Master in Management + Analytics

Asignación de dispositivos para recarga de tarjeta de transporte

Autoría: *Costa, Juan*
Año académico: *2023*

UNIVERSIDAD TORCUATO DI TELLA

Master in Management + Analytics

Asignación de dispositivos para recarga de tarjeta de transporte

---

Por
Juan Costa

Tutor: **Javier Marenco, PhD**
Mayo, 2023

Resumen

Hoy en día las billeteras virtuales están en auge en varios lugares del mundo, uno de esos lugares es América Latina. Los *active users* crecen a un ritmo muy acelerado. Algunos ejemplos de billeteras son MercadoPago, Brubank, Nubank, entre otros. La innovación en este rubro es constante y cada año las Fintech sacan nuevos productos al mercado. En el caso de Argentina, un producto/feature que tuvo mucha adopción fue el *pago con QR.* Las billeteras pueden tener varios flujos de productos, como cuenta en USD, transferencias de dinero, pagos, fondeos, retiros, etc. Un flujo particular, que resulta muy importante, es el de *single player* (recarga de celular, recarga de tarjeta de transporte, pago de servicios, etc.), el cual es una vertiente muy importante en las billeteras virtuales.

En Argentina, la recarga de la tarjeta SUBE se realiza en primera instancia con una billetera virtual (se realiza el pago del monto a cargar) y en un segundo paso se actualiza el saldo en la tarjeta. Este último paso se puede hacer con algunas Apps en un teléfono con NFC o bien desde un *punto físico de recarga*.

En este trabajo se analiza cómo distribuir estos puntos físicos de recarga en las paradas de colectivo de una ciudad, el cual es un problema de tipo *set covering*. El objetivo es usar la menor cantidad de dispositivos brindando un buen servicio al usuario final. Como un análisis extra se suma un estudio para insertar redundancias (dispositivos extra) en zonas importantes o de interés, para eso se tuvo que definir ¿Qué es una zona de interés? y cómo identificarlas

Para este trabajo se consideraron la ciudad de Berlín y el AMBA. Sobre estas zonas se usaron las paradas de colectivo como input, luego se complementan con otras fuentes de información y se asignan los dispositivos según cuatro modelos diferentes. Sobre cada modelo se hace un análisis de los resultados obtenidos.

El primer modelo planteado es por fuerza bruta. Los restantes son de Programación Lineal Entera: uno un modelo estándar, en el siguiente se agregaron restricciones por puntos de interés cercanos y el último suma restricciones de redundancias.

A lo largo del trabajo se demostró las limitaciones del algoritmo por fuerza bruta y la escalabilidad de las soluciones con modelos de Programación Lineal Entera, pudiendo definir de manera precisa y objetiva donde colocar los dispositivos en las ciudades.

Palabras Clave: programación lineal entera – asignación – optimización – set covering problem – paradas de colectivo – recarga de saldo – zonas de interés

Abstract

Today, digital wallets are at an all-time high in many places around the world. One of these places is South America. The active users are growing at a very fast rate. Some examples of digital wallets are MercadoPago, Brubank, Nubank. Innovation in this sector is very high and every year new products/features are released. Wallets can have many rails or segments, a particular one is *single player* (cellphone recharge, charging the transport card, utility payments, etc.), which is very important for any wallet.

In Argentina, the main transport card is called SUBE, and in order to charge it from a digital wallet, one must first pay the amount inside the wallet and then, proceed to impact the new balance in the card. For this step you either need NFC on your phone or to go directly to a physical recharge point, the new balance is instantly charged in the card as soon as you pass the card through the machine.

In this dissertation, the analysis focus on where to put these physical recharge points. The objective is to use as few as possible, while maintaining a good service for the end user. As an extra challenge, a redundancy model was created, it inserts redundancies (extra devices) in important/strategic areas. For this it had to be defined, What is a strategic area? and How can it be identified?

The selected cities/areas were Berlin an AMBA. The bus stops are input to the algorithms, and they are complemented with extra information, and then processed by four different models. For each one, there´s an analysis of the end results.

The first model proposed is by brute force. The rest are of Integer Linear Programming: one is a standard model, another one adds restrictions for nearby points of interest and the last one adds redundancy restrictions.

Throughout the work, the limitations of the brute force algorithm and the scalability of the solutions with Integer Linear Programming models were demonstrated, being able to define precisely and objectively where to place the devices in the cities.

Palabras Clave: linear programing – LP – optimization – set covering problem – bus stops – balance recharge – transport – high density areas

Contenido

Capítulo 1: Introducción

1.1 Problema de negocio

La situación plantea un escenario de desembarco de una tarjeta con saldo recargable para pagar el medio de transporte (colectivos). Esta tarjeta plantea muchos beneficios tanto para el usuario final (pasajeros) como para las empresas de transporte. Teniendo esto en cuenta se desea brindar una buena experiencia al usuario, el mayor *punto de dolor* es la recarga de dicha tarjeta. Se pretende que los usuarios paguen el importe a recargar por alguna billetera virtual y luego tengan la posibilidad de impactar el nuevo saldo mediante los puntos de recarga. Con tan solo apoyar la tarjeta en el dispositivo, el saldo queda impactado sobre la misma.

Los dispositivos pueden localizarse solamente en las paradas. La idea es que un usuario no tenga que caminar más de cierta distancia X desde su parada habitual al dispositivo más cercano. Esto se traduce a que si una parada no cuenta con un dispositivo, hay una parada a una distancia menor a X con un dispositivo.

Dado que los aparatos tienen un costo elevado, se pretende minimizar la cantidad a utilizar. Por otro lado, con dos paradas en igualdad de condiciones, se debe definir un criterio objetivo para priorizarlas, para esto se usaron los puntos de interés que más adelante se analizarán en detalle.

Por último, se analiza cómo identificar zonas donde conviene asignar dispositivos extra, similar a agregar redundancias (dado por la cantidad de pasajeros que van a pasar por esa zona). Para definir un criterio objetivo se analiza la densidad y dispersión de los puntos de interés. Con esto se pueden identificar dichas zonas y forzar redundancias sobre ellas.

El problema original de fondo se conoce como *set cover*, el cual es uno de los 21 problemas NP-completos de Karp.

1.2 Contenido del trabajo

El presente trabajo se estructura en cinco capítulos con el siguiente contenido.

En el capítulo uno se expone la introducción y el planteo formal del problema, su base teórica, se presentan los conceptos básicos a utilizar y se expone literatura relacionada.

En el capítulo dos se realiza un análisis exploratorio. Se investigan los datos de entrada, particularmente las paradas de colectivo y los puntos de interés para las dos áreas bajo análisis (AMBA y Berlín).

En el capítulo tres se descompone el problema, formulando sub problemas más pequeños e independientes entre sí, y con esto se considera al resultado final como la suma de las

soluciones de dichos sub problemas. También se realiza un *deep dive* sobre los puntos de interés para transformarlos en ¨zonas de interés¨.

En el capítulo cuatro se plantean las soluciones algorítmicas y sus resultados. Este capítulo se enfoca en explicar cada uno de los modelos utilizados. Cada vez que concluimos un algoritmo, nos preguntaremos cómo mejorarlo. Las opciones se enfocan principalmente en dos ejes: performance y negocio. Por el lado de performance queremos encontrar una solución escalable (algorítmicamente). Y, por el lado de negocio, nos preguntamos, ¿Cuál va a ser la siguiente pregunta del cliente? ¿Cómo va a querer mejorar el producto en cuanto a la experiencia del cliente?

- Fuerza bruta: Algoritmo de fácil implementación pero con tiempos de cómputo no escalables
- Programación lineal entera: Se plantea el mismo problema pero desde un enfoque de programación lineal entera. Con esto se obtienen soluciones muy eficientemente y de manera escalable. La siguiente pregunta que nos haremos, es cómo complementar la información de las paradas. Después de investigar sobre la materia, se concluyó en complementar la información con los puntos de interés cercanos. Por un lado cada parada contará con un indicador del número de puntos de interés a su alrededor.
- Puntos de interés: Lo más importante que se encontró es ¿Cómo priorizar paradas cuando tengo ambigüedad? Esto se refiere a que da lo mismo poner el dispositivo en la parada X que en la parada Y. Se debería definir un criterio concreto, objetivo y cuantificable para poder sumarlo en el algoritmo. Para esto, se pensó en ponderar los puntos de interés cercanos (hospitales, museos, restaurantes, oficinas, etc.) a una parada para poder darles una prioridad a cada una de ellas. Esto nos permite priorizar soluciones que no agreguen dispositivos pero que escojan combinaciones con paradas ¨más importantes¨ (considerando los puntos de interés cercanos).
- Redundancias: Después de analizar los resultados del modelo anterior en detalle con el cliente, muy probablemente se llegue a la conclusión de que hay zonas estratégicas en la ciudad donde convenga asignar más dispositivos, aunque esas paradas ya estén cubiertas (según el criterio planteado al comienzo). Hay zonas en la ciudad que son muy concurridas y desde un punto de vista del usuario final, agregar dispositivos extra mejora mucho su experiencia del producto. Para esto se tienen que definir estas ¨zonas importantes¨ o ¨zonas de interés¨ y plantear un modelo que agregue redundancias ahí.

En el capítulo 5 se plantea un análisis de los resultados obtenidos con cada modelo mientras se modifica gradualmente el umbral X (distancia máxima entre una parada sin dispositivo a la parada con dispositivo más cercana). Se expone la performance de los algoritmos y se enuncian limitaciones sobre ciertos casos.

Por último, en el capítulo 6 se desarrolla la conclusión, planteando un resumen del trabajo, los resultados obtenidos y se exponen posibles upgrades a implementar.

1.3 Grafos

1.3.1 ¿Qué es un Grafo?

Para resolver el problema planteado, en este trabajo se usa el concepto de grafo y su respectiva estructura de datos para procesamiento.

Un grafo se puede definir como un conjunto de vértices (o nodos) y aristas. Las aristas representan las conexiones entre los vértices.

Supongamos que contamos con un grafo (llamado G) de la Figura 1.

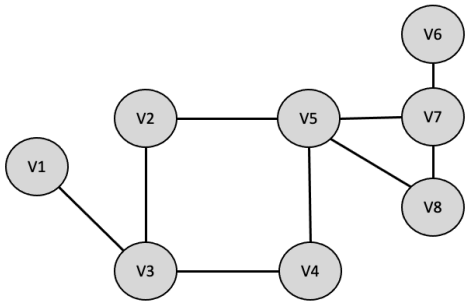

Figura 1 - Ejemplo genérico de un grafo

El grafo se define de la siguiente manera:

$$G = (V, A)$$

Donde V es el conjunto de vértices o nodos, y A es el conjunto de aristas.

Los vértices están definidos como:

$$V = \{V_1 , V_2 , V_3 , V_4 , V_5 , V_6 , V_7 , V_8\}$$

El grado de un vértice representa cuántas aristas inciden en el vértice. En el caso de grafos dirigido se puede definir el grado de entrada y el grado de salida de un vértice. En este trabajo, al solamente manejar grafos no dirigidos, es sencillamente las aristas q incluyen al vértice.

En la ejemplo de la Figura 1 el vértice V4 es de grado dos y el vértice V5 es de grado cuatro.

Las aristas se definen como:

$$A = \{\{V_1 , V_3\} , \{V_3 , V_2\} , \{V_2 , V_5\} , \{V_5 , V_4\} , \{V_3 , V_4\} , \{V_5 , V_8\} , \{V_5 , V_7\} , \{V_7 , V_8\} , \{V_7 , V_6\}\}$$

1.3.2 Grafos dirigidos vs no dirigidos

Una clasificación o característica que es importante mencionar para entender este trabajo es que existen grafos “dirigidos” y grafos “no dirigidos”

En los grafos dirigidos cada arista tiene una “dirección”

$$Arista_1 = (Vertice_4 , Vertice_8)$$

Implicando una conexión del *vértice 4* al *vértice 8*. Pero no implica que haya una conexión del *vértice 8* al *vértice 4*.

En los grafos “no dirigidos” las aristas no tienen dirección, por lo tanto la arista mencionada previamente indica una conexión tanto del *vértice 4* al *vértice 8*, como del *vértice 8* al *vértice 4*. El ejemplo de la sección anterior es un grafo no dirigido.

1.3.3 Matriz de adyacencia

Es una forma de representar las relaciones entre los vértices. En nuestro caso las aristas que tiene el grafo.

$$A'_{ij} = \begin{cases} 1 & Si\ hay\ arista\ de\ V_i\ a\ V_j \\ 0 & En\ otro\ cso \end{cases}$$

Si recordamos el ejemplo mencionado las aristas son:

$$A = \{\{V_1 , V_3\} , \{V_3 , V_2\} , \{V_2 , V_5\} , \{V_5 , V_4\} , \{V_3 , V_4\} , \{V_5 , V_8\} , \{V_5 , V_7\} , \{V_7 , V_8\} , \{V_7 , V_6\}\}$$

Lo cual se traduce a la siguiente matriz:

$$A' = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Es importante notar que para grafos ¨no dirigidos¨ la matriz es simétrica. Otro dato es que también se puede usar la matriz para embeber el costo de ¨utilizar¨ la aristas (donde la arista no existe se puede asignar un costo infinito).

1.3.4 Estructuras de datos utilizadas

En el presente trabajo se utilizaron varias estructuras de datos para trabajar con la información, lo importante a mencionar es que dichas estructuras son derivadas de los conceptos vistos en este capítulo: matriz de adyacencia, lista de aristas, vértices, etc.

En resumen, se creó una clase *Grafo,* encapsulando las estructuras y proveyendo sus propias funciones de optimización (según los distintos métodos que se analizan en el presente trabajo).

1.4 Formalización del problema

1.4.1 Set Covering Problem

Como se mencionó anteriormente, una parada puede tener o no tener un dispositivo, y en caso de que no tenga, una de sus paradas *adyacentes* debe tener un dispositivo.
Definimos:

$$P_i: parada\ i$$

Se define como paradas *adyacentes* a P_i a toda parada P_j que esté a una distancia menor que el umbral máximo *X* de distancia.

Ahora supongamos que tenemos la siguiente información:

$$C_i: conjunto\ de\ paradas\ cubiertas\ cuando\ P_i\ tiene\ un\ dispositivo$$

Es decir que C_i contiene a P_i y todas sus paradas adyacentes. También se define:

$$C : conjunto\ de\ paradas\ de\ colectivo$$

Considerando:

$$n : cantidad\ de\ paradas\ de\ colectivo$$

Y definimos la variable de costo, la cual en este trabajo es constante dado que el costo del dispositivo es siempre el mismo:

$$costo(P_i) :\ el\ costo\ de\ instalar\ un\ dispositivo\ en\ la\ parada\ i$$

Por lo tanto, se puede decir que tenemos que encontrar la mínima cantidad de conjuntos utilizados, de forma tal que la unión de dichos conjuntos, sea el universo completo de paradas.

Este problema se lo conoce bajo varios nombres:

- Problema del conjunto de cobertura

- Dominating Set
- Set Cover Problem

En el problema formal cada conjunto tiene un costo asociado y se minimiza el costo total. En el caso planteado en este trabajo todos los conjuntos tienen como costo un dispositivo, por lo tanto cada conjunto tiene el mismo costo.

1.4.2 Literatura relacionada

Sobre el Set Covering Problem hay mucha literatura, se comenzó a escribir a mediados del siglo pasado, y a partir de la década del 70 hubo un mayor impulso tras la demostración de que efectivamente es un problema NP-completo (Karp,1972). Se pueden encontrar varias definiciones formales del problema a lo largo de la literatura.

El problema se puede plantear de la siguiente manera:

$$minimizar \sum_{i \in \{1,\dots,n\}} costo(P_i) * x_i$$

$$cumpliendo \bigcup_{\forall i \in \{1,\dots,n\}:\, x_i = 1} C_i = C$$

$$considerando\ x_i \in \{0,1\}\ \forall\ \mathrm{i} = 1, \dots, \mathrm{n}$$

Se debe tener en cuenta que este problema es equivalente al de Conjunto Dominante para grafos, en el cual tenemos que elegir la menor cantidad de vértices cumpliendo que todo vértice sea seleccionado o bien sea un vecino de un vértice seleccionado.

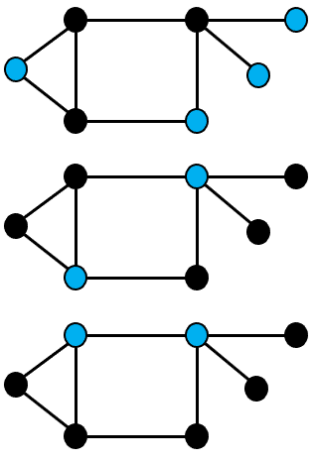

Figura 2 - Ejemplos de grafos y conjuntos dominantes (azul)

Si analizamos los tres ejemplos planteados en la Figura 2 , considerando un costo constante, se puede observar que el conjunto azul del primer caso es dominante pero no es una solución óptima mientras que en los otros dos casos el conjuntos azul sí lo es.

A lo largo de los años se escribieron una gran cantidad de *papers*, libros y tesis sobre estos problemas. Los mismos tratan el problema desde diversos enfoques, ya sea desde soluciones teóricas, métodos de resolución, sus soluciones algorítmicas y aproximaciones computacionales. Es importante notar que para grafos de millones de vertices (o más) las soluciones exactas pueden resultar computacionalmente imposibles de procesar con computadoras modernas, es ahí donde se puede recurrir a heurísticas.

En los últimos 40 años se publicó una gran cantidad de trabajos enfocados en optimizaciones y aproximaciones algorítmicas exponiendo pros y contras de cada modelo. Se pueden observar trabajos en problemas de combinatoria de base binaria y propuestas sobre cómo resolverlos (Crawford, Soto, Mella, Elortegui, Palma, Torres-Rojas, Vasconcellos-Gaete, Becerra, Peña & Misra, 2021), trabajos con sets generados al azar en donde se explora la complejidad algorítmica del método utilizado (Grandoni, Gupta, Leonardi & Miettinen, 2008), y trabajos donde se mezclan dos o más métodos de resolución para formar uno mejor (Wang, Pan, Al-Shihabi, Zhou, Yang & Yin, 2021).

Por último, es importante destacar que este problema y sus variantes tienen aplicaciones diversas como:

- Asignación de turnos laborales (ej. en una aerolínea): Considerando que tenemos un grupo de personas, con distintas tareas (pilotos, tripulantes de cabina, etc.) se desea encontrar una combinación óptima para cumplir con las operaciones, manteniéndose en el marco de las regulaciones vigentes (cantidad de personal mínimo en un turno, cantidad de horas de descanso mínimo entre vuelos, etc.). El problema de una aerolínea es un ejemplo clásico de set cover, con múltiples papers escritos sobre el tema (KasirZadeh, Saddoune & Soumis, 2017)
- Ubicación de antenas de telefonía en una ciudad: Análisis Similar al elaborado en este trabajo, donde hay posibles lugares para colocar antenas y se tiene que dar cierta cobertura en una zona.
- *Random testing* de programas: Al testear un programa se intenta testear el comportamiento del algoritmo en cada rama o bifurcación del código (i.e: en una sentencia IF-ELSE se tiene que testear las 2 salidas). Considerando que un test ¨cubre¨ un determinado conjunto de pasos o ¨bifurcaciones¨, se busca ejecutar la menor cantidad de tests posibles garantizando que toda bifurcación haya sido ¨visitada¨ por algún test.
- Encontrar virus en computadoras: Sobre este tema hay poca literatura disponible, posiblemente por temas de propiedad intelectual. En resumen, es un mix entre el problema conocido como *Regex Golf* y el *Set Covering Problem*. El primero no se estudia en este trabajo, pero en resumen es, dado dos oraciones o strings, encontrar la mínima expresión regular que devuelva el primer string pero no el segundo. De esta manera se pueden mapear sets de strings que son más frecuentes en programas maliciosos que en programas convencionales. Con estos conjuntos se puede armar una heurística basada en el *Set Covering Problem* para predecir si un archivo es malicioso o no.

Capítulo 2: Análisis exploratorio

2.1 Paradas de colectivo

En el presente trabajo se utilizó información de dos ciudades/zonas: Berlín y AMBA. Analizaremos cada una en detalle para tener una idea general de cuál fue el input del problema en cuestión. En el caso de AMBA se obtuvieron 19.006 paradas (ver Anexo 1) con distribución visible en la Figura 3.

Figura 3 - Distribución de paradas en AMBA

En caso de hacer zoom en un área particular, como en la Figura 4, se puede ver que hay muchas paradas a una distancia muy chica. Esto resultó un tema muy importante al definir el umbral *X* que identifica el radio teórico de cobertura de un dispositivo.

Figura 4 - Distribución de paradas en Núñez/Saavedra

En Berlín se obtuvieron 13.111 paradas con la distribución visualizada en la Figura 5.

Figura 5 - Distribución de paradas en Berlín

Sobre cada parada de colectivo se utilizó únicamente su *geolocation* (latitud y longitud) como input para el modelo.

A lo largo del trabajo, analizando posibles variantes surgió la idea de asignar prioridad a una parada sobre la otra (para el caso de empate en la asignación de un dispositivo). Tras investigar distintas maneras posibles, surgió la idea de usar los *puntos de interés* de la ciudad. Mientras más puntos de interés tuviera una parada en sus cercanías, mayor prioridad se le asignaría. El ejemplo más simple es un caso hipotético en el que tenemos solamente dos paradas en una ciudad y con colocar un dispositivo en cualquiera de ellas, la otra ya se encuentra cubierta. La idea es que si contamos la cantidad de hospitales, oficinas, cafés, etc. que tiene cada una a su alrededor, asignemos el dispositivo a la parada que tenga mayor impacto para el usuario final (más puntos de interés cerca).

2.2 Puntos de interés

Se define, en este trabajo, como un punto de interés a toda ubicación que este contenga un tag de:

- Punto turístico: museo, galería, centro de información, hotel o mirador
- Oficina: gobierno o privada
- Gastronomía: restaurante, bar, café, pub,
- Otros: centro religioso, farmacia, ATM, hospital

La idea es que en lugares donde haya mucha concentración de estos puntos también haya mucha concentración de gente.

En el Anexo 1 del presente trabajo se expone cómo se obtuvo la información de puntos de interés. En resumen, hay 8.327 puntos de interés en AMBA con la distribución de la Figura 6, donde se puede observar concentraciones de puntos en la zona céntrica de la Ciudad de Buenos Aires. También resaltan de manera correcta las avenidas más importantes de la ciudad, como la Avenida Cabildo.

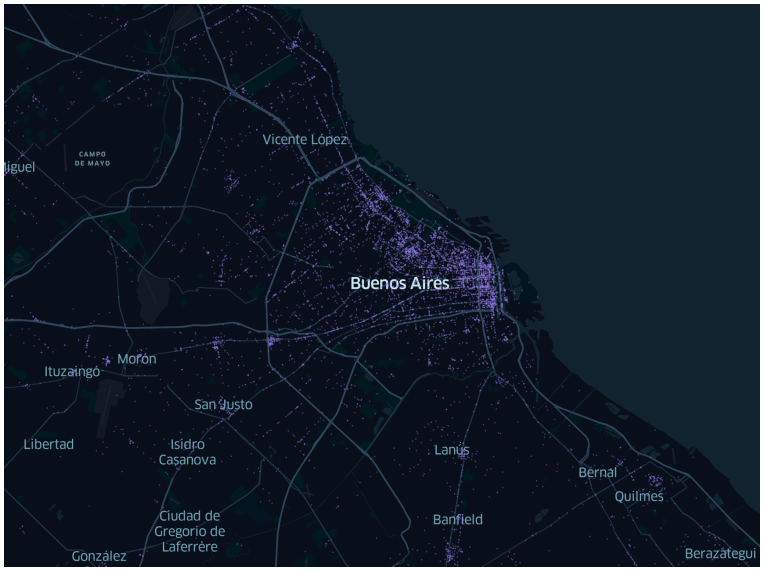

Figura 6 - Distribución de puntos de interés en AMBA

Por otro lado, en Berlín hay 19.748 puntos de interés con la distribución de la Figura 7.

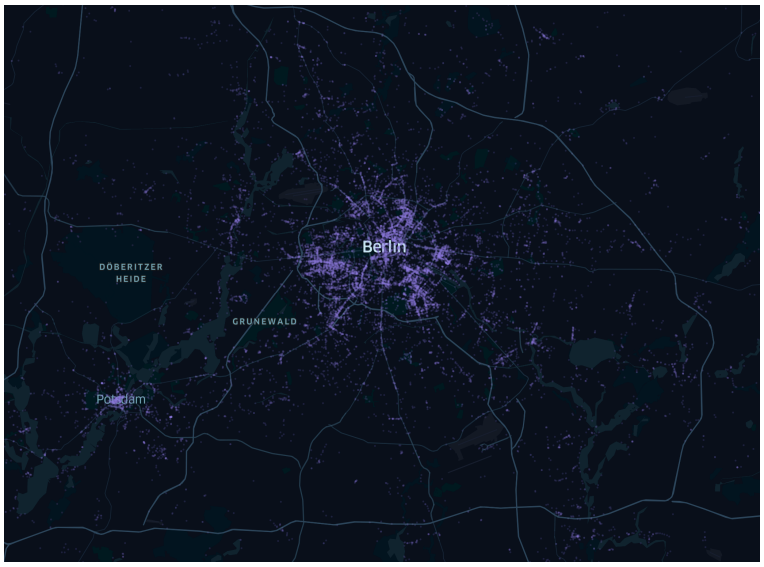

Figura 7 - Distribución de puntos de interés en Berlín

A simple vista ya se pueden ver en las imágenes anteriores grandes concentraciones de puntos en zonas particulares de la ciudad. Más adelante se retomará esta idea para definir el concepto de *zonas de interés*.

Un comentario adicional es que en ambas ciudades los parques y jardines tienen pocos puntos de interés. También en ambas ciudades hay zonas turísticas y de bares con una gran densidad de puntos de interés, ejemplos de esto son Palermo y Prenzlauer Berg.

Capítulo 3: Descomposición del problema

3.1 De geopoints a grafos conexos

Teniendo en cuenta que tenemos las coordenadas de las paradas se intenta usar una estrategia de dividir el problema en partes más pequeñas y simples de procesar. Cómo se puede dividir el problema? La clave es que una parada puede influir en un número finito de paradas "vecinas" o "adyacentes". Cómo identificamos estas paradas? Se calcula la distancia de cada parada contra el resto de la base de datos y se seleccionan aquellas adentro del umbral *X*.

Cálculo de distancias

Una forma muy precisa de calcular la mínima distancia a recorrer por una persona caminado de un punto a otro es usando las APIs de Google. Dichas APIs devuelven un resultado igual al de usar Google Maps como se muestra en la Figura 8.

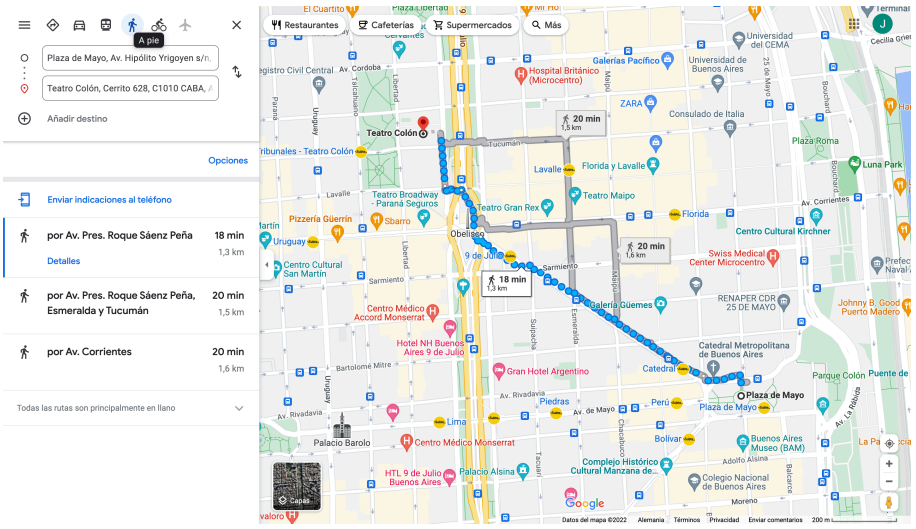

Figura 8 - Ejemplo de distancia entre dos puntos en modo caminata desde Google Maps

El resultado desde Google es más preciso que calcular mediante fórmulas debido a que tiene en cuenta las normas de tránsito en el trayecto.

Otra forma de calcular distancia es la usando la distancia Manhattan que, en espacios de dos dimensiones se define sobre dos puntos (P1 y P2) como:

$$Distancia\ Manhattan(P_1, P_2) = |x_{p1} - x_{p2}| + |y_{p1} - y_{p2}|$$

En este trabajo la distancia se aproximó con la distancia Geodésica[1]. Este método usa aproximaciones con senos y cosenos sobre un elipsoide. Dicha formula esta implementada en la librería GeoPy[2] y el input, al igual que en los ejemplos anteriores, son las coordenadas de dos puntos.

Una vez que obtuvimos para cada parada de colectivo sus respectivos vecinos, podemos definir un grafo con todos los puntos de la ciudad y encontrar las componentes conexas[3]. Cada componente es un problema aislado.

Analizando algunos ejemplos de componentes conexas resultantes se puede ver que:

- El caso más sencillo es cuando una parada no tiene vecinos, entonces está completamente aislada, y por definición deberá llevar un dispositivo
- Un grafo con solo dos paradas: Una de las dos llevará el dispositivo y en la otra no haría falta (a menos que forcemos una redundancia utilizando algún criterio extra)
- Un grafo con tres paradas: En este caso se elige uno de los vértices (parada) de grado 2, si todos son de grado 2 puedo elegir cualquiera, en caso de que solo haya un vértice de grado dos estamos forzados a asignar el dispositivo en esa parada
- Grafos de 4 o más paradas: las combinaciones son más complejas en estos casos. Más adelante se exponen ejemplos visuales de asignaciones para estos casos

Una vez que tenemos las componentes conexas (grafos individuales) se toma a cada uno como un problema separado. Esto se debe a que cada parada dentro de ese grafo no puede afectar a una parada fuera de ese grafo (mediante asignación de un dispositivo), mientras que sí puede afectar a las paradas dentro de ese grafo.

Por lo mencionado anteriormente, minimizar la cantidad de dispositivos utilizados en Berlín (o AMBA) se traduce a minimizar los dispositivos en cada componente conexa. Estas componentes dependen del umbral *X* establecido. Por lo cual se realizó un análisis de sensibilidad, buscando ver cómo impacta en las componentes conexas encontradas, la cantidad de dispositivos asignados y por último, en la performance del algoritmo.

En la siguiente tabla podemos ver cómo aumenta la cantidad de componentes conexas con más de 30 paradas según el umbral *X* para cada ciudad.

		Umbral X (m)							
		25	50	75	100	150	200	250	300
Componentes conexas con más de 30 vértices	**AMBA**	5	12	23	34	70	79	56	31
	Berlín	1	2	2	2	2	5	14	40

[1] Para más información ver: https://en.wikipedia.org/wiki/Geodesics_on_an_ellipsoid
[2] Para más información ver: https://geopy.readthedocs.io/en/stable/#module-geopy.distance
[3] Un grafo o sub grafo es conexo si y solo si para todo par de vértices existe un camino los conecte. Por otro lado, una componente conexa de un gráfico no dirigido es un subgrafo conexo que no forma parte de ningún subgrafo conexo mayor.

Figura 9 - Cantidad de componentes conexas con más de 30 vértices

Por otro lado, en la siguiente tabla podemos ver cómo aumenta el tamaño de la componente conexa más grande según el umbral *X* para cada ciudad.

		Umbral X (m)							
		25	50	75	100	150	200	250	300
Tamaño de la componente conexa más grande	**AMBA**	56	62	131	241	514	3.973	7.927	13.996
	Berlín	32	32	34	38	40	41	59	151

Figura 10 - Tamaño (cantidad de vértices) de la componente conexa más grande

Es importante notar que mientras más grandes sean las componentes conexas, más tiempo va a demorar en resolver (algorítmicamente).

Se seleccionaron algunas imágenes para distintos umbrales *X* con el fin de mostrar cómo el algoritmo separa las componentes conexas en distintos grafos.

En la Figura 11 se puede ver la distribución de componentes conexas cuando el umbral de X es 150m, vemos como el centro porteño tiene algunas componentes de varios km.

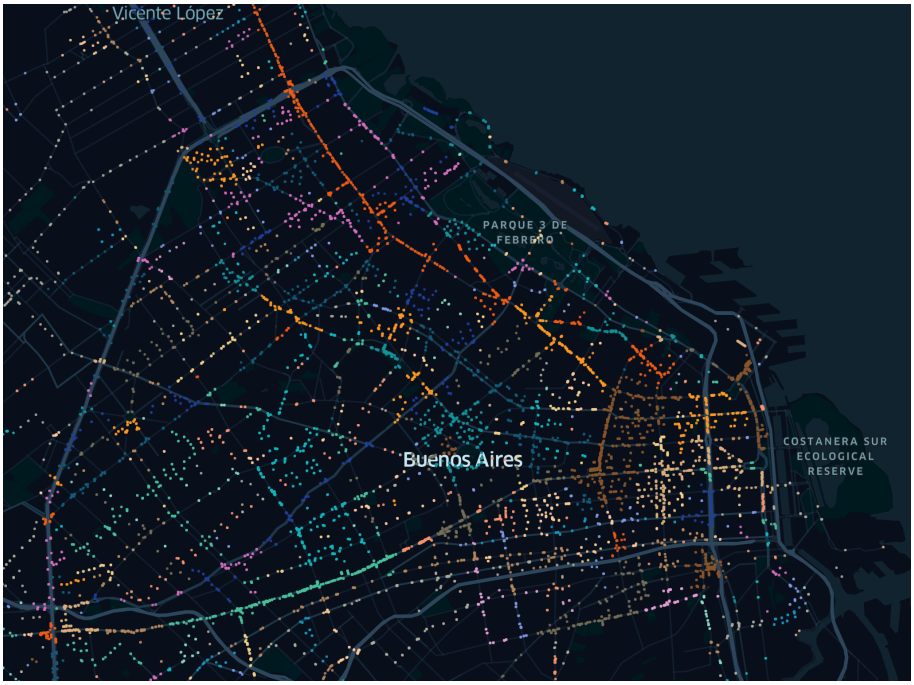

Figura 11 - Componentes conexas definidas por el Umbral de X de 150m

En la Figura 12 se puede ver la distribución de componentes conexas cuando el umbral de X es 200m, vemos como el centro porteño paso a ser mayoritariamente una gran componente conexa que se extiende por Avenida Rivadavia.

Figura 12 - Componentes conexas definidas por el Umbral de X de 200m

En la Figura 13 se puede ver la distribución de componentes conexas cuando el umbral de X es 250m, gran parte de la Capital Federal pertenece a la misma componente conexa.

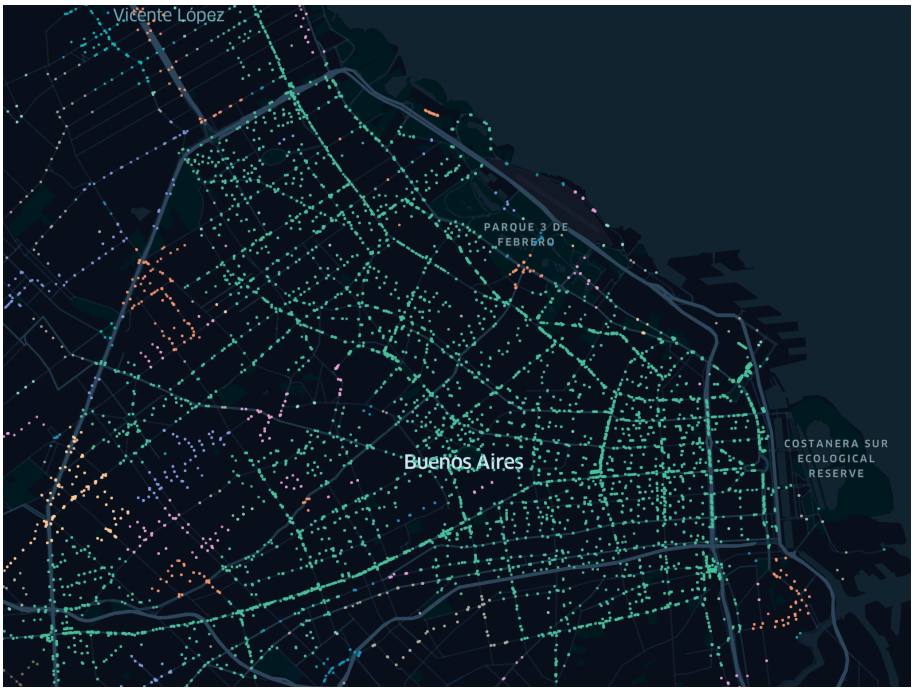

Figura 13 - Componentes conexas definidas por el Umbral de X de 250m

En las imágenes anteriores se expuso visualmente cómo, a medida que crece el umbral *X*, las componentes conexas se van combinando entre ellas, formando componentes conexas más grandes.

Con esta información, el problema se encuentra correctamente separado en sub-problemas de menor tamaño.

3.2 Zonas de interés

Más adelante en el trabajo, nos haremos la pregunta de cómo asignar redundancias, y cómo definir la cantidad. Para esto, se buscó encontrar áreas (o polígonos) donde hay gran concentración de personas. Se asumió que las grandes concentraciones de personas deben estar donde hay gran concentración de puntos de interés.

Por lo tanto, el objetivo es usar los puntos de interés mencionados en la sección anterior, para encontrar clusters de grandes concentraciones. Con cada cluster de puntos se tiene que definir un contorno que contenga a los puntos. Estos contornos definen las áreas/polígonos a las que designamos el nombre de zonas de interés.

Tomemos el caso de AMBA, los puntos de interés[4] tienen la distribución que se muestra en la Figura 14.

Figura 14 - Distribución de Puntos de interés en AMBA

Se buscó generar clusters donde hubiera una mayor concentración de puntos, eliminando los puntos que son considerados como *ruido* y a partir de ahí encontrar el polígono que define cada clúster.

Para encontrar las concentraciones de puntos se usó el algoritmo *DBSCAN*[5](Density Based Spatial Clustering of Applications with Noise). La especialidad del mismo es justamente

[4] Para más información ver Anexo 1
[5] Density Baser Spatial Clustering of Applications with Noise. Ver https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

encontrar muestras de alta densidad dentro de un set de datos. Se utilizó la implementación de la librería *sklearn*.

Lo parámetros de entrada sobre el algoritmo son:

- los puntos a clusterizar
- épsilon: hace referencia a la distancia mínima entre 2 puntos para ser considerados "vecinos" dentro del algoritmo
- min_samples: referencia a la mínima cantidad de puntos dentro de un cluster

El algoritmo define los clusters y todos los puntos que no están dentro de uno, son taggeados como ruido. Modificando los híper-parámetros se puede ser más restrictivo o permisivo en la definición de los clusters.

Después de aplicar el algoritmo *DBSCAN*, ya contamos con los puntos de cada clúster y necesitamos encontrar un polígono que represente su contorno, para esto se utilizó *alphashape*[6].

Un punto importante a notar es que los clusters resultantes de DBSCAN pueden tener cualquier forma, mientras que otros algoritmos (como K-Means) asumen una forma convexa.

En la primera iteración[7] se obtuvieron los resultados de las Figura 15 y Figura 16.

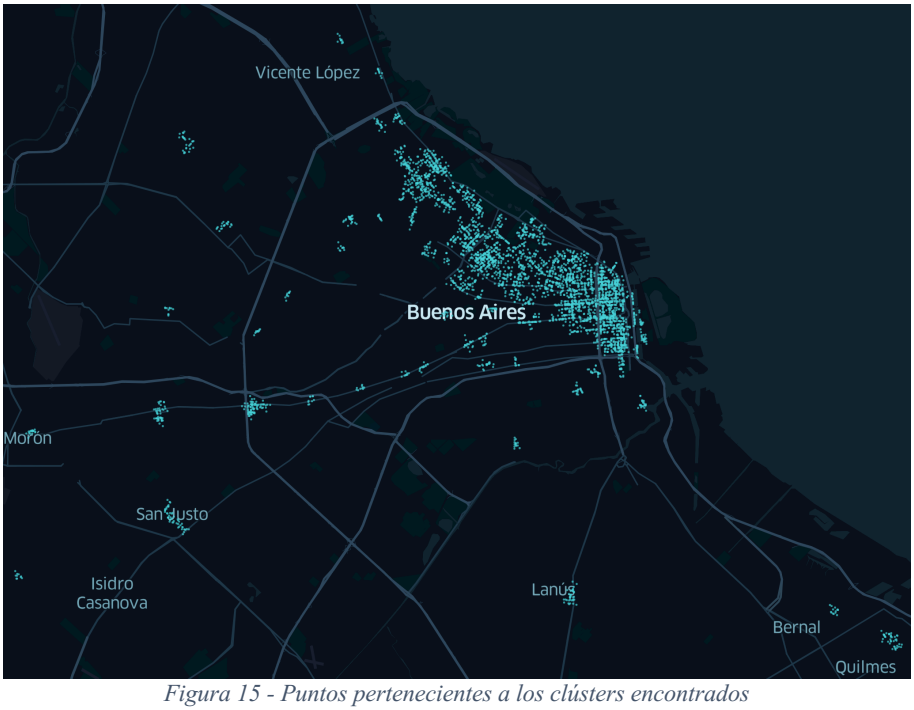

Figura 15 - Puntos pertenecientes a los clústers encontrados

[6] Ver https://pypi.org/project/alphashape/

[7] En la primera iteración de AMBA se configuró épsilon en 0.002 y min_samples en 10

Figura 16 - Puntos pertenecientes a los clústers encontrados. Color dado por el id del cluster

Quienes conocen Capital Federal pueden observar ciertas zonas familiares que ya fueron identificadas como importantes, algunos ejemplos son El Abasto, Once, Constitución y la Estación Lacroze (frente al Cementerio de la Chacarita)

Figura 17 - Polígonos en primer iteración

Como podemos ver en la Figura 17 hay varios clusters que representan un área demasiado grande (mayor a 0.5 km^2). Sobre estos clusters se ejecuta el algoritmo nuevamente para

encontrar una nueva clasificación de los puntos. Para esto necesitamos encontrar los nuevos *hiperparametros* (mediante *grid search*) en esta nueva iteración [8]del algoritmo.

El objetivo dentro de esta nueva iteración es sobre cada una de las zonas que son demasiado grandes, ejecutar el algoritmo nuevamente considerando híper-parámetros más restrictivos. Con esto se logra que algunos puntos que antes se consideraban parte de un mismo cluster se consideren ahora como ruido y "subdividan" a la masa de puntos en varios clusters.

Un ejemplo de esto puede ser que a nivel AMBA las restricciones para encontrar las zonas de concentración en Tigre no son las mismas que en el centro porteño. En el último caso hay que ser más restrictivo.

Figura 18 - Polígonos en segunda iteración

Se definió hacer una tercera [9]y última iteración del algoritmo y los resultados se observan en la Figura 19.

[8] En la segunda iteración de AMBA se configuró épsilon en 0.001601 y min_samples en 10
[9] En la tercera iteración de AMBA se configuró épsilon en 0.001117 y min_samples en 11

Figura 19 - Polígonos en tercera iteración

Con esta información, en las paradas donde corresponda, se asignara el id de la zona de interés a la que pertenece. Esto ayuda más adelante a armar un esquema de redundancias. Esta zonas nos pueden ayudar a construir restricciones del estilo “sobre cada zona de interés asegurar que siempre haya más de N dispositivos¨.

En el caso de Berlín se siguió el mismo procedimiento[10], podemos observar los resultados en las Figura 20 y Figura 21.

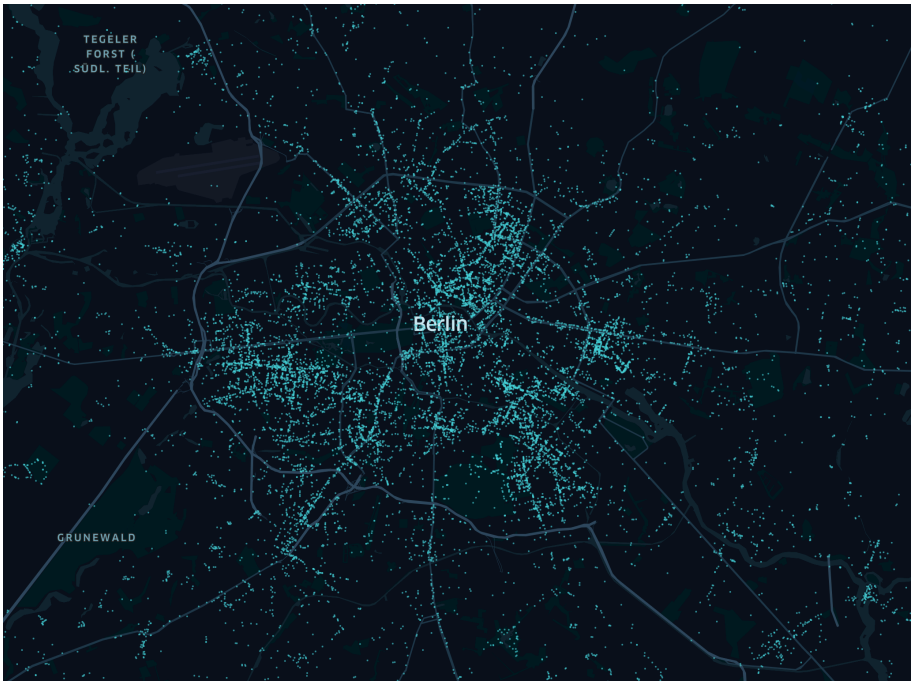

Figura 20 - Distribución de puntos de interés en Berlín

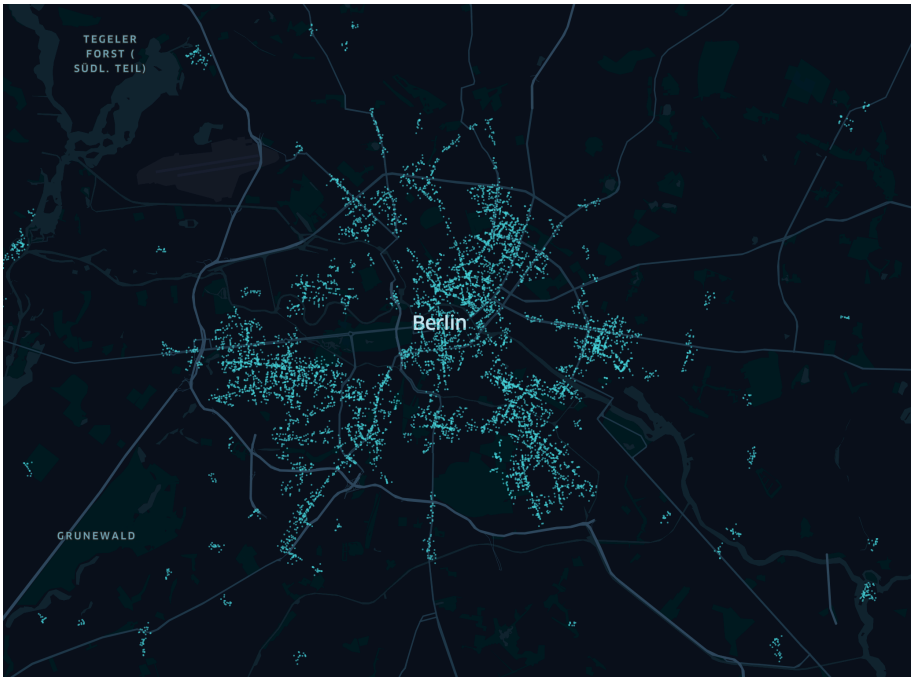

Figura 21 - Puntos en algún clúster en Berlín

[10] En el caso de Berlín los parámetros para la funión DBSCAN fueron:

- Primera iteración: épsilon en 0.002 y min_samples en 10
- Segunda iteración: épsilon en 0.001371 y min_samples en 10
- Tercera iteración: épsilon en 0.001174 y min_samples en 13

Los clusters definidos en la primera iteración se muestran a continuación. Ya se puede ver que hay ciertas zonas que son demasiado grandes y deberemos iterarlas.

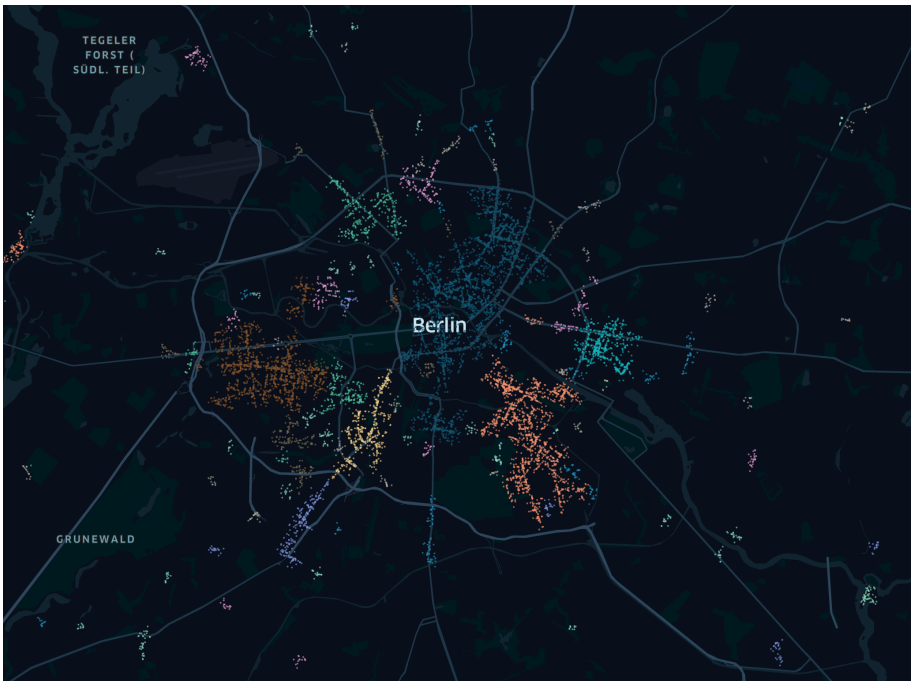

Figura 22 - Puntos en algún clúster coloreados por id del clúster

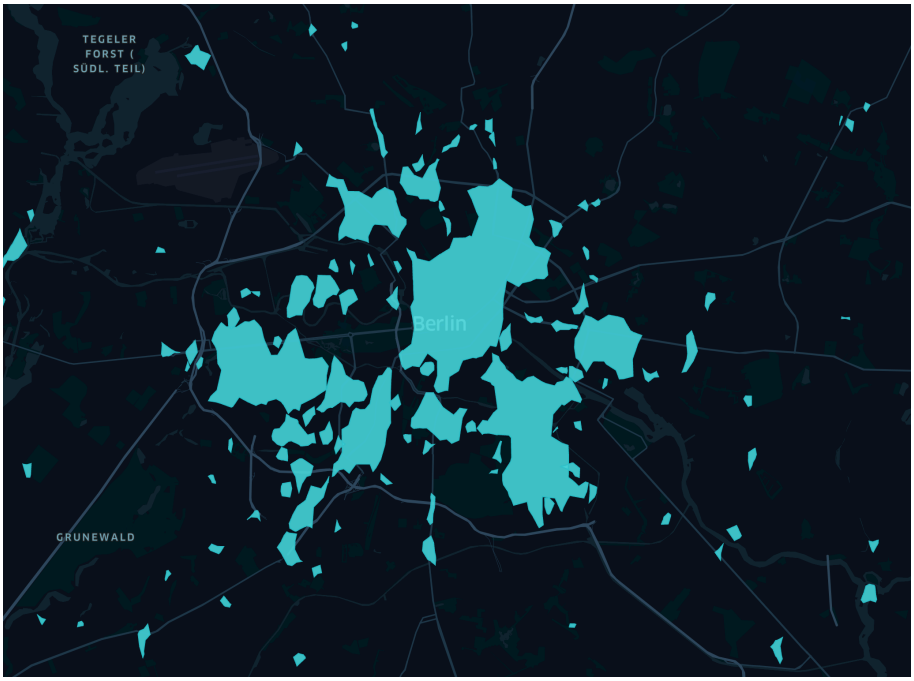

Figura 23 - Polígonos en primer iteración en Berlín

A continuación podemos ver la evolución de las zonas de interés hasta la tercera (y última) iteración, dando como resultado los polígonos necesarios para nuestro análisis.

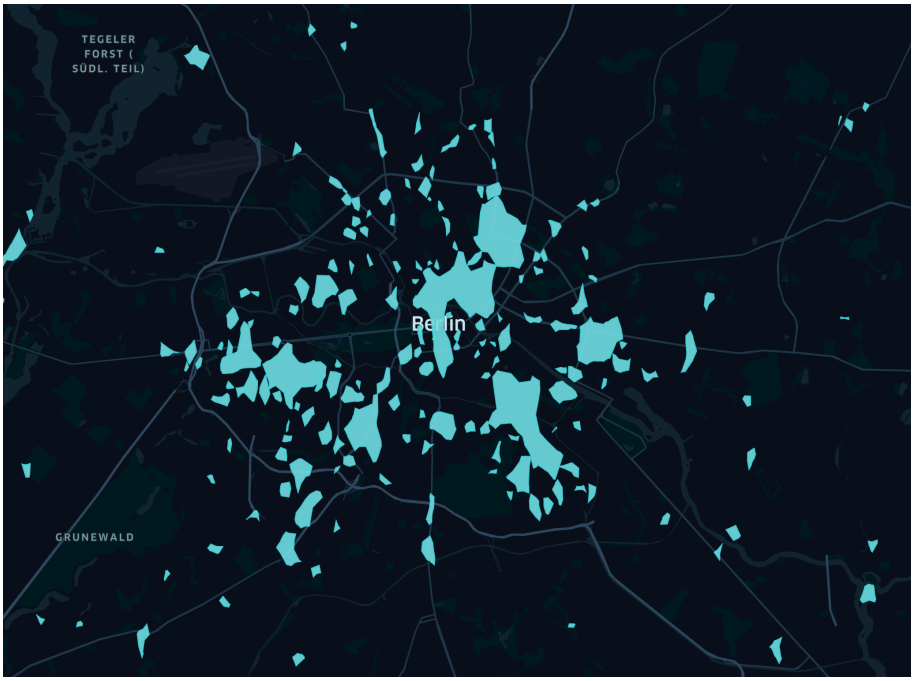

Figura 24 - Polígonos en segunda iteración en Berlín

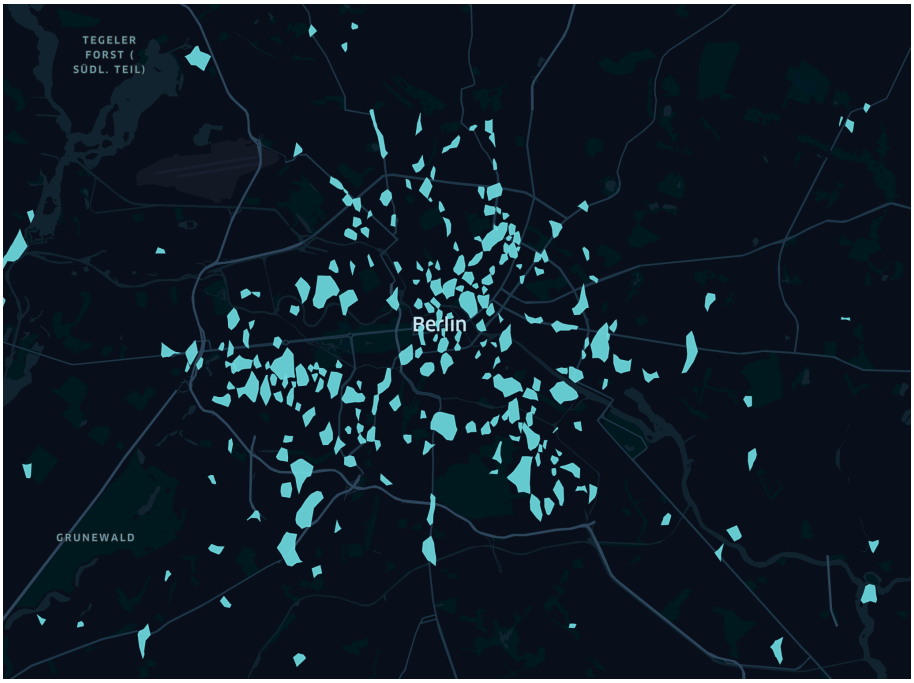

Figura 25 - Polígonos en tercera iteración en Berlín

Capítulo 4: Soluciones algorítmicas

4.1 Fuerza Bruta

Recordando que se intenta resolver el siguiente problema:

Considerando:

$$P : conjunto\ de\ paradas\ de\ colectivo$$

$$costo(P_i) :\ el\ costo\ de\ instalar\ un\ dispositivo\ en\ la\ parada\ i$$

Definimos la Matriz A:

$$A \in \mathbb{R}^{nxn}: matriz\ de\ adyacencia$$

$$A_{ij} = \begin{cases} 1 & si\ distancia(i,j) \leq X \\ 0 & si\ distancia(i,j) > X \end{cases} \quad ;\ \forall\ i,j \in \{1,\dots,n\}$$

Definimos el vector Y:

$$y \in \mathbb{R}^n;\ y_i \in \{0,1\}\ \forall\ \mathrm{i} = 1,\dots,\mathrm{n}$$

Donde:

$$y_i = \begin{cases} 1 & se\ coloca\ dispositivo\ en\ la\ parada\ i \\ 0 & no\ se\ coloca\ dispositivo\ en\ la\ parada\ i \end{cases}$$

Con el siguiente modelo de optimización:

$$minimizar \sum_{\forall\ i \in P} costo(P_i) * y_i$$

$$cumpliendo \sum_{\forall\ i \in P} A_{ki} * y_i\ \geq 1\ \forall\ \mathrm{k} = 1,\dots,\mathrm{n}$$

Con la información presentada anteriormente se planteó el primer algoritmo de resolución, utilizando lo que se conoce como fuerza bruta. La idea se basa en analizar cada componente conexa y probar todas las combinaciones posibles para encontrar el óptimo. En este algoritmo no se tuvo en cuenta la información de puntos o zonas de interés, solamente la información de las paradas de colectivo.

Es importante notar que esta no es una buena solución, siendo un problema NP-completo, una solución por fuerza es muy lenta. El algoritmo es fácil de implementar y es un buen punto de referencia para entender la complejidad del problema y usarlo para comparar contra el resto de los algoritmos implementados (en los casos donde sea posible).

El procedimiento es el siguiente, en un grafo conexo:

1. Se establece la variable temporal de *dispositivos a utilizar* igual a 1
2. Con dicha cantidad de dispositivos hay una cierta cantidad finita de combinaciones de asignación. Se definen todas estas combinaciones en una lista (considerando la cantidad de nodos en el grafo y la cantidad de dispositivos)
3. Se valida si hay alguna combinación en la lista que cumple todas las condiciones del problema.
    - En caso afirmativo, el problema terminó y se encontró el óptimo.
    - En caso negativo, se incrementa la variable de *dispositivos a utilizar* en uno y se vuelve al Step-2 del algoritmo

El algoritmo tiene varios puntos positivos, entre ellos se puede mencionar:

- Es muy simple: tanto la lógica como su programación son relativamente sencillas
- Se garantiza encontrar el óptimo. Dado que iteramos desde el mejor de los casos (un dispositivo) el primer caso afirmativo que encuentre el algoritmo es, al menos, uno de los óptimos. ¿Por qué uno de los óptimos? Si suponemos que el algoritmo devuelve una combinación con N dispositivos, el algoritmo garantiza que no existe un N' más chico que N, dado que ya iteró sobre ese valor. El único punto a destacar es que el algoritmo devuelve la primera combinación exitosa dentro del óptimo, o sea que puede ocurrir que haya una combinación distinta usando la misma cantidad de dispositivos y también cumpla todas las condiciones requeridas.

A continuación, se exponen diversos ejemplos para evidenciar, en forma visual, cómo el algoritmo asignó los dispositivos en cada componente conexa, empezando por la Figura 26.

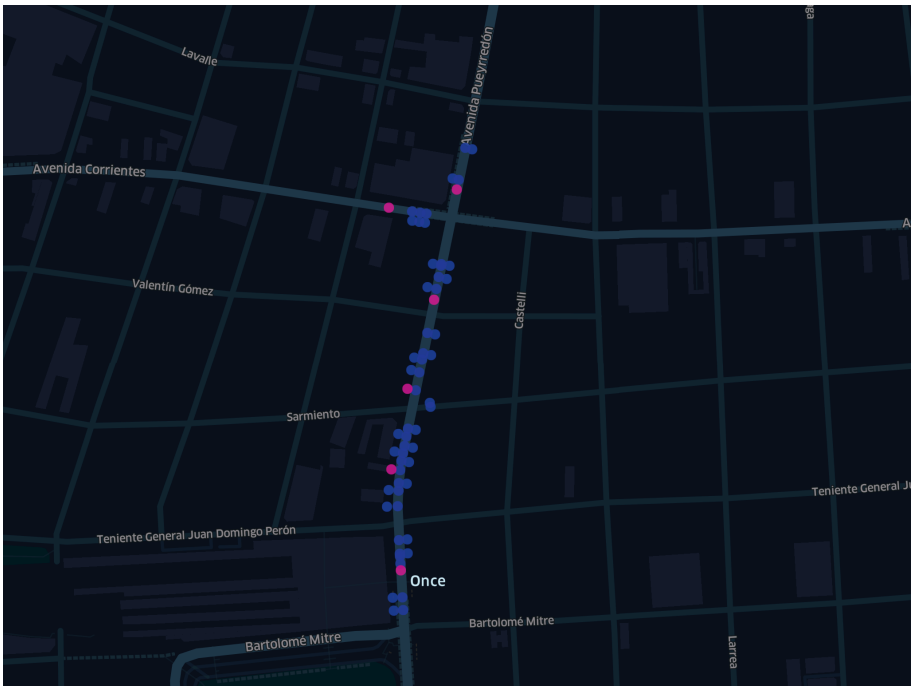

Figura 26 – Umbral X de 50m en una componente conexa de AMBA (Once)

En la Figura 26 vemos como una componente conexa que va mayoritariamente en una avenida, el algoritmo asigna un dispositivo por cuadra, de esa manera se asegura que todas las paradas cumplan el criterio. Este mismo criterio aplica a la Figura 27.

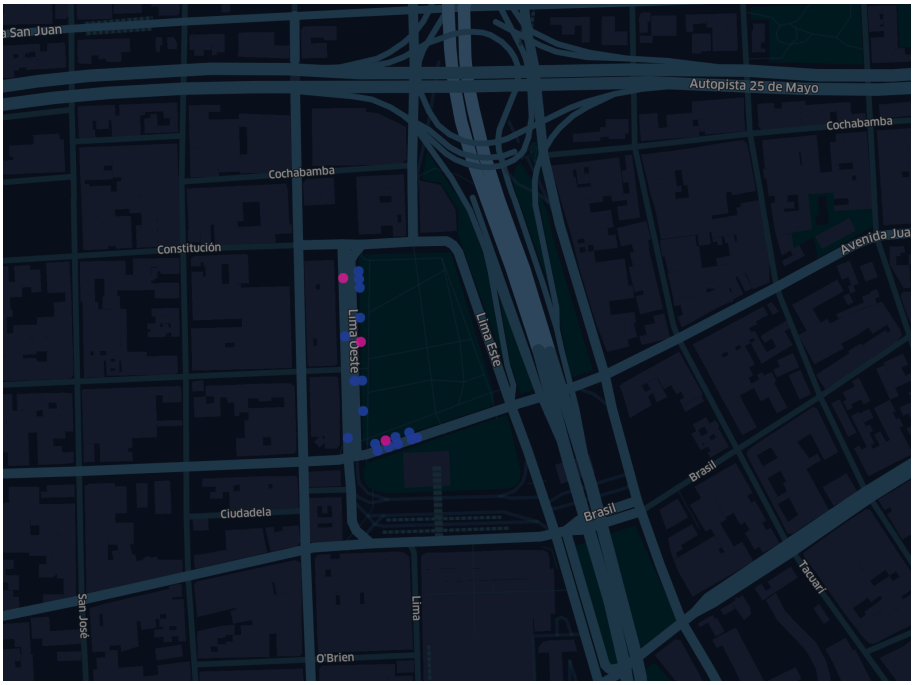

Figura 27 - Umbral X de 50m en una componente conexa de AMBA (Constitución)

En la Figura 28 se analiza un caso donde hay una densidad muy alta de paradas considerando los metros cuadrados bajo análisis.

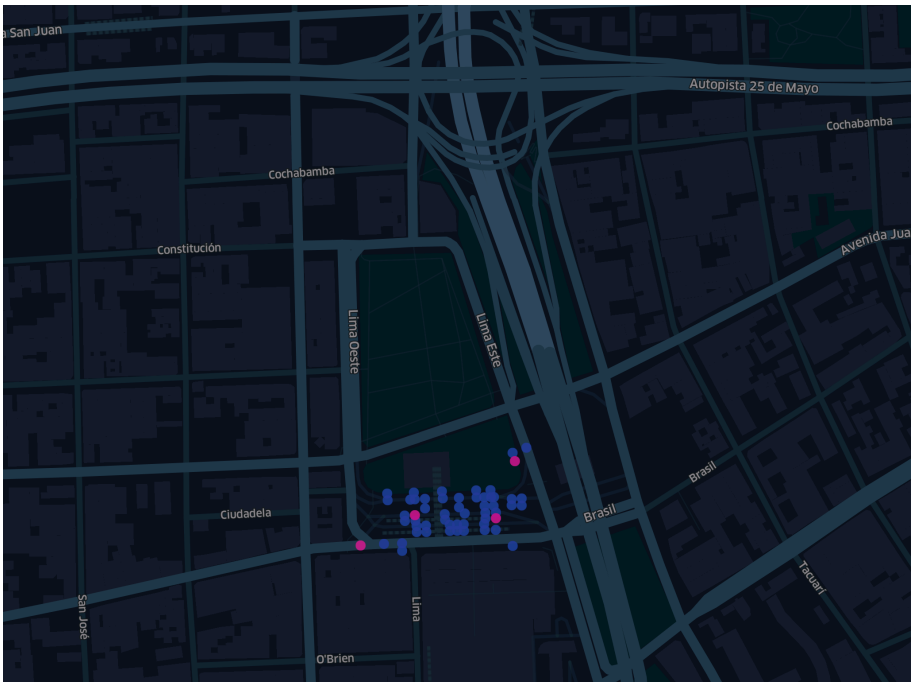

Figura 28 - Umbral X de 50m en una componente conexa de AMBA (Constitución)

En la Figura 29 es otro ejemplo de alta concentración de paradas.

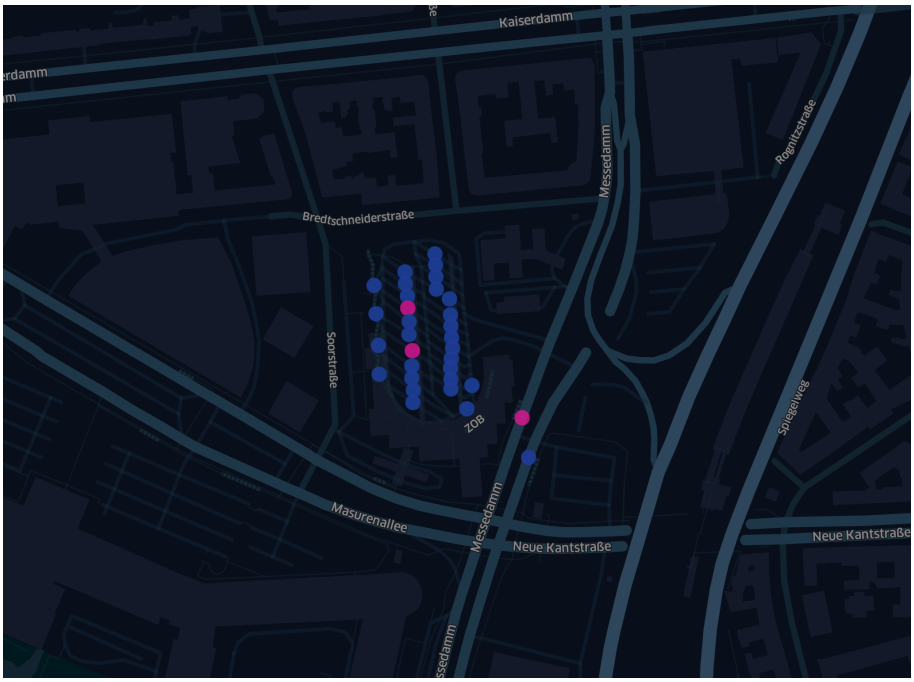

Figura 29 - Umbral X de 50m en una componente conexa de Berlín (Estación central de autobuses)

Con las imágenes expuestas se puede ver la asignación estratégica de cada dispositivo en la respectiva componente conexa.

Por otro lado, el algoritmo tiene un punto en contra que resulta crítico, no es eficiente en el uso de recursos. El tiempo de resolución incrementa exponencialmente al aumentar la cantidad de dispositivos a asignar (lo cual depende de la cantidad de vértices y sus grados).

		Dispositivos Asignados				
		3	4	5	6	7
Tamaño de la componente conexa	22		0,27			
	23	0,03				
	24	0,04	0,60			
	25	0,04				
	26	0,04	0,29	3,46		
	27	0,08				
	28	0,11	0,77	3,10		
	29		1,47	3,75		
	30		1,34			145,66
	31	0,13				
	32	0,44				243,61
	34	0,13	3,02	12,91	54,93	
	36		1,45			
	37		1,92		199,54	
	38	0,58			131,74	
	39	0,86				
	41		9,70	28,73		
	45			50,45		
	48					9.950,53
	50		8,93			
	55		12,32			
	56			190,83		
	58			192,90		
	62				7.740,53	

Figura 30 - Tiempo de resolución promedio (s) según los casos analizados

En casos de componentes conexas de más de 30 nodos (paradas de colectivo) y una distribución de las mismas tal que se necesiten más de 5 dispositivos, los tiempos del algoritmo pasan de unos pocos minutos a varias horas. Más adelante se deja en evidencia que este algoritmo no escala correctamente para el caso de negocio planteado y que su correcta ejecución considerando diferentes umbrales de X demoraría semanas/meses. Considerando esto, el algoritmo por fuerza bruta sólo se utilizó con umbrales bajos, los cuales generan unas pocas componentes conexas complejas y los tiempos de resolución terminaron estando acotados a pocos minutos.

Con lo expuesto en esta sección, se dejaron en evidencia los pros y contras del primer algoritmo de resolución. Se pudo observar gráficamente, en componentes conexas relativamente sencillas, cómo es que el modelo de asignación debe funcionar para usar eficientemente los dispositivos. Para resolver los problemas de este algoritmo se propuso utilizar otro procedimiento de resolución basado en Programación Lineal Entera.

4.2 Programación Lineal Entera – Modelo básico

El algoritmo de fuerza bruta evidenció que se necesita otra solución escalable. Con este fin se planteo un modelo de PLE (Programación Lineal Entera).

Recordando que *X* es el umbral de distancia máxima entre paradas. El modelo básico tiene las siguientes constantes y variables:

Matriz de adyacencia:

$$A_{ij} = \begin{cases} 1 & si\ distancia(i,j) \leq X \\ 0 & si\ distancia(i,j) > X \end{cases} \quad ;\ \forall\ i,j \in \{1,\dots,n\}$$

Variable binaria de asignación de dispositivos:

$$Y_i :\ Y_i \in \{0,1\}\ \forall\ i \in \{1,\dots,n\}$$

La restricción de que una toda parada tiene que tener un dispositivo, o una de sus vecinas, se escribe (en forma generalizada) de la siguiente forma:

$$\sum_{i\in\{1,\dots,n\}} A_{ij}\, Y_i \geq 1\ ;\ \forall\ j \in \{1,\dots,n\}$$

La función objetivo es:

$$\text{minimizar}\left(\sum_{i\in\{1,\dots,n\}} Y_i\right)$$

El modelo planteado es suficiente pero se buscó agregar algunas restricciones extras para optimizarlo.

Utilizando la función:

$$Grado(Vertice\ j)\text{: } numero\ de\ aristas\ incidentes\ al\ vertice\ j$$

Se agrego la siguiente cota:

$$\sum_{i\in\{1,\dots,n\}} Y_i \geq \frac{n}{1 + \max\limits_{j\in\{1,\dots,n\}} Grado(Vertice\ j)}$$

Esta nueva restricción implica un mínimo de dispositivos a utilizar y se basa en el hecho de que en el mejor de los casos, tenemos que utilizar los vértices de grado máximo y están distribuidos de forma tal que cubren de forma óptima el grafo. Un ejemplo básico de esto es el grafo de la Figura 31.

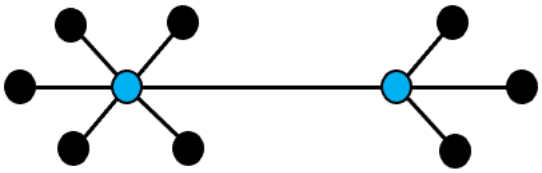

Figura 31 - Grafo de ejemplo para cota inferior

En este caso se cuentan con diez vértices, el grado máximo es cinco y según nuestra cota se deben utilizar por lo menos 1.66 dispositivos. Dado que es un número entero, se deben utilizar por lo menos 2 dispositivos en este grafo.

Es importante notar que el número de dispositivos a utilizar es igual a la cota inferior sólo en casos muy aislados como el ejemplo planteado anteriormente. Esta cota inferior toma más relevancia en casos de grafos grandes para los cuales el vértice con grado máximo es relativamente chico.

Con lo expuesto anteriormente el modelo de PLE se puede resumir en

$$\text{minimizar} \left(\sum_{i \in \{1,\dots,n\}} Y_i \right)$$

$$Y_i : \; Y_i \in \{0,1\} \; \forall \; i \in \{1, \dots, n\}$$

$$\sum_{i \in \{1,\dots,n\}} A_{ij} \, Y_i \geq 1 \; ; \; \forall \; j \in \{1, \dots, n\}$$

$$\sum_{i \in \{1,\dots,n\}} Y_i \geq \frac{n}{1 + \max\limits_{j \in \{1,\dots,n\}} Grado(Vertice \; j)}$$

Con este modelo se logró obtener los resultados para diversos umbrales *X* con el fin de compararlos. En primera instancia, compararemos los tiempos tomando las mismas componentes conexas que en la sección del modelo de Fuerza Bruta, los resultados obtenidos se pueden ver en la Figura 32.

Es importante mencionar que se utilizó una MacBook Pro (2017) con un procesador de 2,5 GHz Intel Core i7 (dos núcleos) y memoria de 16 GB. El *solver* utilizado fue Cplex.

		Dispositivos Asignados				
		3	4	5	6	7
Tamaño de componente conexa	22		0,005			
	23	0,004				
	24	0,005	0,006			
	25	0,005				
	26	0,005	0,007	0,009		
	27	0,004				
	28	0,007	0,006	0,014		
	29		0,008	0,020		
	30		0,008			0,315
	31	0,005				
	32	0,009				0,451
	34	0,008	0,010	0,034	0,167	
	36		0,013			
	37		0,021		0,272	
	38	0,008			0,320	
	39	0,008				
	41		0,021	0,126		
	45			0,178		
	48					67,176
	50		0,043			
	55		0,068			
	56			0,632		
	58			0,792		
	62				59,664	

Figura 32 - Tiempo de resolución promedio (s) para las mismas componentes conexas que en Fuerza Bruta

Cabe destacar que al aumentar *X* por encima de los 200m, las componentes conexas resultantes eran muy grandes, con varios miles de vértices, y sin embargo los tiempos de ejecución eran menores a 10 minutos. Se puede ver un resumen de esto en la Figura 33:

		Dispositivos Asignados							
		33	41	51	90	97	250	479	806
Tamaño de componente conexa	514		0,6						
	518	0,9							
	754			2,0					
	1085					2,9			
	1145				3,7				
	3973						38,3		
	7927							158,5	
	13996								527,3

Figura 33 - Tiempo de resolución promedio (s) en las componentes conexas más complejos

En el siguiente ejemplo, en la Figura 34, se expone la distribución de puntos sobre una componente conexa integrada mayoritariamente por puntos sobre una misma avenida.

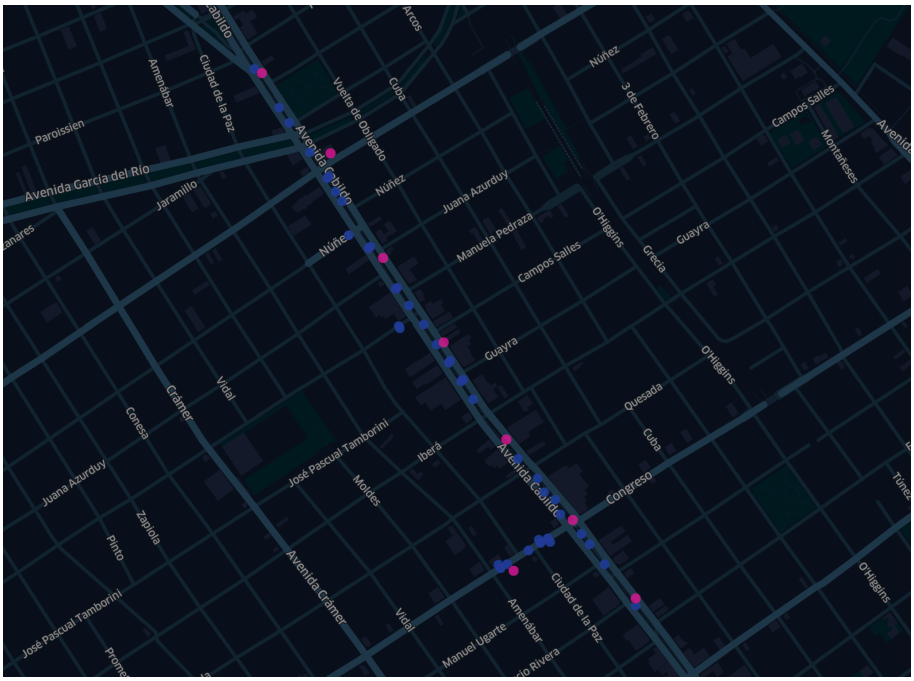

Figura 34 – Umbral X de 100m en una componente conexa de AMBA

Por otro lado, veremos en la Figura 35 un ejemplo de una componente conexa con una concentración de puntos (por km^2) muy alta. Vemos que la distribución de dispositivos es similar al ejemplo expuesto en la sección anterior de este trabajo.

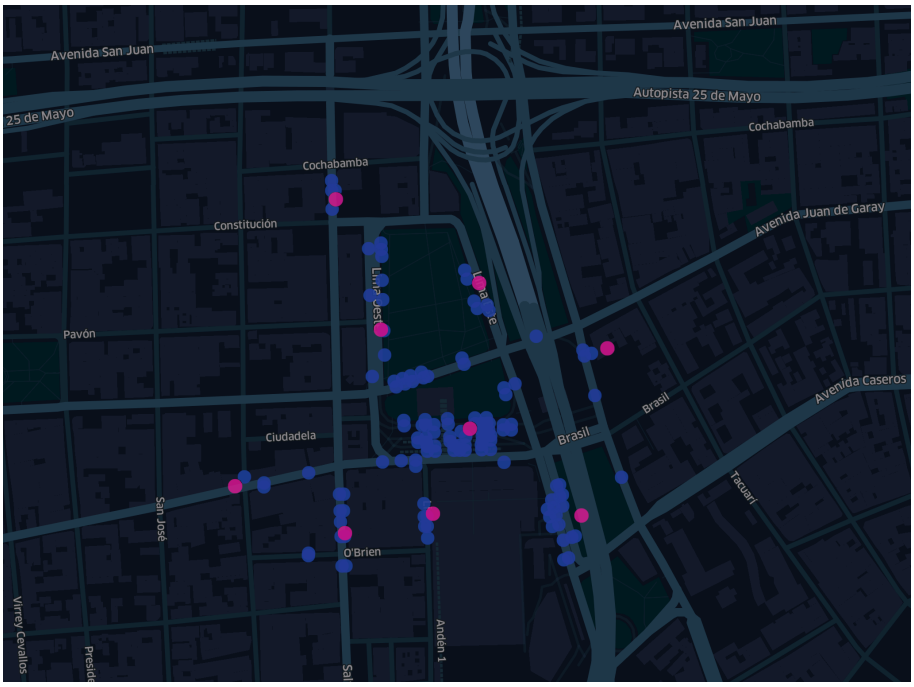

Figura 35 – Umbral X de 100m en una componente conexa de AMBA (Constitución)

Por último, vemos un ejemplo de una componente conexa que ocupa varios km^2. Vemos como los dispositivos colocados forman una especie de grilla de puntos.

Figura 36 - Umbral X de 150m en una componente conexa de AMBA

Con lo expuesto en esta sección, se dejó en evidencia lo escalable que es este modelo de resolución. El siguiente paso es agregarle información extra sobre zonas de interés a cada parada.

4.3 Programación Lineal Entera – Puntos de interés

En esta sección hablaremos de un nuevo modelo de PLE que modifica ligeramente la función objetivo, manteniendo las mismas restricciones pero buscando una forma en la que el modelo priorice alocar los dispositivos en paradas que tengan una mayor cantidad de *puntos de interés* cerca. Dado que ahora la función objetivo tiene dos niveles (dispositivos y *puntos de interés*) tendremos que replantearla de una manera acorde.

Definimos la cantidad de puntos de interés cercanos a una determinada parada:

$$PI_i \in N;\ \forall\, i \in \{1, \dots, n\}$$

Una constante de ponderación:

$$v = \frac{0.1}{\sum_{i \in \{1,\dots,n\}} PI_i}$$

La nueva función objetivo planteada es:

$$\text{minimizar} \left(\sum_{i \in \{1,\dots,n\}} (1 - v * PI_i) * Y_i \right)$$

Analicemos qué implica la constante que está multiplicando a cada variable *Y*. Se tiene en claro que esta acotado de la siguiente forma:

$$0.9 \leq (1 - v * PI_i) \leq 1$$

También se puede ver que:

$$n - 0.1 \leq \sum_{i \in \{1,\dots,n\}} (1 - v * PI_i) \ \leq n$$

La modificación implica "sacarle" unas pequeñas milésimas a las constantes multiplicadoras en la función objetivo, de forma tal que cada vértice contribuya unas milésimas menos al objetivo pero de forma proporcional a la cantidad de puntos de interés cercanos. Esto apunta a que saquemos provecho la función objetivo y que en caso de que sea posible, ante la ambigüedad entre dos paradas, elija la que tenga más puntos de interés cerca. Esto es (obviamente) una simplificación, dado que pueden haber rotaciones más complejas de varias paradas al mismo tiempo pero que en la suma, se beneficien más asignaciones sobre paradas con mayor cantidad de puntos de interés.

Con esto obtenemos que el modelo priorice de forma acorde al criterio establecido. Perdemos el valor entero en la función objetivo (los nuevos valores son números reales) pero recordamos que el valor que nos interesa es la suma de las variables Y.

Más abajo, se encuentran algunos ejemplos de componente conexas en los que se re priorizaron asignaciones con este algoritmo. Es importante destacar que fueron solamente reasignaciones, pero en ningún momento se asignaron más dispositivos que en el modelo anterior. Esto se debe a que elegimos la constante de ponderación *v* de forma tal que nunca lleguen a ser comparables los puntos de interés contra un dispositivo extra.

En la Figura 37 de una componente conexa en San Miguel (AMBA) se puede visualizar el cambio de distribución de dispositivos. Es una componente conexa de 184 vértices y 16 dispositivos. El puntaje[11] por puntos de interés aumentó en un 116% en este caso. Esto se logró sin agregar dispositivos, solamente se rotaron las asignaciones.

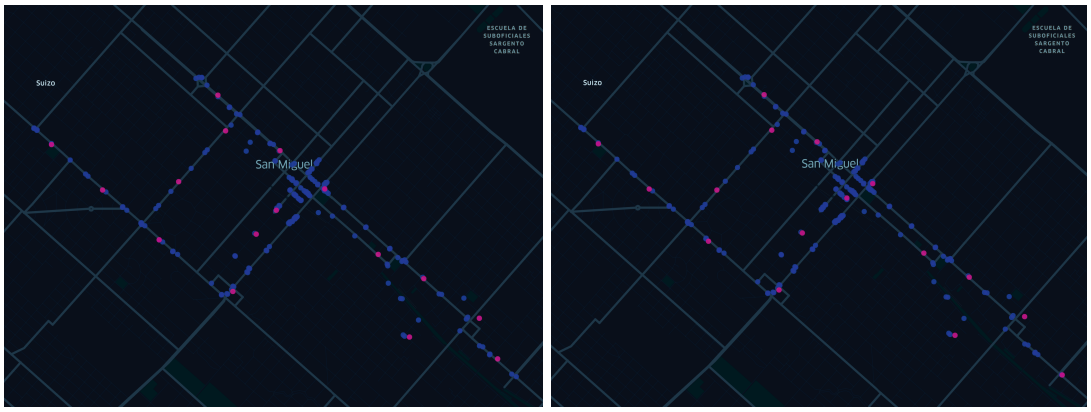

Figura 37 - Umbral X de 300m – PLE (izquierda) vs PLE PI (derecha)

A continuación, en la Figura 38, se evidencia un segundo ejemplo de una componente conexa con reasignaciones. Se cuenta con 191 vértices y se utilizaron 17 dispositivos. Con PLE el puntaje de puntos de interés obtenido es de 100, mientras que con PLE PI es de 121.

Figura 38 - Umbral X de 150m – PLE (izquierda) vs PLE PI (derecha)

[11] El puntaje por puntos de interés hace referencia a la sumatoria de los puntos de interés asociados a los vértices seleccionados para asignar un dispositivo

A continuación, en la Figura 39,se exponen más ejemplos de cómo el algoritmo cambia la priorización dentro de una componente conexa con el fin de obtener un mayor puntaje por puntos de interés impactados.

Información de la componente conexa				Dispositivos utilizados		Puntaje por Puntos de interés		Vértices con distinta configuración
Ciudad	Umbral X	ID de componente conexa	Vértices	PLE	PLE PI	PLE	PLE PI	
AMBA	150	6	251	22	22	41	54	30
AMBA	200	2	216	22	22	34	49	26
AMBA	150	248	191	17	17	100	121	24
AMBA	200	1234	170	19	19	5	9	24
Berlín	300	15	151	25	25	147	168	22
AMBA	250	46	243	27	27	10	24	20
AMBA	150	116	223	9	9	26	41	16
AMBA	150	625	105	9	9	21	30	14
AMBA	75	2598	112	10	10	33	37	12
AMBA	150	169	195	13	13	41	54	12
AMBA	250	1106	128	12	12	5	14	12
AMBA	300	218	184	16	16	6	13	12
Berlin	300	109	101	10	10	10	14	12

Figura 39 - Cambios de priorización – PLE vs PLE PI

Con esto hemos establecido una optimización sobre el modelo original de PLE. Es importante destacar que este modelo asigna la misma cantidad de dispositivos sobre las componentes conexas (en comparación contra el modelo original). Otro punto a tener en cuenta es que los tiempos de resolución no cambian significativamente, por lo que este modelo consume aproximadamente el mismo tiempo que el anterior en encontrar las soluciones. El único tiempo extra consumido es el de procesamiento de los puntos de interés (por única vez).

4.4 Programación Lineal Entera – Redundancias

En los algoritmos anteriores se realizó una asignación según las restricciones planteadas. Muy probablemente al analizar los resultados con el cliente se llegue a la conclusión de que hay zonas estratégicas en la ciudad donde convenga asignar más dispositivos, aunque esas paradas ya estén cubiertas (según el criterio planteado al comienzo). Hay zonas en la ciudad que son muy concurridas y desde un punto de vista del usuario final, agregar dispositivos extra mejora mucho su experiencia del producto.

La respuesta de cómo definimos estas *zonas de interés* se puede encontrar en la respectiva sección dentro de este trabajo.

Considerando que ya contamos con dichas *zonas de interés* el proceso que se definió para agregar estas *redundancias* fue el siguiente:

- En cada vértice se agrega la *zona de interés* a la que pertenece. Si pertenece a más de una se elige una sola de esta lista.
- En cada componente conexa se va a analizar a qué *zonas de interés* se está impactando y sobre cada una de ellas se analiza cuantos vértices del grafo pertenecen a la *zona de interés*
    - Si el conjunto anterior es de tres vértices o menos, todos llevan un dispositivo
    - Si el conjunto anterior es de cuatro vértices o más, se agrega una restricción de que se asignen al menos tres dispositivos en los vértices de la *zona de interés* bajo análisis

Lo enunciado se puede traducir en el modelo de PLE de la siguiente forma:

Zonas de interés involucradas en el grafo:

$$ZI_j \; ; \; \forall \, j \in \{1, \dots, m\}$$

Vértices del grafo en una determinada zona:

$$Vertices\big(ZI_j\big) \; ; \; \forall \, j \in \{1, \dots, m\}$$

Cantidad de vértices del grafo en una determinada zona:

$$Cantidad(Vertices\big(ZI_j\big)\,) ; \; \forall \, j \in \{1, \dots, m\}$$

Con la siguiente restricción:

$$\sum_{i \, \in \, ZI_j} Y_i \geq \text{minimo}\big(3, Cantidad(Vertices\big(ZI_j\big)\,)\big) \; ; \; \forall \, j \in \{1, \dots, m\}$$

Sumando la restricción para cada zona de interés involucrada en la componente conexa, se puede ver que no solo se harán reasignaciones de dispositivos para priorizar las paradas que

estén dentro de dichas zonas, sino que también se agregaran redundancias que antes tal vez eran innecesarias.

En la Figura 40 se puede ver cómo aumenta la cantidad total de dispositivos utilizados en una componente conexa de ejemplo en AMBA y también de forma gráfica cómo variaron las asignaciones.

Información de la componente conexa				Dispositivos utilizados		Extras por redundancias
Ciudad	Umbral X	ID del grafo	Vértices	PLE	PLE Redundancias	
AMBA	300	1	13996	806	917	111
AMBA	250	1	7927	479	566	87
AMBA	200	6	3973	250	304	54
AMBA	250	25	1145	90	91	1
AMBA	200	5	1085	97	111	14
AMBA	200	137	518	33	37	4
AMBA	150	128	514	41	45	4
AMBA	300	58	408	38	41	3
AMBA	150	272	388	33	41	8
AMBA	200	1	367	29	33	4
AMBA	150	2598	336	19	25	6
AMBA	150	50	323	29	36	7
AMBA	150	6	251	22	26	4
AMBA	250	46	243	27	28	1
AMBA	150	116	223	9	11	2
AMBA	250	219	218	22	24	2
AMBA	250	9	204	18	20	2

Figura 40 - Variación de asignaciones

Se toma una componente conexa de la Figura 40 a modo de ejemplo y se comparan los resultados de forma gráfica en la Figura 41. Se puede ver como al agregar las zonas se agregan dispositivos de más, muchas veces en paradas adyacentes.

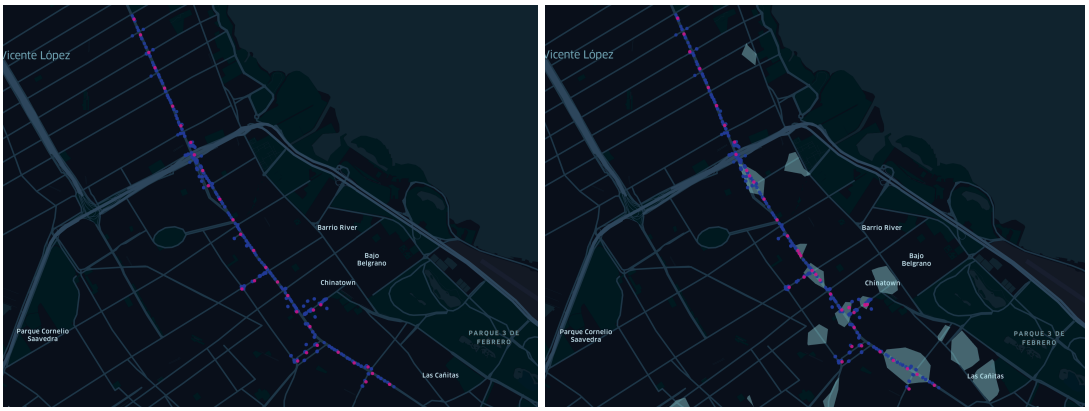

Figura 41 - Umbral X de 150m – PLE (izquierda) vs PLE Redundancias (derecha)

Con lo expuesto en esta sección se establece un modelo concreto para agregar redundancias en zonas de interés con el fin de mejorar aún más la experiencia del usuario final. En las siguientes secciones se analiza el impacto sobre los costos (cantidad de dispositivos) del esquema de redundancias al cliente.

Capítulo 5: Comparación de resultados

5.1 AMBA

En las secciones anteriores se expusieron los distintos algoritmos/modelos utilizados para construir una solución al problema de negocio planteado.

Recordando que en la zona de AMBA se inició con 19.006 paradas de colectivo y 8.327 puntos de interés. Con los puntos de interés se construyeron las zonas de interés. Resumiendo dicha información en la Figura 42.

Figura 42 - AMBA - Paradas de Colectivo (izquierda), puntos de interés (centro) y zonas de interés (derecha)

En secciones pasadas se expuso que las soluciones de PLE son escalables para los distintos umbrales de *X*. En la Figura 43 se comparan los tiempos de ejecución. Recordar que el algoritmo de Fuerza Bruta no se ejecutó para umbrales de *X* mayores a 50m.

	Umbral X (m)							
	25	50	75	100	150	200	250	300
Fuerza Bruta	9.950,53	7.740,53						
PLE	67,18	59,66	0,04	0,17	0,60	38,27	158,48	527,27
PLE PI	0,03	0,01	0,04	0,18	0,60	36,76	158,68	527,94
PLE Redundancias	0,01	0,02	0,04	0,18	0,61	42,80	171,20	449,42

Figura 43 - Tiempo (s) de resolución de la instancia más grande

Si analizamos el tiempo total que se demora en procesar todas las componentes conexas según el umbral X y el algoritmo seleccionado tenemos como resultado la Figura 44.

	Umbral X (m)							
	25	50	75	100	150	200	250	300
Fuerza Bruta	10.451	8.120						
PLE	90	78	32	19	25	52	171	530
PLE PI	30	22	35	21	26	50	172	531
PLE Redundancias	32	25	33	19	25	56	183	452

Figura 44 - Tiempo total (s) de resolución las componentes conexas

En la Figura 44 podemos ver cómo aumenta el tiempo completo del algoritmo.

Uno se puede preguntar:

- ¿Por qué el algoritmo de fuerza bruta no escala a medida que crece la componente conexa?
- ¿Por qué con Programación Lineal Entera sí escala correctamente?

La respuesta a estas preguntas es que el algoritmo de fuerza bruta busca la solución entre todas las combinaciones posibles, lo cual resulta muy ineficiente en términos computacionales. Mientras que PLE aprovecha las optimizaciones algorítmicas encapsuladas en el *solver* para buscar más eficientemente la solución (usando las restricciones y la función objetivo).

Por otro lado, se analiza la cantidad de dispositivos utilizados para un mismo Umbral *X* es la misma para los algoritmos de Fuerza Bruta, PLE y PLE PI. En cambio, el número de dispositivos de PLE con redundancias es siempre mayor o igual que los otros 3. Los resultados de este análisis se pueden ver en la Figura 45.

	Umbral X (m)							
	25	50	75	100	150	200	250	300
Fuerza Bruta	8.591	7.165	6.210	5.587	5.036	4.617		
PLE	8.591	7.165	6.210	5.587	5.036	4.617	4.128	3.602
PLE PI	8.591	7.165	6.210	5.587	5.036	4.617	4.128	3.602
PLE Redundancias	8.807	7.446	6.521	5.911	5.354	4.921	4.431	3.870

Figura 45 - Dispositivos utilizados en función del Umbral X

Se toma el ejemplo de una componente conexa que contiene las paradas enfrente al congreso de la República Argentina con los umbrales X de 100 y 150, analizando las asignaciones con los distintos algoritmos involucrados.

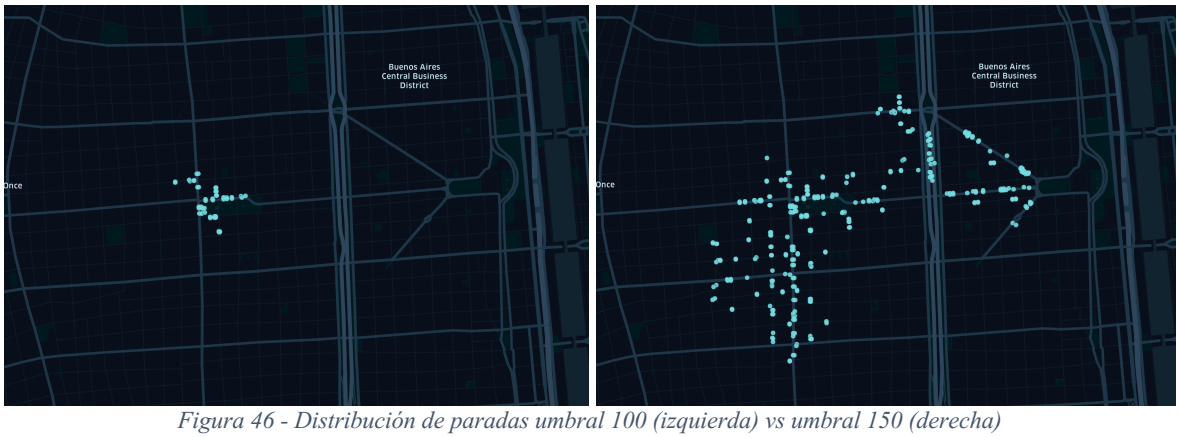

Figura 46 - Distribución de paradas umbral 100 (izquierda) vs umbral 150 (derecha)

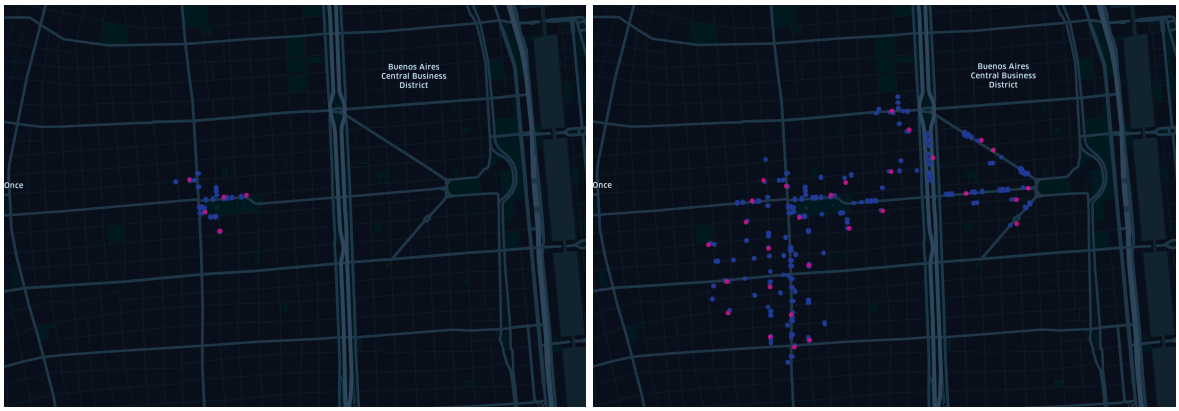

Figura 47 - Asignaciones por PLE Umbral 100 (izquierda) vs Umbral 150 (derecha)

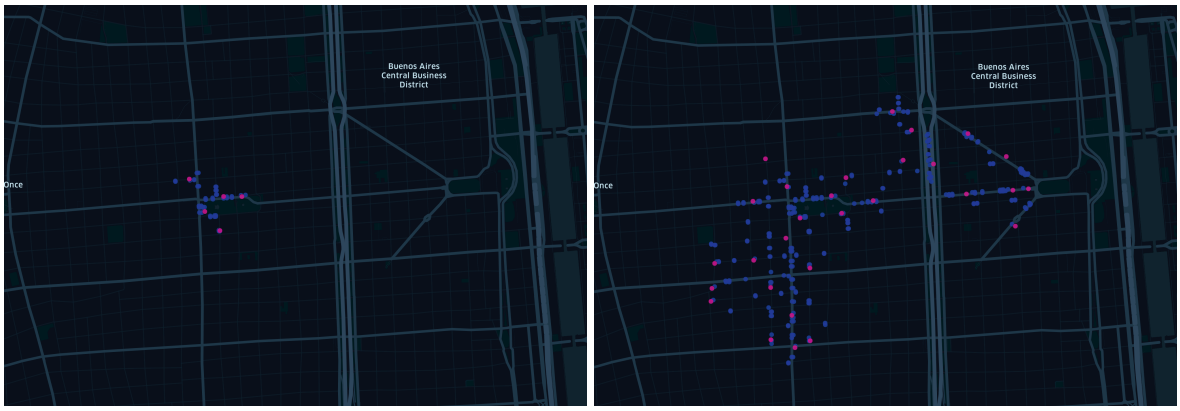

Figura 48 - Asignaciones por PLE PI Umbral 100 (izquierda) vs Umbral 150 (derecha)

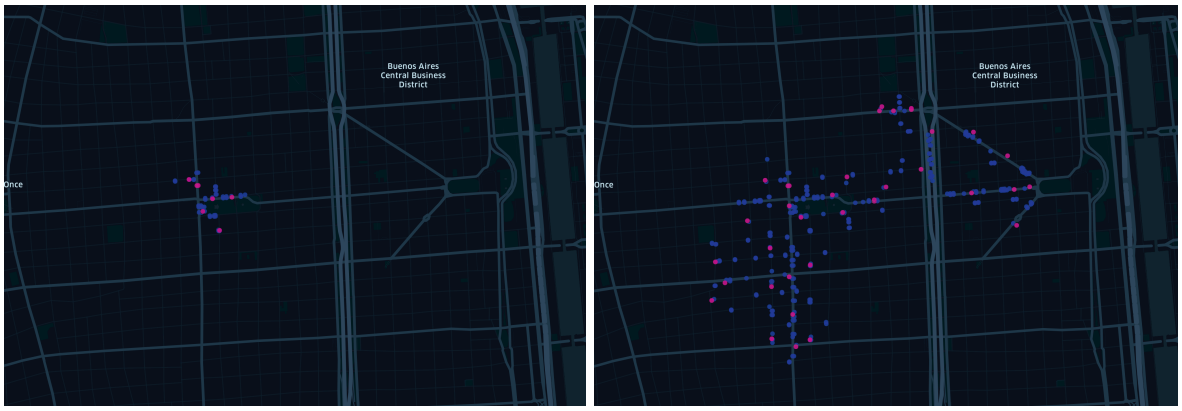

Figura 49 - Asignaciones por PLE Redundancias Umbral 100 (izquierda) vs Umbral 150 (derecha)

Para finalizar, se plantea de forma gráfica la distribución final de dispositivos en AMBA para ciertos umbrales X sobre el modelo básico de PLE y el modelo con redundancias. Se agregan también, para contrastar, las zonas de interés.

Figura 50 - Umbral X 100 - PLE

Figura 51 - Umbral X 150 - PLE

Figura 52 - Umbral X 100 - PLE Redundancias

Figura 53 - Umbral X 150 - PLE Redundancias

En este punto, tenemos suficientes variantes para exponer con el sponsor/cliente. En la conclusión de este trabajo se enuncian posibles mejoras a futuro.

5.2 Berlín

Al igual que en la sección anterior, se exponen los resultados del trabajo sobre la ciudad de Berlín.

Recordando que en la zona de Berlín se inició con 13.111 paradas de colectivo y 19.748 puntos de interés. Con los *puntos de interés* se construyeron las *zonas de interés.* Resumiendo dicha información en las imágenes de la Figura 54.

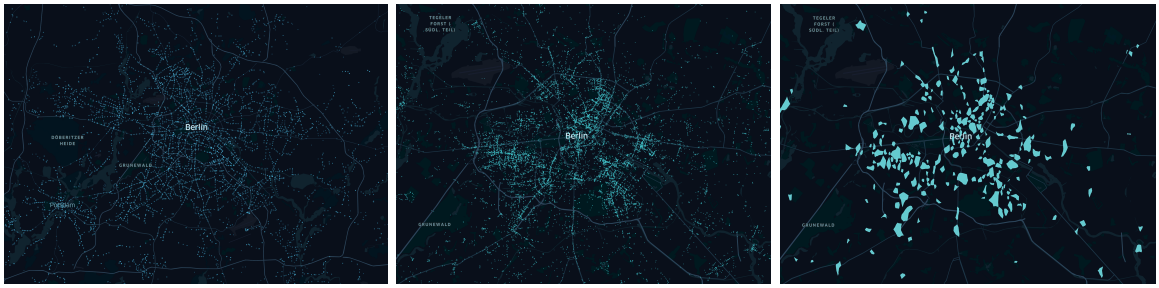

Figura 54 - Berlín - Paradas de colectivo (izquierda), puntos de interés (centro) y zonas de interés (derecha)

Se analiza en la Figura 55 de forma comparativa los tiempos de ejecución para componentes conexas complejas según los distintos algoritmos (el algoritmo de Fuerza Bruta no se ejecutó para umbrales de *X* mayores a 200m)

	Umbral X (m)							
	25	50	75	100	150	200	250	300
Fuerza Bruta	243,61	0,44	0,09	0,04	3,75	28,73		
PLE	0,45	0,01	0,00	0,01	0,02	0,13	0,01	0,07
PLE PI	0,01	0,01	0,01	0,00	0,02	0,01	0,01	0,06
PLE Redundancias	0,01	0,01	0,01	0,00	0,02	0,01	0,01	0,07

Figura 55 - Tiempo (s) de resolución de la componente conexa más complejo

Si analizamos el tiempo total que se demora en procesar todas las componentes conexas según el umbral X y el algoritmo seleccionado tenemos como resultado la Figura 56.

	Umbral X (m)							
	25	50	75	100	150	200	250	300
Fuerza Bruta	390,3	1,5	1,0	1,6	6,3	34,7		
PLE	30,8	35,8	14,3	13,2	13,0	11,0	13,1	15,3
PLE PI	38,3	45,6	21,9	19,5	18,5	16,4	14,2	14,4
PLE Redundancias	38,7	46,4	22,8	19,6	18,3	17,4	13,8	14,6

Figura 56 - Tiempo total (s) de resolución los grafo

En la Figura 56 podemos ver cómo aumenta el tiempo completo del algoritmo.

Como mencionamos en la sección anterior, la cantidad de dispositivos utilizados para un mismo Umbral *X* es la misma para los algoritmos de Fuerza Bruta, PLE y PLE PI. En cambio, el número de dispositivos de PLE con redundancias es siempre mayor o igual que los otros 3. Los resultados de este análisis se pueden ver en la tabla de la Figura 57.

	Umbral X (m)							
	25	50	75	100	150	200	250	300
Fuerza Bruta	8.591	7.165	6.210	5.587	5.036	4.617		
PLE	8.591	7.165	6.210	5.587	5.036	4.617	4.128	3.602
PLE PI	8.591	7.165	6.210	5.587	5.036	4.617	4.128	3.602
PLE Redundancias	8.807	7.446	6.521	5.911	5.354	4.921	4.431	3.870

Figura 57 - Dispositivos utilizados en función del Umbral X

Se toma el ejemplo de componente conexa un que contiene las paradas enfrente al Zoológico de Berlín con los umbrales X de 250 y 300 en la Figura 58.

Figura 58 - Distribución de paradas umbral 250 (izquierda) vs umbral 300 (derecha)

Se analizan las asignaciones de las componentes conexas de la Figura 58 y se exponen los resultados en las Figura 59, Figura 60 y Figura 61.

Figura 59 - Asignaciones por PLE umbral 100 (izquierda) vs umbral 150 (derecha)

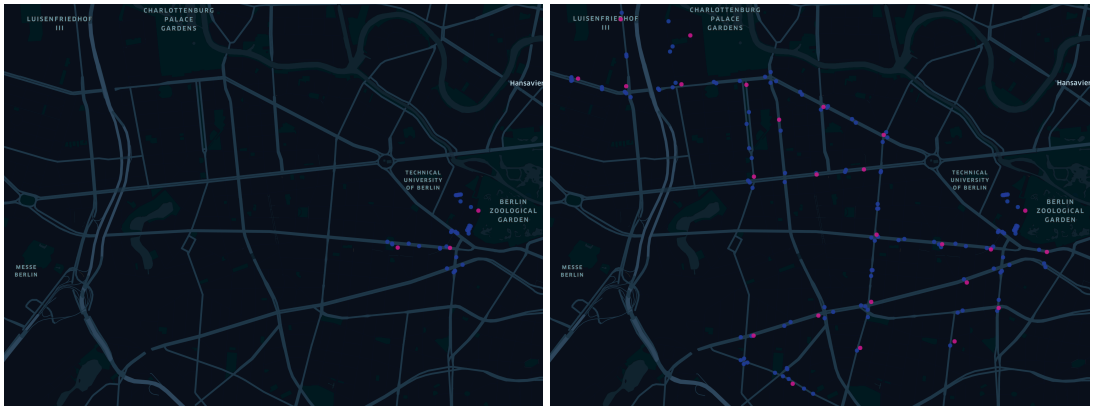

Figura 60 - Asignaciones por PLE PI umbral 100 (izquierda) vs umbral 150 (derecha)

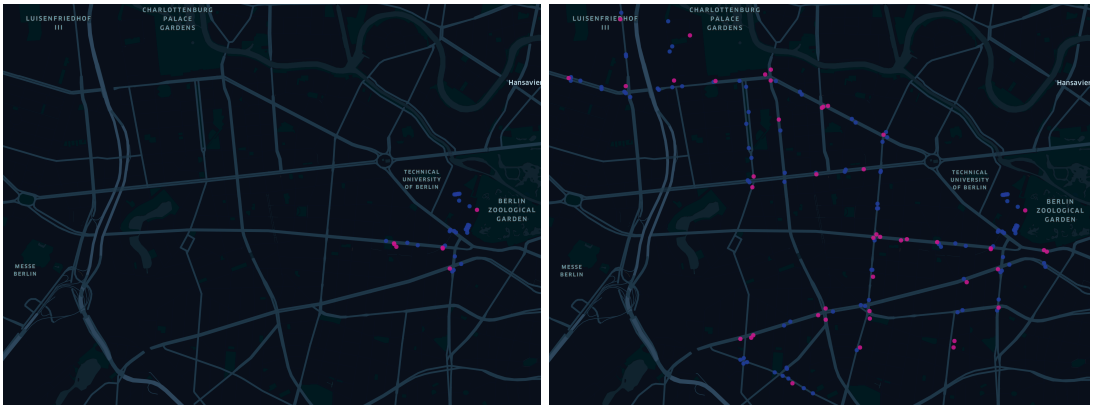

Figura 61 - Asignaciones por PLE redundancias umbral 100 (izquierda) vs umbral 150 (derecha)

Para finalizar, se plantea de forma gráfica la distribución final de dispositivos en Berlín para ciertos umbrales X sobre el modelo básico de PLE (Figura 62 y Figura 63) y el modelo con redundancias (Figura 64 y Figura 65). Se agregan también, para contrastar, las zonas de interés.

Figura 62 - Umbral X 100 - PLE

Figura 63 - Umbral X 150 - PLE

Figura 64 - Umbral X 100 - PLE Redundancias

Figura 65 - Umbral X 150 - PLE Redundancias

Capítulo 6: Conclusión

El problema inicial constaba de un desembarco de una nueva tarjeta de transporte con saldo recargable (similar a la SUBE en Argentina).

El modo de funcionamiento es:

- En primera instancia con una billetera virtual realizar el pago del monto a cargar.
- En una segunda instancia se actualiza el saldo en la tarjeta. Este último paso se puede hacer con algunas Apps en un teléfono con NFC o bien desde un *punto físico de recarga*.

El objetivo es analizar cómo distribuir estos puntos físicos de recarga en las paradas de colectivo de una ciudad de una manera eficiente, utilizando la menor cantidad de dispositivos pero manteniendo un buen servicio para el usuario final.

Se expuso que este requerimiento de negocio se puede mapear a problemas conocidos (Set Covering y Conjunto Dominante) sobre los cuales hay abundante literatura.

La solución fue pensar al problema en términos de grafos:

- Cada parada representa un vértice.
- Si las paradas A y B se encuentran (entre ellas) a una distancia menor o igual al umbral *X*, entonces existe una arista entre el vértice A y vértice B.

Desde el principio se usó una estrategia de dividir el problema en partes más pequeñas y simples de procesar. Lo cual implica dividir el grafo, que representa a toda una ciudad, en sus componentes conexas. El análisis se debe hacer sobre cada uno de estos *sub-grafos* (componentes conexas).

A partir de esto se diseñaron cuatro soluciones:

- Fuerza Bruta
- PLE: Modelo básico de Programación Lineal Entera
- PLE PI: Es el modelo anterior sumando un criterio de priorización de paradas mediante puntos de interés cercanos
- PLE Redundancias: Es el PLE PI sumando un criterio de asignación de redundancias mediante las zonas de interés

Es importante notar que las soluciones diseñadas exponen diversos escenarios y dejan variables a definir con el cliente/sponsor del proyecto (Umbral *X*). También se planteó un paso extra de agregar redundancias acorde a un criterio coherente.

Se expusieron los resultados y se analizaron:

- Casos particulares para constatar cómo se asignan los dispositivos según los distintos algoritmos
- Performance de los distintos algoritmos (a nivel tiempos)
- Escalabilidad de los distintos algoritmos

- Distintos resultados según el Umbral *X:*
    - Cantidad de dispositivos
    - Asignaciones especificas según los criterios establecidos

Considerando lo mencionado anteriormente, se demostró que tres de las cuatro soluciones expuestas escalan correctamente (a nivel performance del algoritmo) para ciudades con una abundante distribución de paradas de autobús en las cuales también se pueden usar umbrales de *X* altos.

Con los resultados obtenidos, se puede plantear/seleccionar una solución formal para el cliente según sus preferencias con respecto a las variables restantes.

Posibles mejoras al modelo/algoritmo pueden ser: mejorar el método de selección cuando dos paradas son equivalentes en el resultado; considerar una mejor métrica para calcular la distancia entre dos coordenadas, tener en cuenta las zonas peligrosas de una ciudad.

Como hemos mencionado, las aplicaciones de este problema son muy diversas. Estos conceptos se pueden utilizar en muchos ámbitos y situaciones como diseño de una nueva ciudad, ubicación de antenas de telefonía, *random testing* de programas, etc. Todos estos ejemplos pueden ser modelados de forma similar al planteado en este trabajo.

Aunque en este trabajo se plantean soluciones exactas, en distintos problemas el número de variables puede ser imposible de procesar de esta forma. Para estos casos se pueden utilizar heurísticas.

Anexo 1: Sets de datos

Las principales fuentes de información se obtuvieron de OpenStreetMap[12] mediante Overpass Turbo[13].

Para obtener las paradas de cada ciudad se ejecutó la siguiente consulta:

```
[out:json][timeout:25];
(
  node["public_transport"="platform"][bus=yes]({{bbox}});
  node["public_transport"="stop_position"][bus=yes]({{bbox}});
);
out body;
>;
out skel qt;
```

Para obtener los puntos de interés de cada ciudad se ejecutó la siguiente consulta:

```
[out:json];
(
        node["tourism"~"museum|gallery|information|hotel|attraction|viewpoint"]({{bbox}});
        node["amenity"~"restaurant|cafe|bar|place_of_worship|pharmacy|atm|hospital|pub"]({{bbox}});
        node["office"~"government|company|yes"]({{bbox}});
);
out center;
```

En Berlín se obtuvieron 13.111 paradas y 19,748 puntos de interés, mientras que en AMBA se obtuvieron 19.006 y 8.327 respectivamente.

Desde OverPass Turbo seleccionamos el área bajo análisis:

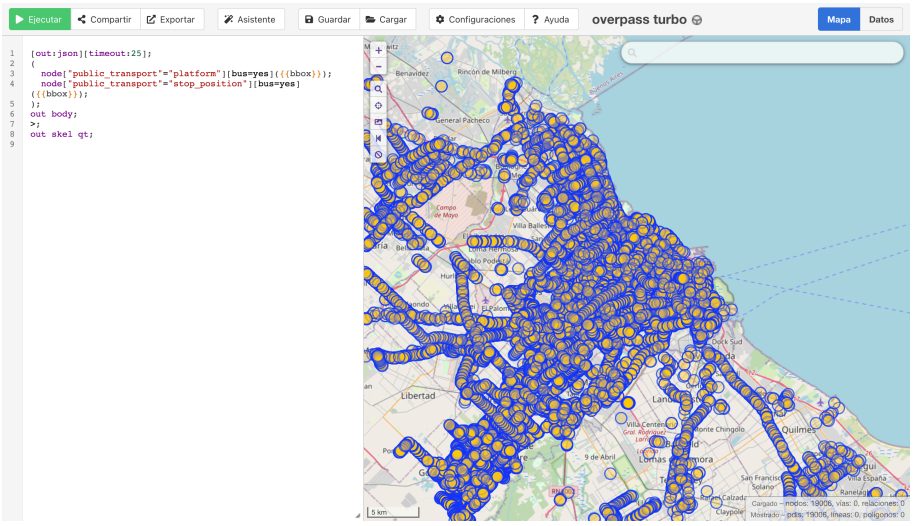

Figura 66 - Descarga de paradas de AMBA – Selección de área a analizar

[12] OpenStreetMap es una base de datos de información geoespacial open source

[13] Overpass Turbo es una herramienta web de para acceder y explotar información de OpenStreetMap. Para más información visitar https://wiki.openstreetmap.org/wiki/Overpass_turbo y https://overpass-turbo.eu/

Desde la sección *Datos* se puede extraer la información fuente:

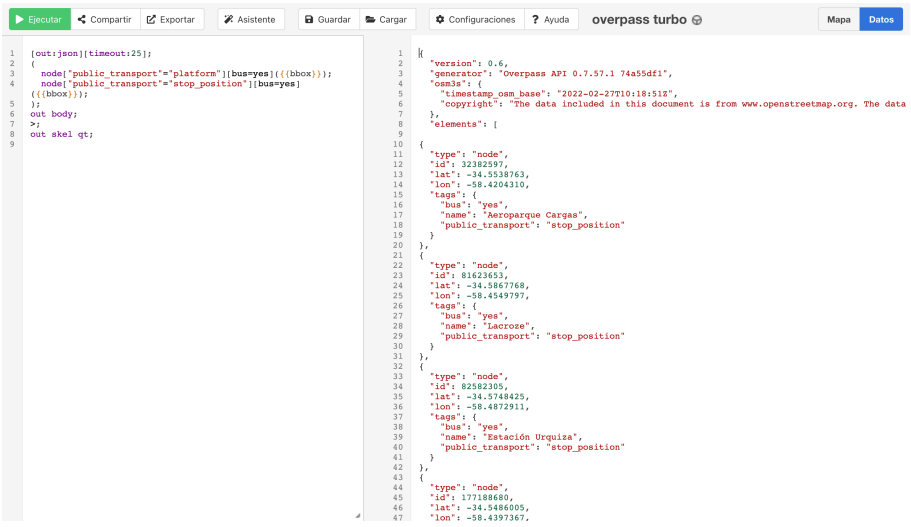

Figura 67 - Descarga de paradas de AMBA

Se repite el proceso pero el código se modifica para obtener puntos de interés del mismo área de análisis:

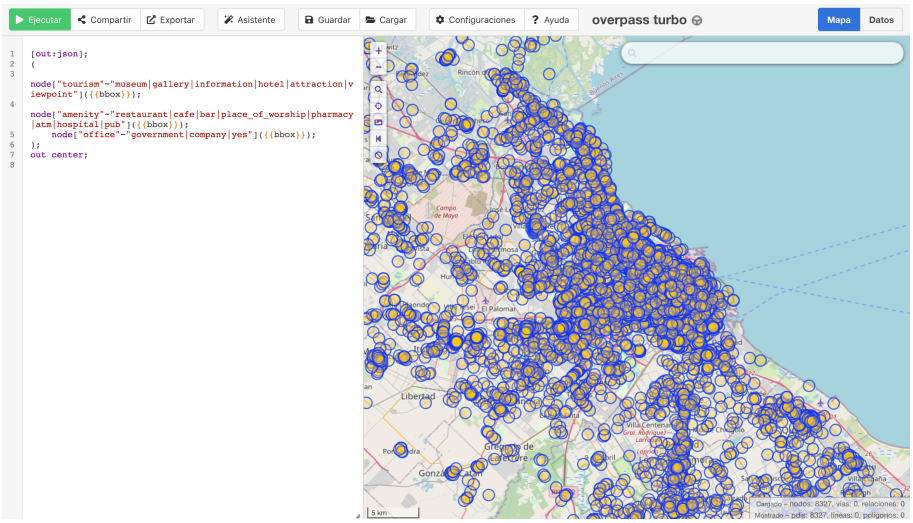

Figura 68 - Descarga de puntos de interés de AMBA

Repetimos el mismo proceso para el caso de Berlín seleccionando el área a análizar.

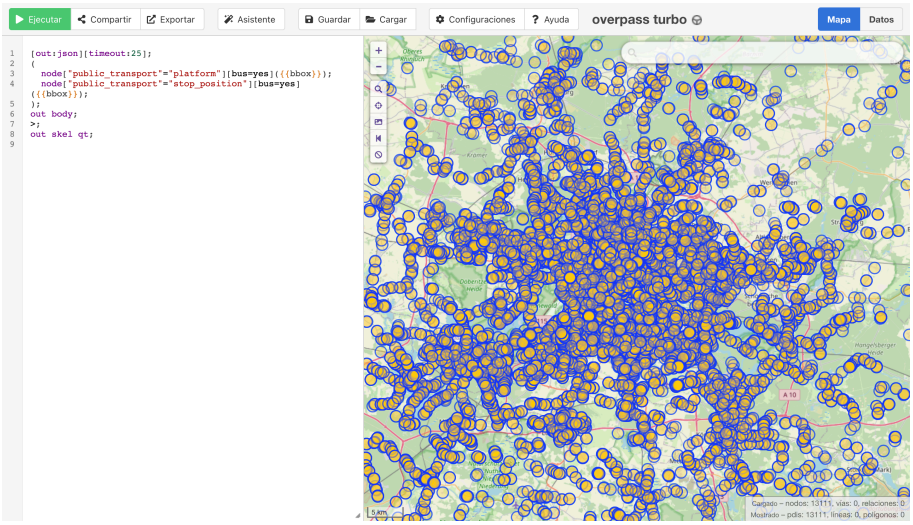

Figura 69 - Descarga de paradas de Berlín – Selección de área a analizar

Nuevamente, desde la sección *Datos* se puede extraer la información fuente:

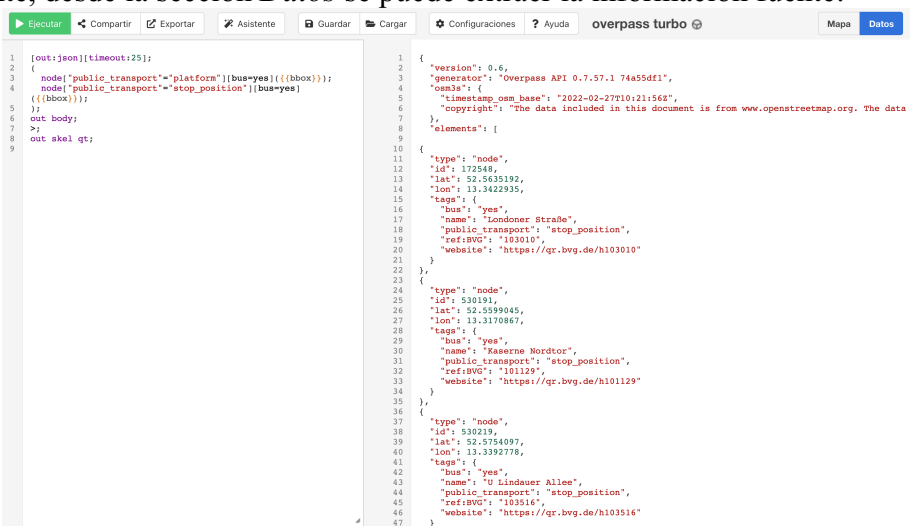

Figura 70 - Descarga de paradas de Berlín – Descargando los puntos

El procedimiento se repite paro ahora extrayendo los puntos de interés.

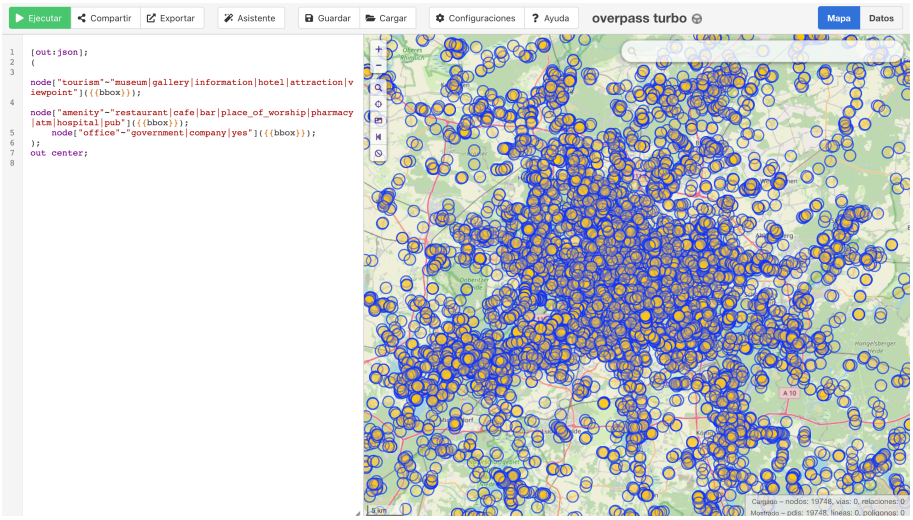

Figura 71 - Descarga de puntos de interés de Berlín

Anexo 2: Tecnologías utilizadas

A lo largo de este trabajo se utilizaron diversas bibliotecas, herramientas y tecnologías. En esta sección se mencionan las más importantes.

Primero, para el procesamiento de datos se utilizó *python* sobre *jupyter notebooks*. Aquí se utilizaron bibliotecas con diversos fines:

- Manipulación y procesamiento de datos:
  - pandas,
  - pandasql
  - geopandas
  - shapely
  - numpy
  - math
  - alphashape
  - DBSCAN (from sklearn)
- Visualizaciones:
  - contextily
  - matplotlib
  - seaborn
  - descartes
- Otras auxiliares:
  - time
  - datetime
  - json
  - os
  - itertools
  - sys

Segundo, para procesamiento de problemas de programación lineal entera se usó el *solver* CPLEX[14], perteneciente a IBM. En el capítulo de comparación de resultados se pudo dejar evidencia de la buena performance del *solver* comparado contra un algoritmo de fuerza bruta.

Por último, para visualización de información geográfica sobre mapas se utilizó Kepler[15], una herramienta muy interesante para mostrar rápidamente miles (o millones) de geopoints. La misma fue creada por el equipo de Visualización de Uber.

[14] Para más información visitar *https://www.ibm.com/analytics/cplex-optimizer*
[15] Para más información visitar *https://kepler.gl/*

Bibliografía

- Karp, Richard M.(1972) *Reducibility Among Combinatorial Problems*. Complexity of Computer Computations, New York: Plenum. páginas 85–103. Recuperado de: https://cgi.di.uoa.gr/~sgk/teaching/grad/handouts/karp.pdf
- Broderick Crawford, Ricardo Soto, Hanns de la Fuente Mella, Claudio Elortegui, Wenceslao Palma, Claudio Torres-Rojas, Claudia Vasconcellos-Gaete, Marcelo Becerra, Javier Peña, Sanjay Misra. Tech science (2021), volumen 71. *Binary Fruit Fly Swarm Algorithms for the Set Covering Problem*. Recuperado de: https://techscience.com/cmc/v71n3/46489/html
- Atoosa KasirZadeh, Mohammed Saddoune & Francois Soumi. Euro Journal of Transportation and Logistics (2017) , volumen 2, páginas 111-137. *Airline crew scheduling: models, algorithms and data sets*. Recuperado de: https://www.sciencedirect.com/science/article/pii/S2192437620300820
- Fabrizio Grandoni, Abhilekh Gupta, Stefano Leonardi , Pauli Miettinen. Foundation of computer science conference (2008). *Set Covering with Our Eyes Closed*. Recuperado de: https://people.idsia.ch/~grandoni/Pubblicazioni/GGLMSS13sicomp.pdf
- Yiyuan Wang, Shiwei Pan, Sameh Al-Shihabi, Junping Zhou, Nan Yang, Minghao Yin. Euro Journal of Transportation and Logistics (2021), volumen 294, páginas 476-491. *An improved configuration checking-based algorithm for the unicost set covering problem*. Recuperado de: https://www.sciencedirect.com/science/article/abs/pii/S037722172100093X
- Douglas West. (2001). *Introduction to Graph Theory* (Capítulo 1)
- Wikipedia (Julio 2022). *Grafo*. Recuperado de: https://es.wikipedia.org/wiki/Grafo
- Wikipedia (Marzo 2023). *Matriz de adyacencia*. Recuperado de: https://es.wikipedia.org/wiki/Matriz_de_adyacencia
- Wikipedia (Diciembre 2020). *Vértice (teoría de grafos)*. Recuperado de: https://es.wikipedia.org/wiki/V%C3%A9rtice_(teor%C3%ADa_de_grafos)
- Wikipedia (Julio 2021). *Problema del conjunto de cobertura*. Recuperado de: https://es.wikipedia.org/wiki/Problema_del_conjunto_de_cobertura
- Cornell Wiki (Diciembre 2020). *Set covering problem*. Recuperado de: https://optimization.cbe.cornell.edu/index.php?title=Set_covering_problem
- Wikipedia (Marzo 2022). *Dominating Set*. Recuperado de: https://en.wikipedia.org/wiki/Dominating_set
- Wikipedia (Noviembre 2022). *Sudoku solving algorithms*. Recuperado de: https://en.wikipedia.org/wiki/Sudoku_solving_algorithms
- Wikipedia (Enero 2023). *Exact cover*. Recuperado de: https://en.wikipedia.org/wiki/Exact_cover
- Explain xkcd (June 2022). *1313: Regex Golf*. Recuperado de: https://www.explainxkcd.com/wiki/index.php/1313:_Regex_Golf
- Wikipedia (Febrero 203). *Component(graph theory)*. Recuperado de: https://en.wikipedia.org/wiki/Component_(graph_theory)