

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



**“OPTIMIZADOR DE RUTA PARA LA COMPRA DE
INSUMOS EN DISTINTOS LOCALES DE SUMINISTROS”**

NICOLÁS ALBERTO PUENTE RIQUELME

**MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA**

Profesor Guía: José Luis Martí Lara
Profesor Correferente: Orlando Andrade Cornejo

Agosto - 2023

DEDICATORIA

A mis padres, hermanos y amigos que me acompañaron a lo largo de esta carrera y me comprendieron durante todo este proceso.

AGRADECIMIENTOS

Me gustaría agradecer a todas las personas que me ayudaron y acompañaron durante mi etapa universitaria.

Primero que todo a mis padres, por darme los recursos y la confianza para poder terminar mi carrera. A mis hermanos por entenderme siempre que tuviera cosas de la U y/o necesitase que me acercaran al metro porque me quedé dormido.

En segundo lugar, a todas esas personas que conocí en la U, muchos de ellos puedo considerarlos grandes amigos (grande modo NiPu) con los que pasamos muy buenos momentos durante todo este tiempo, desde jugar póker con monedas de 10 pesos hasta las pichangas contra el clásico rival. También quiero agradecer a Javiera Romo, por soportar mis quejas, alegrías y frustraciones durante toda la carrera, además de conseguir aún más voluntarios para el desarrollo de este trabajo.

Además, darle las gracias al profesor José Luis Martí por su buena disposición y compromiso durante todo el proyecto y como profesor guía en este trabajo. Además, agradecer a Zippedi por la oportunidad, al equipo de Software y Cloud por hacerme sentir uno más y, en especial a Orlando Andrade y Rodrigo González por su ayuda e interés a lo largo de todo el proyecto.

Por último, queda agradecer a todas las personas que se tomaron la molestia de servir como voluntarios para el desarrollo de este trabajo.

RESUMEN

Resumen— El tiempo de *Pick-Up* dentro de un supermercado depende en gran medida por la experiencia que tenga el cliente sobre su local de compra, por lo que puede volverse un proceso lento e ineficiente para las personas con menos experiencia. En este trabajo se desarrolla una aplicación móvil que utiliza los datos proporcionados por Zippedi para guiar al usuario, generando una ruta basada en un algoritmo de inteligencia artificial.

Luego de un proceso de pruebas utilizando un local como piloto, fue posible concluir que, gracias a las rutas generadas, los usuarios lograron reducir el tiempo para completar una lista de compras en comparación a las personas que simplemente utilizaron un papel.

Palabras Clave— Inteligencia artificial, Pathfinding, Retail, Desarrollo Móvil, *Pick-Up*

ABSTRACT

Abstract—The Pick-Up time within a supermarket is heavily influenced by the customer's familiarity with the shopping location, which can result in a slow and inefficient process for individuals with less experience. This study presents a mobile application developed to utilize data provided by Zippedi in order to guide users, generating a route based on an artificial intelligence algorithm.

Following a testing process using a pilot store, it was possible to conclude that, thanks to the generated routes, users managed to reduce the time required to complete a shopping list compared to individuals who solely used a paper-based approach.

Keywords— Artificial Intelligence, Pathfinding, Retail, Mobile Development, Pick-Up

INDICE DE CONTENIDOS

RESUMEN.....	IV
ABSTRACT	IV
INDICE DE FIGURAS.....	VII
INDICE DE TABLAS.....	IX
INTRODUCCIÓN	1
CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA	2
1.1 Contexto y situación actual	2
1.2 Objetivos	6
1.3 Alcance	6
CAPÍTULO 2: MARCO CONCEPTUAL	9
2.1. Grafos	9
2.2 Algoritmos para planificación de rutas	11
2.2.1 Algoritmo de Dijkstra	12
2.2.2 Algoritmo de búsqueda A*	12
2.3 Problema del vendedor viajero.....	14
2.3.1 Vecino más cercano.....	17
2.3.2 Búsqueda Local	17
2.4. Procesamiento de imágenes	18
2.4.1. Umbral	18
2.4.2. Operaciones Morfológicas.....	19
2.4.2.1. Dilatación	19
2.4.2.2 Erosión	20
2.4.2.3 Apertura.....	21

OPTIMIZADOR DE RUTA PARA LA COMPRA DE INSUMOS EN DISTINTOS LOCALES DE SUMINISTROS

2.4.2.4 Cerrado	22
2.4.2.5 Encontrar perímetro	22
2.4.2.6 Transformada de Sombrero de Copa	23
2.4.2.7 Transformada de Sombrero de Copa Invertido	23
2.5 Algoritmos para dibujo de líneas en dispositivos digitales.....	24
2.5.1 Algoritmo DDA.....	24
2.5.2 Algoritmo de Bresenham.....	24
CAPÍTULO 3: PROPUESTA DE SOLUCION	26
3.1 Primer Ciclo: Pick-Up Routing Algorithm	26
3.2 Segundo Ciclo: Diseño y desarrollo de la aplicación móvil	34
CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN	38
4.1 Descripción del Experimento	38
4.1.1 Conformación de las listas de productos.....	38
4.1.2 Recolección de Datos.....	40
4.2 Desarrollo del experimento	43
CONCLUSIONES.....	50
REFERENCIAS BIBLIOGRÁFICAS.....	54

INDICE DE FIGURAS

Figura 1: Mapa de un local escaneado.	2
Figura 2: Ruta recorrida por el robot de <i>Pick-Up</i>	3
Figura 3: Ruta de <i>Pick-Up</i> para personas.....	4
Figura 4: Razones por la cual los consumidores prefieren <i>Cornershop</i>	5
Figura 5: Etapas de <i>Shape Up</i>	6
Figura 6: Etapas de un ciclo de desarrollo (o <i>Building</i>).	7
Figura 7: Ejemplo de una colina de <i>Shape Up</i>	8
Figura 8: Ejemplo entre un grado dirigido y uno no dirigido.	9
Figura 9: Grafo no dirigido y su matriz de adyacencia.	10
Figura 10: Movimientos permitidos para 4 y 8 vecinos.	11
Figura 11: Algoritmo A* sobre un grafo de grilla.	12
Figura 12: Ejemplo distancia de Manhattan.....	14
Figura 13:Espacio de búsqueda de la función objetivo.	16
Figura 14: Ejemplo utilizando la heurística del vecino más cercano.....	17
Figura 15: Ejemplo de aplicar un umbral a una imagen.....	19
Figura 16: Ejemplo de una operación de dilatación.	20
Figura 17: Representación binaria de una operación de dilatación.....	20
Figura 18: Ejemplo de una operación de erosión.....	21
Figura 19: Representación binaria de una operación de erosión.	21
Figura 20: Ejemplo de apertura sobre una imagen binaria.....	22
Figura 21: Ejemplo de cierre sobre una imagen binaria.	22
Figura 22: Ejemplo de la búsqueda de perímetro sobre una imagen binaria.....	23
Figura 23: Ejemplo de una transformada de Sombrero de Copa.....	23

Figura 24: Ejemplo de una transformada de Sombrero de Copa Invertido.	24
Figura 25: Ejemplo del algoritmo de Bresenham.	25
Figura 26: Mapa de las instalaciones dentro del Jumbo Kennedy, antes y después de su preprocesamiento.	28
Figura 27: Mapa binario inverso.....	29
Figura 28:Nodos de navegación.	30
Figura 29: Mapa de las instalaciones dentro del Jumbo Kennedy	31
Figura 30: Trazado de la ruta entregada por el solucionador de OR-Tools.	33
Figura 31: Trazado de la ruta generada por A*.	33
Figura 32: Mockup de la aplicación móvil.	35
Figura 33: Vista de la descripción y posición del producto en cada góndola.	36
Figura 34: Vista del resumen de Pick-Up.....	36
Figura 35: Posición de los productos en el mapa.	41
Figura 36:Ejemplo de un producto sin foto y/o posición en góndola.	43
Figura 37: Diagrama de caja para los tiempos de la Lista 1.	44
Figura 38:Diagrama de caja para los tiempos de la Lista 2.	45
Figura 39: Diagrama de torta de todos los productos recopilados con la app.	46
Figura 40: Productos más difíciles de encontrar de la Lista 1.	48
Figura 41: Productos más difíciles de encontrar de la Lista 2.	49

INDICE DE TABLAS

Tabla 1: Distribución de los grupos de voluntarios y cantidad de productos.	38
Tabla 2: Ejemplo de papel impreso de la lista 2.	41
Tabla 3: Promedio de tiempos para cada lista. Fuente: Elaboración Propia.	43
Tabla 4: Cantidad Promedio de Productos encontrados por Lista.....	46

INTRODUCCIÓN

Zippedi es una empresa de robótica que utiliza robots autónomos equipados con cámaras y sensores que recorren las tiendas capturando imágenes, las cuales posteriormente son analizadas por algoritmos de inteligencia artificial para obtener información relevante como la verificación de precios, la gestión de inventario y la disponibilidad de productos en los estantes (o góndolas).

Por otro lado, cada vez suelen ser más las personas que utilizan aplicaciones como Cornershop, en donde los usuarios pagan porque otras personas realicen sus compras en los locales de suministros que ellos necesiten. A las personas que trabajan realizando estas compras suelen denominarlas *Shoppers* y el método a utilizar para calificar a estos *Shoppers* suele ser en base al tiempo demorado y la cantidad de productos que suelen encontrar de las listas que se les asignen. En base a lo anterior, este trabajo tiene como objetivo reducir los tiempos de *Pick-Up* utilizando los datos levantados por Zippedi diseñando y desarrollando una aplicación móvil para guiar a estos *Shoppers* a lo largo de su recorrido.

El presente documento fue organizado en los siguientes capítulos:

1. **Definición del Problema:** Se da a entender el contexto y la situación actual en Zippedi, junto con los antecedentes y la problemática que se busca resolver.
2. **Marco Conceptual:** Se definen los conceptos necesarios para entender los temas a tratar en este trabajo.
3. **Propuesta de solución:** Se indica la metodología a utilizar para el diseño y desarrollo de la aplicación de *Pick-Up*.
4. **Validación de la solución:** Se presenta la metodología realizada en el proceso de pruebas, junto al posterior análisis de los datos obtenidos a lo largo de este proceso.

Finalmente se entregan las conclusiones, en donde se da a entender el aprendizaje adquirido al realizar este trabajo, sus limitaciones para continuar con el cumplimiento de los objetivos planteados al inicio del trabajo y terminar planteando el trabajo futuro.

CAPÍTULO 1: DEFINICIÓN DEL PROBLEMA

1.1 Contexto y situación actual

La idea de esta memoria nace como una oportunidad para aprovechar la información constantemente recopilada por el robot de Zippedi, el cual tiene como tarea revisar más de 13 mil productos ubicados en sus respectivas góndolas, para cada uno de los distintos locales de suministros en los que se encuentran. Con esta información, es posible chequear si hay stock, si los precios están bien puestos o si un producto se encuentra en el lugar que le corresponde. Además de esto, el robot va escaneando todo el local, produciendo así un mapa digital como el de la Figura 1.

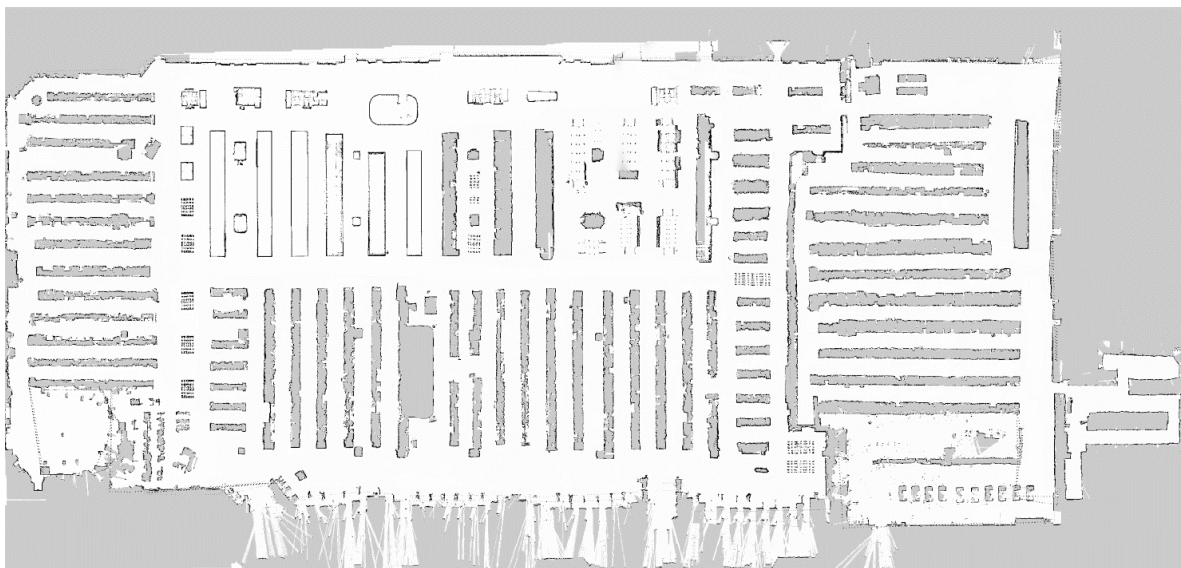


Figura 1: Mapa de un local escaneado.

Entonces, teniendo el mapa de cada local y la ubicación de cada producto dentro de él, es posible representar cada uno de los productos como nodos, y la distancia entre cada uno de ellos se vería representada como los vértices de un grafo. De esta forma, en Zippedi se planteó un optimizador de ruta para el proceso de *Pick-Up* realizado por un robot, el cual realiza un recorrido poligonal debido a que este algoritmo genera unos nodos de navegación que son una proyección de unos 30 cm del borde de los objetos escaneados en el mapa (las góndolas u otros objetos intransitables).



Figura 2: Ruta recorrida por el robot de *Pick-Up*.

Como es posible observar en la Figura 2, este tipo de ruta tiene en consideración evitar cualquier tipo de colisión que pudiese sufrir el robot con el entorno en el cual se encuentra mientras realiza su recorrido, gracias a la gran cantidad de nodos de navegación generados por cada objeto cerrado que se encuentre en el mapa escaneado. Por esta razón, se propone generar nodos principalmente por intersección, junto a los nodos que representan los productos de una lista de compras. De esta manera, se generaría una menor cantidad de nodos de forma que el procesamiento al momento de buscar una ruta óptima a seguir sea menor, y a la vez se tendrían rutas más fáciles de seguir por una persona, obteniendo como resultado una ruta como la de la Figura 3.



Figura 3: Ruta de Pick-Up para personas.

Con este optimizador de ruta para las personas, se busca disminuir el tiempo de *Pick-Up* dada una lista de compras que puede pasar una persona particular o *Shopper* en un determinado local, ya que en gran medida el tiempo de *Pick-Up* suele tener directa relación a la experiencia previa que tenga esta persona sobre la distribución que se tenga en un local y/o la ubicación exacta de un producto en particular en comparación a alguien que carece de este conocimiento. En este sentido, cabe señalar que aun encontrándose en el pasillo correcto, se suele perder tiempo buscando un producto particular mirando las góndolas de arriba abajo, por lo que además de saber qué ruta debe seguir, también se le debe facilitar la posición del producto en la góndola.

Principalmente en el caso de los *Shoppers*, los cuales suelen ir en representación de un cliente, deben ser capaces de cumplir con los pedidos de dicho cliente de la mejor manera posible y al mismo tiempo en el menor tiempo posible, ya que empresas como *Cornershop* suelen prometer un tiempo máximo de espera o en un tiempo previamente acordado siendo responsabilidad del *Shopper* cumplir con estas expectativas, en un rubro que desde el inicio de la pandemia tuvo un gran crecimiento a nivel nacional y en donde los clientes exigen un servicio de calidad que puede verse afectado por problemas que ocurren dentro del mismo local, cuando el *Shopper* se encuentra con que hay productos que realmente no tienen stock y debe consultarle al cliente por alguna alternativa o simplemente desistir de comprar, pudiendo así perder confianza en el *Shopper* o en la empresa que entrega este servicio ya que, como es posible observar en la Figura 4, por una amplia ventaja, la razón

por la cual muchos suelen preferir utilizar *Cornershop* es debido a que el tiempo de entrega es el adecuado, por lo que para mantener la fidelidad de estos clientes es necesario cumplir con sus expectativas.

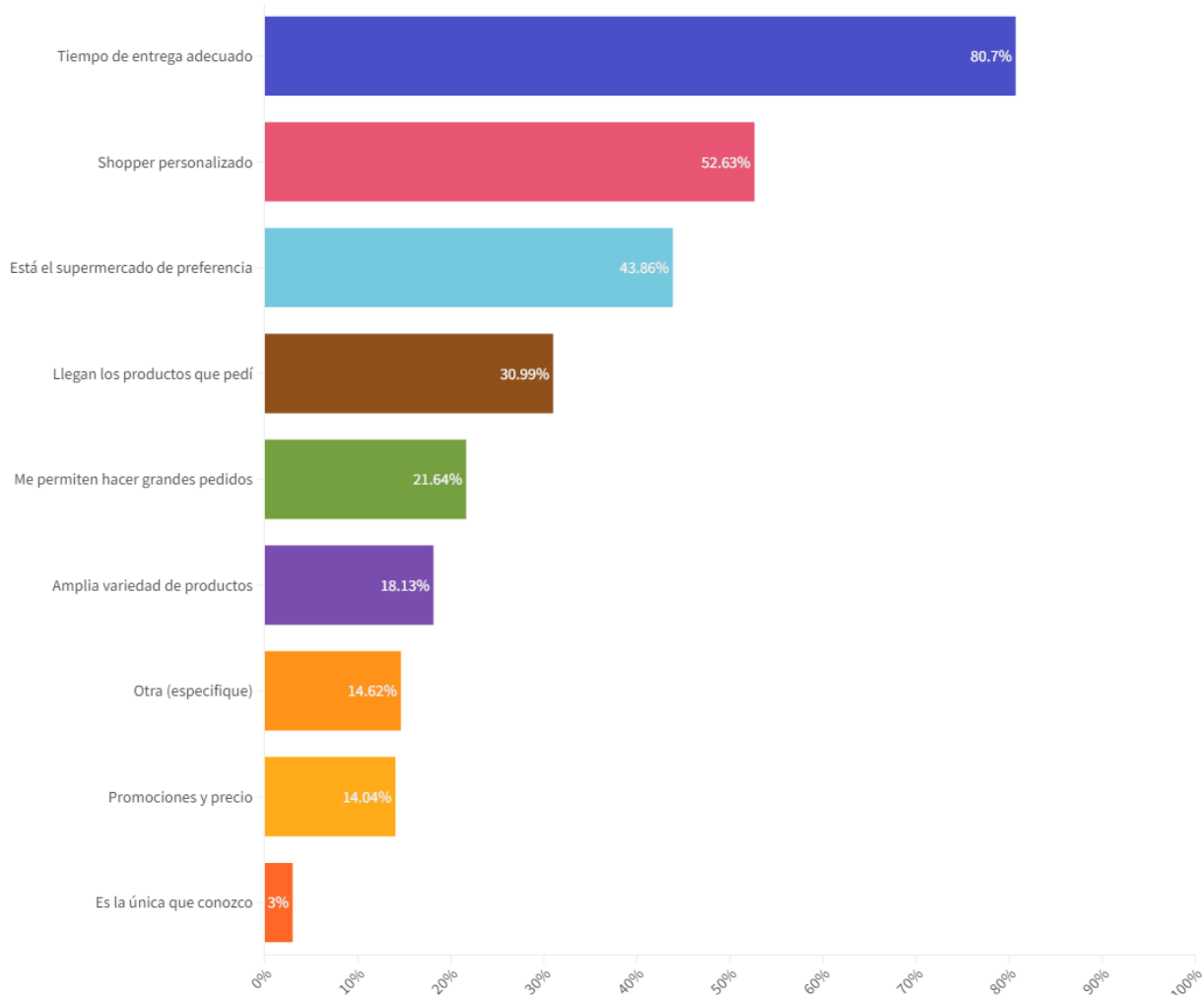


Figura 4: Razones por la cual los consumidores prefieren Cornershop.

Ahora bien, para poder guiar al *Shopper* en su recorrido, se propone diseñar y desarrollar una aplicación móvil, la cual guíe al usuario por el local desde su ingreso hasta llegar a la caja, ya que el tiempo en que demore el usuario en pasar por la caja no está considerado en el alcance de esta memoria. De esta forma, el usuario solo necesitaría de un dispositivo móvil, y específicamente en el caso de los *Shoppers* este requisito ya estaría previamente satisfecho ya que requieren de dicho dispositivo para poder trabajar como *Shopper* y comunicarse con el cliente en caso de ser necesario.

1.2 Objetivos

El objetivo general de esta memoria consiste en reducir los tiempos de *Pick-up* de los productos en tiendas de retail, a través del diseño y desarrollo de un algoritmo de optimización de rutas y una aplicación para el apoyo en dicho proceso de *Pick-Up*.

En cuanto a los objetivos específicos, estos son:

- Diseñar y desarrollar un algoritmo optimizador de ruta en base a la lista de objetos a comprar, para así poder generar una ruta de *Pick-Up* para los *Shoppers*, con el objetivo de reducir el tiempo de *Pick-Up*.
- Construir una aplicación móvil que sirva de apoyo para los *Shoppers* como una guía en su trabajo, indicando el camino más conveniente a seguir y la ubicación de los productos en sus respectivas góndolas.
- Ejecutar pruebas piloto con distintos *Shoppers*, con el objetivo de verificar una reducción en el tiempo de compra dada una determinada lista de productos.

1.3 Alcance

Dado que el estudiante estuvo trabajando en el Departamento de *Software & Cloud* en Zippedi, en esta sección se tiene como objetivo dar a conocer la cultura tecnológica y la metodología de trabajo para la realización de esta memoria, que corresponde a Shape Up.

Shape Up es una metodología ágil de trabajo utilizada en dicho departamento para la planificación, desarrollo e implementación de software hasta la fecha en que se realizó este trabajo. Shape Up está compuesta principalmente de tres etapas, las cuales siguen el orden representado en la Figura 5 que se explican a continuación.

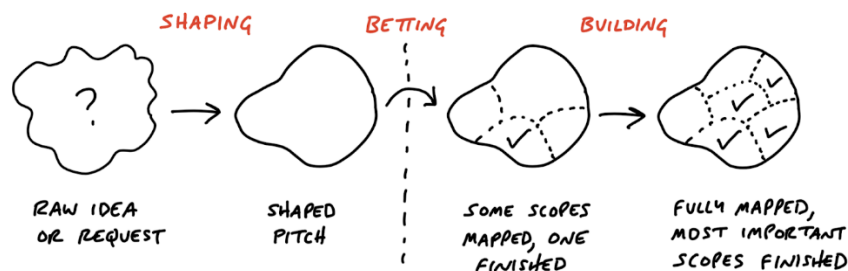


Figura 5: Etapas de Shape Up.

- 1) *Shaping*: En esta etapa se pasa de una idea a un proyecto, que posteriormente recibe el nombre de *pitch*. Para poder generar un *pitch* se suelen establecer:
 - a. La idea o el problema que se busca abordar.
 - b. El apetito, es decir, el tiempo que a destinar para el desarrollo de la solución.
 - c. La solución en base al problema previamente mencionado, aquí se expresan las ideas a tratar.
 - d. Los “*Rabbit Holes*” que son detalles importantes de mencionar, que pueden generar inconvenientes y/o retrasos.
- 2) *Betting*: En esta etapa se escogen los proyectos o *pitches* que serán elaboradas a lo largo del ciclo de trabajo.
- 3) *Bulding*: Esta es la etapa de elaboración o desarrollo de los *pitches* escogidos en la etapa anterior. Esta etapa, a su vez, está conformada de dos fases, desarrollo y *cooldown*. Ahora bien, dependiendo de la empresa esta etapa puede tener distintos tiempos o apetito. En el caso de Zippedi, y tal como se observa en la Figura 6, esta etapa consta de seis semanas de desarrollo y dos semanas de *cooldown*.



Figura 6: Etapas de un ciclo de desarrollo (o *Building*).

La subfase de desarrollo es el momento en que el equipo trabaja en la implementación del proyecto que haya escogido o seleccionado. Esta subfase tiene una reunión semanal de seguimiento (*weekly*) donde se dan a conocer los avances y/o las dificultades que hayan surgido. Ahora bien, este trabajo suele ser organizado por funcionalidades y/o estructuras, de esta forma se obtiene una integración vertical, y distintas partes del proyecto son desarrolladas de forma independiente a las que se les denominan *scopes*.

La subfase de desarrollo suele tener una etapa de desconocimiento, en donde se entiende el cómo hay que dividir el trabajo para formar dichos *scopes*, junto con las tareas a realizar para cada uno de estos, esta etapa puede entenderse como “subir la colina”. Una vez que

se llega “a la cima de la colina”, se tiene conocimiento de qué hacer y cómo hacerlo, por lo que es momento de realizar dichas tareas para así “bajar por la colina” y dar por finalizado el *scope* en cuestión (Figura 7).

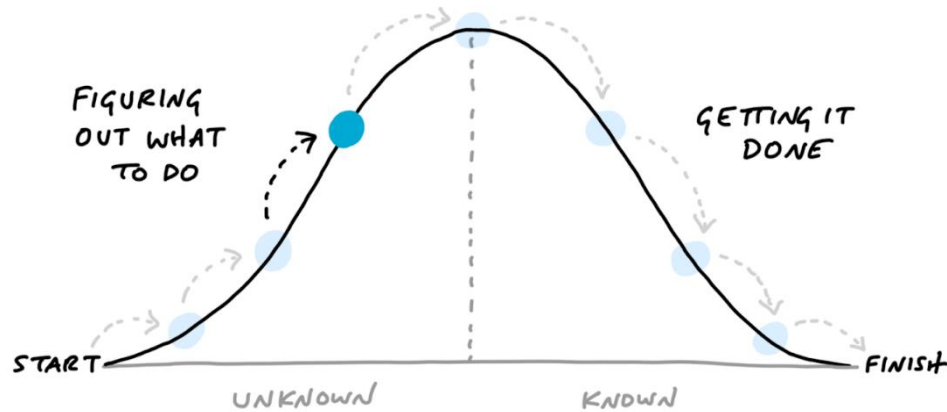


Figura 7: Ejemplo de una colina de *Shape Up*

La fase de *cooldown*, al final del ciclo, tiene como objetivo “bajar el ritmo”, por lo general solucionando bugs o similares junto con la planificación del siguiente ciclo (aquí vuelve a iniciar la etapa de *Shapping y Betting*)

Herramientas de Desarrollo

Dada la constante necesidad dentro de Zippedi de crear y desarrollar nuevas aplicaciones y/o herramientas, se tiene una estandarización de las tecnologías que utilizan. Es por esto que en este trabajo se utilizaron como lenguajes de programación Python para el desarrollo del optimizador de rutas y de la API; mientras que se utilizó React Native (Javascript) para el desarrollo de la aplicación móvil, junto a la biblioteca Redux para el manejo de estados.

CAPÍTULO 2: MARCO CONCEPTUAL

En este capítulo se describe el marco conceptual que actúa como fundamento teórico para el optimizador de rutas, el cual está compuesto por los siguientes temas a abordar: grafos, algoritmos de búsqueda de rutas, problema del vendedor viajero, procesamiento de imágenes y los algoritmos de dibujo de líneas.

Debido a que el objetivo es poder generar una ruta que guíe al usuario a través de un local, es necesario entender el espacio por el cual se va a mover esta persona, para esto surge la necesidad de poder representar los mapas de cada local y el espacio por donde es posible moverse; es aquí donde el procesamiento de imágenes y los algoritmos de dibujo de líneas entran en escena.

2.1. Grafos

Un grafo es un conjunto de vértices V y un conjunto de aristas E , las cuales son un conjunto de pares de vértices de la forma (u, v) tal que $u, v \in V$ [Enya Quetzalli, 2020].

Los grafos dirigidos, o también llamados digrafos, son aquellos que solo tienen uniones unidireccionales, es decir, que para la arista $e_1 = (u, v) \in E$ solo es transitable desde u hacia v pero no viceversa, por lo que debiese existir una arista $e_2 = (v, u) \in E$ que permita transitar desde v hacia u , pero $e_1 \neq e_2$. Ahora bien, un grafo no dirigido es aquel en que sus aristas son direccionales, es decir, que si una arista está conectada entre u y v entonces es posible recorrer en ambas direcciones dicha arista y se representa como una doble arista $(u, v), (v, u) \in E$. Tal como es posible observar en la Figura 8, los grafos dirigidos suelen representar sus aristas con flechas en su dirección correspondiente a diferencia de los grafos no dirigidos.

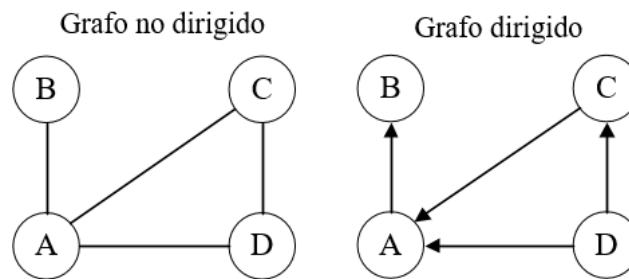


Figura 8: Ejemplo entre un grafo dirigido y uno no dirigido.

Un grafo *ponderado* corresponde a una tupla $G = (V, E, c)$ donde c corresponde al costo asociado a cada arista E siendo un número real y positivo: $c: E \rightarrow \mathbb{R}_0^+$. Este valor de costo suele ser utilizado para representar distintos tipos de problemas de la vida real como el

costo monetario, tiempo requerido o la distancia que hay de un lugar a otro. Ahora bien, para los grafos no dirigidos por cada arista $e_1 = (u, v)$ existe una arista $e_2 = (v, u)$ tal que $c(e_1) = c(e_2)$. Esto no suele pasar en los grafos dirigidos, a pesar de que es posible que se pueda dar el caso; por lo general, al igual que en muchos problemas de la vida real ocurre $c(e_1) \neq c(e_2)$, sucediendo en muchos casos en que la distancia de ida entre dos puntos es distinta a la distancia de vuelta. Lo mismo puede ocurrir con el tiempo o el costo monetario debido a la variación de precio del tag en autopistas dependiendo del momento del día.

Para poder representar un grafo en un computador se suele utilizar una matriz, la cual suele ser conocida como Matriz de Adyacencia de tamaño $V \times V$, en donde los elementos de la matriz están definidos como muestra la Figura 9. En esta se puede observar la representación de matriz de adyacencia. Ahora bien, en caso de que se tratara de un grafo ponderado se reemplaza el 1 por el valor del costo c que tenga la arista.

$$e_{ij} = \begin{cases} 1 \rightarrow (v_1, v_2) \in E \\ 0 \rightarrow (v_1, v_2) \notin E \end{cases}$$

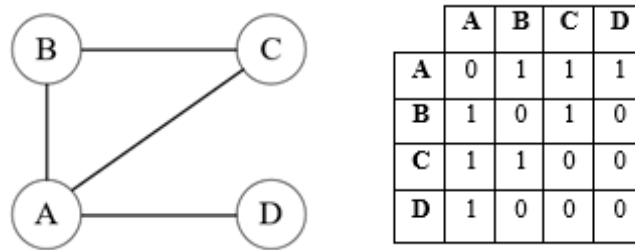


Figura 9: Grafo no dirigido y su matriz de adyacencia.

Este tipo de representación permite realizar operaciones matriciales para determinar varias propiedades del grafo y entre ellas el camino más corto para ir desde un vértice inicial a otro.

Un camino π corresponde a una secuencia de vértices de la forma $(v_1, v_2, \dots, v_{n-1}, v_n)$ los cuales pasar para ir desde el nodo v_1 hacia el nodo v_n , lo que para que un camino exista, necesariamente deben existir aristas que conecten el conjunto de vértices que conforman dicho camino. La longitud de un camino $\pi = (v_1, \dots, v_n)$ corresponde a la cantidad de aristas que se deben atravesar para llegar desde el vértice inicial hasta el final y suele ser descrita como $l(\pi)$. Por lo tanto, el costo asociado a un camino $c(\pi)$ sería la suma del costo de cada una de sus aristas. En un grafo es posible afirmar que un camino es más corto que otro en caso de que para un vértice inicial u y un vértice final v existe un camino π tal que no exista otro camino π' en donde $c(\pi) > c(\pi')$. Nótese que para los grafos no ponderados el costo asociado sería igual al largo de dicho camino, mientras que para un

grafo ponderado sería la suma del costo de todas las aristas de dicho camino. Si π es el camino más corto entonces es llamado la ruta óptima entre el vértice inicial y el final.

2.2 Algoritmos para planificación de rutas

Los problemas de planificación de rutas (*Pathfinding*) tienen una posición inicial y un objetivo o meta al cual llegar, los cuales suelen ser representados como vértices de un grafo. El objetivo de estos algoritmos es que, dado un mapa representado digitalmente, es necesario encontrar un camino π , también llamada solución, que sea óptimo entre un vértice inicial v_s (o punto de partida) y un vértice final v_g (meta). Si la solución resulta ser óptima entonces es el camino más corto para los vértices v_s y v_g .

Es posible que dependiendo del problema este tipo de algoritmos quizás consuman muchos recursos y/o no encuentren la solución óptima. Debido a ello, pueden darse casos en los que se encuentren caminos cercanos al óptimo, los cuales se denominan *subóptimos*, que son lo suficientemente buenos para una cantidad de recursos limitada como puede ser el tiempo de ejecución o la memoria. Cuando un grafo se conoce de antemano y se dispone de tiempo antes de resolver cualquier consulta, se puede tener una fase de pre-procesamiento [Bruno Marín B, 2022] que realiza una serie de cálculos, con el objetivo de generar información útil que pueda ser utilizada para agilizar la consulta de encontrar el camino óptimo entre el vértice inicial y el objetivo.

Por lo general, para poder utilizar este tipo de algoritmos, es necesario transformar o acotar el mapa con el que se va a trabajar a los llamados mapas de malla, malla o grilla, que terminan representando un grafo ponderado por donde se puede o no se puede pasar y el costo asociado a ello. Existen varios tipos de grillas según la cantidad de movimientos permitidos denominado tamaño del vecindario. Algunas de las grillas más comunes son la de cuatro y ocho vecinos, representadas en la Figura 10.

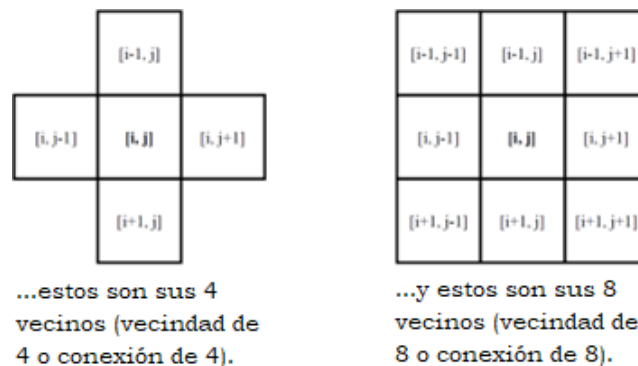


Figura 10: Movimientos permitidos para 4 y 8 vecinos.

2.2.1 Algoritmo de Dijkstra

El algoritmo de Dijkstra tiene como entrada un grafo $G = (V, E)$ y un vértice inicial llamado $s \in V$. Este algoritmo entrega el camino más corto entre s y el resto de los vértices del grafo a pesar de que muy probablemente solo se esté interesado en el camino más corto entre el vértice s y un vértice objetivo. El algoritmo parte desde el vértice s y evalúa todos sus vértices adyacentes buscando el de menor costo, el cual es tomado como un punto intermedio y vuelve a revisar los vértices adyacentes analizando si es posible llegar al resto de vértices con a un costo menor; en caso de ser así se actualiza el costo y se repite el proceso eligiendo al vértice con un costo menor. Este proceso se repite hasta que el vértice no utilizado más cercano resulte ser el vértice inicial s [Jhosimar G. Arias, 2012].

2.2.2 Algoritmo de búsqueda A*

Como el objetivo es encontrar solamente el camino más corto entre dos vértices, parece una pérdida de tiempo y recursos procesar todos los caminos desde un vértice a todos los nodos como lo hace el algoritmo de Dijkstra [Cuevas, 2013], por lo que parece preciso explorar otras alternativas, como A*.

A* es un algoritmo de búsqueda para encontrar el camino de un punto a otro en un mapa de malla o grafo de búsqueda. Este algoritmo es una optimización de Dijkstra utilizando una heurística, o mejor dicho una función heurística $h(v)$, de forma que utiliza la función $f(v) = g(v) + h(v)$ siendo la función $f(v)$ el costo calculado por cada vértice adyacente. En la Figura 11 es posible observar un ejemplo de su funcionamiento sobre una grilla de 8 vecinos con obstáculos.

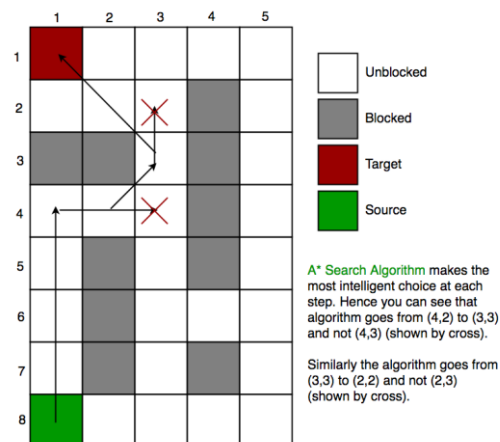


Figura 11: Algoritmo A* sobre un grafo de grilla.

Dentro de sus características, el algoritmo A* es considerado un método completo de búsqueda, es decir, termina encontrando la solución óptima cuando ésta exista para cualquier tipo de grafo. Además, no solo encuentra una solución, sino que la solución encontrada es óptima siempre y cuando se cumpla:

$$\forall v \in V; h(v) \leq h^*(v)$$

En otras palabras, si la función heurística que estima la distancia a la meta no supera nunca la distancia real existente, entonces A* garantiza encontrar la solución óptima. Tal como en el artículo [Krishnaswamy, 2009] en donde se demuestra que hay situaciones en las que A* no rinde de la manera esperada en comparación a otros algoritmos; esto se debe principalmente a que la eficiencia de A* depende de la heurística $h(v)$, debido a que esta es directamente proporcional a la similitud entre el valor de $h(v)$ y el costo real entre el vértice inicial y el objetivo.

Heurísticas para la Búsqueda de caminos

Una heurística estima el costo entre un vértice inicial y el objetivo. Por lo tanto, dependiendo de la heurística que se utilice, se puede ayudar o perjudicar la búsqueda de un posible camino. Por lo que la elección de qué heurística ocupar dependerá directamente del problema y la representación del espacio de búsqueda.

- Distancia de Euclidiana: La distancia entre dos puntos se mide como la longitud del segmento que los une, la cual se deduce a partir del teorema de Pitágoras, para los puntos $A(a_1, a_2, \dots, a_n)$ y $B(b_1, b_2, \dots, b_n)$ donde $a_1, a_2, \dots, a_n$ y $b_1, b_2, \dots, b_n$ son las coordenadas en cada una de sus dimensiones respectivamente, la ecuación general es:

$$|A - B| = \sqrt{\sum_{i=1}^n (|a_i - b_i|)^2}$$

Para un espacio bidimensional, la distancia euclidiana entre los puntos P_1 y P_2 de coordenadas (x_1, y_1) y (x_2, y_2) respectivamente, sería:

$$|P_1 - P_2| = \sqrt{\sum_{i=1}^n (|a_i - b_i|)^2} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Distancia Manhattan: La distancia entre dos puntos corresponde a la suma de las distancias absolutas entre cada una de las coordenadas de A y B, por lo que su ecuación sería de la forma:

$$|A - B| = \sum_{i=1}^d |a_i - b_i|$$

Como ejemplo, en la Figura 12, si los cuadros blancos representaran edificios, la distancia euclidiana (línea verde) sería infactible, por lo que se estaría obligado a bordearlos, tal como la línea azul, amarilla y roja, que representan la distancia de Manhattan mínima entre estos puntos a pesar de ser distintas rutas [eloviparo, 2018].

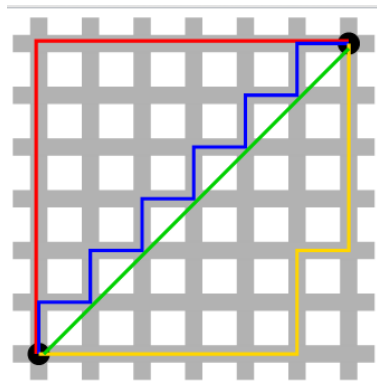


Figura 12: Ejemplo distancia de Manhattan.

Algoritmo D*

El algoritmo D* es un algoritmo de búsqueda incremental para encontrar el camino más corto entre dos puntos. Este algoritmo combina ideas de A* como es la definición de las funciones f y h , además de crear una lista abierta y otra cerrada.

Este algoritmo es utilizado principalmente en problemas de planificación de ruta basada en suposición, por lo que son ampliamente utilizados para robots móviles y vehículos autónomos, en donde el terreno suele ser desconocido por lo que se descubren nuevos obstáculos a medida que avanza.

2.3 Problema del vendedor viajero

El Problema del vendedor viajero, tiene como objetivo encontrar la ruta más eficiente para un vendedor que debe pasar por todos sus destinos. Dicho de esta forma parece un

problema sencillo de resolver si se considera un único destino puesto que pueden existir múltiples formas de llegar; pero elegir la más eficiente estaría directamente relacionada a escoger la ruta más corta. Sin embargo, a medida que aumenta la cantidad de destinos, el cálculo se vuelve mucho más complejo.

Por lo general, este problema suele ser representado como un grafo, en donde los nodos del grafo representan los destinos y las aristas que conectan a los nodos los caminos que los unen, por lo que los pesos de estas aristas representan el costo asociado a recorrer los caminos, pudiendo ser valor monetario, tiempo, distancia, etc. En la teoría se esperaría que los grafos fuesen simétricos, es decir, que el costo de ir de A hacia B sea el mismo que ir de B hacia A, aunque en la práctica no suele ocurrir muy seguido. Por lo tanto, la cantidad de rutas posibles para una cantidad N de destinos estaría dada por la ecuación $(N - 1)!$. Por ejemplo, para un vendedor con 10 destinos se tendría aproximadamente 362.880 posibles rutas para elegir. Mientras que, por otro lado, en caso de que el problema sea simétrico la cantidad de rutas posibles disminuye a la mitad, es decir: $\frac{(N-1)!}{2}$

En ambos casos, sin tener una gran cantidad de destinos el número de rutas posible puede ser muy grande para que un programa sea capaz de manejarlas en un tiempo razonable. Por esta razón este problema es clasificado como “*NP-Hard*” [Garey y Johnson, 1979], es decir que no existe un algoritmo que sea capaz de resolverlo en un tiempo polinomial.

Dada la naturaleza del problema, se han desarrollado ciertas metodologías y heurísticas para calcular soluciones óptimas que quizás no den como resultado la mejor solución (conocida como óptimo global), pero sí soluciones de buena calidad (óptimos locales) en un tiempo reducido de ejecución. Se llaman de esa forma porque se considera que f es la función que calcula la distancia total del recorrido, la cual depende del orden en que se visiten cada una de las ciudades, por lo tanto son las variables de la función. Entonces para distintas configuraciones de estas variables, se obtienen distintos valores de la función f por lo que el conjunto de todas estas posibles soluciones corresponde al “espacio de búsqueda”. Hay que recordar que el objetivo principal es minimizar el valor de f por lo que mientras menor sea el valor de f se tendrán soluciones de mejor “calidad”, por lo que agregando la dimensión de calidad al espacio de búsqueda de la función f daría como resultado la Figura 13 en donde se puede notar montes con distintas alturas en base a la calidad de la solución. Se le llama óptimo local a la cima de uno de estos montes, ya que es la solución de mayor calidad entre todos sus “vecinos”, pero no necesariamente es la solución de mayor calidad para todo el espacio de búsqueda. Mientras que el óptimo global correspondería a la cima del monte más alto de todo el espacio de búsqueda.

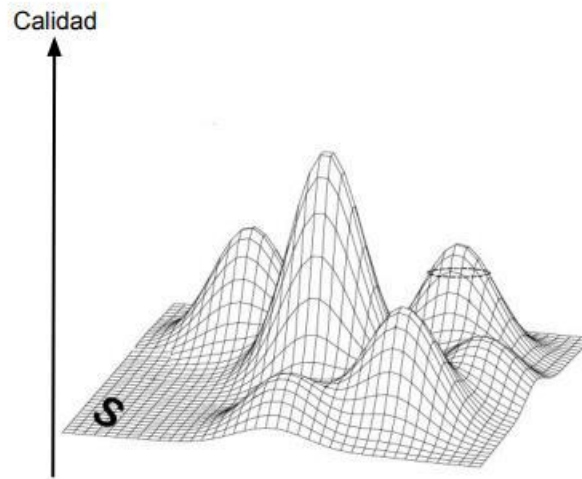


Figura 13:Espacio de búsqueda de la función objetivo.

Ahora bien, existen distintos algoritmos para moverse por este espacio de búsqueda con el objetivo de encontrar el mejor óptimo local, siguiendo los siguientes aspectos:

1. Diversificar: el algoritmo “visita” distintas regiones para identificar secciones del espacio de búsqueda que puedan resultar interesante.
2. Intensificar: el algoritmo se centra en encontrar soluciones de la mejor calidad posible dentro de la región actual del espacio de búsqueda.

En otras palabras, es posible entender la diversificación como ir moviéndose entre estos montes en la búsqueda de uno más “interesante”, mientras que la intensificación vendría siendo el escalar el monte en donde se encuentra el algoritmo con la intención de llegar a la cima u óptimo local.

Los algoritmos constructivos son aquellos que generan soluciones comenzando desde una solución parcialmente vacía, es decir que suelen requerir como definición previa un punto de partida junto a una función miope, siendo esta última la encargada de la toma de decisiones e ir agregando componentes a la solución parcial hasta estar completa. De esta forma es probable que este tipo de algoritmos no encuentre la solución óptima global debido a las soluciones que va tomando a lo largo del camino, pero sí genera una solución candidata. Estas soluciones candidatas se suelen utilizar como punto de inicio para los algoritmos reparadores, los cuales van modificando esta solución candidata con la intención de diversificar o intensificar dependiendo del algoritmo, y así mejorar la calidad de esta solución candidata.

2.3.1 Vecino más cercano

El algoritmo del Vecino más cercano es una de las heurísticas más comúnmente utilizadas para el problema del vendedor viajero, ya que corresponde a un algoritmo constructivo mediante el cual es posible generar una solución inicial en un tiempo reducido. Este método, tal como dice su nombre, consiste en que, una vez que se tenga establecido un nodo de partida, se debe evaluar y seleccionar su vecino más cercano, el cual pasaría a ser el nodo de partida, y así repetir el proceso hasta que no queden más nodos por visitar.

De esta forma, no asegura el óptimo global pero sí una solución aceptable y/o inicial, tal como es posible observar en la Figura 14, en donde si se comienza desde el nodo de más a la izquierda y ϵ es un número positivo mucho menor a 1 y se utiliza esta heurística, poco a poco se recorrerá el camino marcado en rojo, dando como resultado un valor de 10, aunque, por otro lado, si se elige constantemente el vecino de la derecha daría como resultado $8 + \epsilon$.

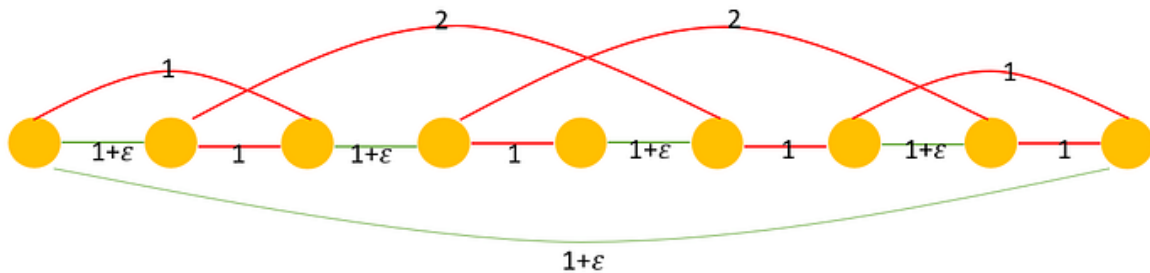


Figura 14: Ejemplo utilizando la heurística del vecino más cercano

2.3.2 Búsqueda Local

La búsqueda local, tal como dice su nombre, es un tipo de metaheurística de búsqueda que establece ciertas estrategias para recorrer el espacio de búsqueda transformando, de forma iterativa, una solución inicial previamente establecida [Melián, Moreno y Vega, 2003]. Dentro de estas estrategias pueden encontrarse el intercambiar el orden entre dos ciudades adyacentes, y el insertar una ciudad en otra posición, entre otros, siempre y cuando este movimiento mejore la calidad de la solución, ya que en caso contrario se le descarta. Dentro de este tipo de búsquedas locales se encuentra la metaheurística *Greedy Descent*, la cual se basa en escoger el movimiento vecino que produce una solución de mayor calidad (intensificación), aunque de esta forma es probable quedar atrapado en un óptimo local sin posibilidad de escapar de él. De manera similar, la Búsqueda Local Guiada combina la búsqueda local con una función de evaluación encargada de salir de un óptimo local

“modificando” la función objetivo considerando información adicional (usualmente llamadas sanciones) como pueden ser la distancia ya recorrida, estimaciones de distancias y/o restricciones de carga. De esta forma diversificarse a lo largo del espacio de búsqueda evitando quedar atrapados en locales subóptimos [Melián, Moreno y Vega, 2003].

Existen diversas metaheurísticas utilizadas principalmente para escapar de óptimos locales como son el caso del Recocido Simulado, el cual controla posibles movimientos de empeoramiento de la solución mediante un criterio de aceptación estocástico, siendo en este caso la probabilidad de aceptación una función exponencial de empeoramiento producido, en donde en un comienzo la probabilidad de aceptar movimientos que empeoren la solución es bastante alta y mientras avanza en su búsqueda se vuelve más selectivo con los movimientos que realiza. Por otro lado, existe la metaheurística de Búsqueda Tabú, la cual combina estrategias de búsqueda local con estrategias de diversificación denominándose una metaheurística con memoria, ya que utiliza información sobre el recorrido realizado llamada “Lista Tabú” para evitar concentrarse en un vecindario con soluciones muy parecidas durante un periodo de tiempo. De esta forma es posible explorar soluciones vecinas incluso si empeoran la calidad de la solución, todo con tal de no verse atrapada en óptimos locales [Melián, Moreno y Vega, 2003].

2.4. Procesamiento de imágenes

En el contexto de este trabajo, el procesamiento de imágenes cumple un rol fundamental para la extracción de información necesaria al momento de generar la ruta a seguir por el usuario. Por lo tanto, se debe entender el formato en que vienen estas imágenes para poder extraer información de ellas.

Los mapas de los locales provistos por Zippedi vienen en imágenes en formato PGM o *Portable Graymap Format*, el cual es una forma para almacenar imágenes en escalas de grises. En este tipo de formatos los píxeles de la imagen se representan en valores numéricos que van desde el 0 hasta el 255, que indica la intensidad máxima de gris o blanco en la imagen, es decir que el 0 representa al negro, por lo tanto, el 255 representa el color blanco, mientras que los valores entre medio representan distintas tonalidades de gris.

2.4.1. Umbral

El umbral de imagen, o *threshold*, en el procesamiento de las imágenes es un tipo de segmentación de imágenes que aísla objetos al convertir imágenes en escala de grises en imágenes binarias [MathWorks, 2020].

El umbral funciona delimitando un valor límite para cada píxel, es decir, cada píxel que se encuentre bajo el umbral tendrá como nuevo valor 0 (negro), mientras que si se encuentra sobre el umbral su nuevo valor será de 255 (blanco) [MathWorks, 2020], tal como es posible observar en la Figura 15.

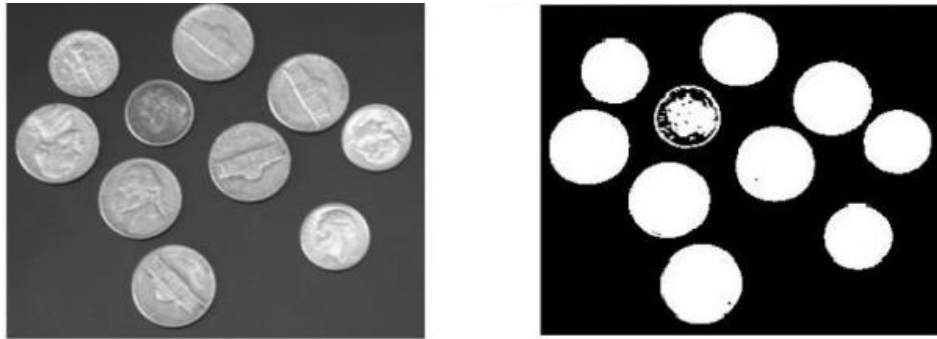


Figura 15: Ejemplo de aplicar un umbral a una imagen.

2.4.2. Operaciones Morfológicas

Las operaciones morfológicas son un conjunto de operaciones utilizadas para el procesamiento de imágenes basadas en formas geométricas. Estas operaciones aplican dichas formas geométricas, también denominadas elementos estructurados (o kernel), que no son más que una matriz que define la forma y el tamaño de la vecindad utilizada en el procesamiento de cada píxel. En este proceso el valor de cada píxel en la imagen de salida se basa en una comparación del píxel correspondiente de la imagen de entrada con sus vecinos.

Las dos operaciones morfológicas más básicas son la dilatación y la erosión. De la combinación de estas operaciones nacen operaciones utilizadas para el procesamiento de imágenes; para el caso de este documento se tendrá en consideración las operaciones de apertura y cerrado.

2.4.2.1. Dilatación

La dilatación añade píxeles a los bordes de los objetos de una imagen; en otras palabras, los hace más grandes, haciéndolos más visibles, y rellenando pequeños huecos que puedan encontrarse dentro de los objetos [MathWorks, 2020], tal como es posible observar en la Figura 16.

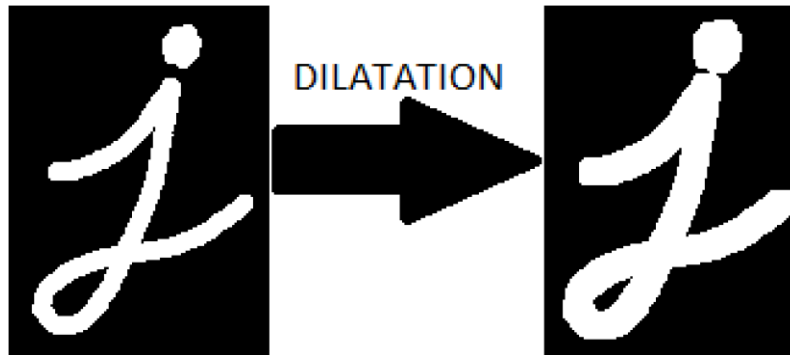


Figura 16: Ejemplo de una operación de dilatación.

La regla utilizada para la dilatación consiste en que el valor del píxel de salida es el valor máximo de todos los píxeles de su vecindario. Por lo que, para una imagen binaria, un píxel se establecerá como 1 si cualquiera de sus vecinos tiene valor 1 (ver

Figura 17).

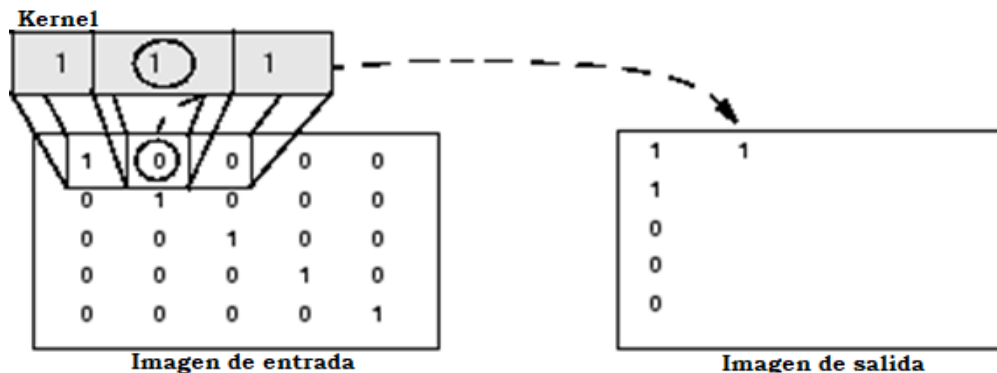


Figura 17: Representación binaria de una operación de dilatación.

2.4.2.2 Erosión

La erosión elimina píxeles del borde de objetos presentes en la imagen pudiendo eliminar píxeles aislados o líneas muy finas dependiendo del elemento estructurado, dejando como resultado solo los objetos más sustantivos, los cuales pueden resultar más pequeños y/o delgados [MathWorks, 2020], como ocurre en el caso de la Figura 18, en donde desaparecen los pequeños puntos blancos en la imagen, además de disminuir el grosor de la letra representada.

La regla utilizada para la erosión consiste en que el valor del píxel de salida es el valor mínimo de todos los píxeles de su vecindario, por lo que, para una imagen binaria, un píxel

se establece como 0 si cualquiera de los píxeles del entorno tiene el valor 0 (ver

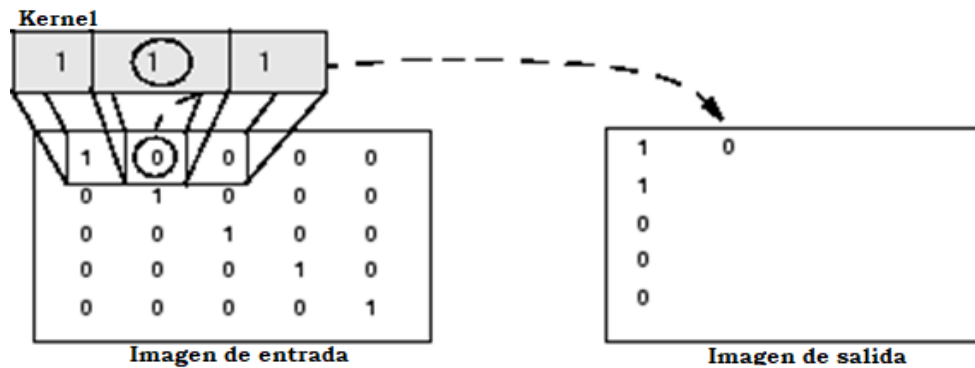


Figura 19).

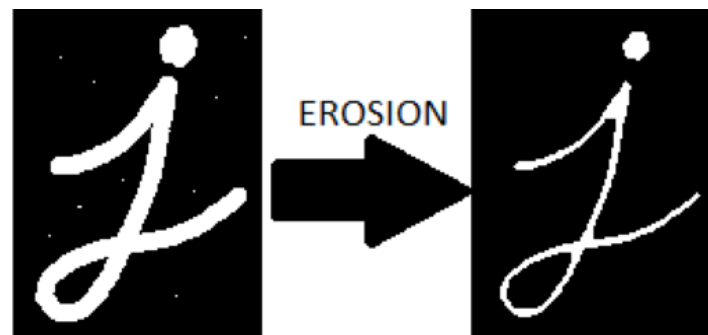


Figura 18: Ejemplo de una operación de erosión.

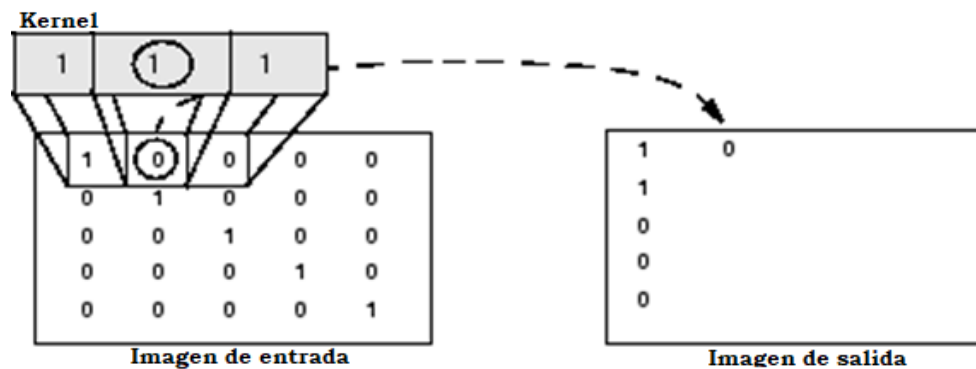


Figura 19: Representación binaria de una operación de erosión.

2.4.2.3 Apertura

Una operación de apertura es el resultado de realizar una operación de erosión seguida de una operación de dilatación, utilizando el mismo elemento estructurante para ambas operaciones. Se sigue este orden debido a que con la erosión se busca eliminar objetos pequeños como píxeles aislados y/o líneas delgadas que puedan ser consideradas como

ruido, y luego con la dilatación se busca conservar la forma y tamaño de los objetos más grandes [MathWorks, 2020], tal como se ejemplifica en la Figura 20, en donde desaparecen los contornos más delgados, dejando solo los objetos de mayor grosor.

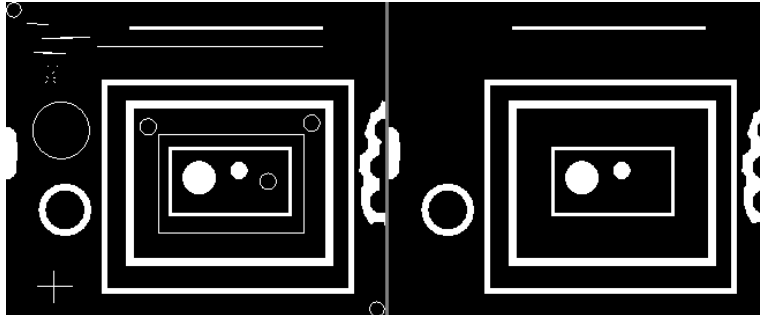


Figura 20: Ejemplo de apertura sobre una imagen binaria.

2.4.2.4 Cerrado

Una operación de cerrado es el resultado de realizar una operación de dilatación seguida de una operación de erosión, utilizando el mismo elemento estructurante para ambas operaciones. Se sigue este orden ya que debido a la operación de dilatación se rellenan los huecos más pequeños que puedan encontrarse en la imagen, ensanchando los bordes de los objetos, los cuales son corregidos posteriormente con la función de erosión, conservando así la forma y tamaño de los huecos y objetos más grandes al interior de la imagen [MathWorks, 2020], como es el caso de la Figura 21, en donde los ligeros huecos dentro de la imagen simplemente se cerraron sin afectar el contorno de dicha figura.

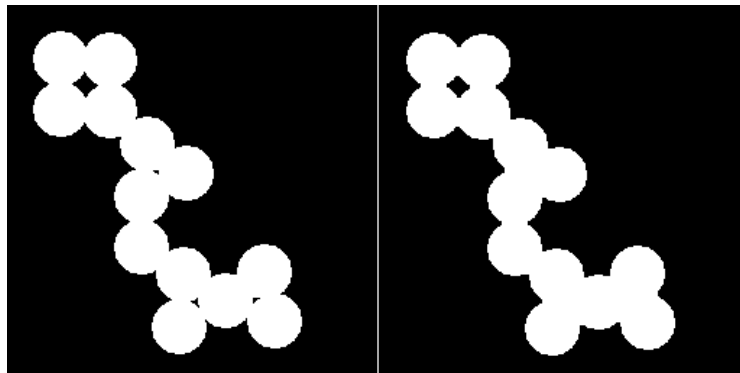


Figura 21: Ejemplo de cierre sobre una imagen binaria.

2.4.2.5 Encontrar perímetro

Para encontrar el perímetro de los objetos dentro de una imagen binaria se sigue como regla que un píxel es parte del perímetro si es distinto de cero y está conectado al menos a

un píxel cuyo valor es cero. Como resultado de esta regla cualquier hueco interior que posea un objeto será considerado como parte del perímetro del objeto [MathWorks, 2020] (Figura 22).

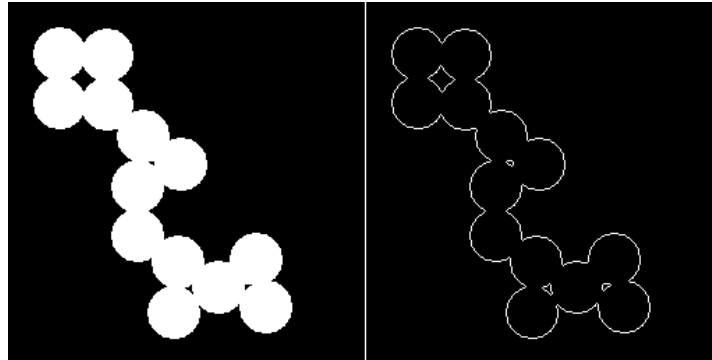


Figura 22: Ejemplo de la búsqueda de perímetro sobre una imagen binaria.

2.4.2.6 Transformada de Sombrero de Copa

Una operación de la transformada de sombrero de copa (*Top Hat*) es el resultado de realizar una operación de apertura sobre una imagen, y luego restar la imagen abierta con la imagen original. Esta operación se suele utilizar para mejorar el contraste en una imagen en escala de grises con una iluminación no uniforme, tal como en la Figura 23.

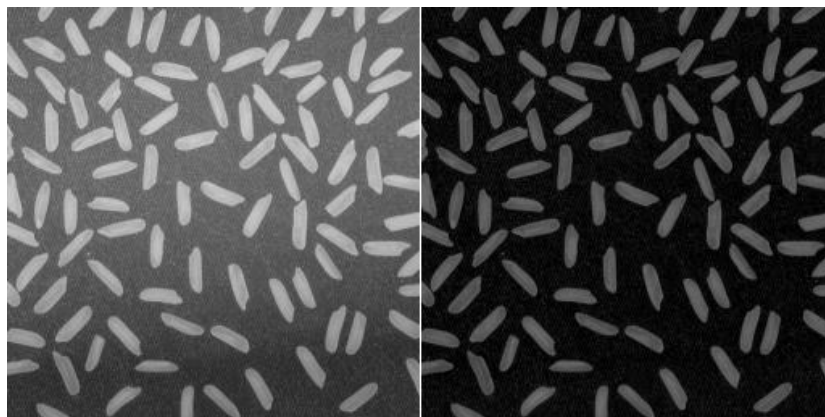


Figura 23: Ejemplo de una transformada de Sombrero de Copa.

2.4.2.7 Transformada de Sombrero de Copa Invertido

Una operación de la transformada de sombrero de copa invertido (*Bottom Hat*) es el resultado de realizar una operación de cierre sobre una imagen y luego restar la imagen abierta con la imagen original. Esta operación aísla los píxeles más oscuros de su vecindad,

por lo que suele ser utilizada para encontrar depresiones de intensidad en una imagen en escala de grises como en la Figura 24.



Figura 24: Ejemplo de una transformada de Sombrero de Copa Invertido.

2.5 Algoritmos para dibujo de líneas en dispositivos digitales

Los algoritmos de dibujo de líneas en dispositivos digitales son los cimientos de la computación gráfica, desempeñando un papel crucial para la creación de gráficos, simulaciones y demás. Esto se debe a que se busca representar algo tan abstracto y continuo como una línea dibujada sobre un papel (y/o similar) en un ambiente discreto como son los píxeles de una pantalla.

2.5.1 Algoritmo DDA

El algoritmo DDA, o analizador diferencial digital, es un algoritmo de línea de conversión de rastreo que se basa en el cálculo de los incrementos de las coordenadas de x o de y según corresponda [Graficación, 2017]. Se efectúa un muestreo de la línea en intervalos unitarios en una coordenada, y se determinan los valores enteros correspondientes más próximos a la trayectoria de la línea para la otra coordenada.

2.5.2 Algoritmo de Bresenham

Es un algoritmo veloz y preciso para el dibujado de líneas debido a que solo realiza cálculos enteros. Este algoritmo determina los puntos en un mapa de bits de n -dimensiones que deben ser trazados (o rellenados) con el fin de formar una aproximación a una línea recta entre dos puntos [Acosta, 2011].

Se parte de la base que las coordenadas de los píxeles en una imagen son coordenadas enteras y que se tiene conocimiento de los extremos del segmento que forma la línea, siendo estas coordenadas (x_0, y_0) y (x_4, y_3) , respectivamente. El algoritmo se basa en probar iterativamente el signo de un parámetro entero, cuyo valor es proporcional a la diferencia entre las separaciones de las dos posiciones de píxel de la trayectoria real de la línea (Figura 25)[Acosta, 2011].

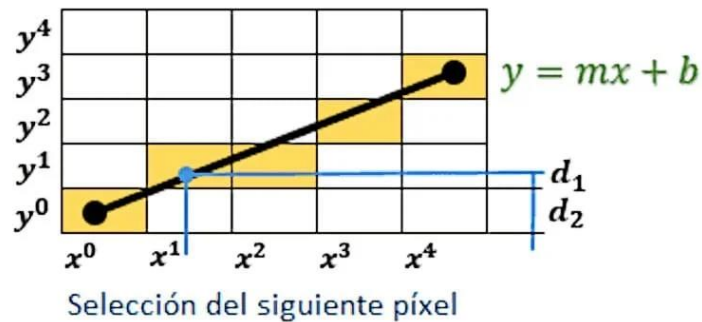


Figura 25: Ejemplo del algoritmo de Bresenham.

El ejemplo anterior corresponde a su versión más básica, el cual funciona para líneas dentro del primer octante (0° a 45°). Sin embargo, el algoritmo se puede ampliar para cubrir el resto de los octantes [Bresenham, 1965].

CAPÍTULO 3: PROPUESTA DE SOLUCION

En este capítulo se explica la manera en la que se desarrolló este trabajo utilizando la metodología de trabajo *Shape Up*. Además, se proporciona una visión simplificada del proceso de generación y almacenamiento de datos necesarios en este trabajo llevado a cabo por el robot de Zippedi.

Dada la duración de los ciclos de *Shape Up*, el desarrollo y la implementación de este trabajo se dividió en dos ciclos, cada uno con una duración de 8 semanas y sin *cooldown*, que corresponden a: *Pick-Up Routing Algorithm*, y *Pick-Up App*.

3.1 Primer Ciclo: Pick-Up Routing Algorithm

Este ciclo tuvo como tareas principales:

- 1) Desarrollar un algoritmo que determinara una ruta de *Pick-Up* para una lista de productos
- 2) Construir un microservicio escalable y de baja latencia capaz de soportar la demanda de múltiples solicitudes
- 3) Desarrollar el *Backend* necesario para la recopilación de información para el periodo de pruebas.

Primero, es necesario entender la forma de modelar el problema que se intenta resolver, ya que el primer objetivo específico es encontrar la ruta más eficiente para que un agente (*Shopper*) pueda recoger todos los productos que necesita para completar su canasta de compras en el menor tiempo posible.

Es posible modelar este problema como el problema del vendedor viajero; de esta forma, las “ciudades” u “objetivos” por los que debe pasar el vendedor viajero corresponden a los productos ubicados en distintas secciones a lo largo del local. Ahora bien, hay que tener en consideración dos restricciones, ya que por lo general gran parte de los locales suele tener una entrada para todos los clientes y la “salida” corresponde a la sección de cajas (siendo aquí donde termina el proceso de *Pick-Up*). En este sentido, cada una de las posibles rutas que se generen tendrán siempre como punto inicial la entrada y como último “objetivo” visitar la sección de cajas.

Teniendo claridad del tipo de problema a resolver se levantaron los siguientes *Scopes*:

- **Recolección de Datos:** Dada una tienda y una lista de productos, es necesario recolectar el mapa y la información más reciente que se tenga sobre los productos que conforman la lista.
- **Limpieza del Mapa:** Procesar el mapa generado por el LIDAR¹, con el objetivo de eliminar ruido y/o errores que pudiese tener.
- **Calculo los nodos:** Calcular los nodos utilizando los pasillos y las intersecciones en los mapas previamente procesados.
- **Cálculo de las matrices de distancia:** Calcular los vecinos de cada nodo junto a su distancia para almacenarlos en *Google Cloud Storage* [Cloud Storage].
- **Cálculo de la ruta:** Utilizando la matriz de distancia previamente calculada y la posición de los productos calcular la ruta a seguir.
- **Levantar la API:** Encapsular las funciones generadas en una API de Flask.

Para obtener el mapa de un local junto con sus correspondientes *Layouts*, es necesario solicitarlos a otro microservicio de Zippedi llamado *Overseer*, ya que este microservicio provee la información más reciente del local solicitado, junto con el *PATH* requerido para descargar el mapa desde *Google Cloud Storage* [Cloud Storage]. Mientras que la posición de los productos se almacena en una base de datos, de aquí se obtiene la posición relativa de cada producto en base al recorrido que hace el robot en cada pasillo junto con la última vez que dicho producto fue visto. Gracias a esto, se descartan todos los productos que no han sido vistos en un periodo máximo de siete días, con la intención de no dar información desactualizada al usuario en su búsqueda.

El siguiente paso corresponde al procesamiento del mapa de cada local, puesto que cada mapa en escala de grises que se obtiene gracias al láser LIDAR¹ que tiene cada robot. No obstante, los mapas no están exentos de errores y/o ruido por lo que se utilizan distintas funciones morfológicas para cerrar los objetos en blanco que en gran medida representan góndolas, muros u obstáculos a lo largo del mapa. Para esto, al mapa en escala de grises se le aplica un umbral para generar una imagen binaria y luego se le realizan una secuencia de funciones morfológicas, dando como resultados las imágenes binaria y preprocesada ejemplificadas en la Figura 26.

Si bien es cierto que el mapa preprocesado pierde cierta información como algunos obstáculos, estos no afectan en mayor medida al usuario, ya que en la escala que se trabaja

¹ LIDAR (Light Detection and Ranging): tecnología de detección remota que utiliza pulsos de luz para medir distancias y generar mapas del entorno.

el mapa son objetos que se pueden considerar como obstáculos en los pasillos o secciones de las cuales no se tiene información de los productos. Un claro ejemplo de esto puede ser la sección de ropas, verduras y/o la sección de congelados de la carne, las cuales suelen estar al nivel del piso, provocando que el robot no sea capaz de tener información sobre estos productos. Por lo tanto, son secciones descartadas por el momento.

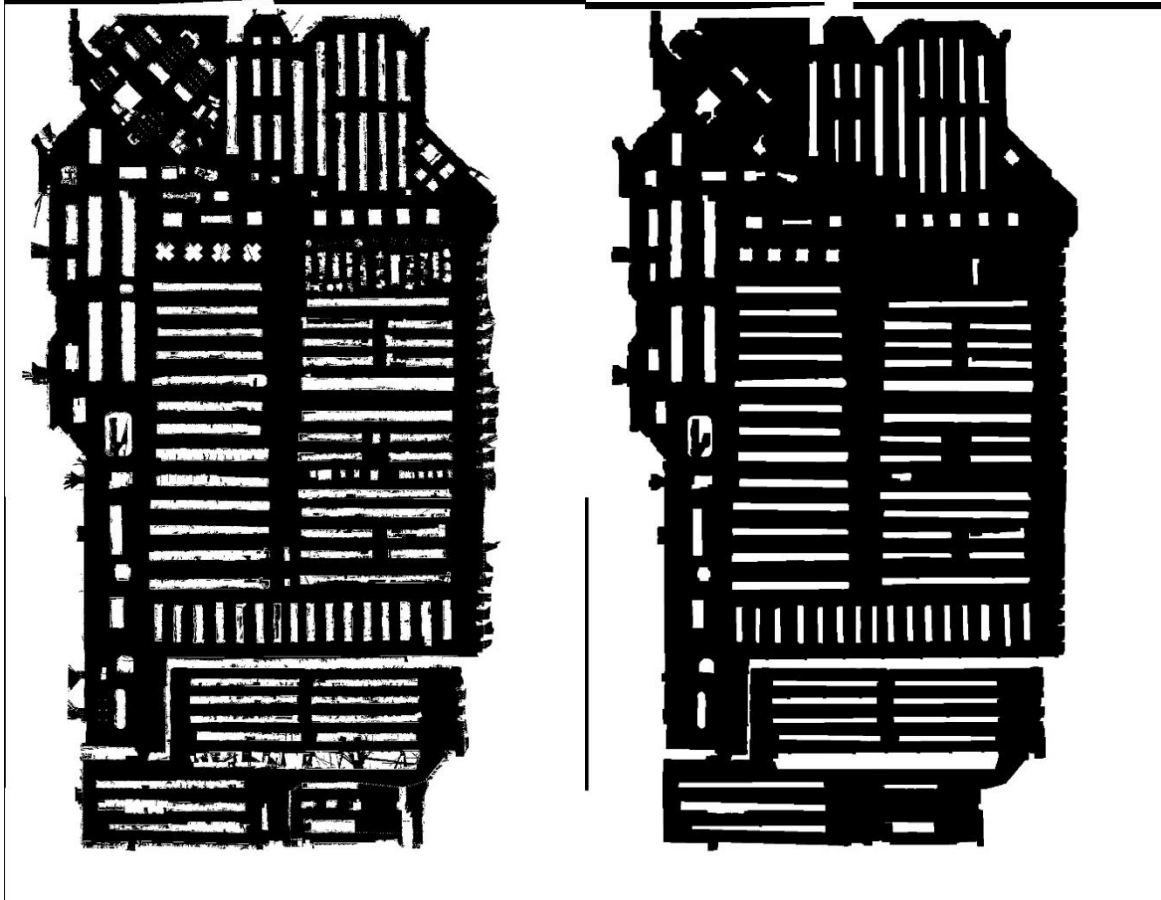


Figura 26: Mapa de las instalaciones dentro del Jumbo Kennedy, antes y después de su preprocesamiento.

Dicha tarea se denomina preprocesamiento porque el verdadero procesamiento del mapa corresponde al siguiente *scope*, que tiene como objetivo el calcular las entradas, salidas e intersecciones entre pasillos, los cuales se denominaron nodos de navegación. Estos nodos de navegación, tal como dice su nombre, permitirán moverse por el mapa sin tener que calcular una gran cantidad de nodos, como puede ser el caso al usar mallas de navegación, las cuales suelen ser muy utilizadas para la generación de agentes autónomos, como la gran mayoría de los *bots* en los videojuegos.

Como se busca obtener las entradas y salidas de los pasillos, se decidió invertir el mapa binario para facilitar el funcionamiento de las funciones morfológicas para encontrar rectángulos horizontales y verticales, los cuales representarán a cada pasillo de un supermercado, tal como se muestra en la Figura 27.

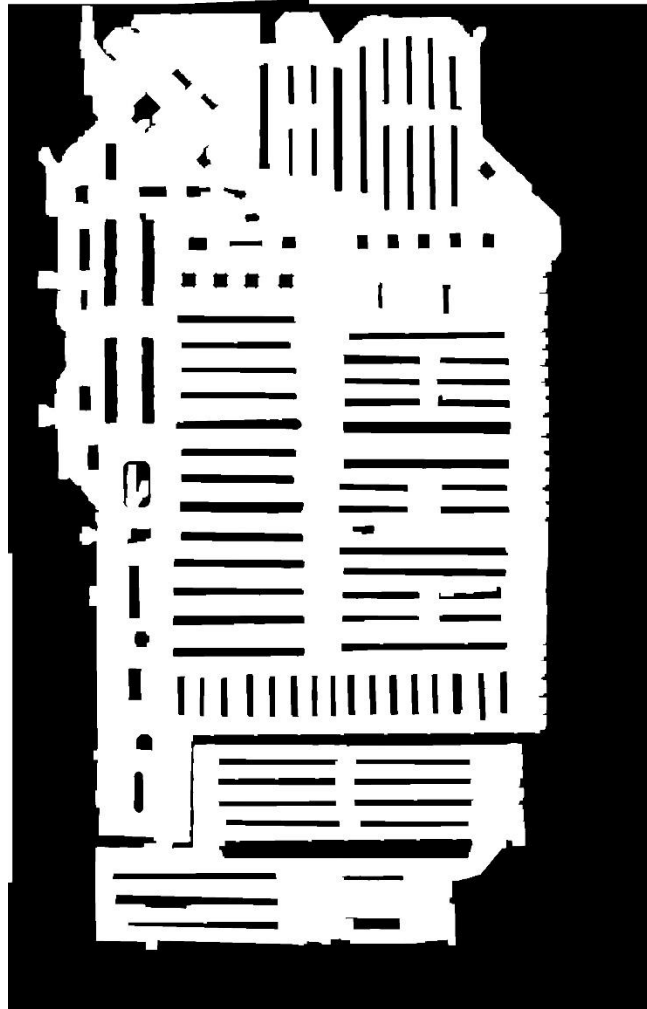


Figura 27: Mapa binario inverso.

Una vez generados los rectángulos verticales y horizontales, se realiza una operación AND a nivel de bits, para así obtener las intersecciones entre los pasillos verticales y horizontales. Por último, al resultado de esta operación se le calcula el centroide, dando como resultado, a su vez, la primera ronda de nodos de navegación, representados en la Figura 28.

La segunda manera para obtener nodos de navegación, y al mismo tiempo poder conocer el nombre de los pasillos, fue utilizando la información de los *Layouts* de cada tienda proporcionada por Zippedi, representados en forma de los *Bounding Boxes* de la Figura 29. Los *Layouts*, por su parte, indican qué sectores suelen ser escaneados en cada sección por

el robot, mientras que los sectores sin *Bounding Box* no son tomados en cuenta para el escaneo de productos.

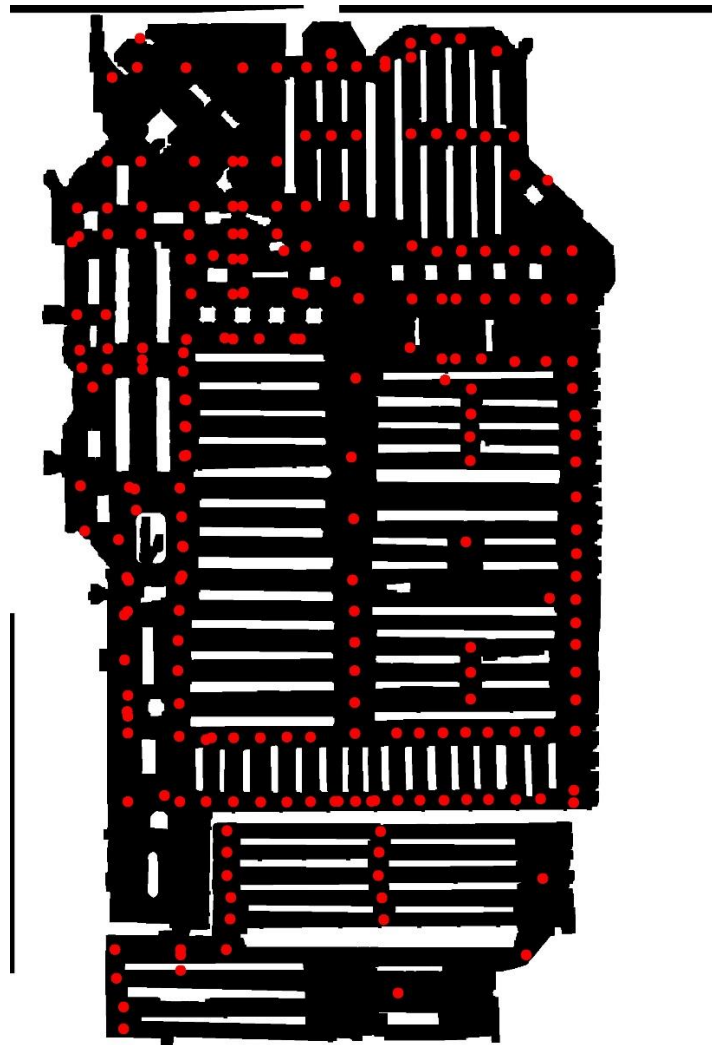


Figura 28:Nodos de navegación.

A cada *Layout* se le calcula el centro de los bordes, y en caso de encontrarse en un objeto o caer dentro de otro *Layout*, se descarta. De esta forma, se busca complementar los nodos de navegación obtenidos previamente para volver transitables todos los pasillos que suelen ser escaneados con regularidad.

Una vez calculados todos los nodos de procesamiento, se continúa con el siguiente *scope*, en donde se calculan los nodos vecinos entre sí utilizando el Algoritmo de Bresenham con el objetivo de verificar si existe una línea recta entre dos nodos sin que haya un objeto de

por medio, es decir que es posible moverse en línea recta entre estos nodos sin obstáculos de por medio. Finalmente, se calcula la matriz de distancias de cada uno de estos nodos, para así dar por terminado el proceso de procesamiento del mapa.



Figura 29: Mapa de las instalaciones dentro del Jumbo Kennedy con sus *Layouts* dibujadas.

Dado que tanto el mapa como los *Layouts* de un supermercado no suelen tener grandes variaciones, se decidió almacenar la matriz de distancias para estos nodos de navegación (o matriz estática) en *Google Cloud Storage* y, de esta forma, ahorrarse este tiempo de procesamiento por cada solicitud que reciba posteriormente la API.

El último paso, previo al cálculo de la ruta óptima, es realizar un procedimiento similar al de los nodos de navegación, pero con los nodos que representarán a los productos, por lo que

se repite el proceso de cálculo de vecinos. De esta forma, se va generando una matriz de distancias entre productos, y además se amplía la matriz de distancias estática con los nuevos nodos, obteniendo así esta nueva matriz de distancias dinámica la cual cambiará dependiendo de los productos seleccionados. Cabe mencionar que hay casos en donde un producto puede encontrarse en más de un pasillo a la vez, por lo que para esta matriz de distancias se considerarán como nodos independientes uno del otro.

Para el cálculo de la ruta óptima se decidió utilizar la herramienta desarrollada por Google: OR-Tools [OR-Tools]. Esta es utilizada para resolver problemas de programación lineal, de restricciones y para el enrutamiento de vehículos o asociados, siendo de suma utilidad para este proyecto. Dado que el optimizador de rutas suele ser comúnmente empleado para resolver problemas de enrutamiento de vehículos, el problema a resolver es equivalente a disponer de un solo vehículo con un punto de entrada y un punto de salida.

Tal como se mencionó anteriormente, es posible que algunos productos puedan estar presentes en más de un pasillo, por lo que se agregan como restricciones al solucionador, de forma que solo elija pasar el cual sea más conveniente para la ruta. Este solucionador recibe como parámetro la matriz de distancia entre los productos, la cual fue ligeramente modificada con la intención de minimizar aún más la recolección de productos en el mismo pasillo, por lo que la distancia entre cada producto quedaría de la forma:

$$p = \begin{cases} 1, & \text{pasillo diferente} \\ 0, & \text{mismo pasillo} \end{cases}$$

$$D'_{ij} = D_{ij} + c \times p$$

siendo c una constante escogida de forma que no afecte en gran medida al resto de distancias. Por otro lado, dada la naturaleza del problema, el solucionador tiene distintas estrategias para buscar una solución, como por ejemplo la estrategia de primera solución. Ésta última, como dice su nombre, busca una solución inicial priorizando aquella que cumpla con las restricciones sin la necesidad de ser óptima o las estrategias de metaheurísticas, como pueden ser la Búsqueda Tabú, *Greedy Descent*, Recocido Simulado o la Búsqueda Local Guiada. Este tipo de estrategias requieren de un parámetro adicional denominado límite de búsqueda. Para la elección tanto de la metaheurística como del límite de búsqueda se realizó un experimento que consistió generar rutas para canastas de 10, 20, 30, 50 y 80 productos con límites de búsqueda de 1, 5, 8, 10 y 15 segundos para cada una de las metaheurísticas previamente mencionadas. Para las canastas de 10 y 20 productos las metaheurísticas entregaron resultados muy similares independiente del límite de búsqueda. Por otro lado, para las canastas de 30, 50 y 80 productos se notó una diferencia dependiendo de la metaheurística y el límite de tiempo, siendo la Búsqueda Local Guiada

la que mejores resultados entregó a medida que la cantidad de productos aumentaba, aunque se notaba cierta tendencia a caer en un óptimo local pasado los 8 segundos, por lo que se escogió como límite de búsqueda un tiempo de 5 segundos dando como resultado una menor latencia de respuesta.

Cabe recalcar que, dado los parámetros del solucionador, resulta inviable añadir productos a la ruta ya generada, ya que la matriz de distancias se vería modificada, así que, en caso de necesitar modificar la ruta a lo largo del recorrido, sería necesario volver a calcular la ruta desde cero.

Por último, el solucionador retorna el orden a seguir de los productos a recoger, por lo que no resulta muy fácil de seguir en caso de que se dibuje esta ruta sobre el mapa representado en la Figura 30.

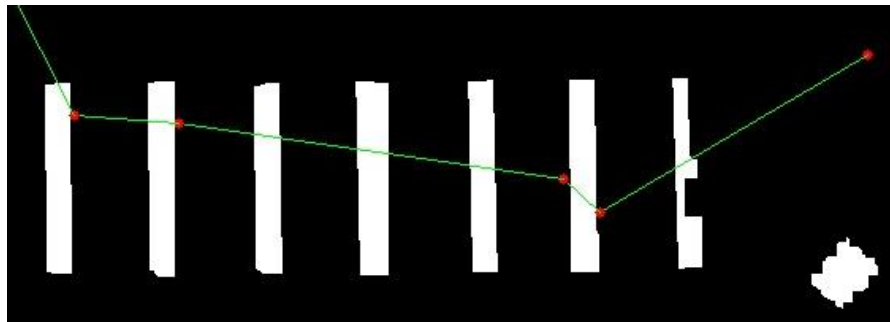


Figura 30: Trazado de la ruta entregada por el solucionador de OR-Tools.

Aquí es donde entran los nodos de navegación a componer estas rutas utilizando el algoritmo A* para calcular la ruta entre la entrada, salida y cada par de productos respectivamente, dando como resultado la ruta de la Figura 31.

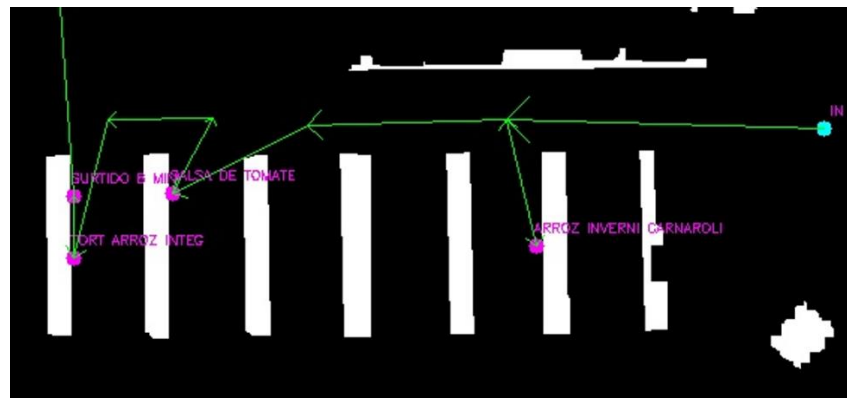


Figura 31: Trazado de la ruta generada por A*.

Una vez que se dio por terminada la ejecución del algoritmo, solo resta el último *scope* que consistió en encapsular todas estas funciones en una API que sería utilizada por la aplicación móvil desarrollada en el siguiente ciclo de trabajo.

3.2 Segundo Ciclo: Diseño y desarrollo de la aplicación móvil

Este ciclo tuvo como objetivo el mejorar la experiencia de compra y *Pick-Up*, y proporcionar una prueba de concepto de una propuesta de valor para las empresas de Delivery.

Si bien la importancia de este trabajo se basa en generar la ruta a seguir por los usuarios, no hay que dejar de lado la experiencia del mismo al momento de utilizar la aplicación, ya que puede influir directamente en el tiempo total de *Pick-Up* si es que hay aspectos que no le quedan del todo claro o simplemente no entiende el funcionamiento de la aplicación móvil y de esta forma afectar directamente la experiencia de *Pick-Up*. Es por esto que se establecieron los siguientes *scopes*:

- Desarrollo de un Producto Mínimo Viable (PMV): El objetivo consistió en el diseño y desarrollo de la aplicación móvil capaz de servir de guía para los usuarios.
- Instrumentación: El objetivo era desarrollar la lógica necesaria dentro de la aplicación para generar los datos que serían posteriormente analizados.
- Testeo: El objetivo era planificar y ejecutar el testeo de la aplicación, junto con la recolección de voluntarios que harían de “testers”.

PMV e Instrumentación

Lo primero a considerar fue el alcance del PMV, ya que el objetivo principal de este es que cumpla con lo necesario para poder realizar el testeo con los voluntarios, por lo que se estableció una interacción fija entre el usuario y la aplicación. Se mantuvo un local y listas de compras fijas (para poder comparar los datos obtenidos en el testeo) y para que, una vez la API responda, se guíe al usuario entregando la información necesaria y al finalizar el recorrido el usuario sea capaz observar un registro de sus acciones.

Si bien la API responde con el orden en el cual se deben recoger los productos, hay que tener en consideración que un usuario que no conozca con anterioridad el local no sabrá con precisión a qué sector dirigirse primero, pudiendo ir de un extremo del local a otro en la búsqueda del primer producto que se le especifica. Por esta razón, el primer requerimiento para la aplicación es la posibilidad de visualizar el mapa, además de tener la información de los productos, junto a su respectiva posición en cada góndola, de forma que

el usuario tenga la mayor cantidad de información para que pueda encontrar cada producto con eficacia; por lo que, como primer paso de diseño se realizó un *Mockup* teniendo como vista principal de la ruta a lo largo del mapa a la Figura 32, mientras que la vista para tener poder tener una información más detallada junto a la posición del producto en la góndola se puede observar en la Figura 33.

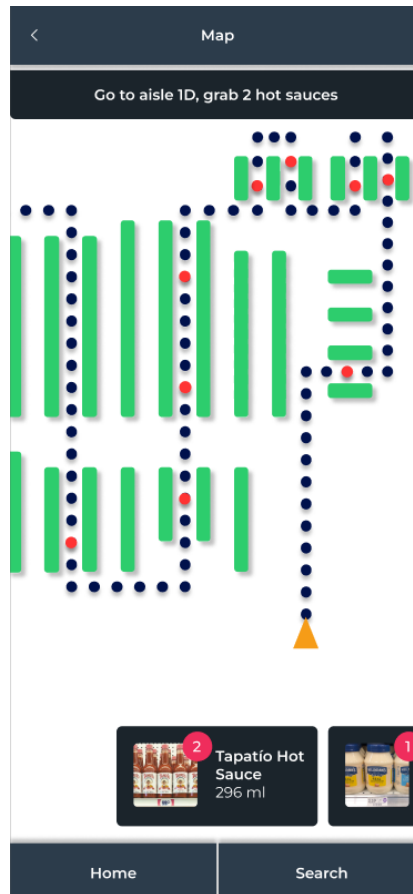


Figura 32: Mockup de la aplicación móvil.

OPTIMIZADOR DE RUTA PARA LA COMPRA DE INSUMOS EN DISTINTOS LOCALES DE SUMINISTROS

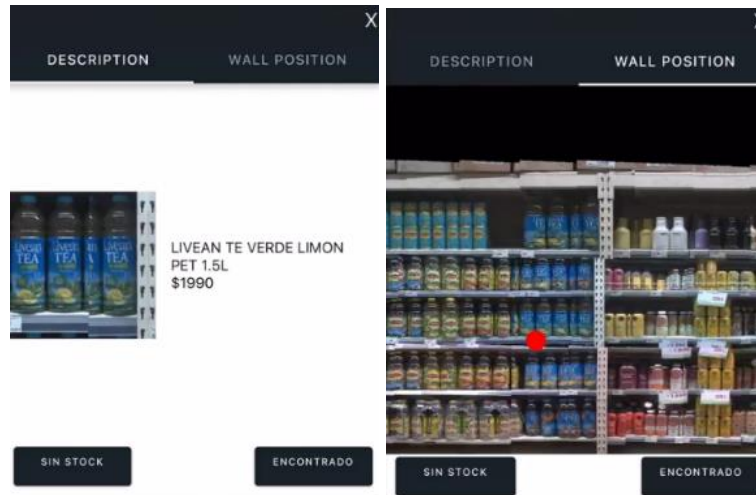


Figura 33: Vista de la descripción y posición del producto en cada góndola.

Una vez el usuario termine de recoger todos los productos de la canasta, la aplicación le indicará que se dirija a la zona de cajas y, posteriormente, un resumen del proceso de *Pick-Up* que realizó (Figura 34). Para generar estos datos, la aplicación empieza a tomar el tiempo desde el momento en que muestra al usuario la ruta a seguir hasta que el usuario presione que efectivamente llegó a la zona de cajas. Junto a ello, la aplicación permite generar el desglose de tiempo entre cada uno de los productos.

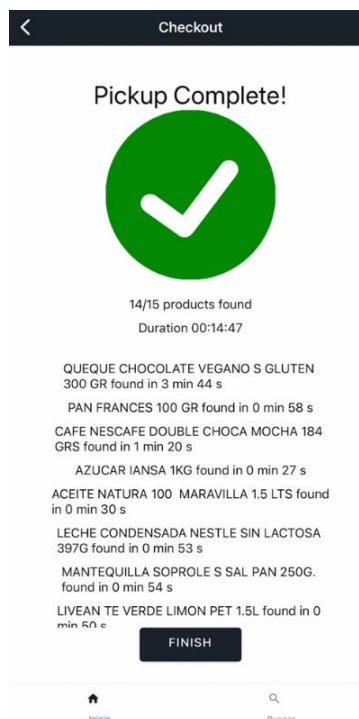


Figura 34: Vista del resumen de Pick-Up.

Los datos obtenidos son enviados en formato JSON para ser almacenados en *Google Cloud Storage* para su posterior análisis.

CAPÍTULO 4: VALIDACIÓN DE LA SOLUCIÓN

4.1 Descripción del Experimento

Para este experimento se plantearon dos listas de productos de distintos tamaños y productos para distribuir a los voluntarios como muestra la Tabla 1. Se separaron ambos grupos con la intención de minimizar el aprendizaje de cada individuo sobre el local de compras y sobre cada lista de compras, debido a que en gran parte son excluyentes entre sí, es decir que los productos que la conformaban se encuentran en distintos pasillos.

	Lista 1	Lista 2
Cantidad de productos	25 productos	15 productos
Con la App	Grupo A	Grupo B
Con un papel	Grupo B	Grupo A

Tabla 1: Distribución de los grupos de voluntarios y cantidad de productos.

4.1.1 Conformación de las listas de productos

Los productos de ambas listas fueron escogidos teniendo en consideración la canasta del IPC del 2018, siendo la Lista 1 una canasta para el hogar que incluía alimento para mascotas, productos infantiles, higiene personal y alimentos varios. Por otro lado, la Lista 2 fue diseñada con algunos productos que pudiesen encontrarse en oficinas o similares, tales como bidones de agua, azúcar, café, bebestibles, artículos de limpieza y/o de impresora con la dificultad extra de tener que buscar productos que no están considerados dentro de la canasta del IPC del 2018, como es la sección de productos orgánicos. Esta decisión se tomó con la intención de recrear el escenario en donde se le solicite a un *Shopper* un producto del cual no tiene un conocimiento previo.

La lista 1 estaba conformada de la siguiente forma:

- 1) Snack Perros sabor churrasco, 65g
- 2) Dulce de coco orgánico
- 3) Galletas Oreo regular, 108 gr

- 4) Salsa Soya Gourmet, 165 cc
- 5) Figura Peppa Pig Amigos Divertidos
- 6) Desodorante Nivea Men Fresh Ice, 150 ml
- 7) Bálsamo Elvive Dream Long, 370 ml
- 8) Papas Fritas Kryzpo Marken 130 gr
- 9) Maíz Gigante Inka Korn, 100 gr
- 10) Barra de Trencito, 150 gr
- 11) Helado San Francisco Tres Leches, 1 lt
- 12) Pasta de Choclo Espinaca Minuto verde, 1kg
- 13) Mermelada Berries Frasco Cuisine & Co, 360 gr
- 14) Yogurt Griego Danone Natural, 150 gr
- 15) Caldo Maggi sabor carne, 15 unidades
- 16) Pony Malta, 330 ml
- 17) Vodka Absolut Raspberri, 750 ml
- 18) Pisco Artesanos Cochiguaz 40°, 700cc
- 19) Detergente Ariel concentrado, 2.8 lt
- 20) Servilleta Nova clásica cóctel, 150 unidades
- 21) Atún lomito en agua Robinson Crusoe ,104 gr
- 22) Saborizante en polvo Cola Cao, 1 kg
- 23) Chocapic, 720 gr
- 24) Disfraz Spiderman Miles Morales Marvel
- 25) Servilletas Spiderman 12 unidades

mientras que la lista 2 estaba conformada por:

- 1) Pan francés Schar sin gluten, 100 gr
- 2) Espirales 49 Carozzi, 400 gr
- 3) Queque chocolate vegano sin gluten, 300 gr

- 4) Papel Equalit laser alcalino A4, 500 hojas
- 5) Marcador Sharpie Twin tip, 3 unidades
- 6) Café nescafé double mocha, 184 gr
- 7) Azúcar Iansa, 1kg
- 8) Aceite natural maravilla, 1.5 lt
- 9) Leche condensada Nestlé sin lactosa, 397 gr
- 10) Mantequilla Soprole sin sal barra, 250 gr
- 11) Livean té verde limón, 1.5 lt
- 12) Desinfectante Lysoform aire de montaña, 360 cc
- 13) Insecticida Raid moscas y mosquitos sin olor, 420 cc
- 14) Bidón de agua Benedictino, 6.5 lt
- 15) CIF crema bioactive limón, 750 gr

De esta forma, los productos quedaron distribuidos tal como se puede observar en la Figura 35, donde los puntos de color verde oscuro representan la ubicación de los productos de la Lista 1, mientras que los puntos de color rosa representan la ubicación de los productos de la Lista 2. Cabe resaltar que hay más de 40 productos (total de productos entre ambas listas) en la imagen debido a que, tal como se mencionó en capítulos anteriores, hay productos que se pueden encontrar en más de un lugar al mismo tiempo.

4.1.2 Recolección de Datos

Para la recolección de datos se tenían dos perfiles de voluntarios:

- 1) Voluntarios con lápiz y papel: a estos voluntarios se les entregó un lápiz y un papel con una de las listas de productos a recoger, teniendo libertad total de buscar las fotos de los productos en internet y/o preguntar dentro del supermercado por la ubicación de los productos.

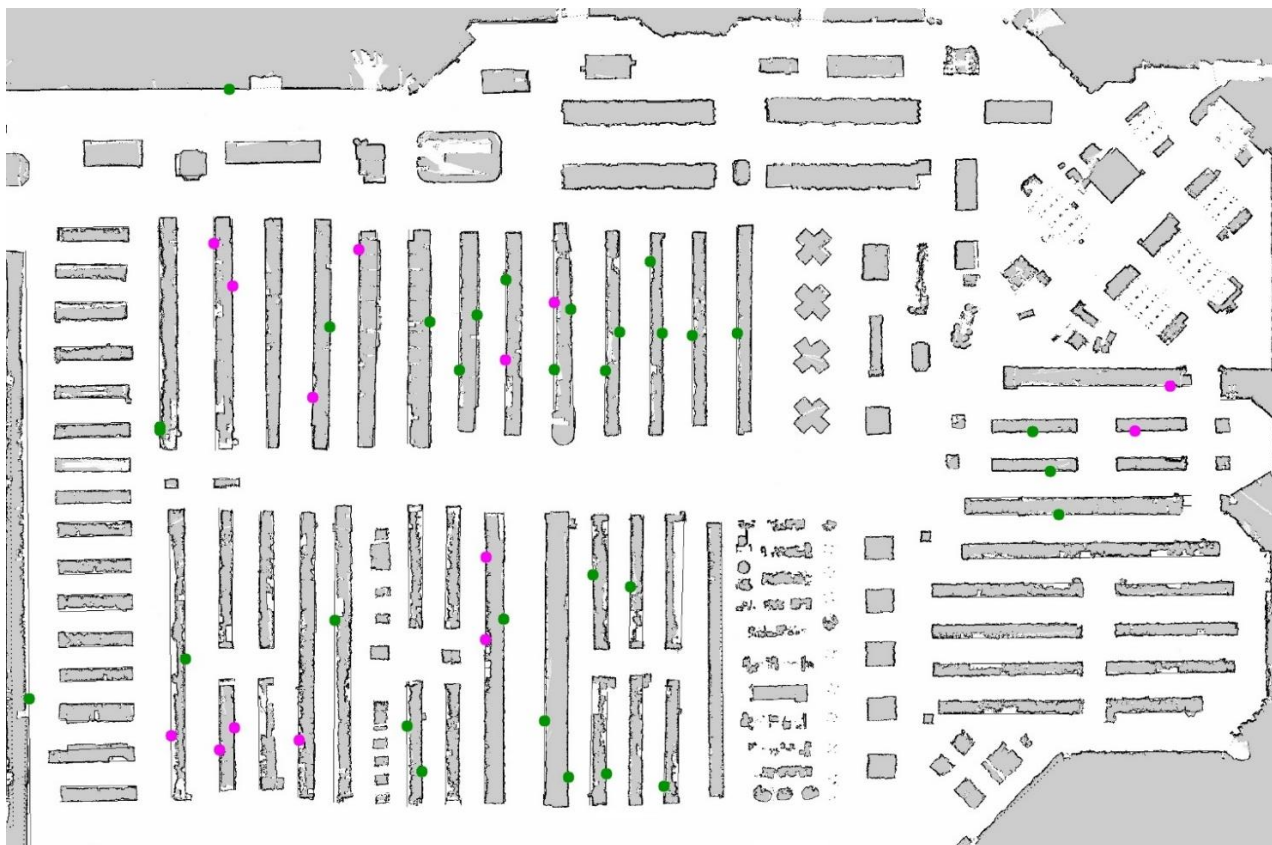


Figura 35: Posición de los productos en el mapa.

Tabla 2: Ejemplo de papel impreso de la lista 2.

PRODUCTO	ENCONTRADO	NO ENCONTRADO
PAN FRANCÉS SCHAR SIN GLUTEN 100 GR		
ESPIRALES 49 CAROZZI 400 GR		
QUEQUE CHOCOLATE VEGANO SIN GLUTEN 300 GR Dilici		
PAPEL EQUALIT LASER ALCALINO A4 500 HJ		
MARCADOR SHARPIE TWIN TIP 3 UN		
CAFÉ NESCAFE DOUBLE MOCHA 184 GR		
AZUCAR IANSA 1KG		
ACEITE NATURA MARAVILLA 1.5 LT		

OPTIMIZADOR DE RUTA PARA LA COMPRA DE INSUMOS EN DISTINTOS LOCALES DE SUMINISTROS

LECHE CONDENSADA NESTLE SIN LACTOSA 397 GR		
MANTEQUILLA SOPROLE SIN SAL BARRA 250G		
LIVEAN TÉ VERDE LIMON 1.5L		
DESINFECTANTE LYSOFORM AIRE DE MONTAÑA 360 CC		
INSECTICIDA RAID MOSCAS Y MOSQUITOS SIN OLOR 420 CC		
BIDON DE AGUA BENEDICTINO 6.5 LT		
CIF CREMA BIOACTIVE LIMÓN 750 GR		

Para obtener su tiempo de *Pick-Up* se utilizó un cronómetro, el cual se iniciaba una vez el voluntario ingresaba al local y terminaba una vez volvía al punto de partida. Su tiempo de *Pick-Up*, junto a la cantidad de productos encontrados y no encontrados, fueron almacenadas en un archivo CSV para su posterior procesamiento.

- 2) Voluntarios con la aplicación móvil: Los voluntarios que utilizaban la aplicación móvil, salían con un margen de 3 a 5 minutos después de la salida de un voluntario con papel desde el mismo punto de partida, de esta forma se buscó evitar que los voluntarios tendieran a seguirse consciente o inconscientemente. Su tiempo de *Pick-Up* fue registrado utilizando la aplicación móvil, además de un cronómetro para tener una concordancia con los tiempos de *Pick-Up* de los voluntarios con papel. A estos voluntarios se les solicitó seguir la ruta generada por la aplicación priorizando siempre la información proporcionada por la aplicación.

El tiempo de *Pick-Up* del cronómetro de cada usuario fue almacenado en un archivo CSV, mientras que el registro de productos encontrados y no encontrados fueron almacenados en *Google Cloud Storage* para su posterior procesamiento.

También se decidió registrar la cantidad de productos encontrados para ambos grupos junto a los productos que por algún motivo no tuviesen stock y/o no se tenga más información de él en los últimos 3 días (tiempo límite de información que utiliza la aplicación móvil), motivo por el cual no es posible mostrarle al usuario una imagen del producto y/o la ubicación del producto en su respectiva góndola, dando como resultado una vista como en la Figura 36.

4.2 Desarrollo del experimento

Se obtuvo la participación de 34 voluntarios, que fueron repartidos en los dos grupos mencionados en la sección anterior, por lo que tanto el grupo A como el grupo B (ver Tabla 1) quedaron conformados por 17 voluntarios cada uno. De ambos grupos se calcularon los tiempos promedios para cada lista de compras, reflejados en la Tabla 3.

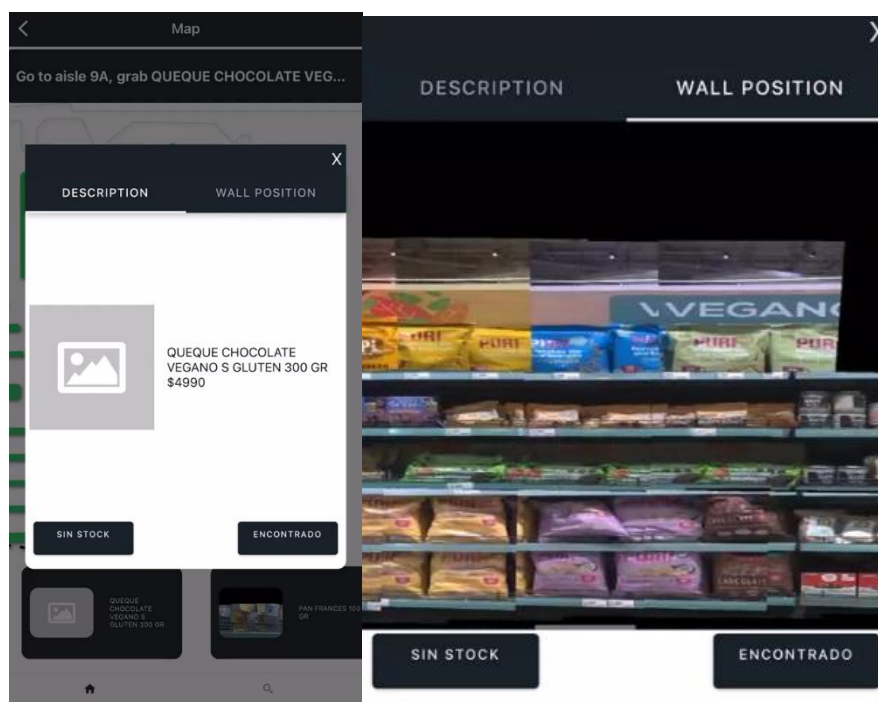


Figura 36:Ejemplo de un producto sin foto y/o posición en góndola.

Tabla 3: Promedio de tiempos para cada lista.

Fuente: Elaboración Propia.

	Lista 1	Lista 2
Tiempo promedio con la App	30:25 [min]	16:21 [min]
Tiempo promedio sin la App	40:43 [min]	27:55 [min]

El primer análisis que se puede hacer es que hubo una reducción de tiempo del 25.3% utilizando la aplicación en comparación al tiempo demorado con un papel para la Lista 1. Por otro lado, también hubo una reducción de tiempo del 41.43% en el caso de la Lista 2.

Si bien, como primera impresión se nota una clara reducción en el tiempo promedio de *Pick-Up* para ambas listas, se graficaron los tiempos de cada persona en un diagrama de caja para poder observar de mejor manera la distribución de los datos, y así visibilizar una posible presencia de valores atípicos o anómalos (también llamados *outlayers*), los cuales en caso de ser significativamente grandes afectan al promedio de tiempo.

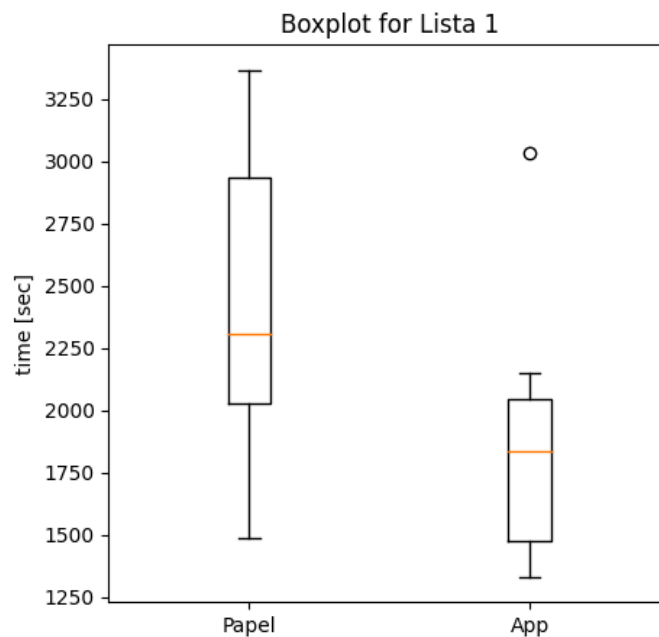


Figura 37: Diagrama de caja para los tiempos de la Lista 1.

Como es posible observar en la Figura 37, el diagrama de caja para los datos recopilados con el papel resulta ser considerablemente largo, yendo desde el primer cuartil con un valor de 2030 [s] o 33:50 [min] hasta el segundo cuartil con un valor de 2937 [s] o 48:57 [min], lo que se traduce en una gran dispersión de datos debido a la diferencia de experiencia dentro de un supermercado entre cada uno de los voluntarios.

Gracias a lo anterior, la mediana tiene un valor de 2309 [s] o 38:29 min y se ubica más cerca del primer cuartil, indicando una asimetría negativa por lo que la mayor concentración de los valores se encuentra sobre la mediana. Por otro lado, los valores obtenidos al utilizar la aplicación móvil muestran una caja más pequeña, presentando una menor dispersión en los datos yendo desde los 1476 [s] o 24:36 [min] hasta los 2043 [s] o 34:06 [min]. Además, el valor de la mediana resultó ser de 1833 [s] o 30:33 [min], lo que representa un valor

bastante similar al promedio previamente calculado y, de esta forma, ubicándose ligeramente más cerca del tercer cuartil, por lo que hay una mayor distribución de los datos bajo la mediana.

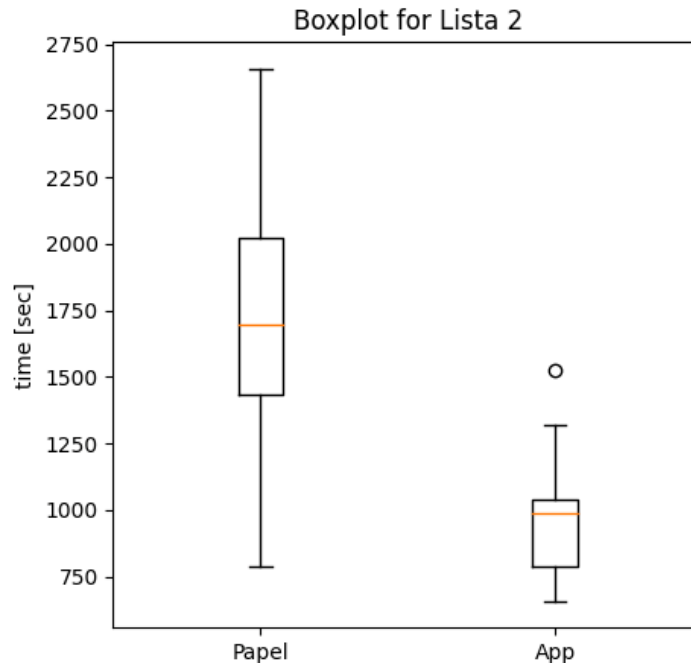


Figura 38:Diagrama de caja para los tiempos de la Lista 2.

Por otro lado, observando los diagramas de cajas de la Lista 2 en la Figura 38, se obtienen resultados similares en comparación a la Lista 1, en donde los valores conseguidos con un papel dieron nuevamente como resultado una caja más larga en comparación a los valores con la aplicación. Sin embargo, lo más destacable resulta ser lo largo de los “bigotes” para los valores con papel, debido a que los valores obtenidos están muy dispersos en ambas direcciones, por lo que es posible afirmar que para algunas personas la Lista 2 no les resultó mayor problema, mientras que para otras personas le tomó más del doble de tiempo.

En conclusión, los valores obtenidos con la aplicación generaron una caja bastante pequeña y los datos en sí resultaron encontrarse menos dispersos. A su vez, los bigotes resultaron ser pequeños e incluso el único valor anómalo con la aplicación se encuentra dentro de los valores más comunes utilizando el papel.

Tal como fue previamente mencionado, además del tiempo también se registraron los productos encontrados y, por consiguiente, los productos no encontrados. Esto debido a que en gran medida el tiempo de *Pick-Up* depende directamente de la cantidad de productos y de la separación entre ellos, considerando también que cada individuo tiene su

propio nivel de paciencia para buscar cada producto de la lista. Teniendo esto en cuenta, pueden existir casos donde el tiempo de *Pick-Up* total aumente debido a la búsqueda de un producto en especial mientras que, por otro lado, puede ocurrir el caso contrario en donde los voluntarios hayan decidido no gastar más tiempo en la búsqueda de algunos productos.

Tabla 4: Cantidad Promedio de Productos encontrados por Lista.

	Lista 1	Lista 2
Promedio App	23.2	14.2
Promedio Papel	21.5	13.5
Porcentaje App	92.7%	94.5%
Porcentaje Papel	86.4%	89.4%

En ambos casos es posible notar que se encontraron más productos con la app que con el papel, llegando a tener una diferencia en la eficiencia del 6.3% para la Lista 1 y de un 5.1% para la Lista 2. Si bien los voluntarios que utilizaron la aplicación en promedio encontraron más productos, también es cierto que no los encontraron todos debido a que el periodo de pruebas se realizó en un lapso de siete días, y por consiguiente algunos productos estaban sin stock al momento de realizar la recolección de datos.

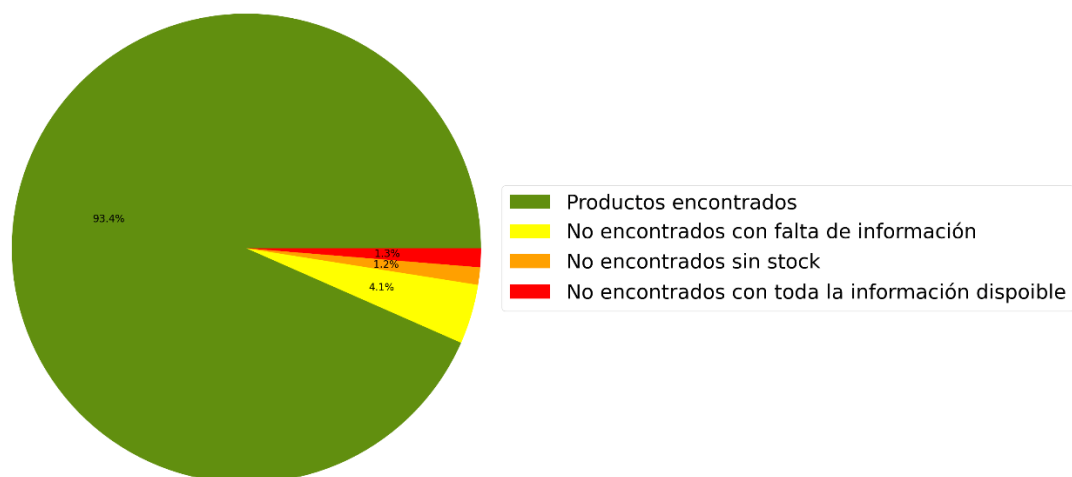


Figura 39: Diagrama de torta de todos los productos recopilados con la app.

Como se observa en la Figura 39, se encontró el 93.4% de un total de 680 productos entre ambas listas utilizando la *app*. Del 6.6% restante, 4.1% corresponden a productos que no fueron encontrados y que, el día del experimento, no se tenía ni la posición en la góndola ni la foto descriptiva del producto, mostrando al usuario la vista señalada en la Figura 31.

Por otro lado, el 1.2% de los productos no encontrados que sí tenían toda la información (Figura 28) realmente no tenían stock, dejando el 1.3% restante como productos que a pesar de tener toda su información disponible (en principio) no fueron encontrados. De estos últimos, algunos voluntarios recalcaron que en la zona señalada por la aplicación se encontraba un producto de distinto gramaje o que, directamente, no coincidía con la descripción de la *app*.

El detalle de los productos no encontrados para cada lista puede verse en la Figura 40 y Figura 41 respectivamente, donde las barras azules representan la cantidad de veces que un producto no fue encontrado por los voluntarios que utilizaron la *app*, mientras que las barras naranjas representan la cantidad de veces que un producto no fue encontrado por los usuarios que usaban un papel.

Los productos que presentaron un mayor problema para los usuarios con el papel resultaron ser, a su vez, los productos encontrados en la sección orgánica y que tenían como dificultad adicional que, en principio, el nombre del producto en sí confundía a los voluntarios con papel, ya que al leer el Queque Vegano y/o el Pan Francés los voluntarios tendían a dirigirse a la sección de panadería del supermercado y generalmente al no encontrar el producto en dicha sección, desistían su búsqueda y era marcado como no encontrado.

Por último, se decidió representar en color rojo la cantidad de días en que, mientras se realizaba dicho experimento, esos determinados productos se encontraban sin stock y, de esta forma, contrastar los productos no encontrados que realmente sí podían ser ubicados contra los productos no encontrados que realmente no tenían stock.

Gracias a los resultados obtenidos, fue posible proporcionar un valor agregado a los datos de la empresa junto a la generación de nuevas ideas de desarrollo, ya que gracias a los comentarios de los voluntarios se propusieron cambios en la interfaz con el objetivo de facilitar su uso y mejorar así la experiencia del usuario.

OPTIMIZADOR DE RUTA PARA LA COMPRA DE INSUMOS EN DISTINTOS LOCALES DE SUMINISTROS

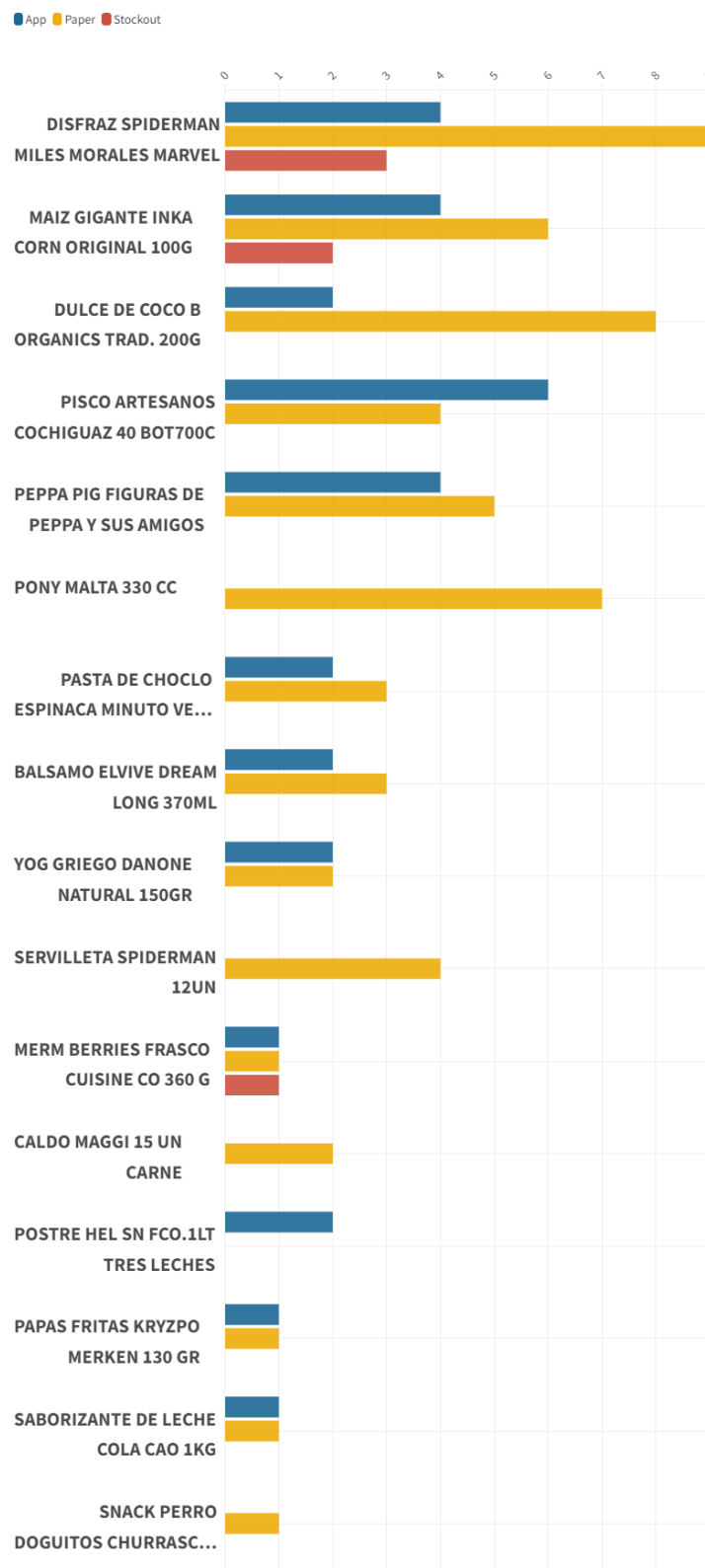


Figura 40: Productos más difíciles de encontrar de la Lista 1.

OPTIMIZADOR DE RUTA PARA LA COMPRA DE INSUMOS EN DISTINTOS LOCALES DE SUMINISTROS

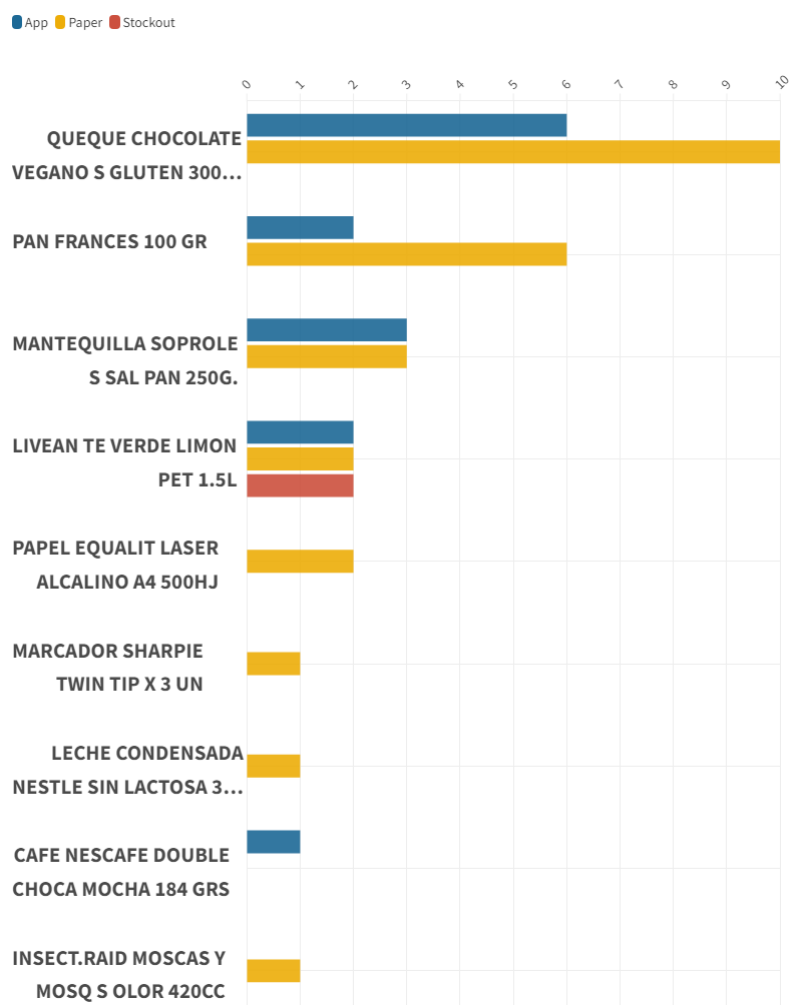


Figura 41: Productos más difíciles de encontrar de la Lista 2.

CONCLUSIONES

El trabajo realizado consistió principalmente en el diseño y desarrollo de software de una aplicación móvil junto a su respectivo *Backend*, expuesto en una API considerando el contexto de desarrollo de la empresa Zippedi SA, la cual, tal como se ha visto a lo largo de este trabajo, proporcionó todos los datos necesarios para el desarrollo de dicha aplicación. Esta herramienta surgió con el objetivo de explorar la idea de reducir los tiempos de *Pick-Up* utilizando los datos para guiar a las personas a lo largo de los locales y así darles un valor agregado a dichos datos.

El algoritmo optimizador de ruta es el resultado de un trabajo de procesamiento de imágenes utilizando funciones morfológicas gracias a la biblioteca de OpenCV. Por otro lado, el trabajo de inteligencia artificial debido al modelamiento del problema como el problema del vendedor viajero con algunas restricciones fijas, tales como el punto de partida y final, junto a la posibilidad de existir restricciones en el caso de los productos que pudiesen encontrarse en más de un lugar a la vez fue resuelto gracias al solucionador de enrutamiento de vehículos de la biblioteca de OR-Tools.

Dada la naturaleza de este problema, se generó un optimizador de rutas, el cual entrega un óptimo local teniendo como límite de búsqueda un periodo de tiempo fijo. Para su desarrollo se utilizó Google OR-Tools debido a la facilidad de su implementación y así lograr cumplir con los tiempos de desarrollo.

Ahora bien, tanto el punto de entrada como de salida están definidos dentro de los “Layouts” que proporciona Zippedi, por lo que, en caso de ingresar por otra entrada, el usuario estaría obligado a caminar hacia la entrada previamente definida o la ruta le resultaría totalmente inútil. Este problema no es tan grave en el caso de la salida, ya que este punto simplemente refleja un punto central dentro de la zona de cajas, por lo que, al recoger los productos más cercanos a la salida para el final del recorrido el usuario queda cerca de la zona de cajas, dando por terminado el proceso de *Pick-Up*, ya que el tiempo que demore está fuera del alcance de este trabajo.

El desarrollo móvil es el resultado de un trabajo de diseño y de un constante replanteamiento de los procesos y formas de visualizar los datos. En gran parte se debe a que, a diferencia del desarrollo web, es necesario tener siempre en consideración el mayor ahorro de recursos posible debido a la gran variedad de dispositivos en el mercado. De igual forma, era relevante la creación de una aplicación capaz de responder de manera eficiente frente a los distintos tamaños de las pantallas sin dejar de lado la experiencia del usuario ante una aplicación totalmente nueva para él/ella.

Una vez terminado el desarrollo de la aplicación móvil, se ejecutó la etapa de recolección de datos utilizando de voluntarios a colegas, familiares y amigos, obteniendo de esta forma un total de 34 voluntarios en un periodo de 7 días ajustándose a la disponibilidad de los voluntarios. Por este motivo, ocurrieron situaciones fuera del experimento que, al mismo tiempo, representan de mejor manera la realidad dentro de un supermercado, algunas situaciones fueron:

- Productos que se quedaban sin stock a lo largo del día
- Productos sin foto y/o información de su posición en la góndola
- Momentos en que el flujo de personas dentro del supermercado era mucho mayor
- Modificación de los pasillos:
 - Cambio de nombre de un pasillo
 - Cambios en las posiciones de los productos en un periodo menor a 24 hrs.

Situaciones como los productos sin stock solían ser mitigadas con horas de diferencia por personal del local, mientras que las modificaciones dentro de los pasillos se corregían automáticamente al día siguiente debido a la forma en que trabaja Zippedi. Por otro lado, la falta de información sobre ciertos productos que afectó a algunos usuarios ocurre principalmente porque existen ocasiones en donde hay pasillos que no suelen ser escaneados por Zippedi en algunos días concretos por distintos motivos como petición del local o porque simplemente el robot encontraba obstáculos frente a él. Debido a esto, algunos productos de los cuales sí se tenía toda la información en un principio a lo largo de la semana de pruebas no pudieron ser registrados por Zippedi y de esta forma, les faltaba la foto y/o su posición dentro del pasillo.

El presente trabajo cumplió con el desarrollo de un algoritmo optimizador de ruta y el diseño y desarrollo de una aplicación móvil capaz de guiar al usuario a lo largo de su recorrido junto con la característica de poder visualizar la posición de cada producto en la góndola correspondiente. Ahora bien, para la verificación efectiva en la reducción de tiempos se ejecutaron pruebas piloto en un local simulando el proceso de *Pick-Up* sin la aplicación, en donde los voluntarios contasen con su propia experiencia, intuición y paciencia para encontrar los productos solicitados. A su vez, existieron voluntarios que utilizaron la aplicación desarrollada para encontrar los mismos productos.

Al momento de comparar ambos grupos fue posible observar una clara diferencia de tiempos entre los usuarios que utilizaron un papel y los que utilizaron la aplicación, siendo

estos últimos por una notoria diferencia más eficientes. Además, también se redujo notoriamente la dispersión entre los datos de tiempo, es decir, que en gran medida los usuarios que utilizaron la aplicación tuvieron por lo general tiempos más similares entre sí, mientras que los tiempos que registraron las personas que simplemente utilizaban un papel fueron más variados, debido muy probablemente a su propia experiencia dentro de un local de suministros.

El esfuerzo por encontrar los productos depende mayoritariamente de la persona, ya que cada persona dedicaba un esfuerzo diferente en la búsqueda de cada producto, mientras que las personas que utilizaban la aplicación, en caso de no encontrar el producto en la posición señalada, simplemente desistían de su búsqueda y continuaban con el recorrido.

Por otra parte, además de notar una reducción en el tiempo de *Pick-Up* para las personas que utilizaban la aplicación, también se observó una mayor cantidad de productos encontrados, ya que, al no tener conocimiento de donde se encontraban los productos, dependía de las personas con la lista desistir, o no, de la búsqueda de cada producto de la lista. De lo contrario, la única forma de que una persona con la aplicación diera por “No encontrado” un producto es que este no estuviese en la ubicación señalada o el producto realmente no tenga stock. Es debido a esto que de todos los productos no encontrados utilizando la aplicación, el 62% corresponden a productos sin la información necesaria disponible al momento de realizar la prueba.

En conclusión, es posible afirmar que el trabajo realizado cumple con su objetivo general de reducir los tiempos de *Pick-Up* gracias al diseño y desarrollo de la aplicación móvil y el algoritmo que genera la ruta.

Como trabajo futuro se considera la opción de modificar el optimizador para posibilitar la opción de gestionar el punto de inicio en base a la posición que el usuario requiera, junto con tener en consideración priorizar recoger ciertos productos debido al peso y/o volumen, ya que es posible que productos muy grandes o de mucho peso como cajas de leche, bebidas y/o bidones de agua se encuentren lejos de la entrada y cuando sea el momento de recogerlos sea necesario reorganizar el carrito de supermercado. De igual forma, también es probable que ciertos productos deban ser recogidos al final del recorrido como es el caso de los productos congelados. Finalmente, el algoritmo optimizador de rutas no está diseñado para quitar o agregar productos a la ruta generada, motivo por el cual en caso de querer añadir productos a la ruta ya generada sería necesario volver a calcular la ruta, pudiendo provocar un proceso menos eficiente.

Como trabajo futuro para la aplicación móvil se tomaron en consideración los comentarios de los distintos voluntarios sobre algunas funcionalidades que no se lograron cumplir

debido a los tiempos de entrega, tales como mejorar la interacción con el mapa, ya que el zoom quedó deshabilitado por razones de eficiencia, junto a la posibilidad de rotar el mapa ya sea de forma manual como automática, con el objetivo de que el usuario sea capaz de orientarse de mejor manera.

Otro de los puntos de mejora está en la visualización gráfica de los nombres de los pasillos, ya que si bien es cierto que la mayoría de los pasillos tienen su cartel correspondiente con el nombre del pasillo y los productos que se podrían encontrar, hay pasillos que no tienen este cartel por lo que algunos voluntarios demoraban en orientarse. Además, se puede tener en consideración modificar la foto de los productos originales, ya que actualmente las fotos son extraídas de las últimas sesiones en que han sido vistas debido a que a la fecha en que este trabajo fue realizado, en Zippedi no se cuenta con una foto modelo de cada uno de los productos de los que se tiene registro.

También puede ser interesante en un futuro recomendar productos similares y/o equivalentes a los usuarios cuando un producto dentro de su lista de compras se encuentre sin stock, siempre y cuando dicho producto se encuentre dentro del mismo pasillo con la intención de no afectar el recorrido original, existiendo la posibilidad de notificar sobre ofertas de dichos productos.

A modo de cierre, se procede a comentar el aporte de la carrera para el desarrollo de este trabajo. La Inteligencia Artificial resultó un pilar fundamental para el desarrollo de este trabajo, desde el reconocimiento del problema, dificultades, restricciones y los distintos enfoques para la búsqueda de soluciones en un tiempo limitado. También, es necesario tener en consideración la importancia de la Ingeniería de Software, ya que este trabajo fue realizado dentro de una empresa en donde si bien no se utilizaba *Scrum*, también se utilizaba una metodología de desarrollo ágil; junto a esto es necesario recalcar la importancia del Diseño de Interfaces Usuarías, ya que de nada serviría tener el mejor algoritmo si es que el usuario no fuese capaz de entender lo que está viendo de forma intuitiva. Por último, la combinación de las bases aprendidas en Estadística Computacional junto al electivo de Visualización de Datos, para poder generar los gráficos presentados junto a sus explicaciones para facilitar el entendimiento al lector.

REFERENCIAS BIBLIOGRÁFICAS

[Acosta, 2011] Antonio Acosta (2011). Algoritmo de Bresenham.

[Bresenham, 1965] J.E. Bresenham (1965). Algorithms for computer control of a digital plotter.

[Bruno Marín B, 2022] Bruno Marín Barrera (2022). Exploiting Direction in Grid Graph to build a Fast and Lighter Subgoal Graph.

[Canasta IPC 2018], <https://www.ine.gob.cl/estadisticas/economia/indices-de-precio-e-inflacion/indice-de-precios-al-consumidor>

[Cloud Storage] Google Cloud Storage. <https://cloud.google.com/storage?hl=es-419#section-3>

[Cuevas, 2013] Danilo A. Cuevas Guzmán (2013). Análisis de Algoritmos Pathfinding.

[eloviparo, 2018] La distancia de Manhattan o distancia euclídea (2018).
<https://eloviparo.wordpress.com/2018/03/13/la-distancia-manhattan-o-la-distancia-euclidea/>

[Enya Quetzalli, 2020] Enya Quetzalli Gómez Rodríguez (2020). Grafos: BFS y DFS.

[Garey y Johnson, 1979] Michael R. Garey, David S. Johnson (1979). COMPUTERS AND INTRACTABILITY A Guide to the Theory of NP-Completeness.

[Graficación, 2017] Algoritmos DDA y Bresenham (2017).
<https://graficacion2017.wordpress.com/2017/08/14/algoritmos-dda-y-bresenham/>

[Jhosimar G. Arias, 2012] Jhosimar George Arias Figueroa (2012). Camino más corto: Algoritmo de Dijkstra.

[Krishnaswamy, 2009] Krishnaswamy, Nikhil. (2009) Comparison of Efficiency in Pathfinding Algorithms in Game Development. <https://via.library.depaul.edu/tr/10>

[Melián, Moreno y Vega, 2003] Belén Melián, José A. Moreno Pérez, J. Marcos Moreno Vega (2003). Metaheuristics: A global view.

[MathWorks, 2020] Tipos de operaciones morfológicas
<https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html>

[Stout, 1996] Stout, B. (1996). Smart moves: Intelligent pathfinding. Game developer magazine, 10, 28-35.

[OR-Tools] Google OR-Tools. <https://developers.google.com/optimization?hl=es-419>