Práctica de Busqueda Local

Hugo Aranda Sánchez
Albert Ruiz Vives
Jack Griffiths Rico
Departamento de Computación

30/10/2023

Índice

1 Introducción

En esta documentación, presentaremos en detalle cómo abordamos el problema de optimización de la distribución de bicicletas en Bicing. El objetivo es mejorar la forma en que las bicicletas se distribuyen en las estaciones para garantizar que los usuarios siempre tengan acceso a ellas cuando las necesiten.

Exploraremos la estrategia que empleamos, los métodos utilizados y los factores clave a considerar. La elección de utilizar la búsqueda local se basa en su eficiencia, su capacidad para adaptarse a las restricciones específicas del problema y su escalabilidad.

A lo largo de la documentación, explicaremos cómo aplicamos la búsqueda local, cómo manejamos las limitaciones y cómo evaluamos la calidad de la solución. Esta guía tiene como propósito demostrar la efectividad de nuestra solución para mejorar la distribución de bicicletas de Bicing.

2 Descripción del problema

En esta práctica, se nos plantea ofrecer un servicio a la empresa Bicing para optimizar la distribución de sus bicicletas a lo largo de sus estaciones de manera que los usuarios estén lo más contentos posibles.

2.1 Elementos del problema

El problema diferencia tres elementos principales que existen dentro de un universo que es la ciudad: existen E estaciones de entre las cuales se distribuyen B bicicletas con la ayuda de F furgonetas que trazan rutas para ayudar en la tarea de redistribución.

2.1.1 Ciudad

La ciudad es un cuadrado de 10 x 10 kilómetros donde la disposición de las manzanas es en cuadrícula total y estas miden 100 x 100 metros. Esto facilita enormemente diversos aspectos como el cálculo de las distancias y los caminos (pueden tomar rutas muy diferentes y recorrer la misma distancia).

2.1.2 Estaciones

Las estaciones se pueden encontrar en los cruces de las manzanas anteriormente descritas.

En ellas cabe un número ilimitado de bicis.

Disponemos de tres informaciones por cada estación dentro del problema que nos indica cómo hemos de distribuir las bicis entre las mismas:

1. Demanda de las bicicletas para la siguiente hora: que nos dice cuántas bicicletas se espera que sean usadas durante la siguiente hora por los usuarios.

2. Bicicletas que habrá a la siguiente hora sin contar los traslados: debido a los movimientos de los clientes de forma independiente a nuestra solución.

3. Bicicletas que no se moverán en la hora actual: las cuales no van a ser usadas por los usuarios esperados para esa hora y están disponibles para movilizar a otras estaciones para crear la solución.

2.1.3 Furgonetas

Estos son los entes que mediante rutas pueden coger bicicletas de una estación y llevarlas como máximo a otras dos en lo que constituye una ruta que sucederá entre la hora actual y la siguiente para la cual queremos optimizar la satisfacción.

Estas furgonetas pueden transportar 30 bicicletas como máximo.

No pueden coger bicis de una estación dos furgonetas diferentes.

2.1.4 Propiedades de la solución

Una asignación de recorridos de furgonetas de distribución de bicicletas contiene algunos elementos más aparte de los anteriores: la satisfacción de Bicing con nuestra solución y el coste del transporte.

La satisfacción se mide simplemente con la recompensa que obtenemos por nuestra labor. La cual aumenta en un euro si transportamos una bicicleta allá donde sea necesitada (no exceda la necesidad de bicicletas para la siguiente hora) y se nos quitará un euro por cada bicicleta que quitemos de un lugar que la necesitaba (no presenta excedente de bicicletas para coger).

El coste de euros por kilómetro recorrido, donde nb es el número de bicicletas, es $((nb + 9)\ div\ 10)$.

La solución, por lo tanto, es para una hora completa donde se indican los traslados de bicicletas.

2.2 Restricciones de la solución

En el problema de optimización de la distribución de bicicletas para la empresa Bicing, las restricciones de la solución son reglas y condiciones que deben cumplirse para garantizar la eficiencia y el funcionamiento adecuado del sistema. Las restricciones del problema se pueden describir de la siguiente manera:

- Demanda de Bicicletas: La distribución de bicicletas debe satisfacer la demanda de bicicletas esperada en la siguiente hora. No debe haber una escasez de bicicletas en una estación si se espera que se utilicen.
- Número de estaciones visitadas: Cada furgoneta puede visitar como máximo dos estaciones en un solo viaje. Esto implica que no se pueden hacer traslados desde una estación a más de dos estaciones en un solo viaje.
- Origen de las furgonetas: El origen de las furgonetas no está determinado inicialmente y debe calcularse de manera que se optimicen los criterios de la solución.
- Límite de capacidad de estaciones: No se pueden agregar o quitar bicicletas en una estación más allá de su capacidad máxima. Si una estación ya tiene suficientes bicicletas, no se pueden agregar más, y si no tiene suficientes, no se pueden quitar más de las necesarias para cumplir con la demanda prevista.
- Restricciones de coste: El coste de transporte se calcula en función del número de bicicletas transportadas en una furgoneta, y el coste por kilómetro recorrido varía según la fórmula proporcionada *((nb + 9) div 10)*. Debe minimizarse el coste total de transporte.

- Satisfacción del cliente: Se busca maximizar la satisfacción del cliente, lo que se mide en función de las recompensas y penalizaciones por los traslados de bicicletas. Se obtiene 1 euro por cada bicicleta que se transporta y que acerca la estación a la demanda prevista, y se resta 1 euro por cada bicicleta que se transporta y que aleja la estación de la demanda prevista.
- Capacidad de las Furgonetas: Cada furgoneta tiene una capacidad máxima de 30 bicicletas. Esto limita la cantidad de bicicletas que se pueden mover en un solo viaje.
- No mover bicicletas con dos furgonetas diferentes: No se pueden mover bicicletas de una estación utilizando dos furgonetas diferentes en la misma hora.
- Tamaño de la ciudad: La distribución de bicicletas se realiza en una ciudad cuadrada de 10x10 kilómetros con manzanas de 100x100 metros.

2.3 Criterios para evaluar la solución

Los criterios de evaluación de las soluciones del problema de optimización del reparto de bicicletas en Bicing son los siguientes:

- Maximización de beneficios: El objetivo es maximizar los ingresos totales. Cada transferencia exitosa (acercar la estación a la demanda esperada) genera un beneficio de 1€.
- Reducir los costes de transporte: El objetivo es reducir los costes generales asociados al transporte de bicicletas. El precio se calcula en base al número de bicicletas transportadas en el camión y el coste por kilómetro recorrido según la fórmula *((nb + 9) div 10)*.
- Satisfacción del cliente: Nos esforzamos en maximizar la satisfacción de los usuarios del Bicing. La satisfacción se mide en función de recompensas y multas por entregar bicicletas. Gana 1 euro por cada bicicleta trasladada que acerque la estación a la demanda esperada y resta 1 euro por cada bicicleta transportada que aleje la estación de la demanda esperada.

Estos criterios son esenciales para evaluar la calidad de la solución. Es imprescindible encontrar un equilibrio entre maximizar los beneficios, minimizar los costes y garantizar la satisfacción del usuario. Una solución eficaz debe lograr un beneficio neto elevado, respetando las limitaciones y manteniendo la satisfacción del usuario.

2.4 Justificación elección de búsqueda local

Optar por utilizar la búsqueda local para resolver el problema de optimización de la asignación de bicicletas en Bicing tiene sentido por varias razones. Primero, la búsqueda local es una forma eficiente de resolver problemas de optimización combinatoria como en este caso. En este ejercicio intentamos maximizar y minimizar aspectos de una solución consistente en traslados de bicicletas entre estaciones. Esta solución de la que partimos forma parte de un espacio de soluciones extenso. La búsqueda local puede explorar este espacio de soluciones de forma iterativa y eficiente. La escalabilidad es otra ventaja de la búsqueda local. Bicing puede tener una red de estaciones de diferentes tamaños y ubicaciones, y las

búsquedas locales se pueden ajustar para tener en cuenta estas diferencias. Además, la flexibilidad de la búsqueda local la hace adecuada para cumplir con limitaciones de problemas específicos, como la capacidad de los camiones, el número máximo de estaciones visitadas y las limitaciones de capacidad de las estaciones.

La esencia del problema está relacionada con la toma de decisiones locales, como trasladar una bicicleta de una estación a otra, lo que se presta bien a la optimización local. La optimización local tiene como objetivo mejorar las soluciones cercanas realizando pequeños cambios que sean relevantes para problemas que requieren soluciones prácticas. Además, dado que el problema es el coste y la rentabilidad de las bicicletas en movimiento, la búsqueda local puede ayudar a encontrar un equilibrio entre maximizar los beneficios y minimizar los costes, algo muy importante para el Bicing.

3 Implementación del proyecto

3.1 Implementación del estado

El estado, según lo explorado en el apartado anterior, se representará como una asignación de rutas a furgonetas que pasan por estaciones recogiendo o dejando bicicletas en ellas.

Para ello, utilizaremos las estructuras de datos:

- Un objeto Estaciones `stations`, con la información de las estaciones de la ciudad (proporcionada por Bejar).
- Un *array* de recorridos `routes`, con tantos elementos como furgonetas, donde el elemento i contiene el recorrido que hace la furgoneta de identificador i en forma de tripleta de `stop`, donde el primero será la parada de origen, el segundo la primera parada y el tercero la segunda parada.
- Un *array* útil para optimización que representa datos de las estaciones.
- Los objetos parada `stop` del array anterior, con la información de una parada en un recorrido. Esta es un par donde el primero es el identificador de la `station` donde se para y el segundo es el impacto que tenemos en su número de bicicletas (positivo indica introducir bicicletas, negativo indica extraer bicicletas).
- Un vector auxiliar `start_stations` que nos indica si la estación de índice i es el inicio de alguna ruta o no.
- Un vector auxiliar `impact_stations` que nos indica el impacto de una estación rápidamente.
- Un valor int *gain* auxiliar que nos permite calcular de forma incremental nuestro heurístico.
- Una matriz de ints *distances* que contiene para cada celda Cij la distancia entre la estación i y la estación j precalculada para no tener que hacer el cálculo cada vez.

3.2 Justificación Operadores

Al final, el conjunto que hemos escogido implementa 4 operadores: *addStop*, *removeStop*, *switchStop* y *jumpStartRoute*.

3.2.1 addStop(int van, int station)

Añade la estación *station* como parada al final de la ruta de la furgoneta *van*.

Para decidir el número de bicicletas que se toman (en caso de origen) o se dejan (en caso de destino), se toman decisiones de la siguiente forma de manera determinista: en caso de que la estación sea de origen, se toman el máximo número de bicicletas disponibles (calculadas como el número de bicicletas no usadas que no son necesarias para cubrir la demanda). Estas bicicletas pasarán a formar parte de las bicicletas en la reserva de la furgoneta. En caso de que la estación sea destino, se dejarán las bicicletas en la reserva de la furgoneta hasta que se cubra la demanda o se agoten las bicicletas en la furgoneta. Obviamente, las bicicletas que se dejan se restarán de la reserva de la furgoneta, por lo que las bicicletas dejadas en el primer destino afectarán a las del segundo.

Para comprobar que esta selección de bicicletas es correcta y no poda soluciones deseables, veamos por casos:

- Si la estación es de origen, sabemos que se nos descontará 1 euro si tomamos una bicicleta que aleje la estación de cubrir la demanda y 0 de lo contrario. Además, sabemos que ganaremos 1 euro si dejamos la bicicleta en una estación que tenga demanda por cubrir y 0 de lo contrario. Por lo tanto, el máximo beneficio que podemos sacar de una bicicleta si la tomamos bajo déficit de demanda es 0 (-1 euro al tomarla y 1 euro en el mejor caso al dejarla). Por lo tanto, hay que coger solo las bicicletas no usadas que estén por encima de la demanda. Además, no tenemos ningún coste por transportar bicicletas además del gas. Por lo tanto, la mejor opción es coger todas las bicicletas que quepan que cumplan esta condición.

- Si la estación es destino, solo ganaremos un beneficio de 1 euro si la estación aún tiene demanda por cubrir. Por lo tanto, solo nos conviene dejar bicicletas mientras la estación tenga demanda. Las demás es mejor dejarlas en reserva para el siguiente destino (y en caso de que no haya demanda, no hay penalización por dejar bicicletas en las furgonetas). Entonces, por una lógica similar a la anterior, esta toma de decisiones es correcta.

Para poder usarlo, debemos comprobar las siguientes restricciones:

- La furgoneta no tiene la ruta llena (es decir, no tiene ya asignadas 3 estaciones).

- En caso de que la estación asignada sea la primera estación de la furgoneta (es decir, su estación de origen), esta estación no debe ser el origen de otra furgoneta y debe tener un número de bicicletas disponibles (no usadas por encima de la demanda) mayor o igual a 0.

- En caso de que la estación asignada sea un destino, debe tener una demanda de bicicletas superior o igual a 0.

Al final de la ejecución de este operador, la furgoneta tendrá añadida en su ruta la parada indicada por los parámetros definidos anteriormente. Además, el número de bicicletas de la estación se ajustará acorde con las bicicletas que se quiten o añadan y, en caso de ser una estación de origen, se registrará para facilitar las comprobaciones posteriores.

El factor de ramificación de este operador es ***O(F*E)*** ya que el número de estados sucesores depende de la combinatoria de las furgonetas (F) con todas las posibles estaciones (E). Las bicicletas se deciden de manera determinista y no afectan en gran medida a la ramificación.

Este operador es obviamente necesario para poder construir estados solución a partir de aquellos que contemplen rutas incompletas o ausentes (como sería el estado "vacío" donde ninguna estación pertenece a ninguna ruta de furgonetas).

3.2.2 removeStop(int van)

Este operador quita la estación que se encuentra en la última posición de la ruta que hace la furgoneta *van*. Es decir, si una furgoneta tiene 1 estación de origen y 2 de destino, quedaría con 1 estación de origen y 1 de destino al final de la ejecución. Asimismo, desde este nuevo estado, volver a ejecutar el operador dejaría a la furgoneta con 1 estación de origen exclusivamente y aplicarlo una última vez dejaría a la furgoneta con una ruta vacía. Para poder usar este operador, hay que comprobar que:

- La furgoneta *van* tiene alguna estación en su ruta.

Al final de la ejecución de este operador, se quitará la última estación de la furgoneta. Además, esta estación tendrá el número de bicicletas ajustado acorde a las bicicletas que se quiten o añadan y, en caso de ser una estación de origen, se le quitará ese estado para que pueda ser usado por otras furgonetas.

El factor de ramificación de este operador es ***O(F)*** ya que depende exclusivamente de a qué furgoneta se le decide aplicar el operador.

Este operador es necesario para explorar todos los posibles estados solución, ya que junto con el operador anterior, ya satisfacen esa propiedad, lo que permite explorar todas las posibilidades como si se tratara de un árbol de retroceso. Por lo tanto, en caso de que se quiera modificar una estación de la ruta o explorar un camino alternativo, basta con usar el operador *removeStop* las veces que se quiera realizar el "retroceso" y luego añadir las nuevas vías con *addStop* (o dejar la ruta restante vacía, por supuesto).

3.2.3 switchStop(int van, int pos, int newStation)

Este operador sustituye la estación en la posición *pos* de la furgoneta *van* por la estación *newStation*. Las bicicletas repartidas en la ruta se recalculan por completo siguiendo la lógica explicada en el operador "AddStop".

Para usarlo, hay que comprobar que:

- La furgoneta *van* tiene una estación en la posición *pos* de su ruta.
- En caso de que *pos* sea 0, como se va a sustituir una estación de origen, debe cumplirse que *newStation* no sea una estación de origen de otra furgoneta y que tenga un número de bicicletas disponibles (no usadas por encima de la demanda) mayor o igual a 0.
- En caso de que *pos* sea 1 o 2, ya que se va a sustituir una estación de destino, *newStation* debe tener una demanda sin cubrir mayor o igual a 0.

Al final de la ejecución de este operador, la furgoneta *van* tendrá sustituida la estación de la posición *pos* por la nueva estación *newStation* y se recalculan todas las bicicletas repartidas durante la ruta para que sea de la forma más eficiente posible (en términos de ganancias). Obviamente, el número de bicicletas de cada estación también se actualizará y los cambios en estaciones de origen, en caso de ser necesario.

El factor de ramificación de este operador es ***O(F*N)*** ya que a cada posible furgoneta (F) se le puede sustituir cualquier posición por cualquiera de las estaciones (N) del problema.

Este operador no es necesario para explorar el espacio de soluciones, ya que los dos operadores anteriores ya satisfacen esa propiedad, pero puede ser útil para hacer cambios en las rutas cuando ya están muy saturadas.

3.2.4 jumpStartRoute(int van, int origStation, int destStation)

Este operador añade a la vez una ruta de origen y un primer destino a una furgoneta *van* con una ruta vacía. Es equivalente a realizar 2 veces el operador *addStop* en una misma furgoneta. La primera vez con *origStation* como parada a añadir y la segunda vez con *destStation*.

Para usarlo, hay que comprobar que:

- La furgoneta *van* tiene una ruta vacía.
- La estación *origStop* no es una estación de origen de otra ruta y tiene un número de bicicletas disponibles superior o igual a 0.
- La estación *destStop* tiene una demanda sin cubrir superior o igual a 0.

Al final de la ejecución de este operador, la furgoneta *van* tendrá añadidas las estaciones *origStop* y *destStop* a la ruta en este orden. Las asignaciones de bicicletas en cada una se calculan de la misma manera que en *addStop*, ya que, como se mencionó anteriormente, esta operación es equivalente a aplicar 2 veces dicho operador. Al igual que en todos los casos anteriores, el número de bicicletas de la estación y si es origen o no de una ruta se actualizará de acuerdo a lo que se necesite.

El factor de ramificación de este operador es $O(F * N^2)$ ya que a cada posible furgoneta (F) se le pueden añadir cualquier estación posible como origen (N) y cualquier otra como destino (N).

Este operador no es necesario para explorar el espacio de soluciones, ya que los dos operadores anteriores ya satisfacen esa propiedad, pero, como se ve en el experimento 1, es necesario para una mejor exploración de este espacio. Ya que añadir una estación de origen sola no aporta ningún beneficio inmediato, si no se hace uso de este operador auxiliar que permite "encender" rutas (de ahí el nombre JumpStart), habría que plantear un heurístico complejo que premie poner nuevos orígenes de ruta, aunque ésto vaya en contra del heurístico de coste de gas (que pretendemos minimizar) y solo potencialmente a favor del heurístico de ganancia de reparto de bicis (que pretendemos maximizar). Por lo tanto, se cree preferible tener este operador presente que un heurístico exageradamente complejo de calcular.

3.3 Estrategia para hallar la solución inicial

Para la creación de la solución inicial hemos decidido usar tres estrategias:

- Solución vacía: Para implementar la solución inicial más simple pensamos en otorgar una ruta nula a todas las furgonetas, consiguiendo así, el menor coste computacional.
- Solución random: Solución simple que se basa en otorgar una ruta aleatoria a cada furgoneta.
- Solución óptima: Solución más compleja con un coste computacional más elevado. La estrategia usada para implementar esta solución se basa en poner todas las estaciones con mayor superávit de bicicletas como las estaciones origen para las furgonetas (haciendo uso de una función TopK que tiene un coste cuadrático respecto al número de estaciones). A continuación, la primera estación destino de la ruta de cada furgoneta es la que está más cerca de la origen (con un coste lineal respecto al número de furgonetas por el número de estaciones), dejando allí el mayor número de bicicletas.

Con estas tres estrategias para crear la solución inicial, podemos observar las diferencias entre usar una solución inicial simple con poco coste computacional, otra solución inicial simple pero con mayor coste computacional y una solución inicial compleja con mayor coste computacional.

3.4 Funciones heurísticas

Las funciones heurísticas que utilizamos son básicamente dos:

- **Heurístico Simple:** El heurístico simple toma en cuenta solo los aspectos de beneficio/penalización a nivel de estación que se nos comentan en los criterios para evaluar la solución. Por lo tanto, este simplemente lo que hace es establecer una proporcionalidad directa con la recompensa (recompensando un euro por bici otorgada que reduce la distancia de la demanda al next) y una proporcionalidad inversa con la penalización (penalizando un euro por bici sustraída de un lugar que aumenta la distancia de demanda al next). Esto se consigue con el valor auxiliar gain que lleva calculado para el estado actual como va la penalización contra la recompensa. Este no es muy bueno, pues obvia completamente el costo de la gasolina. Si bien no es realista, maximiza correctamente la ganancia.

- **Heurístico Complejo:** Este heurístico debe maximizar lo mismo que el anterior mientras minimiza un nuevo factor, el costo de la gasolina. Este se calcula mediante la función descrita en los criterios para evaluar la solución y usa las distancias precalculadas para agilizar el proceso. Por lo tanto, este heurístico usa con proporcionalidad directa el primer heurístico simple de ganancia y añade con proporcionalidad inversa el nuevo factor a minimizar del costo de la gasolina con una resta. De esta forma, maximizando esta operación entre ambos conseguimos maximizar la ganancia y minimizar el costo.

4 Experimentos

Una vez realizadas las tareas de programación y obtenidas las codificaciones para tener diversas formas de abordar el problema, vamos a realizar una exhaustiva experimentación que nos permita definir qué combinación de las mismas nos brindará el mayor rendimiento y potencial para elegir la solución adecuada.

El entorno de experimentación en el que sucederán estos experimentos será el siguiente:

- **PC personal**
  - **Procesador:** AMD Ryzen 5 3600X 6-Core Processor 3.79 GHz
  - **GPU:** NVIDIA RTX 3070 Ti
  - **RAM:** 16.0 GB
  - **Sistema Operativo:** Windows 10 Home
  - **Entorno de desarrollo:** Eclipse

4.1 Determinar Conjunto de Operadores

El primer paso a hacer será definir nuestros operadores, los cuales serán utilizados en el resto de experimentos de ahora en adelante. Esto lo haremos mirando de optimizar el primer criterio (el heurístico simple en el cual no hace falta tener en cuenta el transporte).

Observación	El conjunto de operadores juega un papel esencial en cómo atravesamos el espacio de soluciones. Esto afecta tanto la calidad de la solución que obtenemos como el tiempo que tarda en encontrarla y los pasos que realiza.
Planteamiento	Usaremos 4 conjuntos de operadores según su granularidad (cuán grandes son los pasos que dan): *granularidad gruesa* (grandes pasos a nivel de ruta), *granularidad media* (pasos medios a nivel de parada), *granularidad fina* (pasos a nivel de elementos de la parada) y *granularidad exhaustiva* (pasos muy finos que exploran absolutamente todas las posibilidades en cada paso).
Hipótesis	Usar una granularidad más fina (tener más control sobre las operaciones y las bicicletas que se toman de cada estación) nos permitirá una mejor solución (H0) o la granularidad más fina no mejora la solución.
Método	• Elegiremos 5 semillas aleatoriamente[1](entre 0 y 10000), una para cada réplica de dos inicializaciones. • Usaremos las dos inicializaciones en vez de una. Debido a que a pesar de que los sucesores recorren todo el espacio de soluciones, el heurístico penaliza mucho las mesetas y depende del inicio que tomemos podemos observar o no cómo nuestros sucesores se comportan ante las mismas. Esto es necesario para ver qué tan flexibles son además del tiempo. • Ejecutaremos el experimento una vez por conjunto de operadores, ya que son deterministas. • Experimentaremos con problemas con 25 estaciones, 1250 bicicletas, 5 furgonetas y 5 paradas. Número fijo de bicicletas (1250), estaciones (25) y furgonetas (5), cantidad de paradas fijas (5) ya que no hay mucho en que experimentar en cuanto a la granularidad. • Usaremos la norma *Manhattan* para las distancias (con lo que tendremos $5 \cdot 25 = 125$ distancias). • Usaremos 4 conjuntos de operadores (grueso, medio, fino y exhaustivo). • Haremos 5 réplicas para la operación *Reset*, es decir, para los sucesores no necesitaremos realizar las réplicas ya que sus operadores son deterministas.

A continuación de la definición de la metodología y la hipótesis procedemos a mostrar y valorar los resultados observando los tres factores a comparar.

Dibujaremos boxplots con estos resultados de los factores observables para poder comparar visualmente las diferencias entre los grupos de operadores y poder justificarlas adecuadamente.

Adicionalmente, podemos crear una tabla con la media y desviación estandar de las distribuciones para comparar estos aspectos estadísticos.

Primero analizemos los conjuntos de operadores y sus características:

4.1.1 Conjuntos de operadores

Para los factores de ramificación seguiremos la siguiente nomenclatura: F es el número total de furgonetas disponibles, S el número total de estaciones en el problema, B el número total de bicicletas. También pondremos los valores constantes ya que hemos visto que han resultado significativos en la experimentación (En un caso donde sólo hay 5 furgonetas y 25 estaciones, un coste de $O(F \cdot S)$ no es lo mismo que $O(F \cdot S \cdot 60)$)

Los operadores básicos que hemos creado son:

- **AddStop:** Añade una parada a la ruta de una furgoneta si hay sitio. Si es estación de origen, toma el máximo número de bicis disponibles (no usadas y por encima de la demanda de la estación) y si es un destino, deja las máximas bicis que queden en la furgoneta o hasta satisfacer la demanda. Su factor de ramificación es $O(F \cdot S)$.
- **JumpStartRoute:** Sería el resultado de aplicar 2 AddStop a una ruta vacía de una furgoneta. Su factor de ramificación es de $O(F \cdot S^2)$.
- **SetFullRoute:** Sería el resultado de aplicar 3 AddStop a una ruta vacía de una furgoneta. Su factor de ramificación es de $O(F \cdot S^3)$.
- **RemoveRoute:** Vacía al completo la ruta de una furgoneta. Su único requisito es que haya alguna estación en la ruta. Su factor de ramificación es de $O(F)$.
- **RemoveStop:** Igual que el anterior, pero solo quita la última estación de la ruta en vez de la ruta entera. Su factor de ramificación es de $O(F)$.
- **SwitchStop:** Cambia una estación de la ruta de una furgoneta por otra estación especificada como parámetro. Recalcula las bicis acorde a la mejor estrategia posible (coger máximas disponibles en el origen y en cada destino intentar dejar hasta cubrir demanda). Su factor de ramificación es de $O(F \cdot S)$.
- **ChangeImpact:** Permite cambiar el número de bicis que se cogen (en caso del origen) o dejan (en caso de un destino) en una estación de la ruta. Intenta aprovechar bicis cargadas por la furgoneta que no se usan. Su factor de ramificación es de $O(F \cdot S \cdot 60)$.
- **ChangeFlow:** Igual que el anterior, pero permite "fluir" bicis entre estaciones. Es decir, permite pasar un número especificado de bicis del origen al primer destino o del primer destino al segundo. Su función es redistribuir la carga en la ruta. Su factor de ramificación es de $O(F \cdot 60)$.

[1] Las semillas serán: 4243, 1451, 7452, 6427 y 9920.

- **AddStopOld:** Igual que AddStop, pero permite escoger las bicis tomadas o dejadas en cada estación. Por lo tanto, su factor de ramificación es de $O(F \cdot S \cdot 60)$.
- **AddTwoStopOld:** Igual que JumpStartRoute, pero permite escoger las bicis tomadas y dejadas tanto en el origen como en el destino definidos. Siempre que esas bicis tengan sentido contextualmente. Por lo tanto, su factor de ramificación es mayor, de $O(F \cdot S^2 \cdot 60^2)$.
- **SwitchStopOld:** Igual que SwitchStop, pero permite escoger el número de bicis que tiene la nueva estación. Su factor de ramificación es de $O(F \cdot S \cdot 60)$.

Veamos ahora las granularidades (conjuntos de operadores) escogidas:

4.1.2 Conjuntos de operadores:

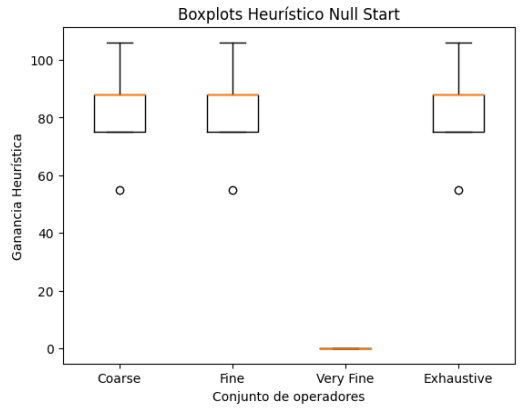
- **Gruesa:** Contiene los operadores JumpStartRoute, SetFullRoute y RemoveRoute. Esencialmente opera sobre el espacio de soluciones a nivel de ruta. JumpStartRoute y SetFullRoute permiten añadir rutas de 2 y 3 estaciones respectivamente (son las únicas opciones interesantes puesto que con solo 1 estación no se gana beneficio). RemoveRoute permite eliminar rutas al completo. Ya que a veces lo mejor es no tomar ninguna acción. Además también permite cambiar rutas ya configuradas por otras por explorar. Por lo tanto permite explorar todo el espacio de soluciones relevante a nivel de ruta.
- **Media:** Contiene los operadores JumpStartRoute, AddStop, RemoveStop y SwitchStop. Principalmente opera sobre las estaciones pero permite acciones de "orden superior" como JumpStartRoute que facilitan la exploración del espacio de soluciones en estados con soluciones esparsas, pero sin llegar a la potencia de configuración de rutas de la granularidad gruesa ni el detalle en bicis operadas de la granularidad fina. Permiten explorar todo el espacio de soluciones útiles gracias a addStop y removeStop que esencialmente cubren todas las posibles configuraciones en un estilo similar al backtracking. Los demás operadores són para facilitar cálculos.
- **Fina:** Contiene los operadores addStop, removeStop, switchStop, changeImpact y changeFlow. Opera exclusivamente a nivel de estación pero además añade operadores que permiten cambiar el reparto de las bicis como changeImpact y changeFlow. Al igual que el anterior, gracias a las operaciones de addStop y removeStop permite explorar todo el espacio de soluciones. Esta granularidad es interesante porque permite ver como distintos operadores "auxiliares" afectan a la búsqueda en el espacio de soluciones, sobretodo comparado con el anterior.
- **Exhaustiva:** Contiene los operadores AddStopOld, AddTwoStopOld, SwitchStopOld y RemoveStop. Este es bastante similar a la granularidad fina en el sentido que tiene operadores principalmente a nivel estación pero también tiene operadores auxiliares que le facilitan el recorrido como AddTwoStop que es similar a JumpStartRoute. Por lo tanto, como tiene AddStopOld y RemoveStop puede explorar todo el espacio de soluciones de una forma similar. La característica interesante de este conjunto es que en cada operación permite configurar el número de bicis que se toman o dejan. Por lo tanto, permite un control al completo del problema.

Generador Inicial	Granularidad	Ganancia	Nodos Expandidos	Tiempo en ms
Vacío	Gruesa	82.4(16.883)	6(0.0)	82.2(5.912)
Vacío	Media	82.4(16.883)	6.8(0.748)	26(1.09)
Vacío	Fina	0.0(0.0)	1(0.0)	10.4(1.854)
Vacío	Exhaustiva	82.4(16.883)	6.8(0.748)	1217(229.47)
Óptimo	Gruesa	57.0(9.38)	1(0.0)	9.4(0.489)
Óptimo	Media	82.4(16.883)	4(0.894)	20.2(1.166)
Óptimo	Fina	82.4(16.883)	4(0.894)	27.0(1.549)
Óptimo	Exhaustiva	82.4(16.883)	4(0.894)	271.2(9.947)

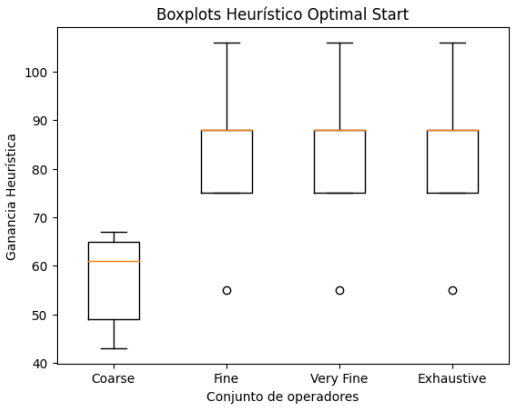
Table 1: Tabla de resultados medios(y con desviación estándar) de cada conjunto

Vemos en la siguiente tabla los resultados generados en este experimento:

Esta tabla ya nos permite ver información bastante interesante que nos puede ayudar a decidir el conjunto de operadores más apropiado para este problema (como por ejemplo el hecho que el conjunto exhaustivo tarde un tiempo 10 o 100 veces superior a los demás), pero para llegar a conclusiones buenas vamos a visualizar los resultados en boxplots:



(a) Heurístico vs Operadores, Inicio vacío

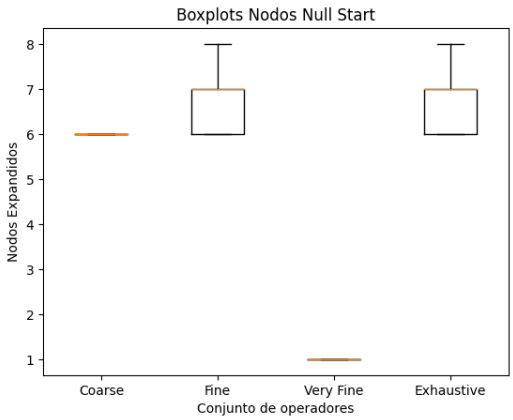


(b) Heurístico vs Operadores, Inicio óptimo

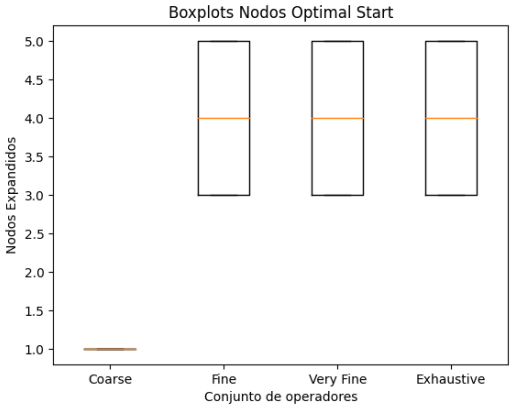
Figure 1: Comparación entre el comportamiento de cada conjunto de operadores dependiendo del estado inicial

Este boxplot nos muestra como rinde cada conjunto de operadores con cada solución inicial. Vemos que en el caso de solución inicial nula el conjunto "Very Fine" es tiene un resultado totalmente nefasto, aunque se comporta igual de bien que la mayoría de los demás conjuntos con la solución óptima. Cosa que nos indica que es posible que "Very Fine" necesite un empuje inicial para funcionar correctamente y se atasca en la parte de iniciación de nuevas rutas. Esto tiene bastante sentido ya que la única forma con la que "Very Fine" puede añadir nuevas estaciones a una ruta es de una en una con AddStop. Como añadir una estación de origen a una ruta no aporta ningún beneficio inmediato, es posible que se quede atascado en una meseta inicial. Por otro lado, vemos como el conjunto "Coarse" se comporta ligeramente peor en el caso de partir del generador de solución inicial óptimo. Esto podría darse por la falta de capacidad de hacer retoques finos que tiene este conjunto de operadores. Pero no sabemos hasta que punto afecta con esta información.

Esta gráfica de nodos expandidos nos muestra cómo se ha comportado cada conjunto



(a) Nodos Expandidos vs Operadores, Inicio vacío



(b) Nodos Expandidos vs Operadores, Inicio óptimo

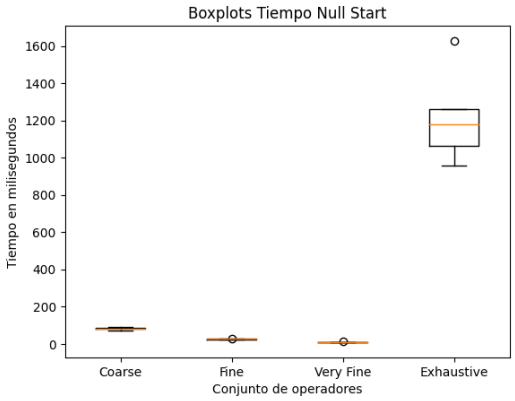
Figure 2: Comparación entre el comportamiento de cada conjunto de operadores dependiendo del estado inicial

de operadores en cada caso. Aquí confirmamos por un lado que "Very Fine" es totalmente incapaz de operar sobre rutas vacías. Esto es malo ya que no sólo hace que sea incapaz de empezar desde la solución vacía, sinó que en cualquier otro estado donde se llegue a conseguir rutas vacías (generadores iniciales random o operaciones de Simulated Annealing), esas rutas estarán condenadas a permanecer vacías para siempre y ser totalmente desaprovechadas. Por lo tanto, la poca flexibilidad de éste conjunto lo hace indeseable. Por otro lado, el conjunto "Coarse" brinda buenos resultados cuando se parte de la solución vacía, puesto que consigue la misma ganancia heurística con menos nodos expandidos, pero por otro lado es totalmente incapaz de expandir nuevos nodos a partir de rutas parciales. Probablemente esto se deba a que eliminar rutas sin añadir nuevas no aporte nuevo beneficio, haciendo que nunca se expandan esos estados. Esto se podría solucionar con un heurístico diferente o usando otros algoritmos como Simulated Annealing que permitan este tipo de movimientos, pero a priori la baja flexibilidad de este conjunto lo hace ligeramente indeseable también.

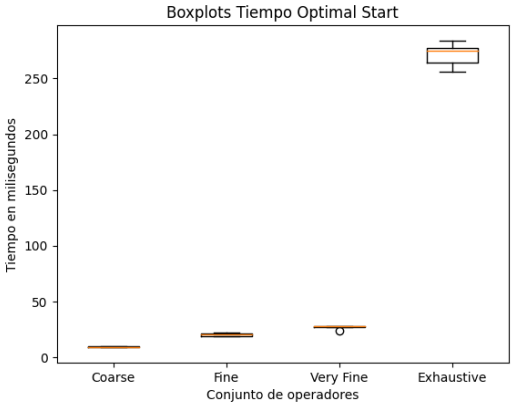
Entonces la cuestión final será sobre los tiempos de ejecución. Puesto que vemos que tanto el conjunto "Fine" como el conjunto "Exhaustive" brindan resultados iguales. De momento no tenemos información para confirmar o desmentir la hipótesis.

Aquí vemos como hay una clara diferencia en los tiempos. Aunque tanto "Fine" como "Exhaustive" rinden igual, "Exhaustive", probablemente debido a sus elevados factores de ramificación, tarda 10 o incluso 100 veces lo que tarda el "Fine" en llegar a una solución final. Por lo tanto, parece que el mejor conjunto a usar es "Fine" debido a su rendimiento, flexibilidad y eficiencia temporal.

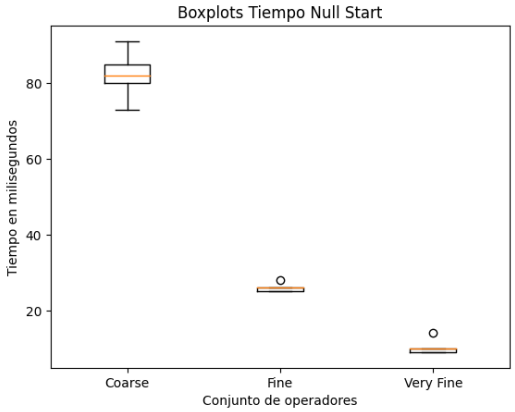
Esto nos desmiente la hipótesis que cuánta más fina la granularidad, mejores resultados. Hemos visto que configuraciones como "Very Fine" directamente son incapaces de explorar el espacio de soluciones pues, aunque teóricamente "AddStop" y "RemoveStop" seria lo único que se requiere para explorarlo todo, el heurístico desincentiva quitar paradas de rutas en este experimento. Con heurísticos más complejos (como por ejemplo el que tiene en cuenta el coste de transporte) o otros algoritmos de búsqueda (Simulated Annealing) podríamos ver otro tipo de resultados, pero de momento parece que funciones auxiliares como "JumpStartRoute" o "AddTwoStop" que circumventan estos casos van mejor. De



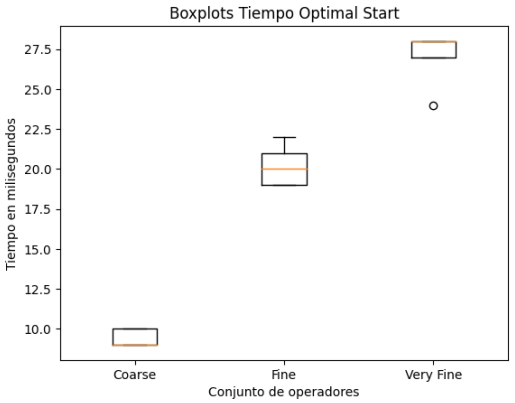
(a) Tiempos vs Operadores, Inicio vacío



(b) Tiempos vs Operadores, Inicio óptimo



(c) Tiempos vs Operadores, sin Exhaustive. Inicio vacío



(d) Tiempos vs Operadores, sin Exhaustive. Inicio vacío

Figure 3: Comparación entre el comportamiento de cada conjunto de operadores dependiendo del estado inicial

hecho, si "Exhaustive" no tuviera este operador auxiliar, probablemente se quedaría atorado como "Very Fine". En conclusión, para maximizar flexibilidad y eficiencia, necesitamos una combinación de operadores con granularidad baja y alta.

4.2 Determinar Estrategia de Solución Inicial

El siguente paso en nuestra exploración experimental de las opciones que tenemos será determinar que solución inicial nos brindará los mayores beneficios. Para ello optimizamos de nuevo según el primer criterio simple.

Observación	La solución inicial también puede tener relevancia en el Hill Climbing puesto que dónde empecemos en el espacio de soluciones puede condicionar la calidad de la solución alcanzamos (si encontramos un máximo local o nos atascamos en una meseta). Además de influir claramente en el coste temporal.
Planteamiento	Usaremos 3 generadores iniciales (dos más complejos y otro trivial): *Inicio Máximo (Complejo)* (buscamos una solución prometedora de forma ágil), *Inicio Rand (Complejo)* (buscamos una solución completamente aleatoria de forma ágil), *Inicio Vacío (Simple)* (tomar el punto inicial que se encuentra dentro del espacio de soluciones más simple que existe).
Hipótesis	La elección de estado inicial es indiferente (H0) o hay métodos mejores que otros.
Método	• Elegiremos 10 semillas aleatoriamente (entre 0 y 10000) [2]. • Ejecutaremos 1 experimento por inicializacion si son deterministas y 3 y la media si son rand. • Experimentaremos con problemas con 25 estaciones, 1250 bicicletas, 5 furgonetas y demanda equilibrada como anteriormente. • Usaremos Hill Climbing. • Compararemos tres factores: los tiempos de ejecución usando *currentTimeMillis*, el coste de la solución y el numero de pasos.

Table 2: Bases de experimentación para la selección de Estado inicial

A continuación de la definición de la metodología y la hipótesis procedemos a mostrar y valorar los resultados observando los tres factores a comparar. Estos nos ayudaran a demostrar o desmentir la hipotesis planteada.

Crearemos una tabla con la media y desviación estandar de las distribuciones para comparar estos aspectos estadísticos.

Estado inicial	Distance	Ganancia	Nodos Expandidos	Tiempo en ms
Null	31460.0(9914.85)	83.2(22.02)	6.5(0.67)	6.20(5.81)
Optim	29240.0(8146.06)	83.5(22.46)	4.1(1.22)	0.8(0.60)
Rand	34876.67(10631.53)	80.47(20.37)	6.90(0.91)	1.07(0.29)

Table 3: Tabla de resultados medios(y con desviación estándar) de cada inicializacion

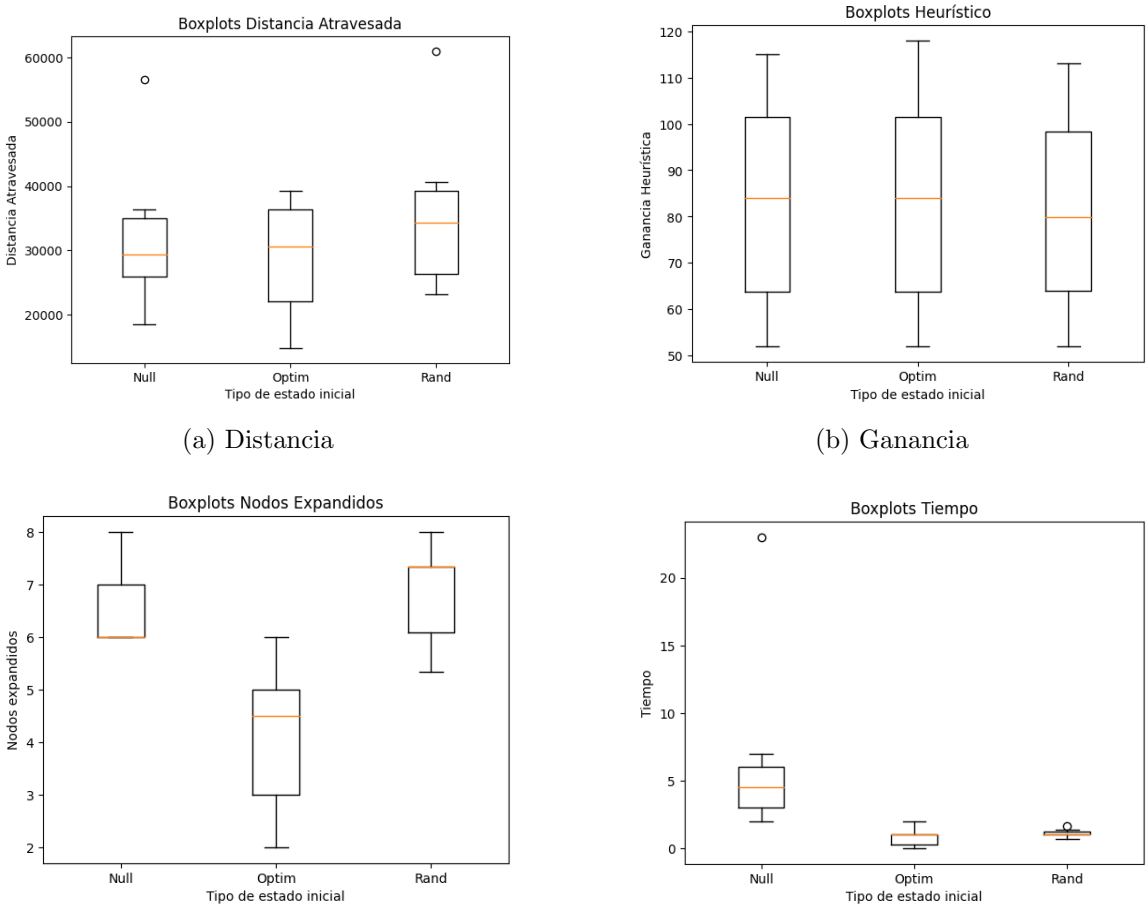
En la figura 3 podemos empezar a ver tendencias que nos apuntan a rechazar nuestra

[2]Las semillas usadas en este experimento son: 9969, 2344, 1705, 7838, 2724, 8407, 1714, 6428, 2806, 4677

hipotesis. En primer lugar para la distancia vemos que la media se encuentra algo reñida entre el inicio vacío y el optimo (estando esta mas favorable para el segundo incluso en desviacion estandar). En ganancia ambas funcionan de forma bastante similar destacando que el random obtiene una peor ganancia de media. Finalmente en nodos epandidos y tiempo es donde vemos la increible superioridad de óptim, el cual con una rápida inicializacion más inteligente baja los tiempos enormemente (tanto de media como en desviación estandar) y los nodos expandidos de media.

Por lo tanto, sospechamos que nuestra hipotesis no es cierta y si hay estados de inicialización que nos aportan un pequeño avance respecto a otros.

Adicionalmente, dibujaremos boxplots con estos resultados de los factores observables para poder comparar visualmente las diferencias entre las soluciones iniciales y poder justificarlas adecuadamente.



(a) Distancia

(b) Ganancia

(c) Nodos expandidos

(d) Tiempo

Figure 4: Boxplots para determinar solución inicial

En estos se ve de forma más clara y explicativa como la solución óptima mitiga ciertos outliers de distancia (que rand y null no son capaces de gestionar) y a nivel de tiempo y nodos expandidos este se encuentra muy por debajo de las dos alternativas pues da algo de trabajo inteligente ya masticado al algoritmo poniendolo en una posición ventajosa en

la escalada a realizar por el Hill Climbing. Esta posición más ventajosa adicionalmente no se ve perjudicada por un bajón de calidad en cuestión del heurístico cosa que es muy positiva. Esta preocupación nace a raiz de que tal vez si comenzamos en un punto encontrado de forma inteligente llegamos a un máximo local demasiado temprano y nuestro algoritmo no es capaz de llegar a opciones mejores. Pero aparentemente alcanza soluciones igual de buenas o mejores que los otros algoritmos.

Esto sumado a su mejor eficiencia generalizada nos hace optar por el mismo desmintiendo nuestra hipotesis positiva dando lugar a pensar que si hay una alternativa mejor a tomar y esta es la óptima.

4.3 Determinar Parametros para SA

En este experimento, nos adentraremos en la optimización de parámetros cruciales para el algoritmo de Simulated Annealing, manteniendo constantes otros elementos importantes, como el escenario, la función heurística, los operadores y la estrategia de generación de la solución inicial. El Simulated Annealing es una técnica metaheurística ampliamente utilizada en la resolución de problemas de optimización combinatoria, y su rendimiento depende en gran medida de la configuración adecuada de sus parámetros. La búsqueda de los valores óptimos de estos parámetros es fundamental para obtener resultados óptimos en la resolución de problemas complejos. En este contexto, exploraremos cómo la variación de los parámetros impacta en la eficacia y eficiencia del Simulated Annealing en un escenario específico, permitiéndonos entender mejor cómo adaptar este algoritmo a diferentes contextos de optimización.

Este experimento se basa entonces en:

Observación	La combinación de los dos parametros del algoritmo puede variar drasticamente sus resultados.
Planteamiento	Probaremos diferntes tipos de lambdas (estrategia de enfriamiento) y k (temperatura) para el mismo escenario con el algoritmo Simulated Annealing.
Hipótesis	Como más grandes sean los parametros mejores seran los resultados obtenidos (H0).
Método	• Elegiremos 10 semillas aleatoriamente (entre 0 y 10000). • Para cada semilla ejecutaremos todas las combinaciones de los dos parametros y calcularemos las medias de los resultados. • Experimentaremos con problemas con 25 estaciones, 1250 bicicletas, 5 furgonetas y demanda equilibrada. • Compararemos los tiempos de ejecución usando *currentTimeMillis*, la distancia total recorrida y el beneficio total.

Table 4: Bases de experimentación para la observación de las diferencias de en hora punta

En este experimento, se observa que el valor de K (temperatura inicial) no afecta significativamente el rendimiento del Simulated Annealing. En cambio, el valor de lambda (factor de enfriamiento) tiene un impacto considerable. Un valor menor de lambda resulta en una mayor ganancia media (aunque no muy notoria), pero también aumenta la cantidad de nodos expandidos y el tiempo de ejecución. La elección de K y lambda debe equilibrarse en función del tiempo disponible y la necesidad de encontrar soluciones óptimas o cercanas. Esto resalta la importancia de ajustar estos parámetros en función de las características específicas del problema.

K	Lambda	Mean Gain	Mean Nodes Expanded	Mean Time (ms)
1	1.0	105.0	801	163
1	0.1	105.0	7501	692
1	0.01	105.0	74601	4567
1	0.001	105.0	100001	5831
5	1.0	101.0	801	96
5	0.1	105.0	7501	558
5	0.01	105.0	74601	4220
5	0.001	105.0	100001	5663
25	1.0	101.0	801	44
25	0.1	107.0	7501	430
25	0.01	105.0	74601	4167
25	0.001	105.0	100001	5535
125	1.0	101.0	801	42
125	0.1	105.0	7501	399
125	0.01	105.0	74601	4070
125	0.001	105.0	100001	5537

Table 5: Tabla que muestra como los valores de K y Lambda afectan a la ganancia heurística obtenida, nodos expandidos y tiempo

Para nuestro problema, hemos decidido usar un valor de K y lambda de 1 en ambos casos, puesto que aunque eso no da la mejor ganancia absoluta, da un valor muy cercano y con 100 veces menos nodos expandidos y un aumento en eficiencia temporal 4 veces mayor.

4.4 Tiempo de ejecución en funcion de tamaño

Ahora una vez definido correctamente y de forma experimental las decisiones que tomamos para nuestro algoritmo exploremos como este se comporta a nivel temporal si escalamos el tamaño del problema. Este crecimiento ira altamente relacionado con el factor de ramíficacion de nuestros operadoes y el espacio de soluciones que hay a recorrer. Este crecimiento es presuntamente polinomico porque el factor de ramíficacion de la gran mayoria de operadores es sobre el numero de camiones por el número de estaciones y en ocasiones este ultimo valor puede llegar a ser cuadrático como peor caso $O(E^2 * F)$. En el caso del espacio de búsqueda también es polinomico con un valor de $O(E^4 * B * F)$, esto se debe a la combinatoria de 3 posibles estaciones en rutas por todas las furgonetas que pueden crear rutas $E^3 * F$ multiplicado al numero de bicis que puede albergar cada estación $E * B$.

Las bases del experimento se resumen en esta tabla:

Observación	El tiempo aumenta a medida que el tamaño crece, hemos de descubrir que funcion creciente sigue.
Planteamiento	Seleccionaremos diferentes tamaños de problema para observar como se comporta el tiempo de ejecución ante una evolucion lineal de tamaño.
Hipótesis	La función será pólinomica, al menos cuadrática respecto al tamaño (H0).
Método	• Elegiremos 5 semillas aleatoriamente (entre 0 y 10000). • Ejecutaremos 5 experimentos por tamaño (uno por semilla pues es determinista). • Comenzaremos con 25 estaciones y unas proporciones 1 a 50 entre estaciones y bicicletas y 1 a 5 entre furgonetas y estaciones. • Las estaciones aumentarán de 25 en 25. • Usaremos Hill Climbing. • Usaremos el heurístico simple con inicialización [blank]. • Compararemos únicamente los tiempos de ejecución usando *currentTimeMillis*.

Table 6: Bases de experimentación para la observación de la evolución temporal

Una vez realizado el experimento obtenemos unos resultados que son bien resumidos por las estadisticas que se pueden observar en la tabla donde mostramos la media y la desviación de tiempo de ejecución (ms):

Tamaño Problema	Tiempo en ms
25 Estaciones	14.6 (1.9235)
50 Estaciones	30.8 (4.3140)
75 Estaciones	55.8 (5.9158)
100 Estaciones	113.8 (8.2014)
125 Estaciones	151.4 (41.0455)

Table 7: Tabla de resultados medios (y con desviación estándar) de cada Tamaño

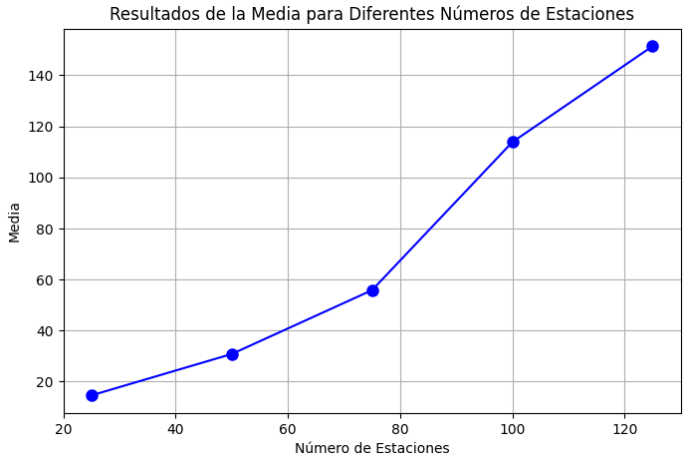


Figure 5: Your Image Caption

En resumen, el experimento proporciona evidencia sólida de que el tiempo de ejecución de un algoritmo de optimización, en este caso Hill Climbing, está intrínsecamente relacionado con el tamaño del problema. Los resultados respaldan mayoritariamente la hipótesis de un crecimiento temporal polinomico con respecto al tamaño del problema (H0), inicialmente se ve una evolución quadrática hacia el final obtenemos valores que apuntarian a esta evolución pero son acompañados por otros más bajos (la desviacione estandar se dispara). Esta conclusión tiene implicaciones importantes para la toma de decisiones en logística, ya que ilustra cómo el aumento en la complejidad del problema, reflejado en un mayor número de estaciones y, por lo tanto, más rutas y combinaciones posibles, se traduce en un aumento de tiempo que evoluciona de forma manejable. Además, la creciente variabilidad en los tiempos de ejecución a medida que el problema se vuelve más grande subraya el hecho de que ante problemas más grandes es posible que las cimas se encuentren en momentos muy diferentes. Estos hallazgos no solo tienen relevancia teórica, sino también aplicaciones prácticas en la gestión de la logística en empresas y organizaciones, ya que permiten anticipar el tiempo necesario para resolver problemas de enrutamiento en entornos de la vida real a tiempo y ajustar en consecuencia los recursos y las expectativas.

4.5 Diferencias entre Heurísticos

En este apartado nos toca ver cómo quedan el Simulated Annealing y el Hill Climbing cara a cara para cada uno de los dos heurísticos que hemos desarollado para la práctica. Puesto que ya disponemos de los resultados de esta experimentación para el heurístico simple, durante este apartado realizaremos la experimentacion del heurístico complejo y los compararemos todos.

Las bases de este experimento se basan en:

Observación	El tiempo aumenta a medida que el tamaño crece, hemos de descubrir que funcion creciente sigue.
Planteamiento	Seleccionaremos el tamaño del problema del apartado 1 y comprobaremos con múltiples ejecuciones como se comportan el algoritmo de Hill Climbing y el de Simulated Annealing según la selección del heurístico simple o el complejo.
Hipótesis	Hill Climbing obtiene mejores beneficios en general que el Simulated Annealing para ambos heurísticos (H0).
Método	• Elegiremos 5 semillas aleatoriamente (entre 0 y 10000). • Ejecutaremos 1 experimento por semilla para Hill Climbing pues es determinista y 3 experimentos por semilla para Simulated Annealing porque tiene un factor random. • Ya disponemos de apartados anteriores simulaciones de Hill Climbing y Simulated Annealing con el heurístico simple en los experimentos 2 y 3. • Experimentaremos con problemas con 25 estaciones, 1250 bicicletas, 5 furgonetas y demanda equilibrada como anteriormente. • Una vez tengamos los nuevos valores compararemos como rinden los heurísticos entre los algorítmos HC y SA. • Compararemos los tiempos de ejecución usando *currentTimeMillis*, la distancia total recorrida y el beneficio total.

Table 8: Bases de experimentación para la observación de las diferencias de beneficio entre heurísticos

Una vez hecha la experimentación podemos mostrar ilustrativamente el coste, la distancia total recorrida y el tiempo de ejecución para Hill Climbing y el Simulated Annealing con el heurístico complejo con estos resultados experimentales:

Semilla	Ganancia	Coste Gasolina	Distancia	Tiempo en ms
3847	67.0	29.5	18400	34
7192	69.0	15.10	6900	56
2356	47.0	12.2	7200	45
6021	64.0	8.5	5900	47
9324	68.0	14.0	7200	26

Table 9: Tabla de resultados del Hill Climbing heurístico complejo

Semilla	Ganancia	Coste Gasolina	Distancia	Tiempo en ms
3847	53.74	25.40	43422.85	114.94
7192	52.93	23.07	31821.06	131.56
2356	38.45	14.39	27334.21	178.39
6021	47.80	8.12	17619.46	119.30
9324	69.90	30.24	28812.74	127.05

Table 10: Tabla de resultados del Simulated Annealing heurístico complejo

Y esto recordando como se comportaban ambos con el heurístico simple tomado nuevas ejecuciones finalmente para usar las mismas semillas de comparación:

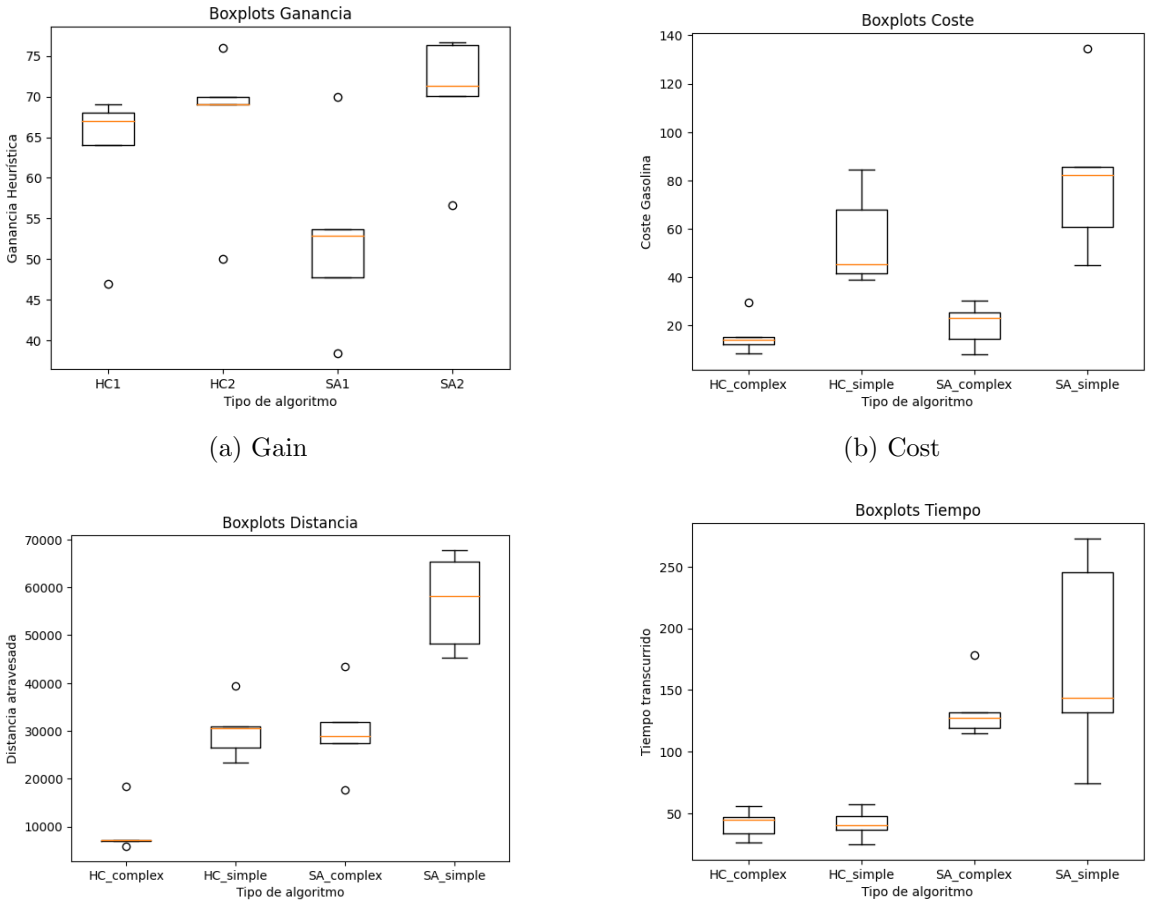
Semilla	Ganancia	Coste Gasolina	Distancia	Tiempo en ms
3847	76.0	84.5	39400	48
7192	69.0	68.0	30500	37
2356	50.0	45.3	31000	40
6021	70.0	38.8	26500	57
9324	69.0	41.4	23400	25

Table 11: Tabla de resultados del Hill Climbing heurístico simple

Semilla	Ganancia	Coste Gasolina	Distancia	Tiempo en ms
3847	76.28	134.60	67823.32	272.94
7192	70.12	85.73	48177.71	144.01
2356	56.67	44.83	45333.33	132.12
6021	71.33	82.26	65403.82	74.53
9324	76.67	60.56	58130.60	245.33

Table 12: Tabla de resultados del Simulated Annealing heurístico simple

Visualicemos boxplots de ambos para obtener una idea de como son estos valores:



(a) Gain

(b) Cost

(c) Distance

(d) Time

Figure 6: Boxplots para determinar que algoritmo y Heurístico produce mejor solución

Podemos observar que todos generalmente exploran ganancias similares, habiendo una sutil bajada en aquellos que usan el heurístico complejo (HC1 y SA1). Esto se debe a que el heurístico complejo tiene en cuenta el coste. Puede parecer que ambos obtienen resultados similares pero si nos centramos en este coste del que hablabamos vemos que el heurístico complejo supone una mejora sustancial de este mismo haciendo que la distancia que hemos de recorrer y el coste gasolina sea mucho menor. Ahora bien, en cuestiones de tiempo no tiene mucho impacto el cambio de heurístico en el Hill Climbing pero en el Simulated Annealing es bastante notoria una perdida de eficiencia asociada a calcular este heurístico más complejo. Esto se debe a que por falta de tiempo no pudimos terminar de optimizar su función y se recalcula a menudo.

Una vez visto esto podemos ver que nuestra hipotesis si bien no es claramente cierta (pues los beneficios son similares) los costes del SA son mayores de media, recorre mas distancia de media y tarda más de media. Así que en la mayoría de aspectos se comporta mejor.

Ahora comparando los dos heurísticos evidentemente el que maximiza solo el gain lleva a este a valores más altos pero a costa de gastos exagerados. En el mundo real el segundo heurístico es mucho más útil e interesante y en Hill Climbing no tiene mucho coste adicional.

4.6 En hora punta

Una vez seleccionado nuestro ganador absoluto en términos de beneficio obtenido, distancia total recorrida y tiempo de ejecución de forma experimental entre todos los algoritmos y configuraciones posibles vamos a explorar como este se comporta con el flag de hora punta activado. Esto es curioso pues todas las decisiones que hemos tomado ahora han sido basadas en una generación de Estaciones normal y ahora rush hour pordía cambiar radicalmente como de capaz es nuestro algoritmo de navegar por el espacio de soluciones nuevas pues la topología del mismo ha podido cambiar y resultar menos favorecedora. Como nuestro algoritmo es generalista que deberia de poder asumir toda clase de situaciones creemos que se desenvolverá de forma similar.

Este experimento se basa entonces en:

Observación	Esta nueva distribución de estaciones si bien no tiene porque afectar pues recorremos todo el espacio puede suponer cambios en inclinaciones y presencia de mesetas en el espacio de soluciones.
Planteamiento	Seleccionamos el algoritmo definitivo obtenido de las anteriores experimentaciones y lo ponemos a prueba con la distribución de estaciones rush hour
Hipótesis	Nuestro algoritmo seleccionado no se ve afectado en gran medida por la nueva distribución de estaciones (H0) o hay cambios en la eficiencia temporal y el resultado.
Método	• Elegiremos 10 semillas aleatoriamente (entre 0 y 10000). • Ejecutaremos 1 experimento por semilla si nuestro algoritmo es determinista o 3 por semilla y hacemos la media si no lo es. • Experimentaremos con problemas con 25 estaciones, 1250 bicicletas, 5 furgonetas y demanda rush hour como anteriormente. • Compararemos los tiempos de ejecución usando *currentTimeMillis*, la distancia total recorrida y el beneficio total.

Table 13: Bases de experimentación para la observación de las diferencias de en hora punta

Una vez hecha la experimentacion como es costumbre mostraremos los *boxplots* y los estadísticos:

Semilla	Ganancia	Coste Computacion Ganancia	Distancia	Tiempo en ms
5689	89	94	22900	16
2314	93	89.4	19300	18
7893	99	106.7	23800	16
4210	80	73.3	14500	12
9375	90	83	25100	14
1248	85	80.9	23700	14
6543	105	137.6	36900	15
8932	96	123.9	33500	15
3657	89	85.4	17100	15
8021	83	80.4	18700	14

Table 14: Tabla de resultados del HC con el flag de hora punta

Dado que los resultados obtenidos al aplicar el flag de hora punta en este experimento son similares a los resultados de experimentos anteriores, podemos concluir que el algoritmo seleccionado se desenvuelve de manera eficiente y consistente en diversas condiciones, incluida la hora punta.

La hipótesis nula (H0), que sugería que el algoritmo no se ve afectado en gran medida por la nueva distribución de estaciones en hora punta, parece estar respaldada por los resultados. Esto indica que el algoritmo es lo suficientemente generalista como para adaptarse a diferentes situaciones y cambios en la topología del espacio de soluciones sin comprometer significativamente su desempeño.

En resumen, la inclusión del flag de hora punta no parece provocar cambios drásticos en el algoritmo seleccionado, y este sigue siendo eficiente en términos de ganancia, distancia y tiempo de ejecución. Esto demuestra la robustez del algoritmo ante variaciones en las condiciones del problema.

4.7 Furgonetas optimas

Sabiendo que la rush hour no cambia el resulado de aplicar el HC en este problema, vamos a probar de buscar otra optimización para el problema. Queremos encontrar el numero optimo de furgonetas para maximizar nuestros resultados.

Este experimento se basa entonces en:

Observación	Intentaremos encontrar el numero de furgonetas que nos permita tener la mayor ganancia sin perder eficiencia.
Planteamiento	Seleccionamos el algoritmo definitivo obtenido de las anteriores experimentaciones y lo ponemos a prueba con diferente numero de furgonetas
Hipótesis	A medida que haya mas furgonetas la ganancia subira, hasta llegar a un punto que hay demasiadas furgonetas y el coste de tenerlas sea mayor (H0).
Método	• Elegiremos 5 semilla aleatoriamente (entre 0 y 10000). • Ejecutaremos 5 experimento por numero de furgonetas hasta que no haya una mejora significativa. • Experimentaremos con problemas con 25 estaciones, 1250 bicicletas y demanda equilibrada y rush hour. • Compararemos los tiempos de ejecución usando *currentTimeMillis*, la distancia total recorrida y el beneficio total.

Table 15: Bases de experimentación para la observación de las diferencias en numero de furgonetas

Furgonetas	Ganancia	Coste Computacion Ganancia	Distancia	Tiempo en ms
5	81.2 (6.352)	91.38 (11.444)	30600 (6169.616)	15.2 (0.748)
10	106.6 (21.22)	110.1 (32.93)	41840 (12201.75)	21.4 (1.14)
15	118.0 (20.35)	122.7 (28.51)	66680 (14687.83)	26.8 (2.94)
20	118.0 (20.35)	122.7 (28.51)	66680 (14687.83)	26.8 (2.94)

Table 16: Tabla de resultados de la demanda equilibrada

Furgonetas	Ganancia	Coste Computacion Ganancia	Distancia	Tiempo en ms
5	93 (10.48)	84.4 (13.65)	15400 (6943.5)	14 (1.17)
10	138 (26.07)	116.5 (18.45)	39500 (7379.98)	18 (1.41)
15	139 (28.35)	123.2 (22.39)	37900 (6149.98)	22 (2.16)
20	138 (24.38)	123.2 (24.99)	37900 (6874.69)	21 (2.83)

Table 17: Tabla de resultados de la demanda rush hour

Como se puede observar en los resultados obtenidos, las furgonetas optimas son las mismas en ambos casos (demanda equilibrada y demanda rush hour). De 5 furgonetas a 10, hay un incremento en el beneficio obtenido (mas considerable en el caso de la demanda en rush hour). El ultimo incremento que podemos observar es cuando pasamos de 10 a

15 furgonetas, es un incremento bastante pequeño que nos indica que nos estamos acercando al numero de furgonetas optimas. A partir de 15 furgonetas, el beneficio no cambia, concluyendo así que las furgonetas optimas son 15 para los dos tipos de demanda.

5 Competencia de trabajo en equipo: Trabajo de innovación

5.1 Descripción del tema

AlphaStar es un sistema de inteligencia artificial desarrollado por DeepMind, una subsidiaria de Alphabet Inc. Conocido por sus impresionantes capacidades en juegos de estrategia en tiempo real, en particular el popular videojuego StarCraft II. AlphaStar utiliza técnicas de aprendizaje profundo, incluidas redes neuronales, para aprender a jugar. Adquiere un amplio conjunto de datos de partidos de alto nivel entre jugadores para poder aprender estrategias y tácticas. Además, utiliza el aprendizaje por refuerzo para mejorar tu rendimiento en función de la retroalimentación que recibes del juego, lo que te permite ajustar y refinar tus decisiones y estrategias con el tiempo.

La característica distintiva de AlphaStar es su capacidad para competir en torneos en línea con jugadores de alto nivel, incluidos jugadores profesionales de StarCraft II. Demostró un alto nivel de habilidad, venciendo a muchos jugadores experimentadas. Sin embargo, el desarrollo de AlphaStar no estuvo exento de desafíos técnicos, ya que StarCraft II es un juego extremadamente complejo con muchas variables y estrategias posibles. AlphaStar debe aprender a comandar y coordinar ejércitos, gestionar recursos y tomar decisiones estratégicas sobre la marcha.

En general, AlphaStar representa un hito importante en la investigación de la IA y contribuye al campo al mostrar cómo la tecnología de la IA se puede aplicar con éxito a problemas complejos y dinámicos que van más allá de los juegos.

5.2 Reparto del trabajo

En nuestro proyecto grupal sobre AlphaStar, hemos decidido distribuir las tareas entre los tres para crear una presentación completa y atractiva. Cada uno de nosotros se centrará en aspectos específicos de AlphaStar, asegurándose de cubrir las dimensiones técnicas, prácticas y éticas.

Un miemmbro se encargará del aspecto técnico del proyecto.Este profundizara en los detalles técnicos de la arquitectura de AlphaStar y los algoritmos de aprendizaje por refuerzo que emplea.

Otro miembro del equipo se centrará en el lado práctico de AlphaStar. Explorará su impacto y significado en el campo de la inteligencia artificial y los videojuegos.

El tercer miembro del equipo adoptará la perspectiva ética y social. Abordará las consideraciones éticas en el desarrollo y la implementación de la IA, discutiendo preocupaciones sobre la equidad y la responsabilidad. Este miembro del equipo también profundizará en la percepción pública de la IA y la importancia del desarrollo responsable de la IA.

5.3 Lista de referencias

Bibliografia General

[1] Vinyals, O., Esgin, T., Simon, I., et al. (2019). Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782), 350-354. DOI:10.1038/s41586-019-1724-z.

[2] DeepMind Blog. (2019). "AlphaStar: Mastering the Real-Time Strategy Game StarCraft II." Retrieved from https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii.

[3] R. -Z. Liu, Y. Shen, and Y. Yu, "Revisiting of AlphaStar," *IEEE Transactions on Games*, DOI:10.1109/TG.2023.3265975.

[4] Ars Technica. (2019). "Level up: DeepMind's AlphaStar achieves Grandmaster level in StarCraft II." Retrieved from https://arstechnica.com/science/2019/10/leveling-up-deepminds-alphastar-achieves-grandmaster-level-in-starcraft-ii/.

5.4 Dificultades para recolectar información

La búsqueda de información sobre AlphaStar de DeepMind puede ser un desafío debido a la disponibilidad limitada de información técnica detallada, la naturaleza en rápida evolución del campo, la terminología compleja utilizada en la investigación de IA, los diferentes niveles de detalle en las fuentes y la naturaleza dispersa de la información. en artículos de investigación, blogs y foros. Además, los aspectos comerciales y confidenciales del proyecto, así como la posibilidad de información errónea, pueden complicar aún más la búsqueda de información precisa y completa.