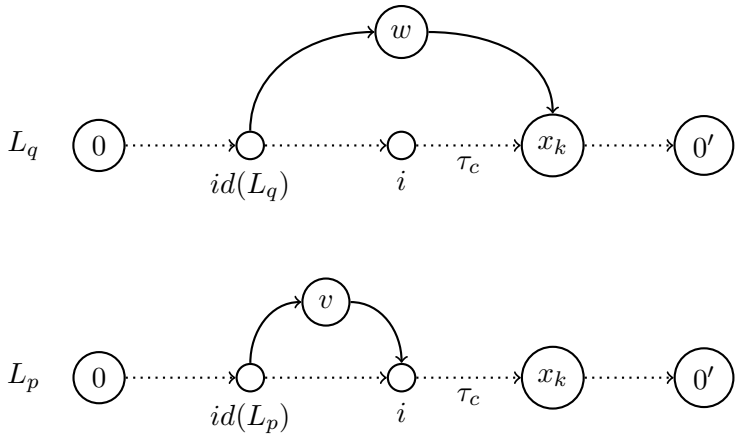

Figura. 5.3: Ejemplo de dominación via w a través de v. Aquí, las extensiones de L_p en donde el dron se extiende a v dominan a todas las extensiones de L_q via w. τ_c representa el costo del tramo realizado por el camión desde i hasta x_k

En la figura podemos ver que, intuitivamente, lo que está pidiendo la regla de dominación es que:

1. Toda extensión que es factible para L_q, también debe ser factible para L_p (condiciones 1, 2, 3, 4).

2. El costo de L_p más su extensión debe ser menor o igual al costo de L_q más su extensión (condiciones 5, 6, 7).

Demostración de la Regla 1: Sean L_p y L_q dos labels que cumplen las condiciones de la Regla 1. Veamos primero que dada una extensión de L_q de la forma $s_q = [(\mathrm{t}, x_1), ..., (\mathrm{t}, x_k), (\mathrm{d}, w)]$, resulta que $s_p = [(\mathrm{d}, v), (\mathrm{c}, x_1), ..., (\mathrm{c}, x_k)]$ es una extensión de L_p. Sea $L'_p = L_p + [(\mathrm{d}, v)]$ (esta extensión es ng-factible por las condiciones 2 y 3, ya que cumple con la definición de las reglas de extensión) y recordemos cómo se actualiza su conjunto de clientes prohibidos ng con respecto a L_p:

$$ng(L'_p) = (ng(L_p) \cup \{v\}) \cap N_i$$

$$E(L'_p) = N \setminus ng(L'_p)$$

Por lo tanto, se tiene que $ng(L'_p) \subseteq ng(L_p)$ pues, por la condición 2, sabemos que $v \notin N_{it(L_p)}$. De esta manera, como por hipótesis L_p tiene un conjunto ng menos restrictivo, se cumple que $ng(L'_p) \subseteq ng(L_p) \subseteq ng(L_q)$.

Veamos qué sucede ahora para cada arco combinado que agregamos a L_p. Recordemos que el conjunto ng se actualiza de la siguiente manera tanto al agregar arcos de camión como arcos combinados:

$$ng' = (ng \cup \{it\}) \cap N_j$$

Luego, por inducción, las extensiones de s_q hasta aquella que visita a x_k pueden ser replicadas como arco combinado en L'_p. Notemos que dada la regla anterior y que $ng(L'_p) \subseteq ng(L_q)$, esta contención seguirá valiendo después de cada extensión, por lo que podemos simular tantos arcos como sea necesario. Por lo tanto, vimos que podemos hacer todas las propagaciones de s_p en orden sobre L_p, lo que significa que s_p es una extensión de L_p.

Veamos ahora que $ng(L_p + s_p) \subseteq ng(L_q + s_q)$. Ya sabemos que $ng(L_p + s_p) \subseteq ng(L_q + [(\mathrm{t}, x_1), ..., (\mathrm{t}, x_k)])$, por lo que falta considerar qué pasa con la última extensión (d,w) de $L_q + s_q$. Es fácil ver que la contención sigue valiendo, ya que luego de agregar un arco de dron, el conjunto ng no puede perder clientes (o bien se mantiene igual, o bien se agrega el cliente visitado). Además, notemos que $E(L_p + s_p) = N \setminus ng(L_p + s_p)$ y $E(L_q + s_q) = N \setminus ng(L_q + s_q)$, por lo que cualquier extensión que se agregue a $L_q + s_q$ para terminar la ruta también es una extensión de $L_p + s_p$.

Luego, podemos afirmar que s_p es una extensión de L_p, y además que toda extensión de $L_q + s_q$ es también una extensión de $L_p + s_p$. Sea τ_c el costo de los movimientos del camión desde i hasta x_k en $L_p + s_p$ y $L_q + s_q$ (ver Figura 5.3), y sea s una extensión de $L_q + s_q$. Como el dron está dentro del camión tanto en $L_q + s_q$ como en $L_p + s_p$, sabemos que $\tau(L_q + s_q) = \tau(L_p + s_p) = 0$, por lo que viendo que $c(L_p + s_p) \leq c(L_q + s_q)$ podemos afirmar que $c(L_p + s_p + s) \leq c(L_q + s_q + s)$:

- $c(L_q + s_q) = c(L_q) + \max\left(\tau(L_q) + \tau_c, t^D_{i_{dq},w} + t^D_{w,i_{tq}}\right) - \sigma(w)$
- $c(L_p + s_p) = c(L_p) + \max\left(\tau(L_p), t^D_{i_{dp},v} + t^D_{v,i_{tp}}\right) + \tau_c - \sigma(v)$

Luego alcanza con ver que

- $c(L_p) + \tau(L_p) + \tau_c - \sigma(v) \leq c(L_q) + \tau(L_q) + \tau_c - \sigma(w)$
- $c(L_p) + t^D_{i_{dp},v} + t^D_{v,i_{tp}} - \sigma(v) \leq c(L_q) + \tau(L_q) - \sigma(w)$

Y simplificando, obtenemos las reglas 5, 6 y 7. □

Notemos que esta regla permite que $L_p = L_q$, por lo que un label puede "autodominarse" si existe una extensión via un cliente v que es mejor que otra via w. Introducimos una nueva definición que permite plantear una regla menos costosa para los casos de autodominación. Es importante notar que esta regla no reemplaza a la regla 1 cuando $L_p = L_q$, si no que ambas reglas se aplican a todos los labels.

Definición 5. *Sea L_1 un label ng-factible en donde el dron está fuera del camión, y $w \in V$. Decimos que $L_2 = L_1 + [(t, x_1)]$ se autodomina via w si para toda extensión de L_2 de la forma $s = [(t, x_2), ..., (t, x_k), (d, w)]$ sucede que $L_1 + [(w, d), (c, x_1), ..., (c, x_k)]$ es ng-factible y domina a $L_2 + s$ (según la primera regla de dominación de RR).*

Regla de dominación 2 (autodominación): Sea L_p un label ng-factible en donde el dron está fuera del camión, $L_q = L_p + (\mathrm{t}, v)$ un sucesor de L_p, y $w \in V$. Si se cumple que:

1. $w \notin N_{it(L_p)}$,

2. $w \in E(L_q)$,
3. $t^D_{id(L_p),w} + t^D_{w,it(L_p)} \leq \tau(L_q)$.

Entonces L_q se autodomina via w.

Notemos que tanto L_p como L_q deben tener al camión y al dron separados. Como además L_p es el label que precede a L_q, entonces necesariamente sucede que el dron está en la misma posición en ambos labels. Es decir, $id(L_p) = id(L_q)$. Esto puede observarse en la Figura 5.4.

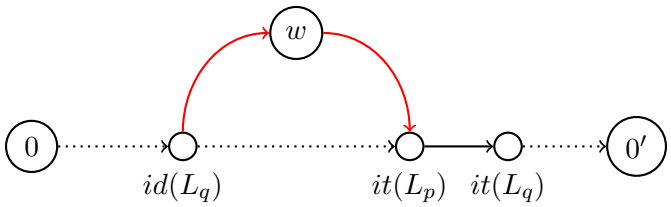

Figura. 5.4: Ejemplo de autodominación. Aquí, L_q es un sucesor de L_p ($L_q = L_p + (\mathrm{t}, v)$) y la extensión de L_p hacia w domina a todas las extensiones via w de L_q.

Lo que está diciendo esta regla es que si el tiempo del recorrido realizado únicamente por el camión en L_q ($\tau(L_q)$) es mayor al tiempo de ambas aristas rojas, entonces la extensión de L_p via w domina a L_q (llamemos a esta extensión L'_q). Esto es fácil de ver pues toda extensión factible desde L_q también lo es desde L'_q, pero además $c(L'_q) \leq c(L_q)$.

Regla de dominación 3: Sean L_p y L_q dos labels ng-factibles en donde el dron está fuera del camión, y sea $w \in V$. Si se cumple que:

1. $ng(L_p) \subseteq ng(L_q)$,
2. $w \in E(L_p) \cap E(L_q)$,
3. $c(L_p) + \tau(L_p) \leq c(L_q) + \tau(L_q)$,
4. $c(L_p) + t^D_{id(L_p),w} \leq c(L_q) + t^D_{id(L_q),w}$.

Entonces L_p domina a L_q via w.

Demostración de la Regla 3: La demostración de esta regla es similar a la de la Regla 1. Sean L_p y L_q dos labels que cumplen con las condiciones de la Regla 3. Comenzamos por ver que toda extensión válida de L_q de la forma $s = [(\mathrm{t}, x_1), ..., (\mathrm{t}, x_k), (\mathrm{d}, w)]$ (es decir, donde el próximo cliente visitado por el dron es w) también es una extensión válida para L_p.

Recordemos como se actualizan los conjuntos ng cuando realizamos una extensión por un arco de camión a un cliente x_i:

$$ng(L'_q) = (ng(L_q) \cup \{it(L_q)\}) \cap N_{x_i}$$

$$ng(L'_p) = (ng(L_p) \cup \{it(L_p)\}) \cap N_{x_i}$$

Es claro a partir de esta definicion que si $ng(L_p) \subseteq ng(L_q)$, entonces $ng(L'_p) \subseteq ng(L'_q)$. Por lo tanto, como por (1) sabemos que $ng(L_p) \subseteq ng(L_q)$, por inducción podemos concluir que $ng(L_p + [(\mathrm{t}, x_1), ..., (\mathrm{t}, x_k)]) \subseteq ng(L_q + [(\mathrm{t}, x_1), ..., (\mathrm{t}, x_k)])$.

En cuanto al conjunto E, por (2) sabemos que la extensión a w está permitida en L_q. Además como s es una extensión de L_q, $L_q + s$ debe ser ng-factible, por lo que necesariamente sucede que $w \in E(L_q + [(\mathrm{t}, x_1), ..., (\mathrm{t}, x_k)])$. Esto significa que $w \neq x_i$ para todo $1 \leq i \leq k$. Por lo tanto, como por (2) sabemos que $w \in E(L_p)$, podemos afirmar que $w \in E(L_p + [(\mathrm{t}, x_1), ..., (\mathrm{t}, x_k)])$. Por otro lado, como vimos en la Regla 1, al agregar el arco de dron en ambas rutas se seguirá cumpliendo la contención de los conjuntos ng, y $E(L_p + s) = N \setminus ng(L_p + s)$ y $E(L_q + s) = N \setminus ng(L_q + s)$, por lo que toda extensión de $L_q + s$ es también una extensión de $L_p + s$.

Sea τ_c el costo de los movimientos del camión desde i hasta x_k en $L_p + s_p$ y $L_q + s_q$. Como en la demostración de la regla 1, nos queda ver que $c(L_p + s) \leq c(L_q + s)$, donde:

- $c(L_q + s) = c(L_q) + \max(\tau(L_q) + \tau_c, t^D_{id(L_q),w} + t^D_{w,it(L_q)}) - \sigma(w)$
- $c(L_p + s) = c(L_p) + \max(\tau(L_p) + \tau_c, t^D_{id(L_p),v} + t^D_{v,it(L_p)}) - \sigma(w)$

Supongamos que $\tau(L_p) + \tau_c \leq t^D_{id(L_p),v} + t^D_{v,it(L_p)}$. Entonces, basta ver que $c(L_p) + t^D_{id(L_p)w} \leq c(L_q) + t^D_{id(L_q)w}$. Por otro lado, si $\tau(L_p) + \tau_c \geq t^D_{id(L_p),v} + t^D_{v,it(L_p)}$, entonces alcanza con probar que $c(L_p) + \tau(L_p) \leq c(L_q) + \tau(L_q)$. Estas expresiones se corresponden con las condiciones 3 y 4, que L_p y L_q cumplen por hipotesis. Por lo tanto, vemos que L_p domina a L_q via w. □

En la Sección 5.2 mencionamos un cambio importante en las reglas de extensión, mediante el cual permitimos que el camión visite al cliente en donde se separó del dron antes de que ambos vehículos vuelvan a unirse. Es en esta regla de dominación en donde utilizamos esto, ya que de no ser así, podríamos tener un caso como el de la Figura 5.5.

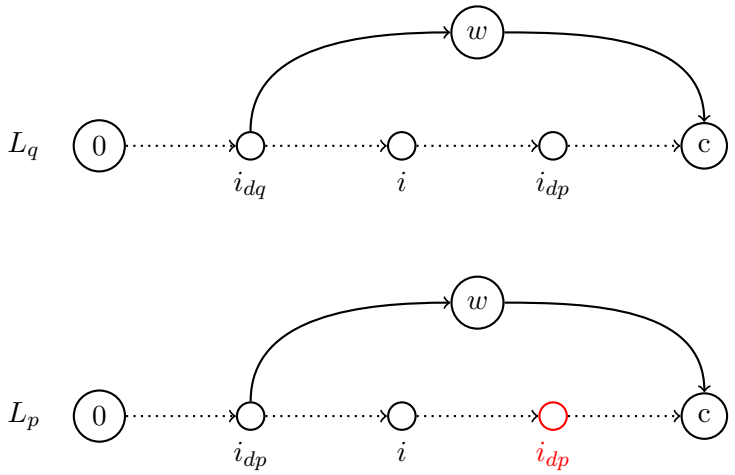

Figura. 5.5: L_p y L_q son dos labels tal que $it(L_p) = it(L_q) = i$, pero hay extensiones de L_q que visitan a i_{dp} antes de realizar el arco de dron, por lo que L_p no podría simular esa extensión si no fuese por el cambio agregado.

Podemos ver que L_q puede tener extensiones en donde el camión visite a i_{dp} por su cuenta, pero esta extensión estaría prohíbida en L_p de no ser por el cambio introducido.

5.3 Labeling bidireccional

El algoritmo de labeling de RR es un algoritmo monodireccional. Es decir, los caminos comienzan a ser generados desde el depósito inicial, y siguen extendiendose hasta alcanzar el depósito final. Righini y Salani (Righini and Salani, 2006) observaron que se puede mejorar mucho un algoritmo de labeling aplicando una búsqueda bidireccional. En este tipo de algoritmos se inicia una búsqueda desde un estado inicial y un estado final, y el algoritmo termina cuando ambas búsquedas se intersecan. En este trabajo proponemos un labeling bidireccional, donde vamos a empezar a extender caminos desde el depósito 0 siguiendo la dirección de los arcos del grafo y desde el depósito $0'$ siguiendo la dirección inversa de los arcos. Para generar la ruta completa, vamos a unir un camino de cada tipo cuando hayan recorrido la mitad del camino aproximadamente.

Recordemos que el problema de TSPD se define sobre un grafo $G = (V, A)$ que, para cada arco $(i, j) \in A$, tiene dos costos t_{ij}^T y t_{ij}^D que representan los costos del camión y del dron para transitar ese arco. Para la implementación del algoritmo bidireccional, vamos a cosiderar la misma red G, pero con los costos invertidos. Es decir, vamos a considerar los costos $t_{ij}^{T^{-1}}$ y $t_{ij}^{D^{-1}}$ tales que $t_{ij}^{T^{-1}} = t_{ji}^T$ y $t_{ij}^{D^{-1}} = t_{ji}^D$. A partir de ahora, vamos a referirnos a los labels sobre A como labels forward, y a los labels sobre A^{-1} como labels backward. Es fácil ver que cualquier ruta forward factible se corresponde con la ruta backward que se obtiene tomando las aristas en el orden inverso y utilizando los costos invertidos (ver Figura 5.6). Sin embargo, es importante notar que las rutas parciales son secuencias de extensiones, las cuales no se pueden invertir directamente para obtener las correspondientes extensiones con los costos invertidos. Por ejemplo, si invertimos las extensiones de la ruta de la Figura 5.6 pasaríamos de tener $[(\mathrm{t}, x_1), (\mathrm{t}, x_2), (\mathrm{d}, x_3), (\mathrm{t}, x_4), (\mathrm{t}, x_5), (\mathrm{d}, x_6), (\mathrm{c}, x_7)]$ a $[(\mathrm{c}, x_7), (\mathrm{d}, x_6), (\mathrm{t}, x_5), (\mathrm{t}, x_4), (\mathrm{d}, x_3), (\mathrm{t}, x_2), (\mathrm{t}, x_1)]$ que no define una ruta válida.

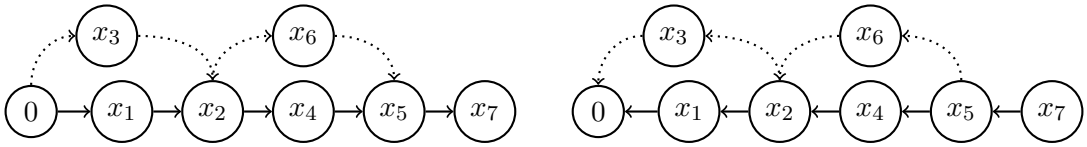

Figura. 5.6: ruta forward junto a su ruta backward correspondiente

Como mencionamos antes, el objetivo del labeling bidireccional va a ser combinar labels forward con labels backward. Sea F un label forward y sea B un label backward. Para poder definir la unión de rutas debemos tener en cuenta la posición de ambos vehículos tanto en F como en B. En este trabajo proponemos los siguientes casos:

1. El dron está dentro del camión tanto en F como en B.
2. El dron está fuera del camión en F
   (a) El dron está dentro del camión en B.
   (b) El dron está fuera del camión en B.

Los casos (2a) y (2b) van a ser equivalentes en este trabajo, pero fueron separados por claridad.

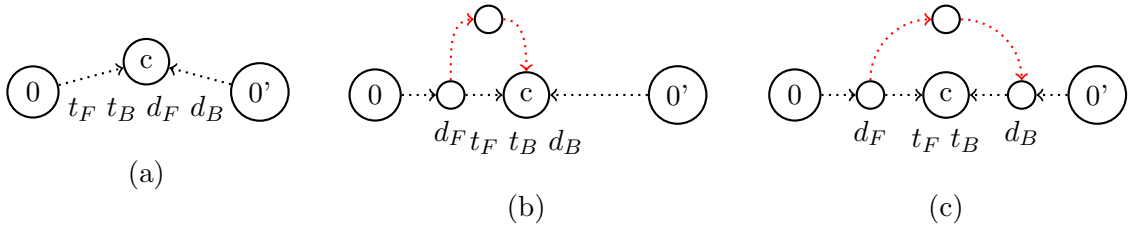

Figura. 5.7: Casos posibles para la unión de un camino forward con otro backward. En estos dibujos, c es el cliente en el que se encuentran los camiones de los labels F y B. En la subfigura (a), el dron se encuentra dentro del camión en ambos labels, por lo que F y B se pueden unir bajo el primer caso. En la subfigura (b), solamente el dron de F se encuentra fuera del camión, por lo que caemos en el caso 2a. Por último, en la subfigura (c) ambos drones se encuentran fuera del camión, por lo que representa al caso 2b.

A partir de estos casos es posible dar una definición para la unión de rutas. Sin embargo, resulta necesario introducir a los labels un nuevo conjunto de extensiones permitidas E^{prop} para garantizar la correctitud de nuestro algoritmo bidireccional. E^{prop} contendrá a todos los clientes salvo por aquellos que hayan sido prohibidos por las reglas de extensión. Es decir, E^{prop} será igual al conjunto E de labels permitidos si ignoramos las reglas de dominación. Esto resulta necesario por un problema en particular: dados un label forward F y un label backward B, puede suceder que por la primera regla de dominación parcial, una extensión de dron por v haya dominado a F via w, y una extensión por w haya dominado a B via v. Esto puede llevar a que algunas rutas no sean generadas al momento de unir labels en el algoritmo bidireccional, lo cual podría hacer que no encontremos una solución óptima.

Teniendo este nuevo conjunto en cuenta, damos la siguiente definición de unión de rutas:

Definición 6. *Sean F y B dos labels forward y backward respectivamente. Sean $t_1^F, ..., t_i^F$ y $t_1^B, ..., t_j^B$ las secuencias de clientes reocorridas por el camión en cada label, y $d_1^F, ..., d_k^F$ y $d_1^B, ..., d_l^B$ las secuencias de clientes reocorridas por el dron. Decimos que F y B se pueden unir si se cumple alguna de las siguientes condiciones:*

- ***Caso 1**: El dron está dentro del camión en F y en B, $t_i^F = t_j^B$, $k(F) + k(B) = |N|$ y $ng(F) \cap ng(B) = \emptyset$. En este caso, definimos a la unión como el label cuyos recorridos realizados por el camión y por el dron son $t_1^F, ..., t_i^F, t_{j-1}^B, ..., t_1^B$ y $d_1^F, ..., d_k^F, d_{l-1}^B, ..., d_1^B$. Vamos a referirnos a este label como $F \cup B$.*
- ***Caso 2a y 2b**: El dron está fuera del camión en F, $t_i^F = t_j^B$, $k(F)+k(B) = |N|-1$, $ng(F) \cap ng(B) = \emptyset$ y existe un nodo v tal que $v \in E^{prop}(F)$ y $v \in E^{prop}(B)$. En este caso, definimos a la unión como el label cuyos recorridos realizados por el camión y por el dron son $t_1^F, ..., t_i^F, t_{j-1}^B, ..., t_1^B$ y $d_1^F, ..., d_k^F, v, d_l^B, ..., d_1^B$. Vamos a referirnos a este label como $F \overset{v}{\cup} B$.*

Teorema 2. *Sean F un label forward ng-factible y B un label backward ng-factible. Si F y B se pueden unir, entonces todas sus uniones son ng-factible.*

Demostración: Vamos a demostrar esta propiedad por inducción en $k(B)$ (la longitud de B).

- Caso Base: si $k(B) = 0$, entonces $F \cup B = F$. Como F es ng-factible, entonces $F \cup B$ también lo es.
- Paso inductivo: Vamos a dividir este paso en distintos casos:
  - Caso 1: Supongamos que F y B se pueden unir siguiendo las condiciones del primer caso de la Definición 6, y sean $t_1^B, ..., t_j^B$ y $d_1^B, ..., d_l^B$ las secuencias de clientes visitadas por el camión y el dron de B respectivamente. Como el dron está dentro del camión en B, la última extensión fue o bien un arco de combinado, o bien un arco de dron. Esto nos genera dos nuevos casos:
    - Caso 1.1: Supongamos que la última extensión de B fue un arco combinado y consideremos la extensión de F hacia t_{j-1}^B mediante un arco combinado. Para ver que esta extensión es válida, supongamos que $t_{j-1}^B \in ng(F)$. Por definición de las reglas ng en las extensiones, esto implica que $t_{j-1}^B \in N_{t_j^B}$, lo cual nuevamente por definición implica que $t_{j-1}^B \in ng(B)$. Sin embargo sabemos por hipótesis que $ng(F) \cap ng(B) = \emptyset$, pero $t_{j-1}^B \in ng(B)$ y $t_{j-1}^B \in ng(F)$. Esto es un absurdo que surge de suponer que $t_{j-1}^B \in ng(F)$. Por lo tanto, $t_{j-1}^B \notin ng(F)$, y F se puede extender por un arco combinado hacia $F' = F + [(\mathrm{c}, t_{j-1}^B)]$.
      Sea B' el label tal que $B = B' + [(\mathrm{c}, t_j^B)]$. Veamos primero que F' y B' se pueden unir. Queremos ver que $ng(F') \cap ng(B') = \emptyset$. Sea x un cliente tal que $x \in ng(F')$. Luego, sabemos que o bien $x \in ng(F)$ o bien $x = t_j^B$. Sin embargo, sabemos que $t_j^B \notin ng(B')$ ya que de ser así, B' no podría extenderse hacia t_j^B para obtener B. Por otro lado, si $x \in ng(F)$ entonces necesariamente $x \in N_{t_j^B}$. Supongamos que $x \in ng(B')$ y $x \in ng(F)$. Esto quiere decir que $x \in ng(B)$ pues $x \in N_{t_j^B}$, pero entonces $\{x\} \subseteq ng(F) \cap ng(B)$. Esto es un absurdo, por lo que no sucede que $x \in ng(B')$ y $x \in ng(F)$. Luego, vimos que ya sea que si $x \in ng(F)$ o $x = t_j^F$, entonces $x \notin ng(B')$. Por lo tanto, F' y B' son dos labels que se pueden unir siguiendo las condiciones del primer caso de la Definición 6, y donde $k(B') < k(B)$. Luego, por definición, $F \cup B = F' \cup B'$ es ng-factible por hipótesis inductiva.
    - Caso 1.2: Supongamos que la última extensión de B fue un arco de dron. Esto significa que debemos extender a F hacia t_{j-1}^B mediante un arco de camión. La justificación de que F se puede extender a $F' = F + [(\mathrm{t}, t_{j-1}^B)]$ es análoga a la del caso anterior.
      Sea B' el predecesor de B en el que el camión visitó a t_{j-1}^B por última vez. Notemos que $k(B') = k(B) - 2$ ya que en B' no se realizaron ni el último arco de dron, ni el último arco de camión. Al igual que en el caso anterior, queremos ver que $ng(F') \cap ng(B') = \emptyset$. Sea x un cliente tal que $x \in ng(F')$. Luego, sabemos que o bien $x \in ng(F)$ o bien $x = t_j^B$. Sin embargo, sabemos

que $t_j^B \notin ng(B')$ ya que de ser así, B' no podría extenderse hacia t_j^B para obtener B. Por otro lado, si $x \in ng(F)$ entonces necesariamente $x \in N_{t_j^B}$. Supongamos que $x \in ng(B')$ y $x \in ng(F)$. Esto quiere decir que $x \in ng(B)$ pues $x \in N_{t_j^B}$, pero entonces $\{x\} \subseteq ng(F) \cap ng(B)$. Esto es un absurdo, por lo que no sucede que $x \in ng(B')$ y $x \in ng(F)$. Por lo tanto, F' y B' son dos labels que se pueden unir siguiendo las condiciones del segundo caso de la Definición 6, y donde $k(B') < k(B)$. Luego, por definición, $F \cup B = F' \overset{d_l^B}{\cup} B'$ es ng-factible por hipótesis inductiva.

- Caso 2: Sean F y B labels que se pueden unir siguiendo las condiciones del segundo caso de la definición 6, y sean $t_1^B, ..., t_j^B$ y $d_1^B, ..., d_l^B$ las secuencias de clientes visitadas por el camión y el dron de B respectivamente. Nuevamente, dividimos la demostración en dos casos:
  - Caso 2.1: Supongamos que la última extensión de B fue un arco de dron o combinado. Esto quiere decir que el dron está dentro del camión en B. Sea $v \in V$ que cumple que $v \in E^{prop}(F)$ y $v \in E^{prop}(B)$ (sabemos que existe por hipótesis) y sea $F' = F + [(\mathrm{d}, v)]$. Queremos ver que $ng(F') \cap ng(B) = \emptyset$. Como el dron está dentro del camión en B y $v \in E^{prop}(B)$, esto quiere decir que $v \notin ng(B)$. Como v es el único cliente que se podría haber agregado a $ng(F')$, podemos afirmar que $ng(F') \cap ng(B) = \emptyset$. Por lo tanto, F' y $B' = B$ son dos labels que se pueden unir siguiendo las condiciones del primer caso de la definición 6, y donde $k(B') = k(B)$. Luego, por definición, $F \overset{v}{\cup} B = F' \cup B'$ es ng-factible por hipótesis inductiva.
  - Caso 2.2: Supongamos que la última extensión de B fue un arco de camión. Consideremos la extensión de F hacia t_{j-1}^B mediante un arco de camión. Sean Sea B' el label tal que $B = B' + [(\mathrm{t}, t_j^B)]$, y $F' = F + [(\mathrm{t}, t_{j-1}^B)]$. Queremos ver que F' y B' se pueden unir. Comencemos por ver que $ng(F') \cap ng(B') = \emptyset$. Sea x un cliente tal que $x \in ng(F')$. Luego, sabemos que o bien $x \in ng(F)$ o bien $x = t_j^B$. Sin embargo, sabemos que $t_j^B \notin ng(B')$ ya que de ser así, B' no podría extenderse hacia t_j^B para obtener B. Por otro lado, si $x \in ng(F)$ encontes necesariamente $x \in N_{t_j^B}$. Supongamos que $x \in ng(B')$ y $x \in ng(F)$. Esto quiere decir que $x \in ng(B)$ pues $x \in N_{t_j^B}$, pero entonces $\{x\} \subseteq ng(F) \cap ng(B)$. Esto es un absurdo, por lo que no sucede que $x \in ng(B')$ y $x \in ng(F)$. Luego vimos que ya sea que si $x \in ng(F)$ o $x = t_j^F$, entonces $x \notin ng(B')$, por lo que $ng(F') \cap ng(B') = \emptyset$. Por otro lado, sabemos que $E^{prop}(F') = E^{prop}(F) \setminus t_{j-1}^B$ y que $E^{prop}(B) \subseteq E^{prop}(B')$. Esto quiere decir que todo v que cumpla que $v \in E^{prop}(F)$ y $v \in E^{prop}(B)$ cumple también que $v \in E^{prop}(F')$ y $v \in E^{prop}(B')$. Por lo tanto, F' y B' son dos labels que se pueden unir siguiendo las condiciones del segundo caso de la definición 6, y donde $k(B') < k(B)$. Luego, por definición, $F \overset{v}{\cup} B = F' \overset{v}{\cup} B'$ es ng-factible por hipótesis inductiva.

Notemos que en el caso 1.2 estamos quitando el arco de dron que visita al cliente d_l^B. Sin embargo, este cliente no será agregado a $E^{prop}(F)$ hasta que eventualmente caigamos en el caso 2.1 (lo cual necesariamente sucede en algún punto, porque el dron comienza el recorrido dentro del camión) por lo que d_l^B será uno de los clientes v que cumplen estar

tanto en $E^{prop}(F)$ como $E^{prop}(B)$. □

Veamos ahora que vale la vuelta. Es decir, que dado un label L ng-factible, existen labels forward y backward tal que uniéndolos se obtiene a L. Para esto, vamos a introducir una nueva definición:

Definición 7. *Sea L un label ng-factible y s una extensión completa de L en donde la secuencia de clientes visitados en s por el camión es $t_1, ..., t_k$, y por el dron es $d_1, ...d_l$. Sea además d el último cliente visitado por el camión en L. Definimos al reverso de s en L como s^r_L, donde s^r_L se genera de la siguiente manera:*

1. *Si el dron está fuera del camión en L, quitamos el primer arco de dron de s generando la secuencia s'.*
2. *Generamos el reverso de s'.*
3. *Recorremos el reverso de s' desde su primer índice hasta el final, y cada vez que encontramos un arco de dron seguido por un arco de camión, swapeamos las posiciones de estos arcos. (Por ejemplo, $[(d, x_1), (t, x_2), (t, x_3), (d, x_4), (t, x_5), (t, x_6), (c, x_7)]$ pasa a ser $[(t, x_2), (t, x_3), (d, x_1), (t, x_5), (t, x_6), (d, x_4), (c, x_7)]$).*
4. *Reemplazamos cada elemento de la forma (t, t_i) y (c, t_i) por (t, t_{i-1}) y (c, t_{i-1}) respectivamente (consideramos que $t_0 = d$).*

Veamos un ejemplo de esto. Sea $s = [(\mathrm{t}, t_1), (\mathrm{t}, t_2), (\mathrm{d}, d_1), (\mathrm{c}, t_3), (\mathrm{t}, t_4), (\mathrm{d}, d_2)]$. Si seguimos los pasos de la definición, vamos a obtener la lista $[(\mathrm{t}, t_3), (\mathrm{d}, d_2), (\mathrm{c}, t_2), (\mathrm{t}, t_1), (\mathrm{t}, c), (\mathrm{d}, d_1)]$.

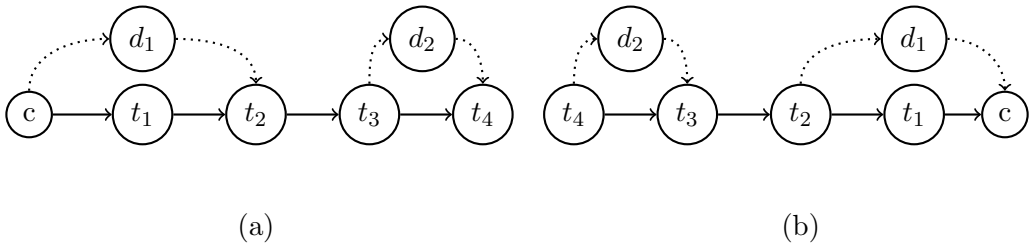

Figura. 5.8: Ejemplo de ruta $s = [(\mathrm{t}, t_1), (\mathrm{t}, t_2), (\mathrm{d}, d_1), (\mathrm{c}, t_3), (\mathrm{t}, t_4), (\mathrm{d}, d_2)]$ (a) y su inverso (b)

Intuitivamente, dado un label forward F, esta definición nos va a permitir reemplazar extensiones completas de F por la unión de F con un label backward. Más específicamente, dado un label forward F y una extensión completa s de F, se tiene que $F + s = F \cup B$ o $F + s = F \overset{v}{\cup} B$ para algún v, donde $B = L_\emptyset + s^r_F$ y $L_\emptyset$ es el label vacío. Esto resulta particularmente útil pues nos permite ver que las rutas generadas por el algoritmo monodireccional también son generadas por el algoritmo bidireccional (si ignoramos las reglas de dominación). Las siguientes propiedades nos permiten formalizar esta idea.

Propiedad 1. *Sea L un label ng-factible, s una extensión completa de L y $L' = L_\emptyset + s^r_L$. Luego, $ng_L \cap ng_{L'} = \emptyset$.*

Demostración: Supongamos que existe un cliente c tal que $c \in ng(L) \cap ng(L')$. Esto quiere decir que tanto en L como L' se visitó a c en algún punto y luego nunca se abandonó su vecindario. Sea $t_1, ..., t_k$ la secuencia de clientes visitados por el camión en L', y sean t_i la posición en la que se encontraba el camión cuando c fue visitado por última vez. Separamos esta demostración en dos casos:

- La última visita a c en L' fue mediante un arco de camión o combinado ($t_i = c$): Sabemos que $L+s$ debe visitar a los clientes $t_1, ..., t_k$ en orden inverso. En particular, debe visitar a los clientes $t_k, ..., t_i$. Sin embargo, sabemos que $c \in ng(L)$ y que no se abandonará su vecindario hasta llegar a t_i en $L + s$, por lo que $L + s$ no es ng-factible. Esto es un absurdo que surge de suponer que existe un cliente c tal que $c \in ng(L) \cap ng(L')$.

- La última visita a c en L' fue mediante un arco de dron ($t_i \neq c$): Como en el caso anterior, sabemos que $L+s$ debe visitar a los clientes $t_k, ..., t_i$. Sin embargo, notemos que la próxima extensión después de visitar a t_i en $L + s$ necesariamente es un arco de camión, y como c está en el conjunto de elementos prohibidos ng, se lo va a quitar del conjunto de extensiones permitidas E^{prop}. Esto significa que cuando $L+s$ realiza esta visita a c mediante un arco de dron, c no va a ser una extensión permitida. Como sabemos que $L + s$ es ng-factible, esto es un absurdo. □

Propiedad 2. *Sea $L_\emptyset$ el label vacío, L un label forward ng-factible y s una extensión completa de L. Luego, $L_\emptyset + s^r_L$ es un label backward ng-factible.*

Demostración: Supongamos que $L' = L_\emptyset + s^r_{L_\emptyset}$ no es ng-factible. Es decir, debe suceder alguna de las siguientes cosas:

1. Existe un cliente c tal que L' lo visita y luego lo vuelve a visitar sin haberse salido de su vecindario.

2. Se realiza un arco de dron hacia un cliente prohibido por el conjunto E^{prop}.

Supongamos primero que sucede (1). Sea $t_1, ..., t_k$ la secuencia de clientes visitados por el camión en L', y sean t_i y t_j ($i < j$) dos posiciones en las que se encontraba el camión cuando c fue visitado, y en donde $c \in N_{t_k}$ para todo $i < k < j$. Separamos esta demostración en dos casos:

- La primera visita a c fue mediante un arco de camión o combinado ($t_i = c$): Sabemos que $L + s$ debe visitar a los clientes $t_1, ..., t_k$ en orden inverso. En particular, debe visitar a los clientes $t_j, ..., t_i$. Luego, en algún punto entre t_j y t_{i+1} se va a visitar a c, y no se va a salir de su vecindario hasta llegar $t_i = c$, por lo que $L + s$ no es ng-factible. Esto es un absurdo, por lo que no existe tal cliente c.

- La primera visita a c fue mediante un arco de dron ($t_i \neq c$): Como en el caso anterior, sabemos que $L+s$ debe visitar a los clientes $t_j, ..., t_i$. Luego, cuando en $L+s$ el camión se encuentra en t_j se visitará a c y no se saldrá de su vecindario hasta llegar a t_i. Sin embargo, notemos que la próxima extensión después de visitar a t_i necesariamente es un arco de camión, y como c está en el conjunto de elementos prohibidos ng, se lo

va a quitar del conjunto de extensiones permitidas E^{prop}. Esto significa que cuando $L+s$ visita a c por segunda vez mediante un arco de dron, c no va a ser una extensión permitida. Como sabemos que $L+s$ es ng-factible, esto es un absurdo.

Por lo tanto, ya demostramos que no sucede (1). Supongamos ahora que sucede (2). Sea d el cliente que fue visitado por el dron a pesar de no ser una extensión permitida, sea $t_1, ..., t_k$ la secuencia de clientes visitados por el camión en L', y t_i y t_j las posiciones en las que se encontraba el camión cuando el dron se desprendió y se volvió a unir al camión para visitar a d respectivamente. Sea p el predecesor de L' y q el predecesor de $L+s$ en los cuales ambos vehículos se encuentran en t_i. Como el arco de dron hacia d viola la ng-factibilidad, o bien $d \in \{t_{i+1}, ..., t_{j-1}\}$ o bien $d \in ng(p)$. Es fácil ver que $d \notin \{t_{i+1}, ..., t_{j-1}\}$ pues de ser así, el arco de dron hacia d tampoco sería válido en $L+s$. Supongamos entonces que $d \in ng(p)$. Notemos que esto quiere decir que $d \in N_{t_i}$, por lo que necesariamente $d \in ng(q)$ (la última extensión de q fue un arco de dron hacia d). Sin embargo esto es un absurdo, pues por la Propiedad 1 sabemos que $ng(p) \cap ng(q) = \emptyset$. Por lo tanto, $d \notin ng(p)$. Luego, como no sucede ni (1) ni (2), se tiene que L' es ng-factible. □

Con esta nueva definición podemos pasar a demostrar que dado un label L ng-factible, existen labels forward y backward tal que uniéndolos se obtiene a L. Notemos que esto no significa que L necesariamente debe ser generado por el algoritmo bidireccional, ya que tanto el label forward como el backward pueden haber sido dominados. Sin embargo, demostramos esta propiedad porque nos es útil más adelante.

Propiedad 3. *Sea L un label ng-factible que comienza en el depósito inicial y termina en el depósito final, y F un predecesor de L. Luego, existe un label backward B ng-factible tal que o bien $F \cup B = L$ o bien $F \overset{v}{\cup} B = L$ para algún cliente v.*

Demostración: Sea F un predecesor de L. Es claro que F es ng-factible, pues de no ser así, L tampoco lo sería. Dividimos la demostración en dos casos:

- Supongamos que la última extensión realizada por F fue un arco combinado o de dron. Es decir, el dron se encuentra dentro del camión. Sea s la extensión de F tal que $F+s = L$, y sea $B = L_\emptyset + s^r_L$. Como ya vimos, B es ng-factible por la Propiedad 2, y $ng(F) \cap ng(B) = \emptyset$ por Propiedad 1. Luego, F y B se pueden unir, y $F \cup B = L$.

- Supongamos que la última extensión realizada por F fue un arco de camión. Sea s la extensión de F tal que $F + s = L$, y sea $B = L_\emptyset + s^r_L$ que es ng-factible por Propiedad 2 y $ng(F) \cap ng(B) = \emptyset$ por Propiedad 1. Como el dron esta fuera del camión en F, por definición también está fuera del camión en B (recordar primer paso en la definición de s^r_L). Sea v el cliente visitado por el dron en la extensión que fue quitada de s^r_L, y veamos que $v \in E^{prop}(B)$.
  Supongamos que $v \notin E^{prop}(B)$, lo que quiere decir que o bien el camión en B visita a v luego de separarse del dron, o bien v se encontraba en el conjunto de clientes prohibidos ng en el momento en que ambos vehículos se separan. Es claro que lo primero no sucede, pues de ser así, el arco de dron a v estaría prohibido también en $F + s$.
  Para ver que no sucede lo segundo, tomamos a B' como el último predecesor de B en donde ambos vehículos estaban juntos y supongamos que $v \in ng(B')$. Por

un argumento similar al de la proposición anterior, esto significa que B' visitó a v anteriormente y aún no se salió de su vecindario, por lo que en $F+s$ pasaría lo mismo y por tanto $F+s$ no es ng-factible. Esto es un absurdo, por lo que $v \in E^{prop}(B)$. Luego, como $ng(F) \cap ng(B) = \emptyset$, $v \in E^{prop}(F)$ y $v \in E^{prop}(B)$, F y B se pueden unir y $F \overset{v}{\cup} B = L$. $\square$

A continuación presentamos una última definición que nos va a ser de utilidad para demostrar el Teorema 3.

Definición 8. *Dado un label L, definimos al label $\lfloor L \rfloor$ como:*

$$\lfloor L \rfloor = \begin{cases} L \text{ si el dron esta dentro del camion en } L \\ L \text{ pero reemplazando los últimos arcos de camión por arcos combinados si el dron} \\ \text{está fuera del camión en } L \end{cases}$$

Por ejemplo, $\lfloor [(\mathrm{t}, x_1), (\mathrm{t}, x_2), (\mathrm{d}, x_3), (\mathrm{c}, x_4), (\mathrm{t}, x_5), (\mathrm{t}, x_6)] \rfloor = [(\mathrm{t}, x_1), (\mathrm{t}, x_2), (\mathrm{d}, x_3), (\mathrm{c}, x_4), (\mathrm{c}, x_5), (\mathrm{c}, x_6)]$

El siguiente teorema nos indica que el algoritmo bidireccional siempre genera al menos una ruta de mínimo costo reducido. Es decir, que es correcto.

Teorema 3. *Dada un label L generado por el algoritmo monodireccional, existe una ruta L' generada por el algoritmo bidireccional que cumple que $c(L') \leq c(L)$.*

Demostración: Vamos a separar la demostración en casos:

- Caso 1: Por la Propiedad 3, existen F y B tal que $L = F \cup B$ y $1 \leq k(F) < |N|$. Como L es un label generado por el algoritmo monodireccional, entonces sabemos que F necesariamente debe ser generado por la parte forward del algoritmo bidireccional. Por otro lado, si la parte backward genera a B, entonces B es ng-factible y podemos tomar $L' = F \cup B = L$, y $c(L') \leq c(L)$.
  Supongamos que el algoritmo backward no genera a B, o bien lo genera pero B es descartado por ser dominado. Queremos ver que existe una ruta B^* generada por el algoritmo backward tal que $c(B^*) \leq c(B)$, $ng(B^*) \subseteq ng(B)$ y $it(B^*) = id(B^*) = it(B)$.
  Supongamos primero que B es generado pero es dominado por la primera regla de dominación de RR. Sea B_1 el label que domina a B en el algoritmo, y sea $B, B_1, B_2, \ldots, B_k$ la secuencia de dominaciones más larga generada por el algoritmo que cumple que B_{i+1} domina a B_i por la primera regla de RR. Notemos que esta secuencia es finita ya que existe una cantidad acotada de labels de largo $k(B)$. Además, como el algoritmo descarta un label luego de ser dominado, no puede pasar que un label B_i domine a otro B_j y luego B_j domine a B_i. Es decir, no hay ciclos de dominación bajo esta regla. Luego, podemos tomar $B^* = B_k$ y $L' = F \cup B^*$ que cumple que $c(L') \leq c(L)$.
  Supongamos ahora que el algoritmo backward no llega a generar a B. Sea B_p un label generado por el algoritmo que cumple que $B_p + s = B$ para alguna extensión s, pero cuya extensión por s fue dominada. Sea $B_p, B_1, B_2, \ldots, B_k$ la secuencia de dominaciones más larga generada por el algoritmo que cumple que:

- Si B_i domina a B_{i-1} via w, entonces B_{i+1} domina la extensión por w de B_i.
- Si B_i domina a B_{i-1} via w a través de v, entonces B_{i+1} domina la extensión por v de B_i.
- Si B_i domina a B_{i-1} por la primera regla de RR, entonces B_{i+1} domina a B_i por la primera regla de RR.

Notemos que a pesar de que por la primera regla de dominación parcial puede haber labels repetidos en esta secuencia, la secuencia es finita ya que existe una cantidad acotada de labels de largo $k(B)$, cada una con una cantidad acotada de extensiones. Luego, existe una extensión s' de B_k tal que $B' = B_k + s'$ se puede unir con F y $c(F \cup B') \leq c(L)$. Notemos que es posible que B' no sea generado por el algoritmo. Sin embargo, el predecesor más largo de B' que genera el algoritmo visita a al menos un cliente más que B_p. Luego, podemos repetir este proceso hasta que eventualmente llegamos a un label B^* con $k(B^*) = k(B)$ que es generado por el algoritmo, y que cumple que $c(L) \geq c(F \cup B^*)$. Como vimos antes, aunque B^* sea dominado por la primera regla de RR, podemos obtener un label L' generado por el algoritmo tal que $c(L') \leq c(F \cup B^*) \leq c(L)$.

- Caso 2: Por la propiedad 3 existen F y B tal que $L = F \overset{v}{\cup} B$ para algún $v \in V$. Como L es un label generado por el algoritmo monodireccional, entonces sabemos que F necesariamente debe ser generado por la parte forward del algoritmo bidireccional. Además, necesariamente debe suceder que $v \in E^{prop}(F)$. Por otro lado, si la parte backward genera a B, entonces podemos unirlos obteniendo una nueva ruta $L' = L$ y tal que $c(L') \leq c(L)$.
  Supongamos que el algoritmo backward no genera a B, o bien lo genera pero B es descartado por ser dominado. Vamos a dividir la demostración en dos casos:

  - El dron está dentro del camión en B: Podemos demostrar que existe un B^* tal que $c(F \overset{v}{\cup} B^*) \leq c(L)$ de manera análoga al primer caso. Lo único que falta notar es que necesariamente $v \in E^{prop}(B^*)$ ya que el dron está dentro del camión en B^*, y por las reglas de dominación $ng(B^*) \subseteq ng(B)$.
  - El dron está fuera del camión en B: Supongamos primero que se genera el label B. Notemos que mientras exista alguna extensión de dron sin dominar en B, podemos unir a F con B pues la unión considera a los conjuntos E^{prop}, en donde no se prohíben las extensiones dominadas. Supongamos entonces que el label B se descarta porque se dominaron todas sus extensiones. Sea $B, B_1, \ldots, B_k$ la secuencia de dominaciones más larga generada por el algoritmo donde B_1 domina la extensión via v en B y además se cumple que:
    - Si B_i domina a B_{i-1} via w, entonces B_{i+1} domina la extensión por w de B_i.
    - Si B_i domina a B_{i-1} via w a través de u, entonces B_{i+1} domina la extensión por u de B_i.

    Si todas las dominaciones son a través de la tercera regla, entonces podemos tomar $L' = F \overset{v}{\cup} B_k$ que cumple que $c(L') \leq c(L)$. Sin embargo, si hay una dominación a través de la primera regla surge un problema. Cómo se muestra en la Figura 5.9, si B_k domina a B via v a través de u, entonces el label backward

debe unir al dron con el camión a través de un arco u. Sea $B' = B_k + [(\mathrm{d}, u)]$. Notemos que B' ya no se puede unir con F porque $k(F) + k(B') = |N|$ y el dron está fuera del camión en F.

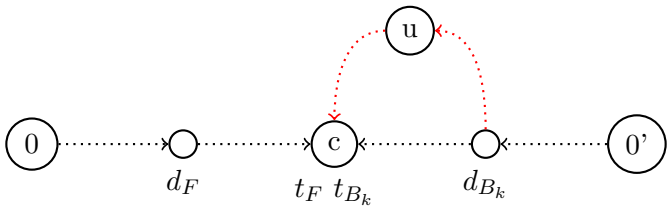

Figura. 5.9: El label B_k domina a B via v a través de u. Esto significa que la extensión de B_k que domina a la de B primero une a los vehículos con un arco de dron vía u, y luego se realizan arcos combinados hasta d_f.

Sin embargo, por definición de dominación sabemos que $c(\lfloor F \rfloor \cup B') \leq c(F \overset{v}{\cup} B)$. Luego, si $\lfloor F \rfloor$ es generado por el algoritmo, podemos tomar $L' = \lfloor F \rfloor \cup B'$. Por otro lado, si no es generado por el algoritmo, siguiendo los mismos pasos que en el primer caso podemos encontrar un label F' generado por el algoritmo tal que $c(L') \leq c(L)$ con $L' = F' \cup B'$. Por lo tanto, vimos que si el algoritmo genera a B entonces podemos encontrar un label L' generado por el algoritmo tal que $c(L') \leq c(L)$ a pesar de que B sea descartado.
Supongamos ahora que el algoritmo backward no llega a generar a B. Sea B_p un label generado por el algoritmo que cumple que $B_p + s = B$ para alguna extensión s, pero cuya extensión por s fue dominada. Sea $B_p, B_1, \ldots, B_k$ la secuencia de dominaciones más larga generada por el algoritmo que cumple que:

- Si B_i domina a B_{i-1} via w, entonces B_{i+1} domina la extensión por w de B_i.
- Si B_i domina a B_{i-1} via w a través de u, entonces B_{i+1} domina la extensión por u de B_i.

Al igual que en los casos anteriores, se puede ver que esta secuencia es finita. Notemos que de haber alguna dominación a través de la primera regla en esta secuencia, es posible que nuevamente caigamos en un caso como el de la Figura 5.9. Luego, si no hay ninguna de estas dominaciones, podemos continuar por inducción para encontrar algún label B^* tal que $L' = F \overset{v}{\cup} B^*$ cumple que $c(L') \leq c(L)$. Caso contrario, continuamos por inducción hasta encontrar un label B^* tal que $L' = \lfloor F \rfloor \cup B^*$ cumple que $c(L') \leq c(L)$. □

El Teorema 3 nos indica que el algoritmo bidireccional es correcto, ya que siempre que el algoritmo monodireccional encuentre una ruta de mínimo costo reducido, también lo hará el algoritmo bidireccional.

6. EXPERIMENTOS COMPUTACIONALES

En esta sección describimos los resultados computacionales obtenidos por el algoritmo propuesto. Primero realizaremos una descripción de los casos de test utilizados en los experimentos. Luego analizaremos los resultados obtenidos por el algoritmo propuesto, y por último compararemos estos resultados con los presentados por Roberti and Ruthmair (2021).

6.1 Casos de test

En este trabajo utilizamos las instancias introducidas por Poikonen et al. (2019) y utilizadas por Roberti and Ruthmair (2021) Este conjunto de tests consiste de instancias con 9, 19, 29 y 39 clientes, con 25 instancias para cada tamaño. Las instancias fueron generadas tomando puntos al azar en un rectángulo de dimensión 100×100. Dados dos clientes i y j y sus coordinadas correspondientes, calculamos los costos para el camión y para el dron de la siguiente manera:

- $t_{ij}^T = \lfloor |x_i - x_j| + |y_i - y_j| \rfloor$
- $t_{ij}^D = \left\lfloor \frac{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{\alpha} \right\rfloor$

Es decir, el camión sigue la norma 1, mientras que el dron sigue la distancia euclídea moviendose a una distancia que es α veces más rápida que el camión. Todas las instancias fueron testeadas para los valores $\alpha = 1, 2, 3$. [1]. En estas experimentaciones, al igual que Roberti y Ruthmair, tomamos a los N_i como los cinco vecinos más cercano a i.

[1] Roberti y Ruthmair toman α=3.003003 porque en lugar de dividir por 3, multiplican por 0.333 para evitar errores numéricos. Esto hace que el óptimo de una instancia dé distinto.

6.2 Resultados para el algoritmo propuesto

		Algoritmo bidireccional			Algoritmo monodireccional		
n	α	Opt	Cpu	BBnodes	Opt	Cpu	BBnodes
9	1	25	0.14	1.24	25	0.15	1.16
	2	25	0.12	1.44	25	0.15	1.16
	3	25	0.11	1.56	25	0.13	2.6
19	1	25	11.36	1.08	25	13.83	1.32
	2	25	5.05	2.12	25	8.62	2.92
	3	25	7.04	5.32	25	14.19	9.2
29	1	25	562.06	2.76	25	849.91	2.2
	2	25	71.50	3.4	25	236.214	5.64
	3	25	76.29	9.28	25	271.84	17.88
39	1	22	1704.18	4	18	2501.41	1.88
	2	25	433.90	5.8	22	1375.37	6.86
	3	24	311.56	12.58	12	1495.97	15.33

Tab. 6.1: Resultados computacionales para el TSPD

La Tabla 6.1 muestra los resultados obtenidos por el algoritmo de TSPD, tanto en su versión bidireccional como en su versión monodirecional. La tabla resume los resultados para cada tamaño de instancia y para cada valor de α. La columna **Opt** indica la cantidad de instancias en las que se llega a la solución óptima dentro del límite de tiempo impuesto. La columna **Cpu** muestra el tiempo promedio de cómputo sobre las instancias resueltas medido en segundos, mientras que la columna **BBnodes** es un promedio del número de nodos de branch and bound que son explorados en las instancias resueltas.

Podemos ver que tanto el algoritmo bidireccional como el monodireccional resuelven todas las instancias de tamaños menores o iguales a 29. Sin embargo, para las instancias de tamaño 39 empieza a notarse la ventaja del algoritmo bidireccional, que resuelve 71 instancias. Por otro lado, el algoritmo monodireccional resuelve tan solo 52 instancias. Además, dentro de las instancias que son resueltas se observa que el algoritmo bidireccional tiene un tiempo de cómputo mucho menor, especialmente en las instancias de mayor tamaño. Por ejemplo, en el caso con $n = 29, \alpha = 3$ vemos que el algoritmo bidireccional llega a ser más de 3 veces más rápido.

Otra tendencia que puede observarse en la tabla 6.1 es que los casos en donde el dron se mueve más rápido obtenemos menores tiempos de cómputo. Esto probablemente se deba a que, como casi siempre va a convenir utilizar al dron, las rutas que no lo utilizan son dominadas rápidamente. Esto hace que se estén generando una menor cantidad total de labels, lo cual mejora significativamente la performance del algoritmo.

En la Figura 6.1 podemos observar el tiempo que toma resolver cada una de las 300 instancias tanto para el algoritmo bidireccional como para el monodireccional. Podemos observar que para $n = 29$ la gran mayoría de las instancias se resuelve en 500 segundos o menos. Esto puede notarse en los colores más claros, que simbolizan una mayor densidad de puntos. Por otro lado, los casos que toman más de 1000 segundos son casos aislados.

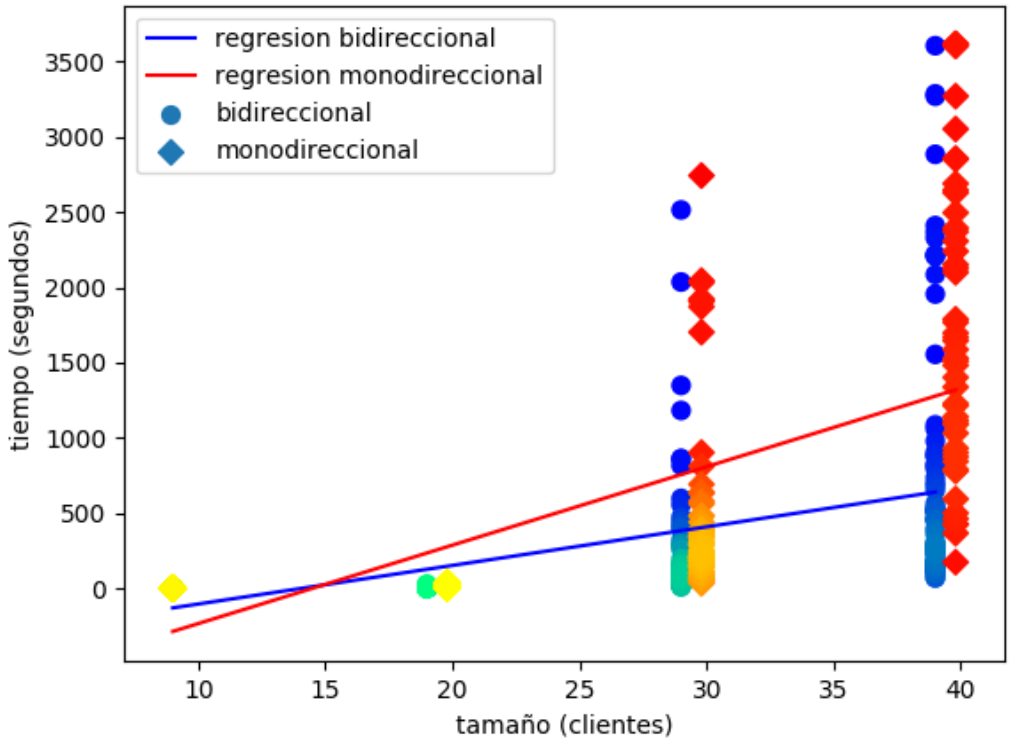

Figura. 6.1: Tiempo de ejecución de RR y de nuestro algoritmo para cada instancia.

Sin embargo, para $n = 39$, los tiempos de resolución empiezan a tener una distribución más uniforme.

Las rectas de la Figura 6.1 fueron calculadas con la técnica de cuadrados mínimos. las mismas nos muestran que el tiempo de cómputo aumenta considerablemente más rápido en el caso monodireccional, reforzando los datos presentados en la Tabla 6.1

6.3 Comparación con algoritmo de Roberti y Ruthmair

		Algoritmo con modificaciones			Algoritmo Roberti y Ruthmair		
n	α	Opt	Cpu	BBnodes	Opt	Cpu	BBnodes
9	1	25	0.14	1.24	25	0.3	4
	2	25	0.12	1.44	25	0.3	4
	3	25	0.11	1.56	25	0.3	4
19	1	25	11.36	1.08	25	47.4	40
	2	25	5.05	2.12	25	20.7	38
	3	25	7.04	5.32	25	17.6	36
29	1	25	562.06	2.76	19	1209.0	50
	2	25	71.50	3.4	24	554.3	118
	3	25	76.29	9.28	23	537	133
39	1	22	1704.18	4	1	3219.7	7
	2	25	433.90	5.8	3	2604.4	7
	3	24	311.56	12.58	7	1414.2	44

Tab. 6.2: Comparación con algoritmo de Roberti y Ruthmair

La Tabla 6.2 nos permite comparar los resultados del algoritmo de Roberti y Ruthmair (RR) con los obtenidos en este trabajo (AM). Para las instancias de tamaño $n = 10$ no se nota mayor diferencia, pero para instancias de mayor tamaño puede observarse el impacto de las mejoras propuestas sobre el algoritmo. En lo que respecta a tiempos, vemos como a partir de $n = 19$ el algoritmo AM resulta unas cuatro veces más rápido que RR. Esta diferencia se mantiene y hasta incrementa para las instancias más grandes. Las mejoras en tiempo se notan principalmente para valores mayores de α que, como mencionamos anteriormente, es donde las nuevas reglas de dominación realmente hacen la diferencia.

Por otro lado, vemos que AM resuelve las 75 instancias de tamaño $n = 29$, mientras que RR comienza a tener problemas y resuelve 66 instancias. Para los casos de $n = 39$ clientes, RR resuelve un porcentaje pequeño de las instancias, mientras que AM sigue resolviendo casi todas las instancias.

Otra diferencia importante se observa en la cantidad de nodos de branch and bound utilizados por cada algoritmo. Se puede ver que AM necesita explorar muchos menos nodos para llegar al óptimo, y esta diferencia va aumentando para instancias más grandes (esto no se observa tan bien para $n = 39$, pero eso se debe a la poca cantidad de instancias que llega a resolver RR).

La Figura 6.2 muestra una comparación entre RR y AM de la cantidad de instancias resueltas para cada α. Por otro lado, la Figura 6.3 compara el tiempo de ejecución entre RR y AM, y por último la Figura 6.4 muestra una comparación de los nodos explorados por cada algoritmo. Se puede observar que los tiempos de ejecución y la cantidad de nodos de branch and bound a explorar crece mucho más rápido en RR. En particular, puede observarse que los tiempos de ejecución de AM para instancias de tamaño $n = 39$ son similares a los de RR para instancias de tamaño $n = 29$. Además, la cantidad de nodos de branch and bound explorados por AM en instancias del máximo tamaño es

considerablemente menor a la cantidad de nodos que explora RR para instancias de $n = 19$.

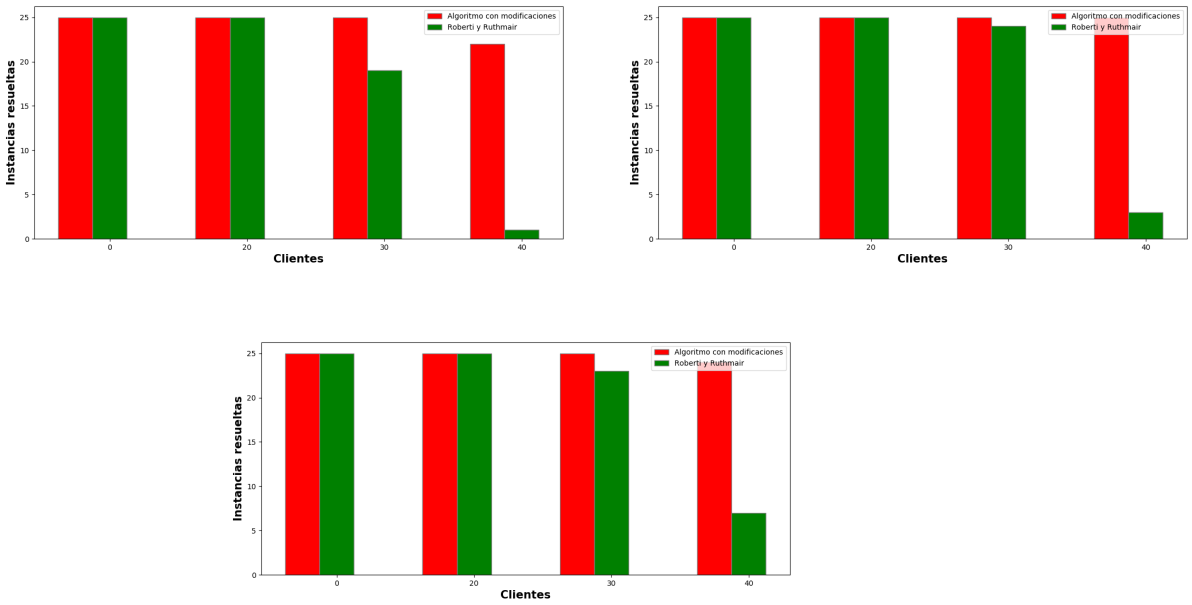

Figura. 6.2: Cantidad de instancias resueltas para cada tamaño de instancia, para $\alpha = 1$ (arriba a la izquierda), $\alpha = 2$ (arriba a la derecha) y $\alpha = 3$ (abajo).

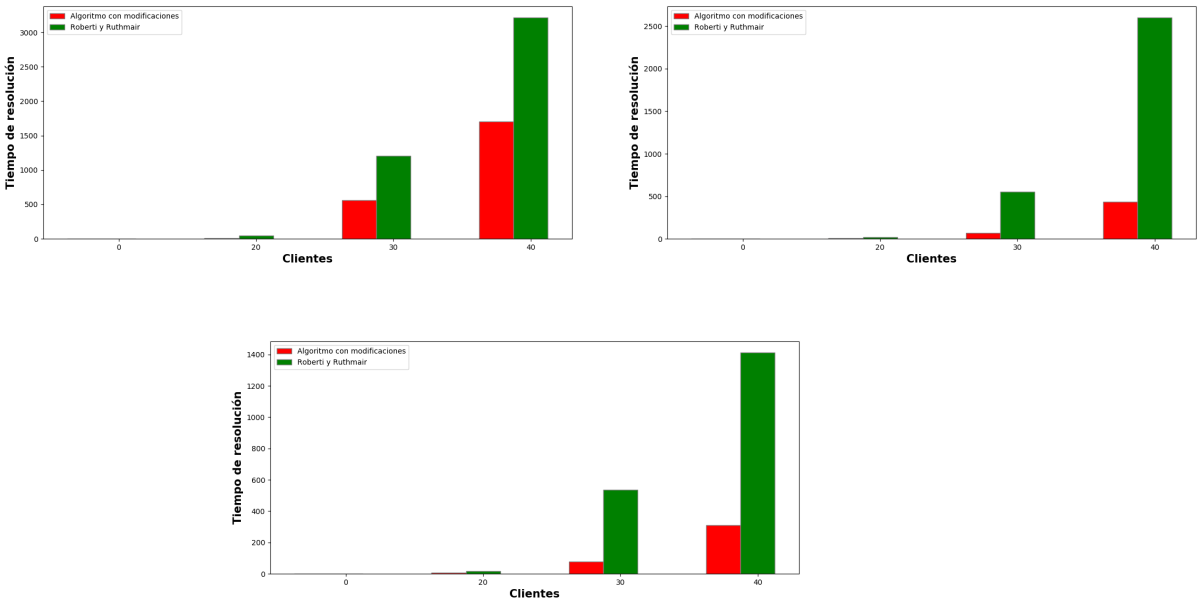

Figura. 6.3: Tiempo de ejecución promedio para cada tamaño de instancia, para $\alpha = 1$ (arriba a la izquierda), $\alpha = 2$ (arriba a la derecha) y $\alpha = 3$ (abajo). Estos gráficos solo tienen en cuenta el tiempo de ejecución de las instancias en donde el algoritmo termina.

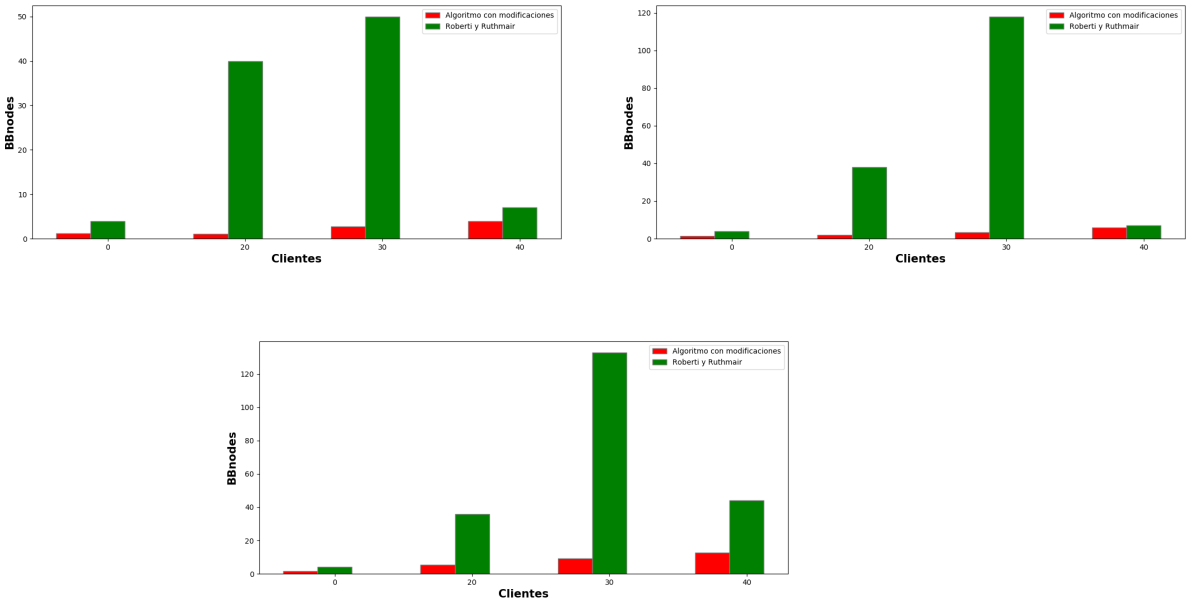

Figura. 6.4: Cantidad promedio de nodos explorados para cada tamaño de instancia, para $\alpha = 1$ (arriba a la izquierda), $\alpha = 2$ (arriba a la derecha) y $\alpha = 3$ (abajo). Estos gráficos solo tienen en cuenta los nodos explorados en instancias en donde el algoritmo termina.

7. CONCLUSIONES Y TRABAJO A FUTURO

En este trabajo estudiamos el problema del viajante de comercio con un dron. Planteamos una solución basada en el algoritmo de Branch and Price propuesto por Roberti and Ruthmair (2021), introduciendo una nueva noción de vecindarios ng, al igual que nuevas reglas de dominación parcial. Estas son mejoras que no surgen de la utilización de distintas técnicas conocidas para este tipo de problemas, si no que salen de observaciones clave sobre el problema en sí. Además, implementamos una versión bidireccional del algoritmo, que ayuda a mejoras los tiempos aún más. Realizamos también una experimentación comparando nuestros resultados contra los mejores que existen en la literatura hasta este momento. De los resultados computacionales podemos concluir que las mejoras propuestas logran reducir considerablemente el tiempo de ejecución del algoritmo, y permiten resolver instancias más difíciles que otros algoritmos no pueden resolver.

En cuanto a posible trabajo futuro, podría considerarse la implementación de técnicas más avanzadas para mejorar los tiempos del algoritmo. Por ejemplo, es posible combinar un *Decremental state space relaxation* (DSSR) que combine las técnicas de *neighbourhood augmentation*, *variable fixing* y generación de columnas. Además, sería interesante adaptar este algoritmo a versiones del TSPD con más restricciones, como por ejemplo darle un límite de autonomía al dron, permitir que el dron visite a más de un cliente por vez, o tener más de un camión y más de un dron por camión. No es obvio cómo adaptar nuestro trabajo a estas variantes del problema sin modificar demasiado el algoritmo de programación dinámica propuesto.

Bibliografía

N. Agatz, P. Bouman, and M. Schmidt. Optimization approaches for the traveling salesman problem with drone. *Transp. Sci.*, 52(4):965–981, 2018. doi:10.1287/trsc.2017.0791.

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice hall, 1993.

Amazon. Amazon Prime Air prepares for dronde deliveries. URL https://www.aboutamazon.com/news/transportation/amazon-prime-air-prepares-for-drone-deliveries.

D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Concorde tsp solver, 2006.

R. Baldacci, A. Mingozzi, and R. Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Oper. Res.*, 59(5):1269–1283, 2011. doi:10.1287/opre.1110.0975.

M. Boccia, A. Masone, A. Sforza, and C. Sterle. An exact approach for a variant of the fs-tsp. *Transp. Res. Proc.*, 52:51–58, 2021a. doi:https://doi.org/10.1016/j.trpro.2021.01.008.

M. Boccia, A. Masone, A. Sforza, and C. Sterle. A column-and-row generation approach for the flying sidekick travelling salesman problem. *Transp. Res. Part C Emerg. Technol.*, 124:102913, 2021b. doi:https://doi.org/10.1016/j.trc.2020.102913.

P. Bouman, N. Agatz, and M. Schmidt. Dynamic programming approaches for the traveling salesman problem with drone. *Networks*, 72(4):528–542, 2018. doi:10.1002/net.21864.

M. Dell'Amico, R. Montemanni, and S. Novellani. Drone-assisted deliveries: new formulations for the flying sidekick traveling salesman problem. *Optim. Lett.*, 15(5):1617–1648, 2021. doi:10.1007/s11590-019-01492-z.

M. Dell'Amico, R. Montemanni, and S. Novellani. Exact models for the flying sidekick traveling salesman problem. *Int. Trans. Oper. Res.*, 29(3):1360–1393, 2022. doi:10.1111/itor.13030.

M. Dell'Amico, R. Montemanni, and S. Novellani. Algorithms based on branch and bound for the flying sidekick traveling salesman problem. *Omega*, 104:102493, 2021. doi:https://doi.org/10.1016/j.omega.2021.102493.

E. Es Yurek and H. C. Ozmutlu. A decomposition-based iterative optimization algorithm for traveling salesman problem with drone. *Transp. Res. Part C Emerg. Technol.*, 91: 249–262, 2018. doi:https://doi.org/10.1016/j.trc.2018.04.009.

Q. M. Ha, Y. Deville, Q. D. Pham, and M. H. Hà. On the min-cost traveling salesman problem with drone. *Transp. Res. Part C Emerg. Technol.*, 86:597–621, 2018. doi:10.1016/j.trc.2017.11.015.

S. Hougardy and X. Zhong. Hard to solve instances of the euclidean traveling salesman problem. *Mathematical Programming Computation*, 13(1):51–74, 2021.

H. Y. Jeong, B. D. Song, and S. Lee. Truck-drone hybrid delivery routing: Payload-energy dependency and no-fly zones. *Int J. Prod. Econ.*, 214:220–233, 2019. doi:https://doi.org/10.1016/j.ijpe.2019.01.010.

G. Lera-Romero, J. J. Miranda Bront, and F. J. Soulignac. Linear edge costs and labeling algorithms: the case of the time-dependent vehicle routing problem with time windows. *Networks*, 76(1):24–53, 2020. doi:10.1002/net.21937.

C. C. Murray and A. G. Chu. The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transp. Res. Part C Emerg. Technol.*, 54:86–109, 2015. doi:10.1016/j.trc.2015.03.005.

S. Poikonen, B. Golden, and E. A. Wasil. A branch-and-bound approach to the traveling salesman problem with a drone. *INFORMS J. Comput.*, 31(2):335–346, 2019. doi:10.1287/ijoc.2018.0826.

A. Ponza. Optimization of drone-assisted parcel delivery. Master's thesis, Universitá Degli Studi di Padova, 2016. URL `https://thesis.unipd.it/handle/20.500.12608/25563`. Accessed 11/17/2022.

G. Righini and M. Salani. Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optim.*, 3 (3):255–273, 2006. doi:10.1016/j.disopt.2006.05.007.

R. Roberti and M. Ruthmair. Exact methods for the traveling salesman problem with drone. *Transp. Sci.*, 55(2):315–335, 2021. doi:10.1287/trsc.2020.1017.

D. Schermer, M. Moeini, and O. Wendt. A branch-and-cut approach and alternative formulations for the traveling salesman problem with drone. *Networks*, 76(2):164–186, 2020. doi:10.1002/net.21958.

Z. Tang, W.-J. v. Hoeve, and P. Shaw. A study on the traveling salesman problem with a drone. In L.-M. Rousseau and K. Stergiou, editors, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 557–564, Cham, 2019. Springer International Publishing.

E. van Dijck and P. Bouman. A branch-and-cut algorithm for the traveling salesman problem with drone. Master's thesis, Erasmus University Rotterdam, 2018. URL `http://hdl.handle.net/2105/44107`.

S. A. Vásquez, G. Angulo, and M. A. Klapp. An exact solution method for the TSP with drone based on decomposition. *Comput. Oper. Res.*, 127:Paper No. 105127, 12, 2021. doi:10.1016/j.cor.2020.105127.

Zipline. flyzipline. URL `https://flyzipline.com`.

7. ANEXO

7.1 Resultados completos para instancias de tamaño 10

Instance	Opt	Cpu	BBnodes	Optimum route duration
poi-10-1-1	Yes	0.122	1	149
poi-10-1-2	Yes	0.378	6	141
poi-10-1-3	Yes	0.088	1	113
poi-10-10-1	Yes	0.11	1	118
poi-10-10-2	Yes	0.098	1	85
poi-10-10-3	Yes	0.149	3	67
poi-10-11-1	Yes	0.136	1	142
poi-10-11-2	Yes	0.124	1	106
poi-10-11-3	Yes	0.113	1	81
poi-10-12-1	Yes	0.157	1	138
poi-10-12-2	Yes	0.124	1	107
poi-10-12-3	Yes	0.098	1	74
poi-10-13-1	Yes	0.129	1	114
poi-10-13-2	Yes	0.116	1	90
poi-10-13-3	Yes	0.193	3	82
poi-10-14-1	Yes	0.145	1	115
poi-10-14-2	Yes	0.072	1	90
poi-10-14-3	Yes	0.083	1	87
poi-10-15-1	Yes	0.177	1	145
poi-10-15-2	Yes	0.131	1	121
poi-10-15-3	Yes	0.124	1	97
poi-10-16-1	Yes	0.12	1	136
poi-10-16-2	Yes	0.099	1	103
poi-10-16-3	Yes	0.089	1	84
poi-10-17-1	Yes	0.105	1	141
poi-10-17-2	Yes	0.114	1	108
poi-10-17-3	Yes	0.096	1	92
poi-10-18-1	Yes	0.158	3	127
poi-10-18-2	Yes	0.085	1	101
poi-10-18-3	Yes	0.152	3	100
poi-10-19-1	Yes	0.108	1	139
poi-10-19-2	Yes	0.236	3	129
poi-10-19-3	Yes	0.18	3	118
poi-10-2-1	Yes	0.314	3	138
poi-10-2-2	Yes	0.075	1	111
poi-10-2-3	Yes	0.217	5	109
poi-10-20-1	Yes	0.105	1	128

poi-10-20-2	Yes	0.047	1	114
poi-10-20-3	Yes	0.071	1	105
poi-10-21-1	Yes	0.121	1	96
poi-10-21-2	Yes	0.086	1	71
poi-10-21-3	Yes	0.115	1	67
poi-10-22-1	Yes	0.173	1	121
poi-10-22-2	Yes	0.107	1	86
poi-10-22-3	Yes	0.161	3	67
poi-10-23-1	Yes	0.145	1	165
poi-10-23-2	Yes	0.114	1	129
poi-10-23-3	Yes	0.106	1	121
poi-10-24-1	Yes	0.114	1	126
poi-10-24-2	Yes	0.132	1	94
poi-10-24-3	Yes	0.095	1	81
poi-10-25-1	Yes	0.122	1	136
poi-10-25-2	Yes	0.175	3	131
poi-10-25-3	Yes	0.062	1	125
poi-10-3-1	Yes	0.231	3	151
poi-10-3-2	Yes	0.046	1	110
poi-10-3-3	Yes	0.111	1	98
poi-10-4-1	Yes	0.129	1	138
poi-10-4-2	Yes	0.09	1	113
poi-10-4-3	Yes	0.088	1	94
poi-10-5-1	Yes	0.121	1	136
poi-10-5-2	Yes	0.152	1	114
poi-10-5-3	Yes	0.086	1	85
poi-10-6-1	Yes	0.102	1	140
poi-10-6-2	Yes	0.177	3	118
poi-10-6-3	Yes	0.069	1	113
poi-10-7-1	Yes	0.117	1	109
poi-10-7-2	Yes	0.1	1	87
poi-10-7-3	Yes	0.096	1	79
poi-10-8-1	Yes	0.134	1	101
poi-10-8-2	Yes	0.114	1	80
poi-10-8-3	Yes	0.089	1	66
poi-10-9-1	Yes	0.106	1	152
poi-10-9-2	Yes	0.109	1	142
poi-10-9-3	Yes	0.097	1	139

Tab. 7.1: Resultados completos para instancias de tamaño 10

7.2 Resultados completos para instancias de tamaño 20

Instance	Opt	Cpu	BBnodes	Optimum route duration
poi-20-1-1	Yes	7.561	1	188
poi-20-1-2	Yes	5.369	3	153
poi-20-1-3	Yes	3.112	1	131
poi-20-10-1	Yes	13.908	1	173
poi-20-10-2	Yes	6.278	3	144
poi-20-10-3	Yes	1.634	1	130
poi-20-11-1	Yes	21.666	1	164
poi-20-11-2	Yes	9.498	3	150
poi-20-11-3	Yes	24.118	17	148
poi-20-12-1	Yes	11.124	1	169
poi-20-12-2	Yes	2.521	1	133
poi-20-12-3	Yes	6.69	5	123
poi-20-13-1	Yes	10.622	1	171
poi-20-13-2	Yes	11.306	7	151
poi-20-13-3	Yes	10.168	9	129
poi-20-14-1	Yes	21.032	1	164
poi-20-14-2	Yes	7.859	5	145
poi-20-14-3	Yes	2.369	1	134
poi-20-15-1	Yes	9.286	1	183
poi-20-15-2	Yes	2.545	1	139
poi-20-15-3	Yes	2.354	1	121
poi-20-16-1	Yes	7.634	1	165
poi-20-16-2	Yes	3.226	1	129
poi-20-16-3	Yes	2.015	1	118
poi-20-17-1	Yes	8.245	1	150
poi-20-17-2	Yes	4.981	1	108
poi-20-17-3	Yes	3.216	1	101
poi-20-18-1	Yes	12.836	1	177
poi-20-18-2	Yes	5.911	3	151
poi-20-18-3	Yes	1.728	1	135
poi-20-19-1	Yes	6.751	1	162
poi-20-19-2	Yes	3.723	1	122
poi-20-19-3	Yes	6.481	5	114
poi-20-2-1	Yes	7.152	1	180
poi-20-2-2	Yes	3.015	1	151
poi-20-2-3	Yes	2.597	1	130
poi-20-20-1	Yes	13.951	1	180
poi-20-20-2	Yes	4.753	1	154
poi-20-20-3	Yes	28.994	19	148
poi-20-21-1	Yes	10.155	1	164
poi-20-21-2	Yes	2.972	1	137

poi-20-21-3	Yes	3.115	1	129
poi-20-22-1	Yes	6.871	1	174
poi-20-22-2	Yes	2.682	1	136
poi-20-22-3	Yes	2.272	1	114
poi-20-23-1	Yes	8.382	1	172
poi-20-23-2	Yes	3.796	1	134
poi-20-23-3	Yes	2.091	1	123
poi-20-24-1	Yes	12.091	1	171
poi-20-24-2	Yes	4.443	1	125
poi-20-24-3	Yes	2.161	1	111
poi-20-25-1	Yes	6.002	1	178
poi-20-25-2	Yes	5.152	3	141
poi-20-25-3	Yes	9.752	11	122
poi-20-3-1	Yes	9.141	1	196
poi-20-3-2	Yes	10.933	5	169
poi-20-3-3	Yes	8.401	5	154
poi-20-4-1	Yes	13.621	3	179
poi-20-4-2	Yes	5.995	3	140
poi-20-4-3	Yes	6.553	7	125
poi-20-5-1	Yes	13.051	1	164
poi-20-5-2	Yes	3.034	1	138
poi-20-5-3	Yes	14.895	17	129
poi-20-6-1	Yes	6.606	1	161
poi-20-6-2	Yes	2.156	1	121
poi-20-6-3	Yes	5.733	5	110
poi-20-7-1	Yes	17.218	1	179
poi-20-7-2	Yes	7.304	3	148
poi-20-7-3	Yes	10.901	7	123
poi-20-8-1	Yes	11.976	1	180
poi-20-8-2	Yes	2.018	1	149
poi-20-8-3	Yes	2.009	1	131
poi-20-9-1	Yes	17.147	1	153
poi-20-9-2	Yes	4.89	1	135
poi-20-9-3	Yes	12.808	13	130

Tab. 7.2: Resultados completos para instancias de tamaño 20

7.3 Resultados completos para instancias de tamaño 30

Instance	Opt	Cpu	BBnodes	Optimum route duration
poi-30-1-1	Yes	260.073	3	205
poi-30-1-2	Yes	293.235	11	164
poi-30-1-3	Yes	16.809	1	140
poi-30-10-1	Yes	92.588	1	195

poi-30-10-2	Yes	94.763	7	159
poi-30-10-3	Yes	19.28	1	139
poi-30-11-1	Yes	122.277	1	200
poi-30-11-2	Yes	33.828	3	162
poi-30-11-3	Yes	50.585	7	144
poi-30-12-1	Yes	116.58	1	210
poi-30-12-2	Yes	25.321	1	160
poi-30-12-3	Yes	37.741	5	150
poi-30-13-1	Yes	859.849	1	200
poi-30-13-2	Yes	32.526	1	157
poi-30-13-3	Yes	59.357	7	147
poi-30-14-1	Yes	2035.08	3	188
poi-30-14-2	Yes	46.484	1	160
poi-30-14-3	Yes	140.381	17	149
poi-30-15-1	Yes	283.339	1	185
poi-30-15-2	Yes	150.352	1	144
poi-30-15-3	Yes	444.137	33	137
poi-30-16-1	Yes	146.28	1	196
poi-30-16-2	Yes	34.407	1	156
poi-30-16-3	Yes	55.878	7	143
poi-30-17-1	Yes	414.51	1	196
poi-30-17-2	Yes	57.954	1	159
poi-30-17-3	Yes	26.997	3	147
poi-30-18-1	Yes	1348.47	23	206
poi-30-18-2	Yes	24.093	1	160
poi-30-18-3	Yes	18.284	1	136
poi-30-19-1	Yes	556.489	5	195
poi-30-19-2	Yes	30.058	1	165
poi-30-19-3	Yes	161.339	24	155
poi-30-2-1	Yes	137.949	5	209
poi-30-2-2	Yes	26.842	1	160
poi-30-2-3	Yes	50.932	7	142
poi-30-20-1	Yes	473.863	1	180
poi-30-20-2	Yes	54.048	3	147
poi-30-20-3	Yes	132.625	21	138
poi-30-21-1	Yes	162.285	1	198
poi-30-21-2	Yes	118.592	17	164
poi-30-21-3	Yes	63.704	11	143
poi-30-22-1	Yes	1182	1	207
poi-30-22-2	Yes	43.301	1	173
poi-30-22-3	Yes	58.897	5	162
poi-30-23-1	Yes	196.831	3	190
poi-30-23-2	Yes	41.517	3	151
poi-30-23-3	Yes	68.302	13	137
poi-30-24-1	Yes	105.502	1	182

poi-30-24-2	Yes	36.236	1	146
poi-30-24-3	Yes	67.608	9	132
poi-30-25-1	Yes	132.02	3	223
poi-30-25-2	Yes	33.937	1	180
poi-30-25-3	Yes	10.162	1	164
poi-30-3-1	Yes	124.924	1	183
poi-30-3-2	Yes	35.658	1	153
poi-30-3-3	Yes	61.635	7	143
poi-30-4-1	Yes	111.7	1	195
poi-30-4-2	Yes	322.906	11	150
poi-30-4-3	Yes	76.864	13	129
poi-30-5-1	Yes	862.687	3	186
poi-30-5-2	Yes	86.202	5	161
poi-30-5-3	Yes	27.411	1	144
poi-30-6-1	Yes	2513.81	1	186
poi-30-6-2	Yes	49.073	3	156
poi-30-6-3	Yes	90.595	13	146
poi-30-7-1	Yes	816.462	3	182
poi-30-7-2	Yes	35.16	3	147
poi-30-7-3	Yes	32.104	5	131
poi-30-8-1	Yes	598.3	3	195
poi-30-8-2	Yes	22.601	1	164
poi-30-8-3	Yes	56.224	7	155
poi-30-9-1	Yes	397.802	1	195
poi-30-9-2	Yes	58.535	5	157
poi-30-9-3	Yes	79.607	13	143

Tab. 7.3: Resultados completos para instancias de tamaño 30

7.4 Resultados completos para instancias de tamaño 40

Instance	Opt	Cpu	BBnodes	Optimum route duration
poi-40-1-1	Yes	826.015	5	221
poi-40-1-2	Yes	536.295	9	172
poi-40-1-3	Yes	382.187	13	154
poi-40-10-1	Yes	883.362	1	214
poi-40-10-2	Yes	448.38	11	177
poi-40-10-3	No	time limit	196	-
poi-40-11-1	Yes	795.072	1	215
poi-40-11-2	Yes	815.435	3	175
poi-40-11-3	Yes	108.567	3	159
poi-40-12-1	Yes	1955.34	1	205
poi-40-12-2	Yes	529.522	5	168
poi-40-12-3	Yes	127.053	1	147

poi-40-13-1	Yes	2409.84	1	226
poi-40-13-2	Yes	277.379	1	179
poi-40-13-3	Yes	245.744	5	161
poi-40-14-1	Yes	1059.83	3	231
poi-40-14-2	Yes	508.461	3	177
poi-40-14-3	Yes	260.805	9	159
poi-40-15-1	Yes	3268.91	3	224
poi-40-15-2	Yes	176.592	1	174
poi-40-15-3	Yes	1086.71	75	161
poi-40-16-1	Yes	2206.95	1	211
poi-40-16-2	Yes	878.012	7	171
poi-40-16-3	Yes	166.704	3	149
poi-40-17-1	Yes	979.337	1	229
poi-40-17-2	Yes	99.29	1	180
poi-40-17-3	Yes	301.614	5	162
poi-40-18-1	Yes	2883.07	13	234
poi-40-18-2	Yes	261.193	3	185
poi-40-18-3	Yes	178.235	3	166
poi-40-19-1	Yes	1554.75	1	194
poi-40-19-2	Yes	176.689	3	157
poi-40-19-3	Yes	358.429	13	140
poi-40-2-1	Yes	462.536	1	213
poi-40-2-2	Yes	121.397	1	173
poi-40-2-3	Yes	145.653	3	152
poi-40-20-1	Yes	2367.55	1	218
poi-40-20-2	Yes	683.085	5	180
poi-40-20-3	Yes	72.107	1	160
poi-40-21-1	Yes	2328.35	3	224
poi-40-21-2	Yes	180.272	1	169
poi-40-21-3	Yes	179.932	3	154
poi-40-22-1	Yes	701.915	3	246
poi-40-22-2	Yes	459.41	17	200
poi-40-22-3	Yes	905.273	51	178
poi-40-23-1	No	time limit	27	-
poi-40-23-2	Yes	463.019	7	193
poi-40-23-3	Yes	653.591	21	173
poi-40-24-1	Yes	3284.44	1	206
poi-40-24-2	Yes	256.106	1	171
poi-40-24-3	Yes	154.136	7	158
poi-40-25-1	No	time limit	17	-
poi-40-25-2	Yes	2084.7	39	180
poi-40-25-3	Yes	284.168	9	153
poi-40-3-1	Yes	2210.65	5	231
poi-40-3-2	Yes	129.492	5	192
poi-40-3-3	Yes	512.037	25	178

poi-40-4-1	No	time limit	3	-
poi-40-4-2	Yes	156.683	1	193
poi-40-4-3	Yes	70.719	1	167
poi-40-5-1	Yes	2219.75	13	237
poi-40-5-2	Yes	564.559	9	188
poi-40-5-3	Yes	218.658	7	159
poi-40-6-1	Yes	519.399	1	186
poi-40-6-2	Yes	107.743	1	149
poi-40-6-3	Yes	294.262	15	140
poi-40-7-1	Yes	736.447	1	202
poi-40-7-2	Yes	400.087	1	162
poi-40-7-3	Yes	126.723	3	144
poi-40-8-1	Yes	3604.36	27	238
poi-40-8-2	Yes	226.392	5	187
poi-40-8-3	Yes	511.526	23	168
poi-40-9-1	Yes	234.003	1	223
poi-40-9-2	Yes	307.476	5	183
poi-40-9-3	Yes	132.788	3	161

Tab. 7.4: Resultados completos para instancias de tamaño 40

7.5 Rutas óptimas por instancia

En esta tabla, las rutas están representadas como la ruta que realizó el camión, y los tramos que realizó el dron. Estos tramos son triplas en donde el primer número indica en qué cliente se separaron el dron y el camión, el segundo indica a qué cliente visitó el dron, y el tercero indica en qué cliente se volvieron a juntar los vehículos.

instance	Truck route	Drone legs
poi-10-1-1	[0, 4, 2, 8, 1, 7, 10]	[[0, 9, 2], [8, 5, 7], [2, 6, 8], [7, 3, 10]]
poi-10-1-2	[0, 4, 2, 1, 7, 10]	[[1, 5, 7], [2, 8, 1], [0, 3, 4], [4, 6, 2], [7, 9, 10]]
poi-10-1-3	[0, 3, 9, 7, 2, 4, 10]	[[0, 8, 7], [4, 1, 10], [2, 5, 4], [7, 6, 2]]
poi-10-10-1	[0, 8, 7, 9, 6, 4, 1, 3, 10]	[[9, 5, 10], [0, 2, 9]]
poi-10-10-2	[0, 2, 7, 6, 9, 8, 10]	[[2, 5, 7], [0, 1, 2], [7, 4, 8], [8, 3, 10]]
poi-10-10-3	[0, 6, 9, 7, 8, 10]	[[7, 5, 8], [6, 4, 9], [8, 3, 10], [9, 2, 7], [0, 1, 6]]
poi-10-11-1	[0, 4, 2, 6, 1, 9, 8, 10]	[[0, 7, 2], [9, 5, 10], [2, 3, 9]]
poi-10-11-2	[0, 4, 2, 8, 9, 5, 10]	[[2, 7, 8], [8, 1, 9], [0, 3, 2], [9, 6, 10]]
poi-10-11-3	[0, 5, 8, 6, 1, 9, 10]	[[8, 7, 6], [0, 4, 8], [9, 2, 10], [6, 3, 9]]
poi-10-12-1	[0, 2, 8, 6, 3, 9, 7, 10]	[[3, 4, 10], [2, 1, 3], [0, 5, 2]]

poi-10-12-2	[0, 4, 3, 2, 9, 7, 5, 10]	[[0, 1, 3], [2, 8, 10], [3, 6, 2]]
poi-10-12-3	[0, 5, 3, 2, 9, 7, 10]	[[9, 8, 10], [5, 1, 3], [0, 4, 5], [3, 6, 9]]
poi-10-13-1	[0, 5, 1, 6, 8, 4, 7, 3, 10]	[[4, 9, 10], [0, 2, 4]]
poi-10-13-2	[0, 5, 6, 8, 4, 7, 3, 10]	[[0, 2, 8], [8, 1, 7], [7, 9, 10]]
poi-10-13-3	[0, 3, 7, 1, 5, 10]	[[0, 9, 3], [1, 8, 5], [3, 4, 7], [5, 6, 10], [7, 2, 1]]
poi-10-14-1	[0, 9, 8, 5, 3, 7, 1, 6, 10]	[[0, 4, 8], [8, 2, 10]]
poi-10-14-2	[0, 9, 8, 3, 7, 10]	[[7, 6, 10], [8, 5, 3], [9, 4, 8], [0, 2, 9], [3, 1, 7]]
poi-10-14-3	[0, 1, 5, 3, 7, 10]	[[0, 9, 1], [5, 8, 3], [1, 6, 5], [3, 4, 7], [7, 2, 10]]
poi-10-15-1	[0, 3, 9, 6, 1, 2, 7, 10]	[[7, 8, 10], [6, 5, 7], [0, 4, 6]]
poi-10-15-2	[0, 6, 3, 9, 8, 7, 10]	[[3, 5, 8], [8, 4, 10], [6, 1, 3], [0, 2, 6]]
poi-10-15-3	[0, 9, 3, 8, 4, 7, 10]	[[7, 6, 10], [3, 5, 8], [8, 2, 7], [0, 1, 3]]
poi-10-16-1	[0, 3, 2, 7, 4, 5, 8, 10]	[[3, 9, 5], [5, 6, 10], [0, 1, 3]]
poi-10-16-2	[0, 3, 1, 2, 5, 8, 10]	[[1, 9, 2], [0, 7, 1], [2, 4, 8], [8, 6, 10]]
poi-10-16-3	[0, 6, 5, 8, 2, 3, 10]	[[3, 9, 10], [2, 4, 3], [0, 1, 8], [8, 7, 2]]
poi-10-17-1	[0, 3, 2, 5, 6, 7, 8, 1, 10]	[[5, 9, 10], [0, 4, 5]]
poi-10-17-2	[0, 8, 7, 6, 5, 2, 10]	[[2, 3, 10], [0, 1, 8], [8, 9, 5], [5, 4, 2]]
poi-10-17-3	[0, 6, 5, 2, 3, 10]	[[2, 9, 3], [3, 8, 10], [6, 7, 5], [0, 4, 6], [5, 1, 2]]
poi-10-18-1	[0, 5, 9, 7, 4, 2, 6, 10]	[[0, 8, 9], [2, 3, 10], [9, 1, 2]]
poi-10-18-2	[0, 5, 9, 7, 4, 10]	[[5, 8, 9], [4, 3, 10], [0, 6, 5], [7, 2, 4], [9, 1, 7]]
poi-10-18-3	[0, 5, 4, 2, 3, 10]	[[0, 8, 5], [4, 7, 2], [3, 6, 10], [5, 9, 4], [2, 1, 3]]
poi-10-19-1	[0, 7, 4, 8, 6, 9, 10]	[[6, 5, 10], [0, 2, 7], [4, 1, 6], [7, 3, 4]]
poi-10-19-2	[0, 8, 6, 4, 7, 10]	[[0, 9, 8], [4, 3, 7], [7, 2, 10], [8, 5, 6], [6, 1, 4]]
poi-10-19-3	[0, 2, 7, 3, 4, 1, 10]	[[0, 9, 3], [1, 6, 10], [4, 8, 1], [3, 5, 4]]
poi-10-2-1	[0, 6, 2, 9, 3, 8, 4, 10]	[[3, 5, 10], [0, 1, 9], [9, 7, 3]]
poi-10-2-2	[0, 4, 9, 7, 8, 10]	[[0, 5, 4], [9, 3, 7], [7, 2, 8], [4, 6, 9], [8, 1, 10]]
poi-10-2-3	[0, 5, 6, 8, 4, 10]	[[8, 9, 4], [5, 3, 6], [6, 2, 8], [0, 1, 5], [4, 7, 10]]
poi-10-20-1	[0, 8, 7, 6, 4, 1, 5, 10]	[[1, 9, 10], [6, 3, 1], [0, 2, 6]]

poi-10-20-2	[0, 5, 9, 1, 7, 8, 10]	[[1, 3, 7], [7, 2, 10], [9, 6, 1], [0, 4, 9]]
poi-10-20-3	[0, 8, 7, 1, 5, 10]	[[5, 9, 10], [1, 4, 5], [0, 3, 8], [8, 6, 7], [7, 2, 1]]
poi-10-21-1	[0, 1, 5, 8, 7, 2, 4, 6, 10]	[[0, 9, 2], [2, 3, 10]]
poi-10-21-2	[0, 6, 7, 8, 5, 1, 10]	[[5, 9, 10], [7, 2, 5], [6, 3, 7], [0, 4, 6]]
poi-10-21-3	[0, 1, 5, 8, 7, 6, 10]	[[0, 9, 5], [6, 4, 10], [7, 3, 6], [5, 2, 8]]
poi-10-22-1	[0, 3, 2, 5, 4, 7, 8, 9, 10]	[[5, 6, 10], [0, 1, 5]]
poi-10-22-2	[0, 3, 2, 4, 7, 8, 9, 10]	[[3, 5, 4], [0, 1, 3], [4, 6, 10]]
poi-10-22-3	[0, 4, 7, 8, 9, 10]	[[0, 5, 4], [9, 3, 10], [4, 6, 7], [7, 2, 8], [8, 1, 9]]
poi-10-23-1	[0, 8, 1, 3, 2, 4, 10]	[[0, 5, 1], [2, 9, 10], [1, 7, 3], [3, 6, 2]]
poi-10-23-2	[0, 4, 7, 3, 6, 9, 2, 10]	[[7, 1, 9], [9, 8, 10], [0, 5, 7]]
poi-10-23-3	[0, 9, 3, 6, 2, 4, 10]	[[2, 8, 10], [0, 7, 9], [9, 1, 3], [3, 5, 2]]
poi-10-24-1	[0, 8, 2, 4, 1, 5, 3, 6, 10]	[[1, 9, 10], [0, 7, 1]]
poi-10-24-2	[0, 8, 6, 3, 1, 10]	[[3, 9, 1], [1, 7, 10], [6, 5, 3], [8, 4, 6], [0, 2, 8]]
poi-10-24-3	[0, 8, 2, 6, 5, 3, 10]	[[6, 1, 5], [8, 9, 6], [0, 4, 8], [5, 7, 10]]
poi-10-25-1	[0, 5, 7, 8, 2, 6, 1, 10]	[[0, 3, 8], [8, 9, 6], [6, 4, 10]]
poi-10-25-2	[0, 1, 4, 6, 9, 10]	[[1, 7, 4], [6, 8, 9], [0, 5, 1], [9, 3, 10], [4, 2, 6]]
poi-10-25-3	[0, 6, 9, 4, 1, 10]	[[6, 8, 9], [4, 3, 1], [1, 5, 10], [9, 2, 4], [0, 7, 6]]
poi-10-3-1	[0, 6, 4, 2, 5, 1, 8, 10]	[[0, 9, 1], [1, 3, 8], [8, 7, 10]]
poi-10-3-2	[0, 9, 6, 4, 2, 5, 1, 10]	[[0, 8, 6], [1, 7, 10], [6, 3, 1]]
poi-10-3-3	[0, 6, 4, 2, 5, 1, 10]	[[0, 8, 6], [6, 9, 4], [4, 3, 1], [1, 7, 10]]
poi-10-4-1	[0, 8, 9, 5, 6, 2, 7, 10]	[[0, 1, 6], [6, 3, 2], [2, 4, 10]]
poi-10-4-2	[0, 7, 6, 5, 9, 8, 10]	[[7, 4, 6], [0, 3, 7], [9, 1, 10], [6, 2, 9]]
poi-10-4-3	[0, 1, 6, 5, 9, 8, 10]	[[0, 7, 1], [6, 3, 5], [1, 4, 6], [5, 2, 10]]
poi-10-5-1	[0, 2, 3, 7, 4, 9, 5, 8, 10]	[[0, 6, 9], [9, 1, 10]]
poi-10-5-2	[0, 8, 9, 4, 7, 3, 10]	[[3, 2, 10], [0, 5, 8], [9, 6, 3], [8, 1, 9]]
poi-10-5-3	[0, 8, 5, 2, 3, 7, 10]	[[7, 9, 10], [2, 4, 7], [5, 1, 2], [0, 6, 5]]
poi-10-6-1	[0, 5, 9, 4, 7, 3, 1, 6, 10]	[[7, 8, 10], [0, 2, 7]]
poi-10-6-2	[0, 5, 4, 1, 9, 10]	[[1, 3, 9], [9, 8, 10], [5, 6, 4], [4, 7, 1], [0, 2, 5]]

poi-10-6-3	[0, 5, 4, 1, 9, 10]	[[0, 7, 5], [1, 6, 9], [4, 3, 1], [5, 8, 4], [9, 2, 10]]
poi-10-7-1	[0, 7, 8, 4, 5, 6, 2, 10]	[[6, 9, 2], [2, 3, 10], [0, 1, 6]]
poi-10-7-2	[0, 7, 8, 4, 3, 2, 10]	[[3, 9, 2], [2, 1, 10], [4, 5, 3], [0, 6, 4]]
poi-10-7-3	[0, 2, 3, 4, 7, 10]	[[7, 8, 10], [3, 6, 4], [4, 5, 7], [2, 9, 3], [0, 1, 2]]
poi-10-8-1	[0, 5, 2, 6, 8, 1, 9, 7, 10]	[[0, 4, 5], [5, 3, 10]]
poi-10-8-2	[0, 9, 6, 2, 5, 10]	[[0, 7, 9], [2, 4, 5], [9, 8, 6], [5, 3, 10], [6, 1, 2]]
poi-10-8-3	[0, 1, 8, 6, 9, 7, 10]	[[1, 5, 6], [6, 2, 9], [9, 3, 10], [0, 4, 1]]
poi-10-9-1	[0, 1, 3, 9, 6, 7, 2, 10]	[[2, 8, 10], [9, 5, 2], [0, 4, 9]]
poi-10-9-2	[0, 2, 7, 3, 1, 10]	[[0, 8, 2], [3, 6, 1], [2, 5, 7], [7, 9, 3], [1, 4, 10]]
poi-10-9-3	[0, 1, 7, 2, 8, 10]	[[0, 9, 1], [2, 6, 8], [1, 4, 7], [8, 5, 10], [7, 3, 2]]
poi-20-1-1	[0, 2, 6, 9, 18, 5, 15, 3, 12, 14, 10, 19, 11, 13, 16, 7, 20]	[[18, 17, 3], [3, 8, 13], [13, 1, 20], [0, 4, 18]]
poi-20-1-2	[0, 2, 6, 18, 5, 15, 12, 14, 19, 13, 16, 7, 20]	[[18, 17, 5], [14, 11, 19], [6, 9, 18], [19, 8, 16], [0, 4, 6], [12, 10, 14], [16, 1, 20], [5, 3, 12]]
poi-20-1-3	[0, 7, 3, 12, 14, 10, 11, 19, 16, 13, 1, 8, 2, 20]	[[7, 17, 3], [8, 9, 20], [3, 5, 12], [10, 4, 16], [0, 18, 7], [16, 6, 8], [12, 15, 10]]
poi-20-10-1	[0, 10, 2, 14, 3, 8, 12, 9, 13, 16, 15, 1, 17, 18, 20]	[[12, 7, 16], [14, 19, 3], [0, 6, 14], [15, 5, 20], [16, 4, 15], [3, 11, 12]]
poi-20-10-2	[0, 18, 15, 16, 13, 9, 12, 8, 14, 2, 10, 20]	[[8, 19, 14], [9, 11, 12], [16, 7, 9], [2, 5, 20], [14, 6, 2], [0, 17, 18], [18, 1, 15], [12, 3, 8], [15, 4, 16]]
poi-20-10-3	[0, 10, 14, 19, 3, 8, 13, 16, 4, 15, 18, 20]	[[18, 17, 20], [19, 12, 8], [13, 9, 4], [15, 7, 18], [4, 1, 15], [10, 5, 14], [0, 2, 10], [8, 11, 13], [14, 6, 19]]
poi-20-11-1	[0, 19, 10, 15, 18, 8, 6, 16, 3, 7, 12, 4, 2, 13, 14, 20]	[[10, 17, 12], [12, 9, 14], [14, 5, 20], [19, 1, 10], [0, 11, 19]]
poi-20-11-2	[0, 5, 10, 15, 18, 8, 6, 16, 4, 2, 13, 14, 20]	[[5, 19, 10], [10, 1, 18], [4, 9, 2], [6, 3, 16], [18, 7, 6], [0, 11, 5], [2, 17, 20], [16, 12, 4]]
poi-20-11-3	[0, 14, 13, 2, 4, 16, 6, 8, 18, 15, 10, 20]	[[14, 19, 13], [13, 17, 4], [0, 11, 14], [6, 7, 8], [4, 12, 16], [16, 3, 6], [8, 5, 15], [15, 1, 10], [10, 9, 20]]
poi-20-12-1	[0, 16, 10, 6, 5, 17, 3, 1, 15, 14, 2, 13, 8, 11, 12, 7, 20]	[[14, 18, 20], [0, 19, 6], [3, 4, 14], [6, 9, 3]]

poi-20-12-2	[0, 5, 17, 3, 1, 14, 2, 13, 8, 11, 12, 7, 20]	[[3, 9, 2], [8, 18, 12], [17, 4, 3], [5, 10, 17], [12, 19, 7], [0, 6, 5], [7, 16, 20], [2, 15, 8]]
poi-20-12-3	[0, 16, 10, 5, 1, 14, 2, 13, 8, 11, 12, 7, 20]	[[7, 19, 20], [10, 6, 5], [0, 9, 10], [8, 18, 11], [5, 4, 1], [11, 3, 7], [13, 15, 8], [1, 17, 13]]
poi-20-13-1	[0, 4, 3, 8, 6, 14, 18, 13, 5, 2, 12, 7, 1, 20]	[[8, 19, 14], [4, 16, 8], [0, 17, 4], [2, 11, 20], [13, 10, 5], [5, 15, 2], [14, 9, 13]]
poi-20-13-2	[0, 4, 3, 8, 6, 19, 14, 18, 9, 11, 7, 12, 1, 17, 20]	[[0, 16, 4], [7, 15, 20], [18, 10, 11], [14, 13, 18], [11, 2, 7], [4, 5, 14]]
poi-20-13-3	[0, 17, 3, 8, 6, 19, 14, 9, 16, 7, 12, 1, 20]	[[9, 18, 16], [1, 11, 20], [16, 5, 7], [14, 13, 9], [17, 2, 6], [0, 4, 17], [6, 10, 14], [7, 15, 1]]
poi-20-14-1	[0, 16, 13, 11, 2, 19, 3, 10, 4, 5, 6, 15, 17, 9, 20]	[[3, 14, 5], [5, 12, 15], [16, 8, 11], [0, 7, 16], [11, 18, 3], [15, 1, 20]]
poi-20-14-2	[0, 9, 17, 15, 12, 6, 14, 2, 11, 13, 16, 20]	[[14, 19, 2], [6, 10, 14], [13, 8, 20], [15, 4, 12], [12, 5, 6], [11, 18, 13], [2, 3, 11], [17, 1, 15], [0, 7, 17]]
poi-20-14-3	[0, 9, 17, 15, 1, 14, 2, 11, 13, 16, 7, 20]	[[11, 19, 13], [16, 18, 20], [17, 12, 15], [1, 10, 14], [15, 6, 1], [14, 4, 2], [13, 8, 16], [0, 5, 17], [2, 3, 11]]
poi-20-15-1	[0, 16, 17, 8, 5, 18, 13, 11, 19, 12, 7, 1, 9, 6, 10, 20]	[[18, 4, 19], [19, 15, 7], [0, 3, 17], [17, 14, 18], [7, 2, 20]]
poi-20-15-2	[0, 3, 1, 7, 12, 4, 13, 18, 5, 17, 16, 20]	[[13, 19, 5], [5, 14, 16], [4, 11, 13], [16, 8, 20], [3, 6, 1], [12, 15, 4], [0, 10, 3], [7, 2, 12], [1, 9, 7]]
poi-20-15-3	[0, 10, 6, 1, 8, 4, 13, 5, 17, 16, 20]	[[5, 14, 17], [16, 12, 20], [4, 11, 13], [0, 9, 10], [17, 18, 16], [6, 2, 1], [13, 19, 5], [8, 15, 4], [1, 7, 8], [10, 3, 6]]
poi-20-16-1	[0, 3, 5, 2, 9, 1, 18, 17, 8, 13, 14, 6, 12, 7, 19, 20]	[[18, 11, 14], [0, 10, 5], [9, 4, 18], [5, 16, 9], [14, 15, 20]]
poi-20-16-2	[0, 3, 5, 9, 1, 18, 17, 13, 14, 6, 12, 20]	[[12, 19, 20], [3, 16, 5], [13, 15, 14], [5, 2, 9], [1, 11, 17], [9, 4, 1], [17, 8, 13], [6, 7, 12], [0, 10, 3]]
poi-20-16-3	[0, 3, 5, 2, 17, 13, 14, 6, 12, 7, 19, 20]	[[0, 16, 3], [14, 15, 6], [5, 9, 2], [17, 18, 13], [13, 8, 14], [2, 11, 17], [6, 1, 19], [3, 4, 5], [19, 10, 20]]
poi-20-17-1	[0, 1, 9, 14, 16, 8, 17, 7, 19, 6, 5, 12, 18, 3, 4, 13, 10, 20]	[[18, 15, 20], [0, 11, 16], [16, 2, 18]]

poi-20-17-2	[0, 10, 3, 18, 6, 19, 7, 17, 8, 16, 14, 1, 20]	[[6, 15, 17], [17, 2, 14], [10, 4, 3], [3, 5, 18], [14, 9, 1], [1, 11, 20], [18, 12, 6], [0, 13, 10]]
poi-20-17-3	[0, 1, 14, 8, 17, 7, 19, 18, 3, 10, 20]	[[17, 15, 7], [10, 11, 20], [7, 6, 19], [0, 13, 1], [1, 9, 14], [8, 16, 17], [19, 12, 18], [14, 2, 8], [3, 4, 10], [18, 5, 3]]
poi-20-18-1	[0, 16, 9, 15, 10, 6, 3, 4, 14, 19, 8, 11, 12, 7, 18, 20]	[[0, 13, 10], [14, 5, 11], [10, 17, 4], [11, 2, 20], [4, 1, 14]]
poi-20-18-2	[0, 18, 19, 8, 14, 4, 9, 15, 10, 7, 20]	[[10, 13, 7], [19, 11, 8], [8, 5, 14], [0, 16, 18], [4, 1, 9], [18, 2, 19], [14, 17, 4], [7, 12, 20], [9, 3, 15], [15, 6, 10]]
poi-20-18-3	[0, 18, 7, 19, 8, 14, 1, 17, 4, 9, 16, 20]	[[8, 12, 14], [4, 10, 9], [0, 3, 7], [9, 2, 16], [16, 13, 20], [19, 11, 8], [1, 15, 4], [14, 5, 1], [7, 6, 19]]
poi-20-19-1	[0, 4, 17, 11, 14, 3, 9, 1, 5, 2, 12, 7, 18, 19, 6, 20]	[[7, 16, 20], [17, 13, 14], [0, 15, 17], [14, 10, 5], [5, 8, 7]]
poi-20-19-2	[0, 19, 18, 7, 1, 5, 9, 3, 14, 11, 6, 15, 20]	[[11, 17, 6], [9, 10, 14], [19, 8, 7], [0, 16, 19], [5, 2, 9], [6, 4, 20], [7, 12, 5], [14, 13, 11]]
poi-20-19-3	[0, 15, 6, 11, 14, 3, 9, 5, 1, 18, 19, 20]	[[6, 16, 11], [11, 13, 14], [9, 12, 1], [18, 7, 19], [14, 10, 3], [19, 4, 20], [0, 17, 6], [1, 8, 18], [3, 2, 9]]
poi-20-2-1	[0, 8, 5, 3, 7, 19, 6, 17, 18, 14, 10, 2, 13, 15, 12, 20]	[[0, 16, 7], [10, 9, 15], [7, 4, 6], [15, 11, 20], [6, 1, 10]]
poi-20-2-2	[0, 8, 16, 6, 17, 14, 10, 13, 15, 11, 12, 20]	[[14, 18, 10], [6, 7, 14], [15, 9, 11], [16, 19, 6], [11, 3, 20], [10, 1, 13], [13, 2, 15], [8, 4, 16], [0, 5, 8]]
poi-20-2-3	[0, 8, 16, 4, 14, 10, 2, 13, 15, 12, 20]	[[14, 18, 10], [10, 17, 2], [12, 11, 20], [16, 19, 4], [13, 9, 15], [4, 6, 14], [8, 7, 16], [2, 1, 13], [15, 3, 12], [0, 5, 8]]
poi-20-20-1	[0, 4, 18, 2, 13, 19, 1, 9, 15, 3, 14, 17, 11, 8, 7, 5, 20]	[[17, 12, 8], [7, 16, 20], [0, 6, 17], [8, 10, 7]]
poi-20-20-2	[0, 16, 7, 8, 11, 3, 15, 9, 19, 13, 2, 20]	[[9, 18, 19], [8, 17, 11], [11, 12, 3], [7, 6, 8], [2, 5, 20], [3, 14, 15], [15, 1, 9], [19, 4, 2], [0, 10, 7]]
poi-20-20-3	[0, 5, 7, 8, 11, 3, 15, 9, 19, 13, 20]	[[8, 17, 11], [0, 16, 5], [11, 10, 3], [3, 14, 15], [19, 2, 13], [7, 12, 8], [9, 4, 19], [5, 6, 7], [15, 18, 9], [13, 1, 20]]
poi-20-21-1	[0, 5, 1, 8, 12, 2, 3, 15, 14, 10, 4, 13, 7, 19, 18, 20]	[[8, 17, 14], [5, 11, 8], [0, 9, 5], [14, 16, 10], [10, 6, 20]]

poi-20-21-2	[0, 18, 13, 4, 16, 10, 2, 12, 8, 1, 5, 20]	[[8, 17, 1], [10, 15, 2], [13, 14, 16], [18, 7, 13], [0, 19, 18], [1, 11, 5], [2, 3, 8], [16, 6, 10], [5, 9, 20]]
poi-20-21-3	[0, 9, 5, 1, 8, 12, 4, 10, 16, 7, 13, 20]	[[0, 18, 5], [13, 17, 20], [10, 6, 16], [7, 19, 13], [16, 15, 7], [4, 14, 10], [1, 2, 12], [12, 3, 4], [5, 11, 1]]
poi-20-22-1	[0, 4, 19, 10, 8, 11, 1, 2, 16, 5, 9, 12, 7, 13, 15, 17, 20]	[[8, 18, 1], [0, 14, 8], [1, 6, 7], [7, 3, 20]]
poi-20-22-2	[0, 17, 13, 15, 18, 8, 11, 1, 2, 16, 9, 12, 7, 3, 20]	[[9, 14, 7], [15, 10, 11], [11, 6, 16], [7, 4, 20], [16, 5, 9], [0, 19, 15]]
poi-20-22-3	[0, 17, 15, 18, 8, 11, 1, 2, 16, 7, 13, 20]	[[15, 12, 18], [7, 3, 13], [11, 6, 1], [1, 9, 2], [18, 10, 11], [2, 5, 16], [0, 19, 15], [16, 14, 7], [13, 4, 20]]
poi-20-23-1	[0, 1, 2, 11, 7, 3, 19, 12, 6, 17, 5, 8, 13, 16, 18, 15, 9, 20]	[[3, 14, 8], [0, 10, 3], [8, 4, 20]]
poi-20-23-2	[0, 1, 2, 7, 3, 5, 17, 8, 13, 16, 18, 9, 20]	[[7, 19, 3], [5, 6, 8], [3, 12, 5], [18, 14, 20], [0, 15, 1], [8, 4, 18], [2, 10, 7], [1, 11, 2]]
poi-20-23-3	[0, 1, 2, 11, 3, 5, 8, 16, 18, 9, 20]	[[9, 15, 20], [18, 14, 9], [0, 19, 1], [11, 12, 3], [8, 4, 16], [5, 17, 8], [2, 10, 11], [16, 13, 18], [3, 6, 5], [1, 7, 2]]
poi-20-24-1	[0, 15, 16, 1, 18, 12, 19, 9, 4, 3, 8, 17, 11, 13, 10, 20]	[[16, 6, 19], [17, 14, 20], [0, 5, 16], [19, 2, 8], [8, 7, 17]]
poi-20-24-2	[0, 15, 16, 1, 18, 12, 19, 6, 8, 5, 11, 13, 10, 20]	[[5, 17, 11], [0, 14, 15], [16, 9, 19], [8, 3, 5], [15, 2, 16], [19, 4, 8], [11, 7, 20]]
poi-20-24-3	[0, 10, 13, 11, 8, 6, 12, 18, 1, 16, 5, 20]	[[13, 17, 11], [18, 15, 16], [0, 14, 13], [12, 19, 18], [11, 4, 8], [16, 2, 5], [6, 3, 12], [5, 7, 20], [8, 9, 6]]
poi-20-25-1	[0, 7, 11, 10, 5, 8, 15, 9, 3, 1, 16, 12, 19, 18, 17, 20]	[[3, 6, 19], [18, 4, 20], [0, 14, 9], [19, 13, 18], [9, 2, 3]]
poi-20-25-2	[0, 17, 18, 13, 16, 12, 2, 9, 6, 4, 7, 20]	[[12, 19, 2], [4, 11, 7], [0, 14, 13], [2, 15, 9], [9, 8, 6], [6, 5, 4], [16, 3, 12], [13, 1, 16], [7, 10, 20]]
poi-20-25-3	[0, 17, 18, 13, 16, 1, 3, 2, 19, 6, 4, 20]	[[1, 15, 2], [18, 14, 13], [6, 11, 4], [0, 10, 18], [16, 12, 1], [13, 5, 16], [4, 7, 20], [19, 8, 6], [2, 9, 19]]
poi-20-3-1	[0, 17, 13, 19, 4, 15, 2, 8, 16, 10, 12, 11, 18, 3, 1, 9, 20]	[[12, 14, 3], [8, 5, 12], [3, 6, 20], [0, 7, 8]]

poi-20-3-2	[0, 13, 4, 15, 2, 8, 10, 12, 3, 18, 9, 20]	[[0, 19, 13], [13, 17, 4], [18, 16, 9], [12, 14, 3], [9, 6, 20], [8, 5, 10], [4, 7, 8], [3, 1, 18], [10, 11, 12]]
poi-20-3-3	[0, 17, 13, 4, 15, 2, 8, 16, 3, 18, 9, 20]	[[18, 14, 9], [8, 11, 16], [9, 12, 20], [13, 7, 2], [17, 19, 13], [3, 1, 18], [2, 5, 8], [0, 6, 17], [16, 10, 3]]
poi-20-4-1	[0, 14, 13, 5, 9, 17, 11, 16, 3, 10, 4, 1, 15, 6, 20]	[[0, 18, 13], [6, 12, 20], [13, 7, 9], [11, 19, 4], [9, 8, 11], [4, 2, 6]]
poi-20-4-2	[0, 12, 6, 1, 15, 16, 11, 17, 9, 5, 13, 20]	[[11, 19, 17], [5, 14, 13], [1, 10, 16], [17, 8, 9], [13, 18, 20], [0, 2, 6], [16, 3, 11], [6, 4, 1], [9, 7, 5]]
poi-20-4-3	[0, 18, 14, 13, 5, 9, 17, 16, 15, 1, 6, 12, 20]	[[9, 19, 17], [16, 4, 1], [1, 3, 6], [12, 7, 20], [6, 2, 12], [5, 8, 9], [0, 10, 5], [17, 11, 16]]
poi-20-5-1	[0, 6, 18, 10, 8, 11, 16, 12, 9, 5, 4, 7, 3, 19, 20]	[[11, 14, 12], [19, 13, 20], [12, 2, 5], [8, 1, 11], [0, 15, 8], [5, 17, 19]]
poi-20-5-2	[0, 19, 9, 5, 2, 16, 1, 8, 10, 6, 13, 20]	[[10, 17, 20], [16, 14, 1], [9, 4, 5], [1, 11, 8], [0, 15, 19], [8, 18, 10], [19, 3, 9], [5, 7, 2], [2, 12, 16]]
poi-20-5-3	[0, 13, 19, 2, 9, 12, 14, 11, 1, 15, 20]	[[1, 18, 15], [11, 8, 1], [2, 7, 9], [0, 6, 13], [14, 10, 11], [19, 5, 2], [13, 17, 19], [12, 16, 14], [15, 3, 20], [9, 4, 12]]
poi-20-6-1	[0, 6, 13, 4, 1, 19, 11, 8, 5, 7, 9, 17, 14, 20]	[[13, 18, 1], [11, 12, 8], [0, 3, 13], [1, 2, 11], [8, 15, 7], [7, 10, 14], [14, 16, 20]]
poi-20-6-2	[0, 6, 18, 1, 4, 2, 19, 11, 5, 7, 9, 17, 20]	[[0, 16, 6], [11, 8, 5], [17, 14, 20], [18, 13, 4], [4, 12, 11], [6, 3, 18], [7, 10, 17], [5, 15, 7]]
poi-20-6-3	[0, 17, 9, 7, 5, 11, 19, 4, 1, 6, 20]	[[4, 18, 1], [7, 15, 5], [17, 16, 9], [0, 14, 17], [19, 13, 4], [1, 12, 6], [5, 8, 11], [6, 3, 20], [9, 10, 7], [11, 2, 19]]
poi-20-7-1	[0, 9, 13, 5, 12, 17, 6, 10, 4, 14, 11, 8, 19, 7, 3, 15, 16, 1, 20]	[[4, 18, 20], [0, 2, 4]]
poi-20-7-2	[0, 9, 13, 5, 12, 4, 14, 16, 11, 8, 19, 7, 3, 15, 20]	[[9, 17, 12], [0, 1, 9], [8, 18, 20], [14, 2, 8], [12, 10, 4], [4, 6, 14]]
poi-20-7-3	[0, 9, 13, 4, 5, 14, 11, 8, 19, 7, 3, 15, 20]	[[3, 18, 20], [4, 17, 5], [14, 2, 19], [0, 1, 9], [9, 12, 13], [13, 6, 4], [5, 10, 14], [19, 16, 3]]
poi-20-8-1	[0, 19, 5, 1, 4, 7, 18, 9, 17, 15, 11, 2, 12, 8, 6, 14, 16, 20]	[[11, 10, 20], [0, 13, 18], [18, 3, 11]]

poi-20-8-2	[0, 16, 14, 6, 12, 2, 11, 17, 18, 7, 1, 19, 20]	[[2, 15, 11], [17, 3, 18], [0, 10, 14], [6, 4, 2], [14, 8, 6], [11, 9, 17], [1, 5, 20], [18, 13, 1]]
poi-20-8-3	[0, 5, 16, 4, 9, 11, 17, 18, 7, 1, 20]	[[0, 19, 5], [17, 12, 18], [7, 10, 1], [16, 14, 4], [5, 8, 16], [1, 13, 20], [4, 6, 9], [9, 2, 11], [11, 15, 17], [18, 3, 7]]
poi-20-9-1	[0, 14, 15, 6, 13, 10, 17, 18, 12, 8, 7, 2, 11, 9, 16, 19, 20]	[[2, 5, 20], [17, 4, 2], [0, 3, 6], [6, 1, 17]]
poi-20-9-2	[0, 14, 5, 3, 13, 10, 8, 7, 2, 11, 9, 16, 19, 20]	[[2, 17, 20], [8, 18, 7], [13, 4, 8], [3, 1, 13], [0, 15, 5], [7, 12, 2], [5, 6, 3]]
poi-20-9-3	[0, 19, 16, 9, 11, 2, 7, 10, 13, 6, 15, 14, 20]	[[2, 18, 7], [10, 17, 6], [0, 12, 16], [16, 8, 2], [6, 1, 15], [15, 3, 14], [7, 4, 10], [14, 5, 20]]
poi-30-1-1	[0, 21, 16, 9, 14, 29, 7, 10, 22, 27, 28, 18, 26, 17, 5, 23, 11, 13, 4, 8, 12, 1, 30]	[[23, 25, 4], [0, 24, 16], [16, 3, 14], [22, 6, 18], [1, 2, 30], [14, 15, 22], [18, 19, 23], [4, 20, 1]]
poi-30-1-2	[0, 21, 24, 16, 15, 3, 14, 29, 7, 6, 19, 11, 23, 5, 13, 8, 12, 1, 2, 30]	[[7, 28, 6], [0, 27, 15], [13, 25, 8], [12, 20, 1], [15, 10, 14], [5, 26, 13], [1, 9, 30], [6, 18, 11], [11, 17, 5], [14, 22, 7], [8, 4, 12]]
poi-30-1-3	[0, 21, 24, 16, 9, 3, 14, 29, 7, 6, 19, 23, 11, 13, 8, 12, 1, 30]	[[3, 22, 14], [6, 18, 19], [0, 15, 9], [14, 10, 7], [8, 4, 12], [12, 2, 1], [9, 27, 3], [19, 17, 23], [7, 28, 6], [23, 5, 11], [1, 25, 30], [13, 20, 8], [11, 26, 13]]
poi-30-10-1	[0, 24, 18, 29, 28, 23, 8, 14, 27, 12, 15, 17, 19, 11, 10, 21, 20, 22, 25, 5, 13, 3, 1, 30]	[[0, 16, 18], [15, 6, 10], [3, 26, 30], [8, 9, 15], [10, 7, 3], [18, 4, 28], [28, 2, 8]]
poi-30-10-2	[0, 13, 5, 25, 22, 20, 21, 10, 11, 19, 15, 14, 8, 23, 28, 4, 18, 30]	[[0, 26, 13], [15, 12, 14], [14, 9, 8], [10, 6, 11], [5, 3, 22], [11, 27, 15], [28, 16, 4], [23, 17, 28], [18, 1, 30], [4, 24, 18], [13, 29, 5], [8, 2, 23], [22, 7, 10]]
poi-30-10-3	[0, 13, 5, 25, 21, 10, 11, 19, 17, 2, 8, 23, 28, 29, 16, 18, 30]	[[13, 20, 5], [5, 22, 25], [28, 15, 29], [17, 14, 2], [8, 12, 28], [18, 4, 30], [25, 3, 21], [0, 1, 13], [10, 7, 11], [11, 27, 17], [29, 26, 16], [16, 24, 18], [2, 9, 8], [21, 6, 10]]
poi-30-11-1	[0, 2, 5, 24, 3, 20, 14, 7, 16, 25, 21, 15, 17, 9, 11, 22, 6, 4, 12, 10, 29, 1, 27, 19, 30]	[[6, 28, 1], [3, 8, 16], [9, 18, 6], [1, 23, 19], [0, 26, 3], [16, 13, 9]]

poi-30-11-2	[0, 2, 5, 24, 3, 20, 14, 16, 25, 15, 17, 9, 22, 6, 11, 1, 27, 19, 30]	[[1, 28, 27], [0, 26, 24], [25, 13, 17], [17, 10, 22], [24, 8, 3], [14, 21, 25], [22, 4, 6], [20, 7, 14], [6, 12, 11], [27, 23, 30], [11, 29, 1], [3, 18, 20]]
poi-30-11-3	[0, 19, 27, 1, 11, 22, 9, 17, 15, 25, 16, 14, 20, 18, 24, 5, 2, 30]	[[19, 29, 1], [5, 23, 2], [9, 21, 15], [15, 13, 25], [22, 12, 9], [16, 8, 20], [20, 3, 18], [0, 28, 19], [25, 7, 16], [1, 10, 11], [11, 6, 22], [2, 26, 30], [18, 4, 5]]
poi-30-12-1	[0, 13, 7, 19, 1, 17, 25, 10, 21, 11, 14, 24, 4, 23, 15, 28, 29, 22, 20, 27, 16, 6, 26, 30]	[[25, 18, 14], [0, 3, 1], [16, 2, 30], [14, 9, 15], [1, 8, 25], [27, 5, 16], [15, 12, 27]]
poi-30-12-2	[0, 26, 6, 5, 20, 22, 29, 28, 15, 24, 14, 11, 17, 1, 25, 10, 7, 13, 30]	[[25, 19, 13], [6, 16, 5], [17, 8, 1], [13, 3, 30], [0, 2, 6], [29, 23, 15], [22, 27, 29], [11, 18, 17], [24, 4, 11], [5, 12, 22], [15, 9, 24], [1, 21, 25]]
poi-30-12-3	[0, 6, 5, 20, 22, 29, 28, 15, 24, 14, 11, 17, 1, 25, 10, 26, 30]	[[1, 19, 25], [0, 16, 6], [22, 12, 29], [15, 9, 24], [17, 8, 1], [14, 18, 11], [6, 7, 20], [29, 3, 28], [25, 23, 26], [11, 13, 17], [26, 2, 30], [24, 4, 14], [28, 21, 15], [20, 27, 22]]
poi-30-13-1	[0, 22, 20, 7, 26, 21, 16, 24, 2, 13, 28, 5, 1, 12, 17, 8, 25, 29, 15, 3, 4, 19, 10, 30]	[[29, 18, 30], [22, 14, 21], [0, 9, 22], [8, 6, 29], [28, 27, 8], [21, 11, 24], [24, 23, 28]]
poi-30-13-2	[0, 9, 10, 4, 19, 15, 29, 17, 1, 5, 28, 13, 2, 24, 16, 7, 20, 30]	[[7, 26, 20], [4, 25, 15], [0, 22, 9], [9, 14, 4], [1, 12, 5], [24, 11, 16], [5, 27, 2], [15, 3, 29], [16, 21, 7], [20, 18, 30], [17, 8, 1], [29, 6, 17], [2, 23, 24]]
poi-30-13-3	[0, 22, 9, 10, 4, 19, 12, 1, 5, 28, 13, 2, 24, 16, 7, 20, 30]	[[22, 29, 10], [16, 27, 7], [2, 23, 24], [10, 15, 4], [12, 8, 1], [19, 25, 12], [5, 6, 13], [13, 18, 2], [24, 11, 16], [1, 17, 5], [4, 3, 19], [7, 21, 20], [0, 14, 22], [20, 26, 30]]
poi-30-14-1	[0, 10, 19, 25, 3, 9, 5, 17, 16, 1, 26, 11, 4, 8, 12, 28, 13, 7, 6, 14, 27, 20, 21, 30]	[[1, 24, 26], [26, 23, 12], [19, 18, 1], [21, 2, 30], [12, 22, 7], [0, 29, 19], [7, 15, 21]]
poi-30-14-2	[0, 21, 6, 7, 13, 28, 12, 8, 11, 1, 16, 17, 5, 9, 3, 25, 19, 10, 30]	[[28, 23, 8], [0, 20, 6], [7, 15, 13], [16, 27, 25], [8, 4, 11], [19, 18, 10], [10, 2, 30], [11, 26, 1], [6, 14, 7], [13, 22, 28], [25, 29, 19], [1, 24, 16]]

poi-30-14-3	[0, 10, 29, 19, 25, 3, 9, 5, 17, 16, 1, 24, 26, 11, 4, 8, 12, 23, 21, 2, 30]	[[16, 22, 26], [19, 20, 9], [23, 15, 21], [9, 14, 16], [21, 6, 30], [29, 18, 19], [11, 13, 12], [0, 27, 29], [26, 28, 11], [12, 7, 23]]
poi-30-15-1	[0, 28, 14, 12, 9, 7, 11, 17, 2, 15, 3, 21, 24, 1, 22, 16, 25, 8, 6, 19, 5, 20, 29, 18, 30]	[[17, 27, 15], [15, 26, 16], [16, 23, 29], [14, 4, 17], [29, 10, 30], [0, 13, 14]]
poi-30-15-2	[0, 18, 29, 19, 6, 8, 25, 16, 22, 1, 21, 3, 2, 17, 11, 7, 14, 28, 30]	[[8, 26, 16], [3, 24, 2], [16, 23, 22], [14, 12, 30], [19, 5, 6], [2, 4, 7], [22, 27, 1], [7, 9, 14], [1, 15, 3], [18, 10, 29], [29, 20, 19], [0, 13, 18]]
poi-30-15-3	[0, 13, 29, 19, 6, 8, 25, 16, 22, 1, 21, 3, 11, 7, 14, 28, 30]	[[16, 27, 22], [6, 26, 25], [21, 24, 3], [29, 20, 19], [0, 18, 13], [13, 10, 29], [11, 9, 7], [22, 2, 1], [7, 17, 14], [19, 5, 6], [3, 4, 11], [25, 23, 16], [1, 15, 21], [14, 12, 30]]
poi-30-16-1	[0, 26, 5, 9, 19, 16, 28, 7, 4, 6, 2, 13, 12, 27, 22, 11, 24, 10, 21, 29, 15, 23, 30]	[[12, 25, 27], [15, 17, 30], [27, 18, 11], [11, 1, 15], [13, 14, 12], [0, 3, 5], [5, 8, 6], [6, 20, 13]]
poi-30-16-2	[0, 5, 9, 19, 16, 28, 7, 6, 20, 12, 25, 27, 22, 11, 24, 10, 21, 23, 30]	[[24, 29, 10], [27, 18, 22], [12, 13, 27], [7, 4, 6], [16, 26, 7], [0, 3, 5], [10, 1, 23], [23, 17, 30], [22, 15, 24], [20, 14, 12], [6, 2, 20], [5, 8, 16]]
poi-30-16-3	[0, 5, 9, 19, 16, 26, 22, 11, 27, 12, 25, 23, 21, 29, 15, 17, 30]	[[19, 28, 16], [23, 24, 21], [12, 14, 25], [27, 13, 12], [16, 7, 26], [26, 6, 22], [11, 20, 27], [0, 4, 19], [29, 1, 15], [25, 2, 23], [17, 8, 30], [15, 3, 17], [21, 10, 29], [22, 18, 11]]
poi-30-17-1	[0, 15, 25, 10, 7, 3, 2, 1, 16, 29, 22, 13, 18, 27, 20, 28, 19, 11, 17, 9, 6, 24, 14, 30]	[[15, 26, 25], [28, 23, 19], [18, 12, 28], [9, 8, 30], [25, 4, 18], [0, 21, 15], [19, 5, 9]]
poi-30-17-2	[0, 15, 3, 2, 1, 16, 29, 22, 13, 18, 27, 20, 12, 17, 11, 6, 9, 24, 14, 30]	[[27, 28, 20], [15, 25, 3], [12, 23, 11], [0, 21, 15], [29, 10, 13], [24, 8, 30], [1, 7, 29], [11, 5, 24], [20, 19, 12], [13, 4, 27], [3, 26, 1]]
poi-30-17-3	[0, 15, 26, 3, 2, 1, 16, 29, 22, 13, 18, 12, 11, 6, 9, 24, 14, 30]	[[18, 28, 12], [22, 27, 13], [13, 20, 18], [11, 23, 6], [0, 17, 15], [3, 7, 2], [2, 4, 22], [15, 25, 26], [24, 8, 14], [26, 10, 3], [14, 21, 30], [12, 19, 11], [6, 5, 24]]
poi-30-18-1	[0, 24, 17, 15, 18, 25, 7, 6, 4, 27, 10, 5, 21, 8, 26, 2, 1, 13, 9, 22, 19, 30]	[[18, 20, 6], [2, 23, 1], [27, 16, 21], [0, 29, 18], [26, 3, 2], [1, 11, 19], [6, 28, 27], [19, 12, 30], [21, 14, 26]]

poi-30-18-2	[0, 12, 22, 9, 13, 1, 2, 3, 26, 8, 14, 21, 27, 6, 7, 25, 24, 17, 30]	[[0, 28, 12], [2, 23, 3], [6, 20, 25], [21, 16, 27], [12, 11, 22], [27, 4, 6], [24, 15, 17], [22, 19, 2], [8, 5, 21], [25, 29, 24], [3, 10, 8], [17, 18, 30]]
poi-30-18-3	[0, 22, 9, 13, 1, 3, 26, 2, 8, 5, 10, 27, 6, 25, 7, 28, 12, 30]	[[6, 29, 25], [7, 24, 28], [5, 21, 10], [25, 20, 7], [26, 16, 8], [8, 14, 5], [22, 11, 1], [3, 23, 26], [0, 15, 22], [27, 18, 6], [28, 17, 30], [10, 4, 27], [1, 19, 3]]
poi-30-19-1	[0, 10, 4, 8, 2, 28, 16, 9, 25, 18, 15, 1, 13, 11, 19, 24, 26, 22, 27, 5, 7, 12, 17, 30]	[[0, 23, 10], [8, 21, 9], [11, 20, 5], [9, 6, 15], [15, 3, 11], [10, 29, 8], [5, 14, 30]]
poi-30-19-2	[0, 17, 13, 11, 19, 24, 26, 22, 3, 16, 28, 2, 8, 29, 4, 10, 7, 12, 30]	[[8, 27, 10], [16, 25, 28], [2, 21, 8], [13, 20, 24], [26, 18, 3], [10, 14, 7], [3, 6, 16], [24, 1, 26], [0, 15, 13], [28, 9, 2], [12, 23, 30], [7, 5, 12]]
poi-30-19-3	[0, 13, 11, 19, 24, 26, 22, 25, 9, 16, 3, 27, 5, 7, 12, 17, 30]	[[16, 21, 3], [17, 23, 30], [5, 10, 7], [13, 20, 19], [12, 4, 17], [19, 1, 24], [26, 15, 22], [25, 6, 9], [9, 28, 16], [22, 8, 25], [24, 18, 26], [7, 14, 12], [3, 2, 5], [0, 29, 13]]
poi-30-2-1	[0, 19, 11, 3, 5, 14, 2, 18, 9, 6, 21, 22, 23, 29, 16, 27, 20, 25, 10, 4, 17, 30]	[[20, 28, 10], [22, 26, 16], [14, 24, 18], [10, 15, 30], [11, 7, 14], [0, 8, 11], [16, 13, 27], [18, 12, 22], [27, 1, 20]]
poi-30-2-2	[0, 11, 3, 14, 18, 9, 6, 21, 29, 16, 13, 25, 20, 10, 4, 17, 30]	[[25, 28, 20], [20, 27, 4], [13, 26, 25], [11, 19, 3], [6, 12, 21], [0, 8, 11], [16, 23, 13], [3, 7, 14], [14, 2, 18], [18, 24, 6], [4, 1, 17], [29, 22, 16], [17, 15, 30], [21, 5, 29]]
poi-30-2-3	[0, 11, 3, 5, 9, 6, 21, 29, 16, 27, 4, 20, 10, 15, 17, 30]	[[29, 22, 16], [6, 18, 21], [27, 13, 4], [9, 12, 6], [11, 7, 3], [5, 24, 9], [20, 28, 10], [15, 1, 17], [3, 2, 5], [10, 23, 15], [4, 25, 20], [0, 8, 11], [21, 14, 29], [16, 26, 27], [17, 19, 30]]
poi-30-20-1	[0, 21, 27, 9, 8, 19, 24, 18, 23, 13, 10, 7, 11, 3, 26, 2, 20, 25, 16, 17, 12, 1, 5, 6, 30]	[[13, 28, 11], [0, 22, 19], [23, 15, 13], [19, 29, 23], [11, 14, 17], [17, 4, 30]]
poi-30-20-2	[0, 6, 1, 5, 16, 25, 22, 11, 7, 3, 13, 23, 24, 19, 8, 9, 27, 21, 30]	[[23, 29, 24], [25, 26, 22], [16, 20, 25], [24, 18, 19], [13, 15, 23], [1, 12, 5], [3, 10, 13], [22, 2, 11], [19, 14, 30], [0, 4, 1], [5, 17, 16], [11, 28, 3]]

poi-30-20-3	[0, 9, 8, 19, 24, 23, 15, 13, 3, 26, 22, 25, 16, 5, 1, 6, 30]	[[13, 28, 3], [5, 12, 1], [22, 11, 25], [15, 10, 13], [1, 4, 30], [26, 2, 22], [25, 20, 16], [16, 17, 5], [3, 7, 26], [9, 27, 19], [0, 14, 9], [19, 21, 24], [24, 29, 23], [23, 18, 15]]
poi-30-21-1	[0, 2, 29, 18, 8, 10, 5, 15, 17, 13, 26, 20, 3, 6, 11, 28, 4, 24, 16, 14, 25, 12, 23, 30]	[[11, 27, 24], [24, 22, 25], [0, 21, 18], [26, 9, 6], [25, 7, 30], [18, 1, 26], [6, 19, 11]]
poi-30-21-2	[0, 23, 7, 12, 14, 25, 4, 28, 11, 6, 3, 20, 26, 5, 10, 8, 2, 30]	[[2, 29, 30], [0, 21, 23], [11, 19, 6], [10, 18, 2], [26, 17, 5], [4, 24, 28], [3, 9, 26], [14, 16, 25], [23, 1, 14], [6, 13, 3], [28, 27, 11], [25, 22, 4], [5, 15, 10]]
poi-30-21-3	[0, 23, 14, 25, 12, 11, 19, 6, 3, 20, 26, 5, 10, 8, 2, 30]	[[11, 28, 19], [19, 27, 6], [6, 21, 3], [26, 17, 5], [3, 15, 20], [10, 18, 8], [5, 13, 10], [25, 4, 12], [14, 16, 25], [12, 22, 11], [0, 7, 23], [23, 24, 14], [2, 1, 30], [8, 29, 2], [20, 9, 26]]
poi-30-22-1	[0, 3, 4, 11, 8, 12, 25, 15, 5, 7, 29, 20, 2, 23, 16, 6, 10, 18, 13, 26, 17, 30]	[[23, 27, 16], [11, 24, 8], [12, 28, 5], [18, 22, 30], [16, 14, 18], [5, 1, 20], [20, 9, 23], [0, 19, 11], [8, 21, 12]]
poi-30-22-2	[0, 3, 11, 21, 12, 25, 15, 5, 29, 20, 23, 27, 14, 13, 26, 17, 30]	[[26, 22, 17], [13, 18, 26], [14, 10, 13], [0, 4, 3], [23, 16, 27], [29, 2, 20], [12, 1, 25], [11, 24, 21], [20, 9, 23], [3, 8, 11], [21, 28, 12], [27, 6, 14], [17, 19, 30], [25, 7, 29]]
poi-30-22-3	[0, 3, 12, 28, 15, 5, 29, 20, 2, 23, 27, 16, 14, 13, 26, 17, 30]	[[28, 24, 15], [17, 21, 30], [12, 11, 28], [29, 9, 20], [3, 8, 12], [5, 7, 29], [16, 10, 14], [0, 4, 3], [20, 1, 23], [15, 25, 5], [14, 19, 13], [23, 6, 16], [26, 22, 17], [13, 18, 26]]
poi-30-23-1	[0, 12, 19, 2, 5, 28, 8, 23, 22, 4, 20, 29, 18, 9, 25, 3, 24, 21, 6, 27, 14, 13, 11, 30]	[[22, 17, 9], [2, 16, 22], [14, 10, 30], [9, 1, 3], [3, 7, 6], [6, 26, 14], [0, 15, 2]]
poi-30-23-2	[0, 2, 23, 22, 4, 20, 29, 18, 9, 25, 3, 24, 21, 6, 27, 14, 13, 11, 30]	[[20, 28, 9], [6, 26, 27], [22, 17, 20], [0, 12, 2], [24, 7, 6], [13, 5, 30], [25, 1, 24], [23, 16, 22], [2, 19, 23], [9, 8, 25], [14, 15, 13], [27, 10, 14]]

poi-30-23-3	[0, 2, 22, 4, 23, 5, 28, 8, 15, 1, 3, 24, 6, 27, 14, 13, 11, 30]	[[0, 19, 2], [23, 16, 28], [13, 12, 30], [3, 25, 24], [14, 10, 13], [24, 7, 6], [27, 26, 14], [6, 21, 27], [4, 20, 23], [28, 18, 1], [2, 29, 22], [22, 17, 4], [1, 9, 3]]
poi-30-24-1	[0, 19, 21, 5, 12, 29, 13, 8, 7, 3, 16, 25, 26, 22, 14, 27, 4, 28, 23, 15, 20, 1, 9, 24, 30]	[[4, 11, 30], [29, 10, 8], [22, 2, 4], [0, 6, 29], [3, 17, 22], [8, 18, 3]]
poi-30-24-2	[0, 19, 5, 12, 29, 10, 3, 25, 26, 6, 17, 2, 23, 28, 4, 15, 20, 1, 9, 30]	[[10, 18, 3], [2, 14, 4], [19, 13, 12], [4, 11, 1], [0, 21, 19], [26, 27, 17], [1, 24, 30], [12, 8, 10], [25, 16, 26], [17, 22, 2], [3, 7, 25]]
poi-30-24-3	[0, 19, 21, 12, 29, 10, 3, 25, 26, 17, 2, 28, 23, 20, 1, 9, 30]	[[1, 24, 30], [3, 18, 25], [0, 15, 19], [10, 14, 3], [12, 13, 29], [25, 16, 26], [19, 5, 21], [28, 4, 23], [21, 8, 12], [2, 22, 28], [29, 6, 10], [17, 27, 2], [23, 11, 1], [26, 7, 17]]
poi-30-25-1	[0, 1, 13, 10, 11, 22, 5, 23, 17, 19, 27, 28, 2, 16, 14, 7, 9, 3, 25, 29, 8, 21, 26, 30]	[[29, 18, 21], [21, 15, 26], [3, 12, 29], [11, 6, 17], [26, 20, 30], [0, 4, 11], [17, 24, 3]]
poi-30-25-2	[0, 26, 21, 8, 18, 29, 3, 9, 7, 28, 27, 17, 23, 11, 10, 20, 13, 1, 30]	[[3, 25, 9], [11, 22, 10], [7, 16, 28], [9, 14, 7], [8, 12, 29], [17, 6, 11], [29, 5, 3], [27, 19, 17], [26, 15, 8], [28, 2, 27], [10, 24, 30], [0, 4, 26]]
poi-30-25-3	[0, 26, 21, 8, 18, 29, 3, 25, 28, 27, 17, 23, 11, 10, 13, 1, 30]	[[0, 24, 21], [13, 22, 1], [21, 15, 18], [25, 14, 28], [3, 9, 25], [23, 6, 11], [11, 5, 10], [10, 4, 13], [28, 2, 27], [1, 20, 30], [29, 7, 3], [18, 12, 29], [27, 16, 17], [17, 19, 23]]
poi-30-3-1	[0, 7, 24, 21, 19, 28, 13, 15, 4, 1, 3, 18, 5, 27, 20, 26, 22, 14, 16, 11, 23, 17, 30]	[[21, 10, 15], [18, 29, 20], [11, 9, 30], [3, 12, 18], [0, 2, 21], [15, 6, 3], [20, 8, 22], [22, 25, 11]]
poi-30-3-2	[0, 17, 23, 11, 16, 14, 8, 20, 18, 12, 3, 15, 13, 28, 19, 21, 7, 30]	[[8, 29, 20], [20, 27, 18], [14, 26, 8], [23, 25, 16], [7, 24, 30], [16, 22, 14], [0, 2, 17], [17, 9, 23], [3, 1, 15], [21, 10, 7], [28, 5, 21], [18, 4, 3], [15, 6, 28]]
poi-30-3-3	[0, 17, 23, 2, 25, 27, 5, 18, 12, 3, 15, 13, 28, 19, 21, 24, 7, 30]	[[5, 29, 18], [25, 22, 5], [0, 11, 17], [19, 10, 21], [28, 6, 19], [15, 4, 28], [3, 26, 15], [2, 8, 25], [17, 9, 23], [18, 20, 12], [23, 16, 2], [21, 14, 30], [12, 1, 3]]
poi-30-4-1	[0, 18, 1, 10, 11, 25, 6, 13, 24, 28, 16, 19, 2, 8, 20, 12, 21, 14, 26, 23, 27, 17, 22, 9, 5, 30]	[[28, 15, 8], [18, 7, 28], [0, 4, 18], [27, 29, 30], [8, 3, 27]]

poi-30-4-2	[0, 5, 9, 22, 17, 2, 19, 16, 8, 20, 12, 21, 14, 13, 6, 25, 11, 10, 1, 18, 4, 30]	[[9, 27, 22], [20, 26, 21], [18, 23, 30], [11, 7, 18], [16, 28, 20], [0, 29, 9], [22, 3, 17], [21, 15, 11], [17, 24, 16]]
poi-30-4-3	[0, 5, 9, 22, 17, 2, 19, 8, 20, 12, 21, 14, 13, 6, 25, 11, 18, 30]	[[17, 28, 2], [19, 24, 20], [0, 29, 5], [20, 23, 21], [2, 16, 19], [9, 27, 22], [14, 15, 25], [5, 4, 9], [22, 3, 17], [11, 7, 18], [21, 26, 14], [25, 10, 11], [18, 1, 30]]
poi-30-5-1	[0, 16, 13, 29, 2, 3, 17, 14, 6, 7, 23, 4, 12, 19, 5, 18, 1, 24, 28, 25, 21, 27, 30]	[[19, 26, 18], [14, 10, 19], [0, 22, 14], [28, 8, 25], [1, 9, 28], [25, 15, 27], [18, 11, 1], [27, 20, 30]]
poi-30-5-2	[0, 16, 13, 29, 2, 3, 17, 23, 4, 12, 19, 5, 18, 9, 28, 8, 25, 21, 27, 20, 30]	[[19, 26, 5], [27, 22, 30], [25, 15, 27], [17, 14, 23], [18, 11, 9], [0, 10, 17], [9, 24, 25], [5, 1, 18], [4, 6, 19], [23, 7, 4]]
poi-30-5-3	[0, 16, 13, 21, 27, 25, 8, 28, 9, 26, 19, 12, 4, 23, 17, 3, 2, 29, 30]	[[27, 20, 28], [21, 15, 27], [26, 5, 19], [19, 1, 4], [0, 22, 13], [23, 10, 3], [3, 6, 29], [9, 18, 26], [29, 14, 30], [28, 24, 9], [13, 11, 21], [4, 7, 23]]
poi-30-6-1	[0, 11, 24, 2, 17, 23, 14, 22, 25, 4, 3, 21, 9, 12, 20, 19, 13, 27, 6, 5, 7, 16, 10, 8, 30]	[[6, 26, 5], [7, 18, 30], [5, 15, 7], [22, 1, 9], [0, 29, 22], [9, 28, 6]]
poi-30-6-2	[0, 24, 2, 17, 23, 4, 3, 9, 12, 19, 13, 27, 6, 5, 7, 10, 8, 30]	[[0, 29, 24], [5, 26, 7], [4, 25, 3], [9, 21, 12], [10, 16, 30], [19, 28, 6], [2, 14, 23], [24, 11, 2], [3, 1, 9], [7, 18, 10], [23, 22, 4], [6, 15, 5], [12, 20, 19]]
poi-30-6-3	[0, 8, 10, 16, 7, 5, 6, 27, 13, 19, 12, 20, 14, 23, 17, 2, 24, 11, 30]	[[2, 29, 11], [5, 28, 13], [7, 26, 5], [14, 25, 23], [23, 22, 2], [20, 21, 14], [19, 9, 20], [13, 1, 19], [11, 4, 30], [16, 15, 7], [10, 18, 16], [0, 3, 10]]
poi-30-7-1	[0, 17, 10, 19, 6, 27, 1, 11, 4, 3, 15, 20, 5, 9, 25, 28, 14, 18, 12, 26, 23, 29, 30]	[[28, 24, 18], [18, 7, 26], [1, 2, 3], [0, 13, 19], [15, 22, 5], [19, 8, 1], [5, 21, 28], [26, 16, 30]]
poi-30-7-2	[0, 29, 23, 26, 12, 28, 25, 9, 5, 20, 15, 3, 1, 27, 19, 10, 30]	[[12, 24, 28], [20, 22, 3], [5, 21, 20], [28, 18, 25], [1, 11, 27], [10, 8, 30], [26, 7, 12], [19, 6, 10], [25, 14, 5], [27, 2, 19], [0, 17, 29], [29, 13, 23], [23, 16, 26], [3, 4, 1]]

poi-30-7-3	[0, 10, 19, 27, 1, 3, 15, 20, 5, 9, 25, 12, 26, 23, 29, 30]	[[9, 28, 25], [25, 18, 12], [29, 17, 30], [1, 22, 3], [26, 16, 23], [20, 14, 5], [27, 11, 1], [0, 6, 10], [19, 2, 27], [23, 13, 29], [15, 21, 20], [10, 8, 19], [3, 4, 15], [5, 7, 9], [12, 24, 26]]
poi-30-8-1	[0, 18, 29, 25, 14, 1, 8, 2, 20, 23, 21, 13, 16, 15, 24, 12, 26, 17, 10, 5, 4, 28, 30]	[[12, 27, 26], [0, 11, 29], [13, 19, 24], [10, 6, 30], [26, 9, 10], [24, 3, 12], [29, 7, 20], [20, 22, 13]]
poi-30-8-2	[0, 18, 28, 4, 5, 26, 27, 12, 24, 15, 21, 13, 20, 2, 8, 1, 14, 25, 30]	[[26, 17, 12], [24, 16, 15], [0, 11, 28], [5, 9, 26], [25, 6, 30], [12, 3, 24], [28, 10, 5], [20, 7, 2], [15, 19, 21], [21, 23, 13], [2, 29, 25], [13, 22, 20]]
poi-30-8-3	[0, 18, 28, 4, 5, 26, 27, 24, 21, 13, 16, 7, 2, 8, 1, 25, 30]	[[13, 15, 16], [16, 20, 7], [2, 14, 8], [27, 19, 24], [26, 12, 27], [8, 29, 25], [5, 9, 26], [4, 17, 5], [24, 3, 21], [28, 10, 4], [25, 6, 30], [21, 23, 13], [7, 11, 2], [0, 22, 28]]
poi-30-9-1	[0, 22, 9, 5, 28, 2, 19, 26, 15, 1, 4, 7, 13, 27, 3, 14, 25, 11, 18, 20, 23, 30]	[[11, 24, 18], [19, 21, 1], [0, 16, 5], [18, 17, 30], [5, 6, 19], [1, 8, 4], [3, 29, 25], [25, 10, 11], [4, 12, 3]]
poi-30-9-2	[0, 23, 11, 10, 25, 29, 27, 13, 7, 4, 1, 15, 26, 19, 2, 16, 5, 9, 22, 30]	[[11, 24, 10], [1, 21, 15], [9, 20, 30], [0, 18, 11], [16, 17, 9], [10, 14, 29], [13, 12, 4], [4, 8, 1], [19, 6, 16], [29, 3, 13], [15, 28, 19]]
poi-30-9-3	[0, 23, 11, 10, 25, 29, 27, 13, 7, 4, 1, 15, 26, 2, 16, 22, 30]	[[10, 24, 25], [0, 20, 11], [25, 17, 29], [16, 9, 22], [2, 6, 16], [22, 5, 30], [7, 21, 4], [26, 28, 2], [29, 14, 27], [27, 3, 13], [11, 18, 10], [13, 12, 7], [1, 19, 26], [4, 8, 1]]
poi-40-1-1	[0, 5, 12, 24, 37, 30, 11, 39, 18, 13, 7, 2, 28, 16, 33, 38, 19, 36, 26, 8, 6, 17, 31, 15, 27, 14, 10, 34, 4, 1, 32, 22, 40]	[[0, 35, 12], [12, 29, 30], [22, 3, 40], [2, 25, 15], [30, 9, 2], [1, 20, 22], [15, 21, 14], [14, 23, 1]]
poi-40-1-2	[0, 5, 24, 37, 11, 39, 13, 2, 28, 16, 33, 38, 19, 26, 36, 21, 15, 31, 27, 10, 34, 4, 1, 32, 20, 40]	[[39, 30, 13], [10, 25, 4], [27, 23, 10], [11, 18, 39], [21, 17, 31], [0, 12, 5], [2, 9, 26], [26, 8, 36], [20, 22, 40], [4, 3, 20], [37, 29, 11], [13, 7, 2], [36, 6, 21], [31, 14, 27], [5, 35, 37]]

poi-40-1-3	[0, 5, 12, 24, 37, 13, 2, 28, 16, 33, 38, 19, 26, 36, 21, 15, 27, 10, 34, 1, 4, 20, 40]	[[2, 39, 28], [5, 35, 12], [20, 32, 40], [0, 29, 5], [27, 25, 10], [34, 22, 4], [13, 7, 2], [24, 9, 13], [28, 11, 16], [21, 17, 15], [12, 30, 24], [15, 31, 27], [16, 18, 19], [26, 8, 36], [19, 6, 26], [4, 3, 20], [36, 14, 21], [10, 23, 34]]
poi-40-10-1	[0, 23, 13, 25, 33, 2, 38, 36, 7, 8, 6, 1, 20, 19, 37, 26, 21, 10, 22, 32, 11, 34, 18, 15, 39, 35, 9, 16, 4, 3, 40]	[[32, 31, 18], [37, 30, 21], [21, 29, 32], [36, 27, 8], [8, 28, 37], [0, 24, 2], [15, 5, 16], [2, 12, 36], [18, 17, 15], [16, 14, 40]]
poi-40-10-2	[0, 3, 14, 12, 33, 2, 38, 36, 7, 8, 6, 1, 20, 37, 26, 21, 22, 32, 29, 31, 17, 15, 35, 40]	[[15, 39, 35], [6, 28, 20], [31, 11, 17], [0, 4, 3], [36, 24, 7], [22, 34, 31], [8, 30, 6], [14, 23, 12], [7, 5, 8], [17, 18, 15], [37, 9, 21], [21, 10, 22], [35, 13, 40], [20, 19, 37], [3, 16, 14], [12, 25, 38], [38, 27, 36]]
poi-40-10-3	-	-
poi-40-11-1	[0, 13, 7, 24, 22, 31, 17, 28, 21, 16, 11, 30, 15, 5, 2, 23, 34, 29, 32, 12, 37, 6, 1, 3, 10, 35, 9, 20, 33, 4, 38, 19, 39, 40]	[[2, 27, 29], [7, 25, 17], [0, 26, 7], [3, 14, 4], [17, 8, 2], [4, 36, 39], [29, 18, 3]]
poi-40-11-2	[0, 26, 25, 31, 17, 11, 30, 15, 5, 2, 23, 34, 32, 12, 1, 3, 10, 35, 9, 33, 4, 38, 19, 13, 40]	[[30, 27, 2], [12, 6, 1], [1, 29, 3], [13, 39, 40], [19, 7, 13], [11, 16, 30], [26, 22, 25], [9, 14, 4], [4, 36, 19], [23, 8, 32], [3, 18, 9], [2, 20, 23], [17, 21, 11], [25, 28, 17], [0, 24, 26], [32, 37, 12]]
poi-40-11-3	[0, 13, 19, 38, 4, 33, 9, 35, 10, 3, 1, 29, 34, 23, 2, 5, 15, 30, 17, 31, 22, 7, 40]	[[38, 27, 4], [22, 26, 7], [0, 39, 13], [13, 36, 19], [17, 16, 31], [1, 37, 29], [31, 25, 22], [19, 24, 38], [29, 32, 34], [34, 12, 2], [3, 6, 1], [7, 28, 40], [2, 8, 15], [15, 11, 30], [4, 20, 33], [35, 14, 3], [33, 18, 35], [30, 21, 17]]
poi-40-12-1	[0, 20, 16, 37, 13, 33, 26, 7, 24, 27, 1, 18, 15, 17, 30, 3, 32, 2, 31, 34, 11, 19, 29, 14, 12, 5, 9, 39, 23, 10, 38, 40]	[[23, 36, 10], [0, 25, 13], [10, 35, 40], [34, 6, 11], [19, 4, 39], [11, 28, 19], [18, 8, 17], [13, 22, 18], [17, 21, 2]]
poi-40-12-2	[0, 20, 16, 37, 13, 33, 26, 7, 24, 1, 18, 15, 17, 30, 3, 32, 2, 31, 34, 6, 19, 29, 14, 12, 5, 10, 38, 35, 40]	[[0, 28, 37], [7, 27, 24], [6, 39, 29], [37, 25, 33], [5, 23, 10], [24, 11, 18], [29, 9, 5], [15, 21, 17], [18, 8, 15], [17, 4, 6], [10, 36, 40], [33, 22, 7]]

poi-40-12-3	[0, 20, 37, 13, 33, 7, 24, 1, 18, 3, 30, 32, 2, 31, 34, 6, 19, 29, 14, 12, 5, 10, 38, 35, 40]	[[12, 39, 10], [34, 4, 19], [7, 27, 24], [24, 11, 1], [1, 8, 18], [10, 36, 40], [3, 17, 32], [33, 22, 7], [13, 26, 33], [32, 21, 34], [14, 23, 12], [19, 9, 14], [18, 15, 3], [37, 16, 13], [20, 25, 37], [0, 28, 20]]
poi-40-13-1	[0, 31, 22, 23, 5, 20, 37, 19, 27, 12, 33, 1, 35, 6, 14, 25, 34, 36, 24, 7, 15, 2, 3, 21, 8, 29, 32, 16, 18, 38, 40]	[[16, 39, 40], [23, 30, 20], [3, 28, 8], [12, 17, 25], [8, 13, 32], [25, 11, 36], [32, 26, 16], [36, 4, 3], [20, 9, 12], [0, 10, 23]]
poi-40-13-2	[0, 18, 16, 32, 26, 29, 13, 8, 21, 15, 3, 2, 4, 34, 25, 14, 33, 12, 19, 37, 20, 23, 22, 31, 40]	[[0, 39, 18], [12, 27, 19], [33, 1, 12], [19, 17, 37], [25, 35, 14], [37, 5, 20], [2, 36, 4], [16, 28, 29], [14, 6, 33], [15, 7, 2], [31, 10, 40], [20, 9, 23], [29, 24, 15], [4, 11, 25], [23, 30, 31], [18, 38, 16]]
poi-40-13-3	[0, 31, 5, 20, 37, 19, 27, 12, 25, 34, 4, 2, 3, 15, 21, 8, 29, 26, 32, 16, 18, 38, 22, 40]	[[18, 39, 38], [12, 35, 25], [27, 33, 12], [4, 36, 2], [5, 9, 20], [25, 11, 34], [37, 6, 27], [15, 7, 21], [22, 10, 40], [8, 13, 29], [34, 14, 4], [20, 17, 37], [38, 23, 22], [0, 1, 5], [21, 28, 8], [29, 30, 18], [3, 24, 15]]
poi-40-14-1	[0, 16, 27, 12, 24, 30, 38, 2, 10, 37, 7, 13, 28, 33, 6, 26, 34, 25, 23, 1, 17, 3, 29, 36, 20, 32, 11, 35, 21, 5, 4, 40]	[[1, 39, 3], [27, 31, 30], [30, 19, 10], [21, 18, 40], [28, 22, 23], [29, 14, 11], [0, 15, 27], [11, 8, 21], [10, 9, 28]]
poi-40-14-2	[0, 16, 30, 24, 38, 2, 37, 7, 13, 33, 26, 34, 25, 23, 1, 17, 3, 29, 20, 32, 11, 21, 5, 40]	[[32, 35, 11], [0, 15, 16], [16, 27, 30], [17, 36, 29], [25, 31, 17], [37, 9, 13], [38, 19, 2], [2, 10, 37], [33, 6, 26], [5, 4, 40], [11, 8, 21], [20, 14, 32], [29, 39, 20], [21, 18, 5], [26, 22, 25], [30, 12, 38], [13, 28, 33]]
poi-40-14-3	[0, 21, 35, 11, 32, 20, 29, 3, 17, 1, 23, 25, 34, 26, 6, 33, 28, 13, 7, 38, 24, 30, 16, 40]	[[20, 36, 29], [13, 37, 38], [24, 15, 30], [29, 39, 3], [11, 14, 32], [17, 22, 1], [21, 5, 35], [38, 10, 24], [6, 9, 28], [3, 31, 17], [30, 2, 16], [16, 4, 40], [32, 8, 20], [28, 12, 13], [0, 18, 21], [1, 19, 6], [35, 27, 11]]
poi-40-15-1	[0, 25, 18, 26, 14, 23, 7, 34, 5, 28, 12, 29, 20, 6, 16, 33, 10, 17, 30, 19, 38, 37, 21, 32, 36, 15, 2, 24, 31, 13, 11, 4, 40]	[[16, 39, 10], [24, 27, 40], [10, 9, 30], [30, 1, 36], [36, 3, 24], [7, 35, 12], [0, 8, 7], [12, 22, 16]]

poi-40-15-2	[0, 25, 18, 26, 14, 23, 7, 34, 28, 12, 29, 20, 39, 33, 10, 30, 17, 32, 21, 36, 15, 24, 31, 2, 40]	[[33, 38, 10], [29, 6, 20], [36, 3, 15], [2, 4, 40], [15, 13, 24], [34, 5, 28], [23, 27, 34], [30, 1, 21], [0, 8, 23], [10, 9, 30], [24, 11, 2], [39, 19, 33], [20, 16, 39], [12, 22, 29], [28, 35, 12], [21, 37, 36]]
poi-40-15-3	[0, 25, 18, 14, 23, 7, 34, 28, 12, 29, 20, 39, 19, 38, 37, 32, 36, 15, 24, 31, 2, 40]	[[34, 35, 28], [20, 33, 39], [28, 27, 12], [14, 26, 23], [0, 22, 18], [18, 21, 14], [37, 17, 32], [29, 16, 20], [23, 30, 7], [39, 9, 19], [36, 3, 15], [19, 10, 37], [31, 13, 2], [32, 1, 36], [15, 4, 24], [7, 5, 34], [24, 11, 31], [12, 6, 29], [2, 8, 40]]
poi-40-16-1	[0, 11, 33, 28, 39, 6, 35, 17, 21, 7, 34, 13, 10, 26, 19, 32, 4, 16, 5, 38, 1, 29, 14, 27, 18, 24, 2, 9, 12, 15, 37, 40]	[[1, 36, 18], [34, 31, 32], [12, 30, 15], [0, 25, 17], [32, 8, 4], [15, 20, 40], [4, 22, 1], [18, 23, 12], [17, 3, 34]]
poi-40-16-2	[0, 37, 15, 28, 6, 35, 17, 21, 7, 34, 13, 10, 26, 19, 32, 8, 16, 5, 38, 36, 18, 27, 24, 2, 20, 40]	[[6, 33, 17], [7, 31, 34], [10, 22, 32], [17, 25, 7], [0, 12, 15], [18, 9, 2], [32, 4, 8], [34, 3, 10], [8, 1, 38], [28, 39, 6], [15, 30, 28], [38, 14, 36], [2, 23, 20], [36, 29, 18], [20, 11, 40]]
poi-40-16-3	[0, 11, 33, 35, 6, 39, 21, 7, 26, 19, 32, 8, 16, 5, 38, 36, 18, 27, 24, 2, 20, 37, 40]	[[39, 34, 7], [0, 30, 11], [38, 31, 36], [33, 25, 35], [2, 23, 20], [35, 28, 6], [26, 13, 19], [6, 17, 39], [36, 29, 18], [24, 9, 2], [20, 12, 40], [8, 4, 16], [11, 15, 33], [32, 22, 8], [16, 1, 38], [7, 3, 26], [19, 10, 32], [18, 14, 24]]
poi-40-17-1	[0, 10, 8, 23, 22, 6, 13, 14, 29, 5, 11, 35, 2, 3, 37, 27, 9, 15, 12, 24, 17, 28, 19, 38, 33, 21, 20, 34, 18, 16, 40]	[[8, 39, 14], [2, 32, 37], [37, 31, 15], [14, 36, 2], [38, 30, 21], [15, 26, 28], [28, 4, 38], [0, 25, 8], [34, 1, 40], [21, 7, 34]]
poi-40-17-2	[0, 8, 23, 22, 6, 5, 29, 2, 32, 27, 9, 15, 12, 17, 28, 19, 38, 33, 21, 7, 18, 16, 40]	[[6, 36, 5], [7, 34, 18], [9, 31, 12], [18, 10, 16], [8, 11, 23], [23, 13, 6], [12, 24, 17], [2, 3, 32], [28, 39, 19], [19, 26, 33], [16, 1, 40], [17, 4, 28], [29, 35, 2], [33, 30, 21], [5, 14, 29], [32, 37, 9], [21, 20, 7], [0, 25, 8]]

poi-40-17-3	[0, 16, 18, 7, 21, 30, 33, 38, 19, 4, 9, 27, 32, 2, 29, 5, 22, 6, 23, 8, 40]	[[38, 39, 19], [6, 36, 23], [29, 35, 5], [7, 34, 21], [27, 31, 32], [21, 26, 30], [0, 25, 16], [33, 17, 38], [19, 15, 4], [30, 28, 33], [18, 20, 7], [4, 24, 9], [16, 10, 18], [8, 1, 40], [32, 37, 2], [5, 3, 22], [23, 11, 8], [9, 12, 27], [22, 13, 6], [2, 14, 29]]
poi-40-18-1	[0, 34, 12, 23, 39, 24, 36, 2, 1, 26, 8, 19, 29, 17, 10, 5, 32, 7, 15, 20, 9, 22, 27, 38, 14, 35, 3, 16, 33, 6, 40]	[[2, 37, 19], [5, 30, 32], [24, 25, 2], [19, 21, 5], [0, 18, 24], [14, 28, 6], [32, 4, 15], [15, 13, 22], [6, 11, 40], [27, 31, 14]]
poi-40-18-2	[0, 34, 12, 39, 24, 1, 2, 19, 29, 10, 30, 32, 7, 15, 20, 9, 27, 22, 38, 3, 16, 6, 40]	[[2, 37, 19], [6, 35, 40], [22, 31, 38], [38, 28, 16], [24, 26, 1], [10, 21, 30], [30, 5, 32], [0, 11, 34], [1, 36, 2], [16, 33, 6], [15, 13, 20], [7, 4, 15], [12, 25, 39], [39, 18, 24], [34, 23, 12], [19, 8, 29], [20, 14, 22], [29, 17, 10]]
poi-40-18-3	[0, 34, 12, 39, 24, 1, 2, 19, 29, 10, 5, 30, 32, 20, 9, 22, 27, 35, 3, 16, 33, 40]	[[2, 37, 19], [12, 36, 39], [22, 31, 35], [33, 28, 40], [24, 26, 1], [1, 25, 2], [34, 23, 12], [10, 21, 5], [20, 13, 9], [30, 15, 32], [35, 38, 16], [32, 4, 20], [29, 17, 10], [39, 18, 24], [16, 6, 33], [19, 8, 29], [0, 11, 34], [9, 14, 22], [5, 7, 30]]
poi-40-19-1	[0, 10, 36, 7, 27, 17, 26, 18, 6, 15, 14, 13, 1, 29, 30, 3, 21, 5, 34, 12, 25, 38, 19, 24, 4, 28, 31, 32, 16, 40]	[[10, 37, 26], [3, 33, 5], [6, 20, 14], [0, 35, 10], [1, 8, 3], [31, 22, 40], [14, 39, 1], [25, 9, 4], [26, 23, 6], [4, 11, 31], [5, 2, 25]]
poi-40-19-2	[0, 10, 35, 36, 7, 27, 17, 18, 6, 15, 14, 39, 8, 3, 34, 5, 12, 25, 38, 24, 9, 11, 32, 40]	[[8, 33, 3], [12, 30, 25], [15, 1, 14], [39, 20, 8], [14, 13, 39], [9, 31, 11], [0, 16, 10], [11, 28, 32], [17, 37, 18], [10, 26, 17], [25, 19, 38], [3, 2, 34], [32, 22, 40], [38, 4, 9], [34, 21, 5], [18, 23, 15], [5, 29, 12]]
poi-40-19-3	[0, 16, 13, 14, 1, 29, 24, 19, 38, 25, 12, 30, 3, 8, 20, 6, 18, 17, 27, 7, 36, 35, 10, 40]	[[18, 37, 17], [8, 34, 20], [12, 2, 30], [0, 31, 16], [3, 21, 8], [13, 39, 14], [30, 5, 3], [38, 4, 12], [16, 32, 13], [20, 33, 6], [17, 26, 7], [36, 15, 10], [10, 22, 40], [24, 9, 38], [14, 11, 29], [6, 23, 18], [29, 28, 24]]

poi-40-2-1	[0, 10, 31, 38, 32, 30, 37, 9, 35, 34, 23, 8, 4, 19, 11, 22, 18, 7, 5, 33, 16, 1, 13, 17, 39, 20, 26, 36, 12, 15, 40]	[[13, 29, 17], [4, 28, 16], [0, 25, 10], [16, 24, 13], [34, 14, 4], [17, 27, 26], [10, 21, 37], [37, 2, 34], [26, 3, 12], [12, 6, 40]]
poi-40-2-2	[0, 10, 25, 31, 32, 38, 9, 37, 35, 34, 23, 8, 11, 22, 18, 7, 5, 33, 16, 1, 13, 17, 29, 26, 36, 12, 15, 40]	[[17, 39, 26], [31, 30, 38], [36, 27, 12], [13, 24, 17], [25, 21, 31], [38, 14, 35], [35, 2, 34], [34, 28, 11], [18, 4, 13], [12, 3, 40], [11, 19, 18], [26, 20, 36], [0, 6, 25]]
poi-40-2-3	[0, 10, 25, 31, 32, 38, 9, 37, 35, 34, 23, 8, 18, 7, 5, 33, 16, 1, 13, 17, 29, 26, 36, 12, 15, 40]	[[0, 28, 32], [16, 27, 13], [32, 30, 38], [13, 24, 17], [7, 22, 16], [26, 20, 36], [17, 39, 26], [38, 21, 9], [36, 2, 12], [9, 4, 35], [18, 11, 7], [8, 19, 18], [15, 6, 40], [12, 3, 15], [35, 14, 8]]
poi-40-20-1	[0, 24, 25, 29, 13, 26, 17, 10, 21, 4, 20, 5, 23, 15, 35, 14, 36, 19, 6, 27, 18, 12, 9, 39, 30, 32, 28, 2, 31, 40]	[[26, 34, 10], [30, 33, 28], [21, 8, 15], [0, 7, 25], [28, 38, 40], [9, 22, 30], [25, 16, 13], [13, 37, 26], [15, 3, 27], [10, 11, 21], [27, 1, 9]]
poi-40-20-2	[0, 7, 13, 37, 26, 10, 11, 21, 23, 15, 35, 14, 36, 19, 6, 1, 22, 39, 30, 32, 28, 2, 31, 40]	[[37, 34, 10], [28, 33, 2], [0, 29, 7], [11, 4, 21], [23, 5, 15], [7, 25, 13], [21, 20, 23], [32, 38, 28], [13, 16, 37], [19, 27, 6], [36, 3, 19], [15, 8, 36], [1, 12, 22], [22, 9, 32], [10, 17, 11], [6, 18, 1], [2, 24, 40]]
poi-40-20-3	[0, 31, 2, 38, 33, 28, 32, 30, 39, 22, 1, 19, 36, 35, 15, 23, 21, 11, 10, 26, 37, 7, 40]	[[19, 34, 36], [1, 27, 19], [7, 25, 40], [37, 24, 7], [22, 18, 1], [33, 6, 32], [39, 12, 22], [36, 3, 35], [23, 20, 21], [31, 8, 33], [15, 4, 23], [21, 5, 11], [0, 29, 31], [26, 13, 37], [35, 14, 15], [10, 16, 26], [32, 9, 39], [11, 17, 10]]
poi-40-21-1	[0, 24, 26, 28, 38, 39, 21, 10, 18, 20, 23, 15, 27, 17, 25, 7, 12, 2, 32, 34, 22, 8, 9, 11, 3, 36, 1, 5, 33, 37, 14, 40]	[[11, 35, 40], [15, 30, 25], [21, 31, 10], [9, 19, 11], [2, 16, 9], [0, 13, 38], [10, 29, 15], [38, 4, 21], [25, 6, 2]]
poi-40-21-2	[0, 14, 36, 3, 5, 1, 33, 35, 22, 34, 32, 2, 12, 7, 25, 17, 27, 23, 20, 39, 38, 28, 26, 24, 40]	[[22, 8, 32], [27, 29, 23], [36, 37, 3], [32, 6, 2], [3, 19, 35], [2, 16, 7], [35, 9, 22], [0, 11, 36], [39, 21, 38], [23, 18, 20], [38, 4, 28], [20, 10, 39], [28, 31, 24], [17, 30, 27], [24, 13, 40], [7, 15, 17]]

poi-40-21-3	[0, 14, 37, 1, 5, 33, 35, 22, 34, 32, 25, 17, 27, 10, 20, 23, 39, 38, 28, 26, 24, 40]	[[38, 31, 28], [39, 30, 38], [23, 29, 39], [28, 21, 26], [20, 18, 23], [32, 16, 25], [10, 15, 20], [0, 13, 14], [35, 9, 22], [24, 8, 40], [5, 36, 33], [22, 6, 34], [33, 19, 35], [27, 12, 10], [34, 2, 32], [1, 3, 5], [26, 4, 24], [25, 7, 27], [14, 11, 1]]
poi-40-22-1	[0, 23, 6, 31, 26, 9, 33, 7, 3, 1, 21, 12, 20, 19, 14, 36, 5, 38, 34, 11, 29, 4, 10, 24, 25, 35, 17, 22, 32, 40]	[[20, 39, 36], [24, 37, 35], [36, 30, 38], [34, 28, 24], [3, 27, 21], [21, 18, 20], [38, 13, 34], [0, 8, 9], [33, 16, 3], [35, 15, 40], [9, 2, 33]]
poi-40-22-2	[0, 22, 15, 8, 9, 26, 31, 7, 3, 12, 18, 13, 11, 29, 34, 38, 5, 30, 36, 14, 19, 20, 39, 23, 40]	[[15, 37, 26], [12, 21, 18], [31, 2, 7], [3, 27, 12], [18, 1, 13], [23, 32, 40], [7, 16, 3], [0, 17, 22], [36, 24, 39], [22, 35, 15], [39, 6, 23], [13, 4, 11], [11, 10, 38], [38, 25, 30], [30, 28, 36], [26, 33, 31]]
poi-40-22-3	[0, 23, 6, 31, 26, 33, 16, 27, 12, 18, 13, 11, 29, 34, 38, 14, 19, 36, 30, 28, 22, 32, 40]	[[19, 35, 28], [18, 25, 13], [38, 24, 14], [12, 21, 18], [0, 39, 23], [33, 8, 16], [6, 15, 31], [23, 7, 6], [11, 4, 34], [22, 17, 40], [28, 37, 22], [14, 20, 19], [16, 1, 27], [26, 2, 33], [13, 10, 11], [27, 3, 12], [34, 5, 38], [31, 9, 26]]
poi-40-23-1	-	-
poi-40-23-2	[0, 8, 32, 17, 34, 26, 9, 1, 28, 35, 29, 14, 10, 3, 27, 33, 22, 18, 12, 20, 30, 15, 38, 39, 40]	[[32, 31, 17], [1, 4, 28], [18, 37, 12], [15, 5, 38], [9, 7, 1], [20, 19, 30], [34, 25, 9], [35, 24, 33], [38, 2, 40], [12, 6, 20], [17, 11, 34], [22, 16, 18], [28, 21, 35], [0, 23, 32], [33, 13, 22], [30, 36, 15]]
poi-40-23-3	[0, 39, 38, 15, 30, 20, 12, 18, 33, 27, 3, 10, 29, 14, 28, 1, 9, 34, 17, 11, 2, 40]	[[12, 37, 18], [15, 36, 30], [34, 32, 17], [11, 31, 2], [17, 26, 11], [20, 22, 12], [27, 13, 3], [33, 16, 27], [9, 25, 34], [18, 6, 33], [38, 5, 15], [28, 7, 1], [2, 23, 40], [0, 8, 38], [29, 35, 14], [14, 4, 28], [1, 24, 9], [30, 19, 20], [3, 21, 29]]
poi-40-24-1	[0, 39, 24, 15, 18, 26, 1, 5, 3, 19, 21, 28, 38, 20, 37, 2, 16, 8, 4, 7, 23, 35, 29, 34, 10, 36, 25, 9, 27, 22, 40]	[[3, 33, 28], [35, 32, 34], [25, 30, 40], [34, 17, 25], [18, 13, 3], [37, 31, 4], [28, 14, 37], [15, 12, 18], [0, 11, 15], [4, 6, 35]]

poi-40-24-2	[0, 22, 27, 9, 29, 34, 10, 32, 7, 4, 8, 16, 2, 37, 14, 21, 19, 3, 5, 1, 26, 18, 24, 40]	[[32, 35, 7], [24, 39, 40], [21, 33, 3], [10, 36, 32], [29, 17, 34], [3, 13, 26], [26, 12, 18], [4, 6, 16], [37, 20, 14], [2, 38, 37], [18, 15, 24], [7, 23, 4], [0, 11, 27], [34, 25, 10], [14, 28, 21], [27, 30, 29], [16, 31, 2]]
poi-40-24-3	[0, 24, 15, 18, 26, 5, 3, 19, 21, 37, 20, 6, 35, 23, 32, 10, 34, 25, 9, 27, 22, 40]	[[10, 36, 34], [26, 33, 5], [25, 30, 9], [9, 29, 27], [20, 16, 6], [27, 11, 22], [34, 17, 25], [0, 28, 24], [32, 7, 10], [21, 31, 37], [3, 14, 21], [6, 4, 23], [18, 38, 26], [37, 2, 20], [22, 39, 40], [5, 1, 3], [23, 8, 32], [15, 12, 18], [24, 13, 15]]
poi-40-25-1	-	-
poi-40-25-2	[0, 30, 28, 5, 34, 19, 31, 21, 9, 36, 33, 7, 26, 2, 6, 3, 13, 11, 24, 16, 38, 17, 35, 1, 8, 14, 40]	[[3, 37, 11], [21, 29, 9], [26, 23, 2], [5, 22, 21], [6, 18, 3], [16, 15, 17], [7, 10, 26], [9, 4, 7], [2, 27, 6], [11, 32, 24], [17, 12, 40], [0, 39, 30], [24, 20, 16], [30, 25, 5]]
poi-40-25-3	[0, 14, 8, 1, 32, 16, 24, 20, 13, 3, 6, 2, 26, 23, 10, 36, 33, 31, 19, 34, 5, 30, 25, 39, 40]	[[23, 7, 10], [36, 29, 31], [16, 38, 24], [13, 11, 3], [6, 4, 23], [25, 22, 39], [31, 21, 34], [14, 15, 16], [0, 35, 14], [10, 9, 36], [20, 27, 13], [34, 28, 30], [30, 12, 25], [39, 17, 40], [3, 18, 6], [24, 37, 20]]
poi-40-3-1	[0, 13, 28, 24, 2, 33, 7, 8, 22, 14, 23, 17, 27, 29, 32, 26, 31, 30, 19, 3, 12, 35, 20, 5, 38, 37, 34, 1, 36, 21, 4, 18, 16, 40]	[[21, 39, 40], [0, 15, 28], [37, 25, 21], [8, 11, 17], [3, 10, 37], [28, 6, 8], [17, 9, 3]]
poi-40-3-2	[0, 16, 18, 4, 1, 34, 37, 38, 20, 35, 12, 30, 31, 26, 32, 29, 23, 14, 22, 8, 7, 28, 13, 40]	[[13, 39, 40], [1, 36, 34], [8, 33, 7], [4, 25, 1], [28, 24, 13], [32, 27, 29], [14, 9, 8], [7, 2, 28], [12, 3, 30], [37, 5, 38], [18, 6, 4], [30, 10, 32], [23, 11, 14], [34, 21, 37], [0, 15, 18], [29, 17, 23], [38, 19, 12]]
poi-40-3-3	[0, 13, 28, 7, 8, 22, 14, 23, 17, 29, 32, 26, 31, 30, 12, 35, 37, 34, 1, 4, 16, 40]	[[0, 39, 13], [1, 38, 4], [34, 36, 1], [17, 27, 29], [13, 24, 28], [32, 19, 31], [37, 21, 34], [4, 25, 16], [23, 15, 17], [28, 9, 7], [31, 3, 30], [12, 5, 35], [35, 20, 37], [29, 6, 32], [30, 10, 12], [14, 11, 23], [8, 33, 14], [7, 2, 8], [16, 18, 40]]
poi-40-4-1	-	-

poi-40-4-2	[0, 33, 1, 21, 4, 24, 13, 36, 22, 14, 23, 20, 34, 37, 9, 3, 6, 7, 8, 10, 29, 30, 15, 35, 18, 31, 40]	[[31, 39, 40], [0, 38, 33], [4, 27, 36], [30, 17, 15], [15, 12, 18], [7, 11, 30], [23, 5, 9], [36, 2, 14], [3, 32, 7], [18, 25, 31], [14, 16, 23], [9, 28, 3], [1, 19, 4], [33, 26, 1]]
poi-40-4-3	[0, 31, 18, 15, 35, 12, 17, 11, 3, 6, 9, 37, 34, 20, 22, 36, 13, 24, 4, 21, 1, 33, 40]	[[6, 29, 37], [11, 28, 3], [33, 27, 40], [37, 23, 20], [1, 26, 33], [12, 30, 17], [18, 8, 12], [31, 39, 18], [21, 38, 1], [13, 32, 24], [24, 2, 4], [36, 5, 13], [3, 7, 6], [17, 10, 11], [20, 14, 22], [22, 16, 36], [4, 19, 21], [0, 25, 31]]
poi-40-5-1	[0, 13, 24, 17, 18, 7, 39, 28, 22, 2, 30, 27, 14, 33, 12, 16, 31, 26, 1, 5, 15, 20, 6, 34, 4, 10, 29, 32, 38, 25, 40]	[[33, 36, 31], [39, 23, 30], [0, 21, 17], [30, 35, 33], [20, 11, 4], [38, 8, 40], [15, 19, 20], [31, 37, 15], [4, 3, 38], [17, 9, 39]]
poi-40-5-2	[0, 8, 32, 29, 10, 4, 20, 15, 5, 1, 26, 16, 12, 33, 23, 9, 7, 39, 22, 28, 21, 13, 40]	[[20, 37, 15], [23, 36, 9], [33, 31, 23], [39, 30, 22], [29, 34, 10], [15, 19, 1], [9, 35, 7], [8, 38, 32], [28, 17, 21], [7, 18, 39], [21, 25, 40], [10, 6, 4], [32, 3, 29], [22, 2, 28], [4, 11, 20], [16, 14, 33], [0, 24, 8], [1, 27, 16]]
poi-40-5-3	[0, 8, 32, 29, 34, 6, 20, 11, 37, 36, 31, 16, 12, 33, 23, 9, 35, 39, 7, 18, 21, 13, 40]	[[31, 26, 16], [13, 24, 40], [39, 2, 18], [35, 22, 39], [37, 17, 36], [8, 38, 32], [23, 30, 35], [18, 28, 13], [34, 4, 6], [16, 27, 33], [11, 15, 37], [36, 1, 31], [29, 3, 34], [20, 5, 11], [32, 10, 29], [0, 25, 8], [33, 14, 23], [6, 19, 20]]
poi-40-6-1	[0, 38, 33, 19, 30, 5, 16, 34, 32, 3, 18, 20, 39, 1, 4, 14, 25, 35, 21, 37, 12, 15, 29, 28, 2, 9, 23, 22, 36, 27, 31, 40]	[[18, 17, 4], [0, 8, 30], [25, 13, 37], [30, 24, 5], [2, 10, 9], [9, 26, 40], [4, 6, 14], [5, 7, 18], [37, 11, 2]]
poi-40-6-2	[0, 33, 19, 30, 24, 5, 16, 32, 17, 3, 1, 4, 14, 25, 35, 21, 37, 29, 28, 2, 23, 22, 36, 27, 31, 40]	[[1, 39, 4], [35, 12, 37], [29, 10, 23], [32, 7, 17], [24, 11, 32], [14, 6, 25], [23, 9, 36], [4, 34, 14], [17, 18, 3], [3, 20, 1], [36, 26, 40], [37, 15, 29], [25, 13, 35], [33, 8, 24], [0, 38, 33]]

poi-40-6-3	[0, 31, 27, 22, 23, 2, 28, 29, 37, 21, 35, 25, 14, 4, 1, 3, 16, 5, 30, 19, 33, 40]	[[4, 39, 1], [0, 38, 31], [31, 36, 27], [21, 32, 35], [3, 20, 16], [14, 18, 4], [5, 17, 30], [19, 8, 40], [30, 24, 19], [37, 11, 21], [23, 10, 2], [16, 34, 5], [27, 26, 22], [25, 6, 14], [35, 13, 25], [1, 7, 3], [29, 12, 37], [22, 9, 23], [2, 15, 29]]
poi-40-7-1	[0, 2, 1, 3, 14, 35, 20, 16, 4, 33, 34, 5, 39, 30, 36, 10, 26, 18, 32, 27, 24, 6, 19, 22, 23, 28, 21, 15, 12, 13, 38, 40]	[[27, 31, 6], [39, 29, 10], [10, 25, 27], [5, 37, 39], [0, 11, 1], [33, 7, 5], [16, 17, 33], [6, 8, 40], [1, 9, 16]]
poi-40-7-2	[0, 38, 13, 12, 28, 23, 19, 6, 24, 27, 32, 18, 26, 10, 36, 39, 30, 7, 4, 33, 16, 20, 35, 14, 3, 40]	[[36, 29, 30], [28, 21, 23], [33, 17, 16], [3, 11, 40], [13, 8, 28], [0, 2, 13], [30, 5, 7], [16, 1, 3], [23, 15, 19], [19, 22, 6], [32, 37, 36], [24, 25, 32], [4, 34, 33], [6, 31, 24], [7, 9, 4]]
poi-40-7-3	[0, 38, 13, 12, 15, 21, 31, 27, 32, 18, 26, 10, 39, 30, 7, 4, 20, 16, 35, 14, 3, 40]	[[26, 36, 10], [4, 34, 20], [16, 33, 14], [10, 29, 39], [12, 28, 15], [32, 25, 18], [15, 23, 21], [31, 22, 27], [39, 37, 30], [30, 5, 7], [7, 11, 4], [18, 24, 26], [21, 19, 31], [13, 8, 12], [3, 9, 40], [0, 2, 13], [14, 1, 3], [27, 6, 32], [20, 17, 16]]
poi-40-8-1	[0, 9, 34, 18, 19, 11, 20, 13, 27, 26, 25, 30, 24, 28, 7, 39, 36, 16, 3, 5, 15, 6, 17, 33, 22, 1, 23, 35, 31, 37, 40]	[[28, 29, 16], [35, 21, 40], [13, 10, 25], [16, 2, 5], [6, 14, 22], [22, 32, 35], [30, 12, 28], [0, 38, 13], [25, 4, 30], [5, 8, 6]]
poi-40-8-2	[0, 31, 35, 23, 1, 14, 17, 6, 15, 5, 25, 4, 30, 7, 28, 26, 27, 20, 11, 18, 34, 9, 40]	[[30, 39, 7], [31, 37, 35], [28, 36, 26], [35, 32, 1], [18, 29, 40], [6, 8, 15], [4, 24, 30], [0, 21, 31], [5, 2, 25], [20, 13, 11], [1, 22, 14], [14, 38, 17], [11, 19, 18], [15, 3, 5], [17, 33, 6], [26, 10, 20], [7, 12, 28], [25, 16, 4]]
poi-40-8-3	[0, 9, 34, 18, 11, 20, 27, 26, 24, 30, 4, 25, 5, 15, 6, 17, 33, 14, 1, 23, 35, 31, 40]	[[26, 39, 24], [20, 38, 27], [0, 37, 34], [11, 29, 20], [6, 28, 14], [34, 19, 18], [31, 21, 40], [5, 3, 15], [18, 13, 11], [23, 12, 31], [25, 36, 5], [30, 16, 4], [4, 2, 25], [1, 32, 23], [24, 7, 30], [15, 8, 6], [27, 10, 26], [14, 22, 1]]

poi-40-9-1	[0, 9, 34, 13, 15, 16, 11, 2, 12, 18, 32, 25, 23, 35, 7, 29, 1, 39, 10, 20, 22, 30, 4, 14, 21, 37, 3, 5, 40]	[[25, 36, 23], [23, 28, 29], [12, 26, 32], [2, 24, 12], [32, 19, 25], [0, 8, 15], [3, 33, 40], [21, 38, 37], [22, 27, 21], [37, 17, 3], [15, 31, 2], [29, 6, 22]]
poi-40-9-2	[0, 9, 34, 13, 15, 8, 33, 3, 5, 22, 30, 4, 28, 7, 35, 23, 1, 39, 10, 26, 24, 12, 2, 11, 16, 40]	[[5, 38, 22], [33, 37, 3], [7, 36, 23], [39, 32, 24], [1, 29, 39], [4, 27, 28], [2, 19, 40], [28, 6, 7], [34, 17, 33], [3, 21, 5], [0, 31, 34], [24, 18, 2], [30, 14, 4], [22, 20, 30], [23, 25, 1]]
poi-40-9-3	[0, 9, 34, 13, 15, 16, 11, 12, 24, 26, 10, 39, 1, 23, 35, 7, 28, 4, 30, 14, 21, 5, 40]	[[21, 38, 5], [14, 37, 21], [23, 36, 35], [0, 33, 9], [9, 31, 13], [35, 29, 7], [7, 6, 28], [4, 22, 14], [24, 18, 26], [1, 19, 23], [16, 32, 12], [12, 2, 24], [13, 8, 15], [5, 17, 40], [15, 3, 16], [26, 20, 10], [10, 25, 1], [28, 27, 4]]