PROYECYO DE GRADO

FACULTAD DE INGENIERÍA

PROBLEMA DE RUTEO PARA FLOTA DE VEHÍCULOS HETEROGÉNEA CON VENTANAS DE TIEMPO Y MÚLTIPLES ORÍGENES Y DESTINOS

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN COMPUTACIÓN

CHRISTIAN GONZALEZ
christian.m.gonzalez.perez@gmail.com

IGNACIO DA CUNHA
dacunha.igna@gmail.com

Tutor: FRANCO ROBLEDO
frobledo@fing.edu.uy

MONTEVIDEO, DICIEMBRE 2020

Índice general

Resumen

En el presente trabajo se estudia la problemática actual de una empresa estatal, la Administración Nacional de Correos. Se trata de un problema de planificación de rutas de un grupo heterogéneo de vehículos que deben pasar por ciertos puntos a recoger paquetes. Los vehículos tienen diferentes capacidades y pueden partir desde orígenes distintos así como pueden finalizar su recorrido en diferentes destinos. Los puntos cuentan con diversas características como ventana de tiempo, prioridad, volumen a ser recogido y restricciones de ciertos vehículos. El objetivo es que los vehículos pasen por la mayor cantidad de puntos, contemplando sus prioridades, y tratando de encontrar la mejor ruta para cada vehículo de manera que la distancia total recorrida sea mínima.

Para la resolución de dicho problema se implementaron técnicas de programación genética, mas precisamente, algoritmos evolutivos. El análisis experimental se realizó utilizando un conjunto de instancias del problema construidas en base a coordenadas reales (pertenecientes a puntos válidos de Montevideo) generadas de forma aleatoria y fue comparado frente a un algoritmo ávido. Como adicional al proyecto se presenta un conjunto de servicios desarrollados para la Administración Nacional de Correos la cual se encarga de recibir solicitudes de clientes y se encarga de realizar las peticiones de ejecución al algoritmo evolutivo.

El trabajo realizado abre la puerta a futuras investigaciones sobre el estudio de métodos de resolución eficientes para problemas de índole similar.

Palabras clave: VRP (Vehicle Routing Problem), ventanas de tiempo, vehículos heterogéneos, pick-up, algoritmos evolutivos, optimización

Capítulo 1

Introducción

Cada vez más las empresas u organizaciones dedicadas al transporte y entrega de mercadería buscan optimizar la recogida y/o entrega de sus productos con el fin de mejorar la calidad y el desempeño de los servicios, de forma de reducir costos y aumentar la eficiencia en el uso de recursos. Gracias al avance tecnológico se avanza constantemente en esta línea.

Bajo este universo, la optimización de rutas de una empresa de levante/entrega de mercadería que tiene una cantidad elevada de puntos por los cuales debe pasar resulta de gran interés ya que es un elemento clave en logística. Se debe tener en consideración varios aspectos tales como el numero y capacidad de los vehículos de la flota así como también otros requerimientos específicos que pudieran venir desde el cliente, por ejemplo, una ventana de tiempo.

El problema de enrutamiento de vehículos (VRP en ingles) [1] ha sido ampliamente estudiado en la literatura [2].

El VRP puede ser definido como un problema de diseño de rutas que recorren un conjunto de puntos, cumpliendo con una serie de restricciones. El objetivo es maximizar o minimizar una determinada función objetivo. Generalmente el problema se compone de un depósito con varios vehículos y clientes, donde las rutas se definen de tal forma que se cumplan todas las restricciones establecidas al problema. Las restricciones varían ampliamente de un problema a otro, abarcando desde la capacidad de los vehículos, cumplir con una determinada demanda, un cierto horario, etc. Un posible valor a maximizar podría ser el beneficio neto y un posible valor a minimizar podría ser el costo total.

Este tipo de problema tiene diversas aplicaciones en la realidad, resultando un campo de gran interés y que se mantiene en constante investigación, tratándose de problemas complejos de resolver, clasificados como NP-hard [3]

El problema a estudiar tiene características similares a la variante del problema de ruteo de vehículos con flota heterogénea, múltiples depósitos y ventanas de tiempo [4] con restricciones de asignación descrita segun Tummel, Franzen, Hauck y Jeschke [5]. Este problema cuenta, además, con una serie de elementos o restricciones adicionales que lo hacen más complejo. Cada cliente tiene una ventana de tiempo y no puede ser visitado por fuera de esa ventana. Se cuentan con vehículos de diferentes capacidades y restricciones de visita a los clientes, es decir, no todos los vehículos pueden visitar a todos los clientes. Por otra parte algunos vehículos tienen puntos fijos asignados, es decir, puntos por los que deben pasar de forma

obligatoria, esto resulta en una posible limitante a la hora de establecer rutas.

Por lo tanto en este proyecto se busca encontrar soluciones a la variante VRP de recogida de mercadería que tenga en cuenta: flota heterogénea de vehículos, que puedan partir de puntos distintos y volver de la misma manera a puntos distintos donde cada vehículo cuenta con su propia hora de partida y llegada, diferente capacidad y ciertos puntos fijos por los cuales el vehículo deba pasar de forma obligatoria. Por otro lado, para los puntos (clientes) se tiene en cuenta: ventana de tiempo (horario en el cual el cliente se encuentra disponible), tiempo de espera estimado (tiempo que le lleva recoger las mercaderías al vehículo que pasa por allí). Los clientes tienen, además, una prioridad, por lo que se debe evitar dejar puntos de alta prioridad sin recoger. Por último cada punto puede tener un subconjunto de vehículos de forma tal que solo estos estén habilitados a pasar por el punto. En suma, se tienen las restricciones: un vehículo debe pasar necesariamente por sus puntos fijos en caso de tener alguno; Si un punto tiene vehículos asignados significa que por ese punto solo pueden pasar esos vehículos; Un vehículo no puede cargar más volumen que su capacidad permitida; Un vehículo no puede pasar por un punto en un horario que el punto no admita. Se trata de un problema de minimización donde se busca obtener una solución con la menor cantidad de puntos pendientes, la menor distancia recorrida y que pase por la mayor cantidad de puntos prioritarios cumpliendo todas las restricciones.

La heurística escogida para resolver este problema es Algoritmos Evolutivos [6]. Una de las razones para la utilización de esta técnica es su eficiencia para resolver problemas de ruteo de alta complejidad (NP-hard).

El trabajo incluyó una investigación bibliográfica del tema, profundizando en los aspectos mas relevantes a tener en cuenta para el problema a tratar. Por un lado se hizo énfasis en la variante relacionada a la flota heterogénea con múltiples depósitos y ventanas de tiempo así como los métodos de resolución.

Finalizado el diseño, implementación del modelo y su correspondiente ajuste paramétrico se evaluó la eficiencia del algoritmo diseñado utilizando un generador de datos reales implementado por el propio equipo.

El presente documento se encuentra organizado de la siguiente manera: En el capítulo 2 se describe el problema en forma detallada, se presenta su formulación matemática y se da un caso de ejemplo para la mayor comprensión del mismo. El capítulo 3 introduce el concepto de algoritmos evolutivos explicando brevemente su marco histórico, su funcionamiento y características más importantes como son la representación de las soluciones, la función de fitness y los operadores correctivos. El capítulo 4 introduce el estado del arte mostrando una serie de trabajos relacionados al problema planteado. En el capítulo 5 se presentan las herramientas y bibliotecas utilizadas, se describe la solución propuesta, se detallan como son los datos de entrada y salida del algoritmo, como se codificaron las soluciones, la función de fitness empleada así como los mecanismos de selección, recombinación y mutación empleados. En este capítulo se presentan también los mecanismos para la generación e inicialización de la población inicial y por último se describe el operador correctivo utilizado.

En el capítulo 6 se describe como fue implementado el generador de instancias de prueba. En este capítulo se presenta el entorno de ejecución utilizado para las pruebas y se describe el proceso de configuración paramétrica. Por último se realiza

la evaluación experimental del algoritmo comparándolo con un algoritmo greedy.

El capítulo 7 muestra el desarrollo de una API de integración que permite ejecutar el algoritmo evolutivo a través de servicios REST y de esta forma poder utilizar el algoritmo fácilmente ya sea desde una aplicación móvil o desde una pagina web.

Por último en el capítulo 8 se presentan las conclusiones del trabajo realizado así como el trabajo a futuro para seguir con la investigación.

Capítulo 2

Análisis del problema

Este capítulo presenta el problema abordado en el marco del proyecto. En la Sección 2.1 se presenta el problema con mayor detalle y su contexto. En la Sección 2.2 se presenta la formulación matemática del problema descrito. Por último, en la Sección 2.3 se presenta una instancia utilizada a modo de ejemplo para complementar la comprensión del problema a resolver.

2.1. Descripción del problema

La Administración Nacional de Correos [7] se encarga, entre otras muchas tareas, de la entrega y recogida de paquetes. Actualmente cada día se debe calcular la mejor ruta de distribución/recogida de paquetes en el área regional para la flota de vehículos existente. La flota utilizada para la entrega no necesariamente es la misma que se utiliza para la recogida de paquetes. Este proyecto se centra en la recogida de paquetes, pudiendo ser adaptado para posteriormente utilizarlo con las condiciones de la modalidad de entrega.

El problema que se plantea se trata de un problema de enrutamiento de vehículos (variante de VRP). La Administración Nacional de Correos describe el problema de la siguiente manera: Se dispone de un conjunto de vehículos heterogéneos que difieren en capacidad. Estos vehículos son utilizados para levantar paquetes en diferentes puntos geográficos y pueden partir desde diferentes puntos de origen así como también pueden tener diferentes puntos de finalización, pudiendo ser el punto de origen diferente al punto de destino. Cada vehículo tiene un horario de salida de su origen y un horario de llegada a su destino el cual debe ser respetado. Un vehículo puede tener asignado uno o más puntos fijos en su trayectoria. Un punto fijo de un vehículo es un punto por el cual el vehículo debe necesariamente pasar a recoger por cuestiones de negocio. En cada punto los paquetes son heterogéneos entre si, es decir, difieren en volumen. Cada punto tiene una prioridad asignada, a mayor prioridad más importante es pasar por ese punto y recoger los paquetes. Cada punto tiene predefinido un tiempo de espera, es decir, el tiempo estimado que se espera que un vehículo este en dicho punto para recoger todos los paquetes. No es posible pasar por un punto y recoger solamente algunos paquetes y dejar el resto pendientes. Pasar por un punto implica necesariamente recoger todo el volumen allí presente. Un punto, además puede tener vehículos asignados, esto quiere decir que por ese punto solo pueden pasar los vehículos que estén asignados. Por último cada

punto dispone de un horario donde esta disponible. Un vehículo solo puede pasar por un punto en el rango horario donde el punto este disponible.

Por las condiciones mencionadas, los puntos van cambiando cada día y los vehículos pueden no ser los mismos del día anterior e incluso si llegaran a ser los mismos pueden partir de lugares totalmente diferentes (destino del día anterior por ejemplo). Ante este escenario resulta poco viable que una tarea de esta magnitud se este realizando actualmente de forma manual, siendo una persona la encargada de tener en cuenta todas las restricciones mencionadas y encontrar así una ruta para cada vehículo. El problema planteado cae en la categoría de problemas NP-hard[3], para los cuales puede que no exista un algoritmo que garantice encontrar la solución óptima en un tiempo razonable o que garantice siquiera encontrar una solución. Los métodos de resolución como la programación dinámica o la programación lineal no son capaces, por lo general, de resolver eficientemente instancias de dimensiones grandes para este tipo de problemas ya que demandan una enorme capacidad de cómputo y complejidad algorítmica [8] [9]. Al tratarse de un problema NP-hard y en base a todas las condiciones que presenta el problema resulta sencillo cometer errores en un análisis manual obteniendo soluciones no factibles y de esta forma dar lugar a inconsistencias en la jornada laboral. Estas inconsistencias pueden ser desde haber asignado erróneamente un punto de recogida a un vehículo que no tenía permitido pasar por ese punto hasta asignar puntos a vehículos de forma tal que sea imposible cumplir con el horario disponible (ventana de tiempo) de alguno de los puntos.

Por lo anteriormente mencionado surge la motivación de emplear algoritmos evolutivos (AE)[6] los cuales resultan en una buena opción para atacar este tipo de problemas ya que garantizan encontrar una solución buena (puede ser la óptima) en un tiempo razonable en comparación con otros métodos los cuáles pueden no terminar o tardar un tiempo no práctico dependiendo del problema a atacar.

La solución debe devolver por cada vehículo los puntos asociados en orden por los cuales el vehículo debe pasar. En caso de quedar puntos pendientes (puntos que no fue posible asignarlos a un vehículo) se debe devolver también una lista de puntos pendientes junto a la solución. Una solución es mejor que otra si no tiene en su lista de pendientes ningún punto con mayor prioridad y si la cantidad de puntos pendientes es menor. En caso de empate, se decide por la solución que presente la menor distancia recorrida. Para la empresa es más importante no dejar puntos pendientes con mayor prioridad que tener menos cantidad de puntos pendientes. Es decir, se prefiere pasar a buscar un punto de prioridad X antes que pasar a buscar N puntos de una prioridad Y, siendo $Prioridad(X) > Prioridad(Y)$, $N >> 1$.
En resumen se tiene la siguiente información para vehículos y puntos:

Por cada vehículo se tiene en cuenta:

- Identificador del vehículo
- Origen (coordenadas)
- Destino (coordenadas)
- Hora de salida del vehículo (origen)
- Hora de llegada del vehículo (destino)

- Capacidad del vehículo
- Lista de puntos fijos (puntos por los que el vehículo debe pasar de forma obligatoria)

Por cada punto se considera:

- Identificador del punto
- Lugar (coordenadas)
- Tiempo de espera
- Vehículo asignado
- Volumen
- Horario disponible
- Prioridad.

Las siguientes **restricciones** deben ser tenidas en cuenta en el modelo del problema:

- Un vehículo debe pasar necesariamente por sus puntos fijos en caso de tener alguno. Dichos puntos deben ser factibles de recoger.
- Si un punto tiene vehículos asignados significa que por ese punto solo pueden pasar esos vehículos.
- Un vehículo no puede cargar más volumen que su capacidad permitida.
- Un vehículo no puede llegar más tarde que su hora de llegada establecida.
- Un vehículo no puede pasar por un punto en un horario que el punto no admita.

Como se mencionó anteriormente se trata de un problema de minimización dónde se busca obtener una solución con la menor cantidad de puntos pendientes, la menor distancia recorrida y que pase por la mayor cantidad de puntos en base a su prioridad cumpliendo todas las restricciones.

2.2. Formulación matemática

A continuación se presenta la formulación matemática para el problema planteado:
Dado los siguientes elementos:

- Un conjunto de vehículos $V = \{v1, .., vk\}$ que parten desde un conjunto de orígenes potencialmente distintos $O = \{o1, .., on\}$ hacia un conjunto de destinos diferentes $D = \{d1, .., dm\}$, $n \leq k$, $m \leq k$.

- Un conjunto de puntos $P = \{p1, .., pn\}$ ubicados en orígenes diferentes $OP = \{op1, .., opn\}$
- La funcion $origP : P \rightarrow OP$ establece el origen de cada uno de los puntos.
- La funcion $orig : V \rightarrow O$ establece el origen de cada uno de los vehículos.
- La funcion $dest : V \rightarrow D$ establece el destino de cada uno de los vehículos.
- La función $CapV : V \rightarrow \mathbb{R}^+$ que dado un vehículo, devuelve su capacidad.
- La función $Cap : P \rightarrow \mathbb{R}^+$ que dado un punto, devuelve su capacidad.
- La función distancia, $dist : O \times \{O \cup D\}\} \rightarrow \mathbb{R}^+$.
- La función tiempo, $tiempo : O \times \{O \cup D\}\} \rightarrow \mathbb{R}^+$.
- La función de costo asociada a la prioridad de los puntos recogidos por cada vehículo y la distancia recorrida para pasar por dicho punto, $cost : \mathbb{N}^+ \times \mathbb{R}^+\} \rightarrow \mathbb{R}^+$.
- La función $PuntosFijos : V \rightarrow P$ que devuelve los puntos fijos del vehículo.
- La función $VehiculosEnPunto : P \rightarrow V$ que devuelve los vehículos permitidos por un Punto.
- La función $HoraLLegada : V \rightarrow \mathbb{N}^+$ que devuelve la hora de llegada establecida de un vehículo.
- La función $HoraMin : P \rightarrow \mathbb{N}^+$ que devuelve la hora mínima en la que el punto esta disponible (en minutos). Para un punto que este disponible a partir de las 08:00 la función devolverá 480.
- La función $HoraMax : P \rightarrow \mathbb{N}^+$ que devuelve la hora máxima en la que el punto esta disponible (en minutos).

El problema consiste en hallar una función $f : P \rightarrow V \times \mathbb{N}$ para recoger a los n puntos en k vehículos que determine la asignación de puntos a vehículos y el orden en que serán recogidos desde sus respectivos origines hacia los diferentes destinos, minimizando la función de costo total (CT) de la asignación, donde $f^{-1}(v_i, j)$ indica el punto recogido en el lugar j para el vehículo i con $orig(f^{-1}(v_i, -1)) = orig(v_i)$ y $orig(f^{-1}(v_i, ult(v_i)) = dest(v_i)$ donde $ult(v_i)$ devuelve el lugar (número de recogida) del último punto recogido por el vehículo v_i

$$CT = \sum_{v_i, ult(v_i)} [\sum_{j=0}^{ult(v_i)} cost(dist(orig(f^{-1}(v_i, j-1)), orig(f^{-1}(v_i, j))))]$$

Sujeto a:

$$\forall v \in V, f^{-1}(v, i) \in PuntosFijos(v), i = 1, ..., ult(v)$$

$$\forall p \in P, f(p) \in VehiculosEnPunto(p)$$

$$\forall v \in V, \sum_{i=0}^{ult(v)} Cap(f^{-1}(v,i)) \leq CapV(v)$$

$$\forall v \in V, \sum_{i=0}^{ult(v)} tiempo(orig(f^{-1}(v,i-1)), orig(f^{-1}(v,i))) \leq HoraLLegada(v)$$

$$\forall v \in V,$$

$$HoraMin(f^{-1}(v,i)) \leq \sum_{i=0}^{ult(v)} tiempo(orig(f^{-1}(v,i-1)), orig(f^{-1}(v,i))) \leq HoraMax(f^{-1}(v,i))$$

$$, i = 1, ..., ult(v)$$

2.3. Caso de ejemplo

A continuación se describe una instancia de ejemplo del problema planteado. Considérese el mapa de la Figura 2.1 en el cual se muestra un conjunto de 4 puntos P1, P2, P3 y P4 a ser recogidos, un origen y un destino.
Para este ejemplo, se dispone de un único vehículo a efectos demostrativos y para que el lector finalice de comprender el problema a abordar. Las características de cada punto y del vehículo se detallan a continuación. Se considera, además, que el tiempo del origen al punto 1 es de 30 minutos, el tiempo del punto 1 al punto 3 es de 120 minutos y el tiempo del punto 3 al destino es de 120 minutos también.
Vehículo:

- Hora de salida del vehículo (origen): 09:00
- Hora de llegada del vehículo (destino) : 15:30
- Capacidad del vehículo : 500l
- Lista de puntos fijos (puntos por los que el vehículo debe pasar de forma obligatoria): P1

Punto 1 (P1):

- Identificador del punto: 1
- Tiempo de espera : 30 min
- Vehículo asignado : 1
- Volumen: 100l
- Horario disponible: 09:00-10:00
- Prioridad: 2

Punto 2 (P2):

- Identificador del punto: 2

- Tiempo de espera : 30 min
- Vehículo asignado : 2
- Volumen: 150l
- Horario disponible: 09:00-10:00
- Prioridad: 1

Punto 3 (P3:

- Identificador del punto: 3
- Tiempo de espera : 60 min
- Vehículo asignado :
- Volumen: 300l
- Horario disponible: 11:00-15:00
- Prioridad: 2

Punto 4 (P4):

- Identificador del punto: 4
- Tiempo de espera : 120 min
- Vehículo asignado : 1
- Volumen: 30l
- Horario disponible: 16:00-17:00
- Prioridad: 1

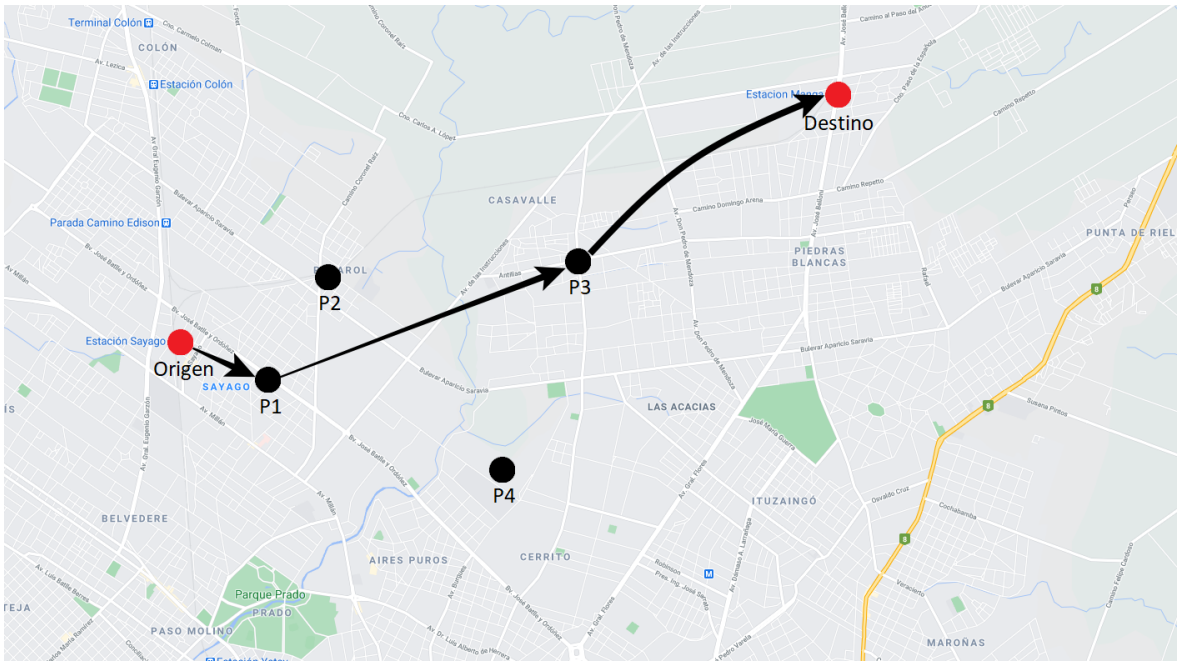

Figura 2.1: Caso de ejemplo.

Para esta solución el vehículo parte del origen y como primera instancia recoge al punto 1. Esto es posible ya que el vehículo 1 tiene como punto fijo al punto 1. Por otra parte el tiempo para llegar del origen a dicho punto es de 30 minutos, es decir, que el vehículo llega a dicho punto a las 9:30, horario en el cual el punto sigue disponible. La capacidad es de 100l y la del vehículo es de 500l, por lo que no se incumple ninguna restricción en este punto de la solución. Posteriormente y a pesar de que el punto 2 esta más cerca se opta por recoger el punto 3. Esto se debe a que el punto 2 tiene como vehículo permitido otro diferente al vehículo que esta circulando actualmente, por lo que no puede pasar a recogerlo. Algo similar ocurre con el punto 4. Este punto tiene una ventana de tiempo de 1 hora, la cual está comprendida entre las 16 y las 17 horas. El vehículo debe llegar al destino a lo sumo a las 15:30 horas por lo que resulta imposible pasar por el punto 4 con el vehículo actual. El punto 3 está disponible entre las 11 y las 15 horas. El tiempo del punto 1 al punto 3 es de 120 minutos. El vehículo llega al punto 1 a las 9:30 y parte del mismo a las 10 ya que el tiempo de espera es de 30 minutos. Por lo que llega al punto 3 a las 12. El volumen del punto 3 es de 300l y al vehículo aún le quedan 400l disponibles de capacidad. Por último el tiempo del punto 3 al destino es de 120 minutos y el tiempo de espera en dicho punto es de 60 minutos. Por lo que el vehículo llega a destino a las 15 horas, sobrando 30 minutos. En esta solución el vehículo 1 recoge entonces, al punto 1 y 3 dejando como puntos pendientes el punto 2 y el punto 4. Este ejemplo a pesar de contar únicamente con 4 puntos y un solo vehículo no resulta intuitivo encontrar una solución óptima al problema que logre cumplir con todas las restricciones que el mismo plantea. Por lo que esto da un panorama de la dificultad del problema a enfrentar para casos realistas dónde la cantidad de puntos y vehículos es mucho mayor abriendo un enorme abanico de posibilidades y combinaciones para obtener una solución factible.

Capítulo 3

Algoritmos evolutivos

El presente capítulo describe la técnica utilizada para resolver el problema planteado en el capítulo 2. En la Sección 3.1 se introducen los algoritmos evolutivos como técnicas para resolver problemas de optimización mostrando brevemente su marco teórico e histórico y se describen los principales conceptos asociados a estos métodos.

3.1. Algoritmos evolutivos

En esta sección se hace una breve introducción a los algoritmos evolutivos como técnica para resolver problemas de optimización detallando los conceptos fundamentales para poder comprender el resto del informe de forma más clara y concisa.

3.1.1. Marco Teórico

Los algoritmos evolutivos son técnicas estocásticas que simulan el proceso de evolución natural de las especies para resolver problemas de búsqueda, aprendizaje y optimización. Los programas 'evolutivos' fueron sugeridos desde los inicios de la era de la computación, entre 1948 y 1966. Alan Turing investigó las relaciones sobre evolución natural y aprendizaje. Propuso desarrollar programas que se modificaran a si mismos, siendo capaces de jugar ajedrez y simular otras actividades inteligentes y sencillas, utilizando técnicas y mecanismos evolutivas [10] [11]. En 1966 Neumann János planteó mecanismos evolutivos para desarrollar autómatas con una capacidad de cómputo similar a las másquinas de Turing. Habló de poblaciones de individuos autómatas trabajando en conjunto, compartiendo información y comunicándose entre sí [12]. Entre 1953 y 1956 Nils Barricelli desarrolló un modelo computacional sobre la evolución darwiniana donde menciona la importancia de utilizar la recombinación, la selección por aptitud y el operador probabilístico de mutación [13].

Woodrow Bledsoe y Hans Bremermann (1960–1961) establecieron el concepto de algoritmos evolutivos. Entendieron la evolución como un mecanismo de optimización capaz de aplicarse para mejorar algoritmos de patrones utilizando redes neuronales. Propusieron la idea de codificación binaria, el uso de valores de aptitud y utilizar poblaciones. Por otra parte vieron la importancia de utilizar operadores de mutación probabilísticos para intentar evitar que una solución caiga en un óptimo

local. De esta forma en 1965 presentan su estudio teórico sobre el funcionamiento de los algoritmos evolutivos determinando la probabilidad de mutación óptima para resolver problemas que se pueden separar linealmente [14] [15].

En 1963 Newell y Simon proponen el General Problem Solver el cual puede resolver problemas simples empleando técnicas evolutivas guiadas por heurísticas que determina el usuario [16].

En 1965 Ingo Rechenberg propone una serie de estrategias de evolución que funcionan a través de métodos de ajustes discretos aleatorios los cuales se inspiran en el mecanismo de mutación que se puede observar en la propia naturaleza [17].

Diez años después, en 1975 Holland propone un método de búsqueda genética sentando las bases de la computación evolutiva[18].

3.1.2. Introducción

Un algoritmo evolutivo es, una técnica iterativa, conocida como generación, a la cual se le aplican operadores estocásticos sobre un conjunto de datos, a los que se les conoce comunmente como población [6]. Esta población se puede generar de dos maneras, a través de un procedimiento aleatorio o utilizando heurísticas que dependen del problema que se quiere resolver. Dentro de la población existen individuos. Cada individuo esta compuesto por un genoma compuesto por alelos [19]. Cada uno de estos individuos representan una posible solución del problema. A cada individuo se le asocia un valor de fitness que se calcula a partir de una función de evaluación o función de fitness que determina que tan adecuado es el individuo para resolver el problema. Una vez que se determinan los valores de fitness de cada individuo estos se someten a un proceso de selección donde se escogen qué individuos formaran parte de la siguiente generación. Una vez seleccionados los individuos se procede a utilizar operadores de recombinación o cruzamiento para generar descendientes con nuevas características que puedan dar nuevas soluciones y abrir el abanico de posibilidades. A efectos de evitar caer en óptimos locales los algoritmos evolutivos emplean un operador, el de mutación. Este operador tiene lugar generalmente con baja probabilidad y en caso de ocurrir afecta alguna característica del individuo que de forma natural no se hubiera manifestado. [20]

En forma general se puede decir que un algoritmo evolutivo consta de cuatro etapas bien diferenciadas [21]:

- Evaluación, donde se asigna un valor de fitness a cada uno de los individuos de la población
- Selección, donde se determinan los candidatos para formar parte de la siguiente generación de individuos de la población.
- Operadores evolutivos los cuales generan descendientes utilizando los individuos seleccionados mediante operadores que simulan la evolución natural (recombinación, mutación, etc)
- Reemplazo, encargado de realizar el cambio de una generación a otra.

Existen multitud de políticas de selección y reemplazo las cuales permiten establecer las características del algoritmo evolutivo. Por ejemplo privilegiar los individuos

más adaptados (elitismo), aumentar la presión selectiva, incrementar la diversidad genética, etc. Los operadores evolutivos determinan el mecanismo de exploración del espacio de búsqueda del problema. Un algoritmo evolutivo finaliza mediante un criterio de parada previamente establecido. Este criterio de parada depende fuertemente del problema a resolver. Generalmente esta determinado por un número de generaciones, es decir, el algoritmo se detendrá al llegar a dicho número de generaciones. Otros criterios de parada utilizados pueden ser la detección de convergencia, utilizar un tiempo máximo de ejecución, detener la ejecución del algoritmo si el mismo no obtiene mejoras en un determinado número de generaciones, etc.

Una vez se cumple la condición de parada el algoritmo evolutivo retorna la mejor solución encontrada en ese momento.

Por lo tanto el objetivo principal de un algoritmo evolutivo es mejorar el valor de fitness de los individuos de una población con la ayuda de operadores evolutivos, habitualmente selección, recombinación y mutación.

A continuación se presenta, a modo de resumen, el funcionamiento básico de un algoritmo evolutivo.

```
inicializar(P(0))
t = 0 {contador de generación}
Mientras !criterioParada hacer
 evaluar(P(t))
 padres = seleccion(P(T))
 hijos = operadoresEvolutivos(padres)
 nuevaPoblacion = reemplazo(hijos, P(t))
 t++
Fin Mientras
Retornar mejorIndividuoHallado
```

3.1.3. Representación de la solución

En biología se denomina cromosoma a cada una de las cadenas de ADN que se encuentran en el núcleo de las células. Los cromosomas son los responsables de la transmisión de información genética. Pueden existir formas o valores alternativos del gen llamadas alelos [19][22]. Se denomina genotipo a la información contenida en el genoma de un individuo. El genotipo puede verse como lo que potencialmente puede llegar a ser un individuo. Por lo tanto los rasgos específicos y observables de un individuo constituyen su fenotipo. A partir del genotipo y del desarrollo se origina el fenotipo de un individuo. Los algoritmos evolutivos utilizan estos conceptos para dar una representación de las soluciones. Se utilizan los cromosomas como un vector de genes donde cada valor de cada gen es un alelo.

3.1.4. Función de fitness

La aptitud o adecuación de un individuo determina, en biología, la capacidad de dicho individuo para adaptarse a su entorno. Esta aptitud se relaciona con la probabilidad que tiene el individuo para sobrevivir en el medio y de esta forma conseguir reproducirse y tener descendencia.

Una función de fitness se encarga de evaluar cada uno de los individuos y determina que tan aptos son para resolver el problema a tratar. La función de fitness es una de las principales herramientas para que el algoritmo evolutivo pueda llegar a obtener una buena solución ya que determina los individuos candidatos a sobrevivir, es decir, aquellos individuos a los cuales se les aplicarán los operadores evolutivos.

La función de fitness está muy relacionada con la función objetivo del problema a resolver. Es posible que un determinado genotipo tenga un fenotipo asociado que no represente una solución factible en el alcance o dominio del problema [23]. Algunos enfoques para tratar a individuos no factibles son los siguientes [24]:

- Evitarlos. En general es un procedimiento difícil, complica los procesos de codificación y decodificación.
- Descartarlos. Es una opción simple pero puede conducir a la pérdida de características que podrían ser útiles para el problema. Si se mantiene diversidad en la población se evita la probabilidad de converger antes de tiempo y llegar a una solución no deseada. De la misma manera mantener diversidad ayuda a evitar los óptimos locales.
- Penalizarlos. Se penaliza el valor de fitness de los individuos que se detectan no factibles. Este mecanismo no evita completamente que sobrevivan individuos no factibles. Para problemas complejos es necesario realizar estudios de índole empírica para establecer un correcto mecanismo de penalización.
- Corregirlos. Cuando se detecta un individuo no factible se procede a corregirlo aplicándole ciertas transformaciones que hagan que ese individuo se vuelva factible alterando lo menor posible sus características. Con este método el algoritmo se asegura trabajar siempre con individuos factibles. Se puede perder diversidad o caer en un óptimo local si no se aplica un operador de corrección adecuado.

3.1.5. Operadores evolutivos

Los algoritmos evolutivos cuentan con operadores de selección, recombinación o cruzamiento y mutación.

Selección

La selección es el proceso mediante el cual algunos individuos en una población son seleccionados para reproducirse basados en su aptitud, es decir, su valor de fitness. Se denomina 'selección dura' cuando solamente se seleccionan los mejores individuos de cada generación. Por otro lado la 'selección blanda' surge cuando se utilizan mecanismos probabilísticos para mantener como padres a individuos que tengan aptitudes relativamente bajas. Alguno de los operadores de selección más habituales son los siguientes [25] [26]:

- Selección proporcional. Se asigna una probabilidad de selección proporcional al fitness relativo.

- Selección por rango (k). Se ordenan los individuos en relación a su fitness y se escogen los k mejores.
- Selección por torneo (k,r). Se escogen k individuos de los cuales se seleccionan los r con mejor fitness.
- Selección estocástica universal. Se utilizan punteros a modo de ruleta para eliminar el sesgo de la selección proporcional.

Dependiendo del problema a resolver conviene utilizar un operador de selección u otro.

Recombinación

En biología la reproducción consiste en la creación de un nuevo individuo a partir de dos progenitores (reproducción sexual) o de un único progenitor (reproducción asexual). Durante la reproducción sexual ocurre la recombinación o cruzamiento. El operador de recombinación o cruzamiento es un operador que se aplica sobre dos cromosomas para generar descendientes con características de ambos progenitores. En algoritmos evolutivos los operadores de cruzamiento se basan en estos principios. A continuación se presentan los operadores de cruzamiento más habituales [27]. Cabe aclarar que depende de cada problema puede ser mejor uno u otro operador o incluso diseñar uno propio.

- Cruzamiento de un punto (SPX). Dado dos genomas correspondientes a los padres, se corta a ambos en un punto de corte común y se intercambian los trozos de los cromosomas de ambos padres generando así dos nuevos descendientes con las características de ambos progenitores.
- Cruzamiento de n puntos (NPX). Análogo al cruzamiento de un punto pero esta vez con n puntos de corte.
- Cruzamiento uniforme (UX). Se seleccionan n puntos de intercambio en cada uno de los progenitores y se intercambian estos puntos seleccionados dando lugar a nuevos descendientes que heredan ciertas características puntuales de sus progenitores.

Mutación

En biología una mutación es un cambio en la secuencia de ADN de un individuo. Las mutaciones pueden suceder por un error en la copia del ADN en el proceso de división celular o malformaciones genéticas posteriores a la etapa de gestación de los progenitores. En algoritmos evolutivos el operador de mutación tiene como principal objetivo introducir variedad a la población de forma aleatoria. De esta forma se permite una mayor exploración del espacio de búsqueda. La exploración es un mecanismo útil para evitar que un algoritmo quede atrapado en óptimos locales. De esta forma se da la posibilidad de llegar a zonas del espacio de búsqueda que no hubiera sido posible alcanzar con los individuos de la población actual antes de producirse dicha mutación en su información y características. Alguno de los operadores de mutación más habituales son los siguientes [28]:

- Mutación de inversión. Consiste en modificar una posición determinada del genoma de un individuo de forma aleatoria. Se basa en la mutación presente en la evolución natural.
- Mutación de inversión múltiple. Análogo al caso de mutación de inversión pero modificando n posiciones del genoma del individuo.

Capítulo 4

Estado del arte

Esta sección presenta una revisión bibliográfica acerca de la historia relacionado a los problemas de ruteo de vehículos.

4.1. Contexto

La logística industrial es utilizada por las compañías con el fin de generar ventajas competitivas. En este contexto resulta fundamental el proceso de distribución, por lo que determinar rutas de la forma mas óptima posible resulta de gran interés. Con esta motivación han surgido multitud de modelos y métodos que tratan este problema con el fin de mejorar el desempeño logístico. El ruteo de vehículos consiste en determinar cómo distribuir o asignar en un cierto orden una serie de puntos que pueden representar clientes, paquetes, etc. Esta asignación se lleva a cabo cumpliendo ciertas restricciones sujetas al problema a resolver. Estas restricciones están fuertemente relacionadas con la naturaleza del problema. Un problema de ruteo de vehículos no solo busca una solución que satisfaga todas las restricciones del problema si no que busca, también, la mejor solución posible. Los objetivos de un problema de ruteo pueden ir desde obtener una ruta con menor costo, recorrer las menores distancias posibles, no dejar puntos de recogida sin atender, minimizar los tiempos totales de transporte, etc.

A continuación se describe la evolución histórica de los problemas de ruteo, se presenta la formulación matemática de los modelos más relevantes y se realiza una breve introducción a los diferentes mecanismos utilizados para resolver estos problemas.

4.2. Evolución histórica

El primer problema formalmente reconocido como un problema de ruteo fue el del agente viajero o TSP (Travelling Salesman Problem) introducido por Flood en 1956. Inicialmente el problema plantea la problemática de un agente vendedor que debe visitar una cierta cantidad de destinos (ciudades) en un solo viaje. De esta forma debe iniciar y finalizar su recorrido en la ciudad de la cual partió. El agente debe calcular cual es la mejor ruta para visitar todas las ciudades una sola vez y poder finalizar su viaje en la ciudad de la cual partió de tal manera que la distancia total

recorrida sea mínima.
En 1959 George Dantzig y John Ramser presentan el problema TSP generalizado como una variante más compleja del TSP dada sus restricciones [29]. La variante consiste en minimizar el costo de repartir mercancía desde un depósito a un conjunto de puntos de reparto (clientes) utilizando vehículos con cierta capacidad. Este trabajo se utiliza como base para el desarrollo de posteriores formulaciones las cuales aumentan la cantidad de variables y restricciones implicadas en el problema, aumentando así su complejidad [30] [31] [32].
El TSP generalizado puede ser considerado como un VRP o un CVRP, es decir, como un problema de ruteo de vehículos donde la capacidad de la flota se convierte en una restricción a la hora de formular el problema. La función objetivo del CVRP busca minimizar el costo total de recorrido para repartir a todos los puntos. Se asume que la flota de vehículos es homogénea y que todos parten del mismo deposito. El CVRP encuentra una cantidad exacta de rutas con mínimo costo definido como la suma de los costos de los arcos que forman los recorridos. De esta forma cada recorrido parte del centro de distribución, cada punto o centro es visitado por un solo vehículo y la suma de la demanda de los puntos visitados no excede la capacidad del vehículo.
El CVRP cuenta con dos escenarios diferenciados. Por un lado si el costo de ir de un punto i a un punto j es igual al costo de ir de un punto j a un punto i entonces estamos ante el caso del CVRP simétrico. En caso de que ir de un punto i a un punto j no cueste lo mismo que ir de un punto j a un punto i estamos ante el caso del CVRP asimétrico. Por otra parte del CVRP surgen dos grandes problemas, el VRP homogéneo y el VRP heterogéneo. En el VRP homogéneo los puntos utilizan los mismos recursos de distancia, retornos, ventanas de tiempo mientras que en el VRP heterogéneo cada nodo maneja diferentes recursos, ya sea flota de vehículos heterogénea, depósitos, etc.
En 1960 aparece la primera referencia al TSP múltiple (m-TSP) de la mano de Miller, Tucker y Zemlin. Se trata de una generalización del TSP donde se cuenta con un deposito y m vehículos, es decir m agentes viajeros. El objetivo consiste en construir exactamente m rutas, una por vehículo. De esta forma cada cliente es visitado una vez por un vehículo. Cada ruta tiene que comenzar y finalizar el recorrido en el deposito de partida y puede contener a lo sumo n clientes, $n \leq m$.
En 1969 aparece el TSP probabilístico o PTSP gracias al trabajo de Taillman. El PTSP tiene como fin encontrar el recorrido de menor costo entre múltiples nodos donde cada nodo cuenta con una probabilidad asociada a la presencia o no de consumidores que requieren ser atendidos [32].

A continuación se muestra en forma esquemática la evolución histórica de los problemas de ruteo partiendo del problema TSP:

- TSP
    - m-TSP
        - m-TSPTW
    - PTSP
        - m-PTSP
    - VRP o CVRP
        - VRP homogéneo
        - VRP heterogéneo

4.2.1. VRP Homogéneo

Dentro del VRP homogéneo existen cuatro tipos. Por un lado el DVRP o distancia VRP, el VRPTW o VRP con ventanas de tiempo, el VRPB o VRP con retornos y por ultimo el SDVRP o VRP con entregas divididas.

Dentro del DVRP existe una variante ampliamente estudiada, el DCVRP o VRP de distancia y capacidad [30] [31] [32]. El DVRP es la primera variante del CVRP desarrollada por Toth y Vigo. En esta variante se reemplaza la restricción de capacidad por la de tiempo y longitud de arcos (linea que une dos nodos). En caso de existir restricción tanto en la capacidad como en la distancia el problema deriva en el DCVRP [30] [33] [34]

El VRPTW es un VRP que surge en 1967 en el que cada ciudad/nodo/punto tiene asociado un intervalo de tiempo en el cual el lugar puede ser visitado. Si se pasa por un punto de entrega antes del tiempo mínimo donde el punto esta disponible entonces se debe esperar hasta que el punto este habilitado. No se puede pasar por una ciudad después del horario máximo donde el punto esta disponible. Este problema tiene como objetivo encontrar recorridos con costo mínimo, es decir, donde cada recorrido visite el origen, los puntos una sola vez y la demanda de los puntos visitados no exceda la capacidad del vehículo. El VRPTW cuenta con varias variantes entre las que cabe resaltar el VRPTD o VRP con ventanas de tiempo fijas (1986), VRPMTW o VRP con múltiples ventanas de tiempo (1988) o el VRPSTW o VRP con ventanas de tiempo blandas (1992) [30] [31] [32] [34]

El VRPB surge en 1985 en el que cada cliente puede solicitar o entregar mercancías. En este problema por cuestiones de complejidad se suele asumir que las entregas deben realizarse antes que las recogidas de mercadería. Por otra parte la cantidad de mercadería recogida y entregada suele ser fija. [30] [34] En el VRPB se tienen dos escenarios, por un lado se tienen k puntos de recogida y por otro lado l puntos de entrega.

Existe una variante que combina el VRPTW y el VRPB que surge 1994 llamada VRPBTW.

El SDVRP surge en 1989, se trata de una variante del VRP en la cual se permite que el mismo cliente pueda ser atendido por diferentes vehículos. Esta condición

surge ante la posibilidad de que un vehículo no tenga la capacidad suficiente pare recoger toda la mercadería de un punto. De esta forma si un segundo vehículo pasara a buscar el restante de mercadería podría minimizarse el costo total ya que de otra forma el vehículo que no pudo recoger todo debería volver al deposito y regresar a buscar el resto de mercadería.

En el año 1995 surge una variante que combina el SDVRP y el VRPTW llamada SDVRPTW es decir VRP con entregas fraccionadas y ventanas de tiempo [30] [31] [32].
A continuación se muestra en forma esquemática la evolución histórica del VRP Homogéneo:

- VRP homogéneo
  - DVRP
    - DCVRP
  - VRPTW
    - VRPMTW
    - VRPTD
    - VRSTW
    - VRPBTW
  - VRPB
    - VRPBTW
  - SDVRP
    - SDVRPTW

4.2.2. VRP Heterogéneo

El VRP heterogéneo se puede clasificar en 7 tipos detallados brevemente a continuación.

- VRPHF o VRP con flota de vehículos heterogénea (1994, Golden Assad, Levy y Gheyesens). En esta variante los costos y capacidades de los vehículos cambian, asumiendo una cantidad ilimitada de vehículos de diferentes tipos. Para cada ruta se define la flota de vehiculos a utilizar.

- PVRP o VRP con vehículos periódico (1991, Russell y Gribbin). En esta variante existen M días para la planificación de una ruta mientras que en el VRP normal se suele utilizar un solo día.

- Multi Trip VRP o VRP con múltiples viajes (1990, Fleischman). En esta variante cada vehículo puede realizar varios recorridos en el mismo periodo de tiempo planificado [35] [36] . Esta variante implica no solo la planificación de rutas sino que también la asignación de las mismas a los diferentes vehículos disponibles.

- Multi Depot VRP o VRP con múltiples depósitos (1993, Chao, Golden y Wasil). En esta variante existen múltiples depósitos donde los puntos están ampliamente distribuidos entre los depósitos, por lo que no es posible realizar varios VRP independientes. Es decir, no es posible ejecutar tantos VRP como depósitos existan dada la aleatoriedad de los puntos entre los depósitos. En estos casos se emplea el MDVRP [30] [31] [32] [34]

- MCVRP o VRP con múltiples capacidades (1998, Guan y Zhu). En esta variante se transportan múltiples cantidades de objetos, es decir, capacidad mayor a 1 [37] .

- MOVRP o VRP con múltiples objetivos (1995, Boffe). En esta variante se utilizan diferentes objetivos como puede ser el costo, beneficio, distancias, ventanas de tiempo, nivel de satisfacción con el cliente, etc [32]

- SVRP o VRP estocástico (1983, Stewart y Golden). En esta variante existen algunos componentes aleatorios como pueden ser la demanda de cada consumidor (VRPSTT), cuando los tiempos de atención y de llegada a un punto son aleatorios o cuando cada consumidor tiene cierta probabilidad de estar en el lugar de atención o de estar ausente al momento de llegar el vehículo.

A continuación se muestra en forma esquemática la evolución histórica del VRP Heterogéneo:

- VRP heterogéneo
    - VRPHF
    - PVRP
    - Multi Trip VRP
    - Multi Depot VRP
    - MCVRP
    - MOVRP
    - SVRP

4.3. Formulación matemática para problemas de interés

En esta sección se describen algunas posibles formulaciones matemáticas para los problemas de mayor relevancia para el problema abordado en este informe.

4.3.1. Traveling Salesman Problem

El TSP ha recibido la atención y el estudio de multitud de matemáticos ya que se trata de un problema sencillo de describir pero complejo a la hora de resolver [38]. Por otro lado se trata de un problema base, aplicable para innumerables problemas de este estilo.

Al tratarse de un problema combinatorio el TSP resulta ser un problema altamente complejo (Np-Hard) de resolver de forma computacional con métodos estándar.

El TSP puede formularse de diferentes maneras, pudiéndose representar desde un grafo donde cada ciudad es un nodo del mismo y las lineas dibujadas entre los nodos son los arcos del grafo. Cada arco tiene asociado un costo o distancia que es lo que se quiere optimizar. Cuando el vendedor puede ir de forma directa a cada ciudad (nodo del grafo) se dice que el grafo esta completo. Se define ruta como el subconjunto de arcos (líneas) que unen los nodos (ciudades). El TSP consta de dos casos diferenciados, por un lado si el arco va de un nodo i a un nodo j o si va de un nodo j a un nodo i (problema asimétrico). En el segundo caso resulta indiferente ir del nodo i al j o del nodo j al i (problema simétrico). Se muestra a continuación con mayor detalle ambos casos.

Caso asimétrico

En esta variante cada nodo del grafo debe contener una línea apuntando a el y otra saliendo de el. En estos casos es posible generar subrutas aisladas y cerradas, por ejemplo, A(a,b) y A(b,c) y A(c,a), donde A(i,j) representa un arco desde el nodo i al j. Una formulación correcta del problema debe evitar que ocurra este problema, esto se logra añadiendo restricciones adicionales al problema. Según Hoffman, el problema puede ser formulado de la siguiente manera [39]:

Sea

$$x_{ij} \begin{cases} 1 & \text{si } A(i,j) \text{ esta en la ruta} \\ 0 & \text{en caso contrario} \end{cases}$$

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, c_{ij} \neq c_{ji}$$

s.a:

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i = 1, ..., n$$

$$\sum_{i=1}^{n} x_{ij} = 1, \forall j = 1, ..., n$$

$$\sum_{i \in k} \sum_{j \in k} x_{ij} \leq |K| - 1, \forall k \subseteq 1, ..., n$$

$$x_{ij} \in 0{,}1, \forall i, j = 1, ..., n$$

k subconjunto no vacío de nodos. La función objetivo busca encontrar la ruta con menor costo cumpliendo con todas las restricciones especificadas. La primer restricción limita la cantidad de arcos que salen de un nodo i a otro nodo diferente. La segunda restricción acota la cantidad de arcos que entran al nodo. De esta forma cada nodo recibe únicamente una línea de un nodo y se dirige únicamente a un nodo. La tercer restricción limita la cantidad de subrutas y la última restricción define el dominio de la variable de decisión x.

Caso simétrico

En esta variante el costo para ir del nodo i al nodo j es el mismo que el costo de ir del nodo j al nodo i. Sea x_r = 0,1 la variable de decisión donde r $\in$ A son todos

los arcos del grafo y c_r el costo de recorrer el arco r. Una posible formulación del problema es la siguiente:

$$min \frac{\sum_{j=1}^{n} \sum_{k \in J} c_k x_k}{2}$$

s.a:

$$\sum_{k \in J}^{n} x_k = 2, \forall i = 1, ..., n$$

$$\sum_{j \in E_k} \leq |K| - 1, \forall k \subseteq 1, ..., n$$

$$x_r = 0{,}1, \forall r \in E$$

J es un conjunto de todos los arcos sin dirección conectados al nodo j y E es un subconjunto de todos los arcos sin dirección conectados al subconjunto no vació de nodos K.

La función objetivo busca encontrar arcos que minimicen el costo del recorrido total. La primer restricción indica que solo pueden haber dos arcos conectados al nodo j. La segunda restricción evita las subrutas a la hora de formar rutas y la tercer restricción indica que x_j tiene el valor de 1 si el arco es recorrido.

Como se mencionó anteriormente el TSP resulta un problema atractivo para su estudio dada la facilidad para ser planteado y la complejidad para ser resuelto. Esta facilidad para plantearlo ha desencadenado multitud de variantes del mismo problema. Cada variante agrega nuevas restricciones al mismo haciéndolo más complejo de resolver y a su vez adaptado a infinidad de problemas de la vida real. A continuación se presenta el Vehicle Routing Problem (VPR), variante más complejo de resolver que el TSP donde es posible tener más de una ruta.

4.3.2. Vehicle Routing Problem

Una posible formulación del VRP es la propuesta por Toth y Vigo en 2002 [40]:

Sea un grafo G = (N,A), donde N = 0, 1, .., n es el conjunto de n + 1 puntos (nodos) y A es el conjunto de arcos entre nodos. Se utiliza n+1 puntos ya que el 0 representa el origen (depósito) y los restante n valores representan los puntos (nodos). Sea P = $N \backslash 0$ al conjunto que contiene los puntos sin el deposito.

Cada punto $i \in P$ necesita que le entreguen una cierta cantidad de un producto Q_i que se encuentra en el depósito O. Se cuenta con una flota de K vehículos homogénea con capacidad Q los cuales parten y finalizan su jornada en el depósito O para cumplir con la entrega de los productos a todos los puntos. Por otra parte se cuenta con los costos del recorrido, es decir, ir de un punto i a un punto j tiene asignado un costo no negativo c_ij. Se define ruta al conjunto ordenado de puntos que se deben visitar, es decir $R = i_1, ...i_n$, donde $i_1 = i_2 = 0 = O$.

Por lo tanto cada punto debe ser visitado una única vez por cada ruta, todas las rutas deben iniciar y finalizar en el depósito y la suma de la demanda de los puntos pertenecientes a una ruta es menor o igual a la capacidad del vehículo asignado a esa ruta.

De la misma forma que en el TSP tenemos la variante asimétrica y la variante simétrica. La primera implica que el costo $c_ij \neq c_ji$ mientras que en la variante simétrica $c_ij = c_ji$.

Sea

$$x_{ij} \begin{cases} 1 & \text{si el vehículo viaja del punto i al j} \\ 0 & \text{en caso contrario} \end{cases}$$

Una posible formulación del problema es la siguiente:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

s.a:

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i \in P$$

$$\sum_{i=1}^{n} x_{ij} = 1, \forall j \in P$$

$$\sum_{i=1}^{n} x_{i0} = K$$

$$\sum_{j=1}^{n} x_{0j} = K$$

$$\sum_{i \in P'} \sum_{j \notin P'} x_{ij} \geq \alpha(P'), \forall P' \subseteq P, P' \neq \emptyset$$

$$x_{ij} \in 0{,}1, \forall i, j \in N$$

Se busca minimizar los costos de recorrer todas las rutas. La primer restricción asegura que un punto se visite una única vez. Si un vehículo visita un punto el vehículo debe partir de ese punto. Las siguientes restricciones determinan que un numero K de vehículos deben partir y volver al deposito. La penúltima restricción es para que se cumpla el requisito de capacidad del vehículo y la ultima restricción determina el dominio de las variables de decisión.

4.3.3. Variantes VRP

Gran parte de los modelos y algoritmos utilizados para resolver problemas de ruteo no son capaces de captar todo el conjunto de restricciones de un problema practico. Una de estas restricciones, es que un vehículo deba pasar por la localización de un punto de entrega en un cierto intervalo de tiempo (ventana de tiempo). Incorporar este tipo de restricciones da lugar a problemas mixtos que juntan programación, optimización y ruteo.

Se describe a continuación los modelos de TSP con ventanas de tiempo (TSPTW), el problema de Rutas de Vehiculo con ventanas de tiempo (VRPTW), el problema de Cargas y Descargas con ventanas de tiempo (PDPTW) con ventanas de tiempo (DARPTW).

Los modelos presentados a continuación siguen el enfoque de Desrochers (1988). Se tomaran las siguientes consideraciones: Sea $N = 2, 3, ...n$ el conjunto de clientes y $A = (i, j)/i, j \in N \subset 1, i \neq j$

Otras formulaciones a los problemas con ventana de tiempo pueden ser encontradas en los trabajos de Dagazno, o en los trabajos de Tsitsiklis (1992).

TSP con ventanas de tiempo (TSPTW)

Se trata de un TSP en el que cada ciudad/nodo tiene asociado un intervalo de tiempo $[c_i, f_i], i = 1, ..n$ en el que la ciudad puede ser visitada. Si se pasa por una ciudad i antes del instante c_i entonces hay que esperar hasta ese momento. No se puede pasar por una ciudad después del instante f_i. Se asume que el tiempo de partida inicial es c_1.

Sea d_ij la distancia entre la ciudad i y la ciudad j.
Sea t_ij el tiempo de trayecto entre las ciudades i y j.

Sean las variables:

$$x_{ij} \begin{cases} 1 & \text{si el vehículo viaja del punto i al j de forma inmediata} \\ 0 & \text{en caso contrario} \end{cases}$$

Sea D_i el tiempo de llegada a la ciudad i

Una posible formulación del problema es la siguiente:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

s.a:

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i = 1, .., n$$

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = 0 i = 1, .., n$$

$$x_{ij} = 1 => D_i + t_{ij} \leq D_j, i = 2, .., n; j = 1, .., n;$$

$$x_{ij} = 1 => D_j \geq c_1 + t_{1j}, j = 2..n;$$

$$c_i \leq D_i \leq f_i, i = 1..n;$$

$$x_{ij} \in \{0, 1\}, i, j = 1, .., n$$

VRP con ventanas de tiempo (VRPTW)

Se trata de un VRP en el que a cada ciudad se le asocia un intervalo de tiempo $[c_i, f_i], i = 1, ..n$. Solamente en ese intervalo de tiempo la ciudad puede ser visitada. Si se llega a la ciudad i antes del instante c_i se debe esperar hasta ese momento. No se puede llegar a la ciudad i después del instante f_i. Se asume que el tiempo de salida inicial es c_1

Sea d_ij la distancia entre la ciudad i y la ciudad j. Sea t_ij el tiempo de trayecto entre las ciudades i y j.

Sean las variables:

$$x_{ij} \begin{cases} 1 & \text{si el vehículo viaja del punto i al j de forma inmediata} \\ 0 & \text{en caso contrario} \end{cases}$$

Sea D_i el tiempo de llegada a la ciudad i
y_i la carga que lleva el vehículo cuando llega a la ciudad i, i = 2,..,n.

Una posible formulación del problema es la siguiente:

$$min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

s.a:

$$\sum_{j=1}^{n} x_{ij} = 1, \forall i = 2, .., n$$

$$\sum_{j=1}^{n} x_{ij} - \sum_{j=1}^{n} x_{ji} = 0, i = 1, .., n$$

$$x_{ij} = 1 => D_i + t_{ij} \leq D_j, i = 2, .., n; j = 2, .., n;$$

$$x_{ij} = 1 => D_j \geq c_1 + t_{1j}, j = 2..n;$$

$$x_{j1} = 1 => D_j + tj1 \leq c_1, j = 2..n;$$

$$c_i \leq D_i \leq f_i, i = 1..n;$$

$$x_{ij} \in \{0, 1\}, i, j = 1, .., n$$

Problema de carga y descarga con ventanas de tiempo (PDPTW)

En este problema cada cliente i necesita que se le envíe paquetes desde un origen o a un destino d. Para conseguir esto se dispone de m vehículos que parten de una ciudad de origen oc a la que regresan al finalizar su recorrido. Cada punto j de carga o descarga debe ser visitado dentro de un intervalo de tiempo $[c_j, f_j]$. Se busca minimizar la distancia total a recorrer por los vehículos y las rutas asignadas a los mismos.

Sea:

$$O = \{o/o \in N\}$$

el conjunto de origenes

$$DS = \{d/d \in N\}$$

el conjunto de destinos

$$V = O \cup D \cup \{1\}$$

c_{ij} es la distancia entre los puntos i y j, $\forall i, j \in V$. t_{ij} es el tiempo de viaje entre los puntos i y j, $\forall i, j \in V$ Q es la capacidad máxima de cada vehículo (vehículos homogéneos).

$$x_{ij}^k \begin{cases} 1 & \text{si el vehículo k viaja del punto i al j.} \\ 0 & \text{en caso contrario} \end{cases}$$

Sea D_i el tiempo de llegada al punto i por algún vehículo, $\forall i, j \in O \cup DS$
y_i la carga que lleva el vehículo cuando llega a la ciudad i, $\forall i \in O \cup DS$

Una posible formulación del problema es la siguiente:

$$min \sum_{k=1}^{m} \sum_{i=1 \in v}^{n} \sum_{j=1 \in v}^{n} c_{ij} x_{ij}^k$$

s.a:

$$\sum_{k=1}^{m} \sum_{j \in V} x_{ij}^k = 1, \forall i \in O$$

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{ji}^k = 0, i \in O, k = 1, .., m$$

$$\sum_{j \in V} x_{i+j}^k - \sum_{j \in V} x_{ji}^k = 0, i \in O, k = 1, .., m$$

$$x_{ij}^k = 1 => D_i + t_{ij} \leq D_j, i, j \in O \cup DS, k = 1, .., m$$

$$c_i \leq D_i \leq f_i, i = 1..n;$$

$$x_{i_o j}^k = 1 => y_i + q_i \leq y_i, i \in O, j \in O \cup DS, k = 1, .., m$$

$$x_{i_{ds} j}^k = 1 => y_i - q_i \leq y_i, i \in O, j \in O \cup DS, k = 1, .., m$$

$$0 \leq y_i \leq Q, i \in O \cup DS$$

$$x_{ij} \in \{0, 1\}, i, j = 1, .., n, k = 1, .., m$$

4.4. Mecanismos de resolución

Se pueden clasificar los métodos de resolución para problemas de ruteo en tres categorías:

- Métodos exactos
- Heurísticas
- Metaheurísticas

4.4.1. Métodos Exactos

Estos métodos se consideran efectivos para problemas de porte pequeño no superando los mismos los 20 depósitos para problemas como TSP y los 50 depósitos para problemas como el VRP y alguna de sus variantes [35]. Esto se debe al costo computacional que implicaría utilizar estos métodos para una mayor cantidad de depósitos, además de que cuanto mas compleja sea la variante a resolver mayor serán los tiempos. Dentro de los métodos exactos se pueden encontrar tres tipos:

- Métodos de búsqueda directa de árbol. La búsqueda se lleva a cabo sobre todos los nodos del arbol segun ciertos criterios de búsqueda [41]. Según el criterio utilizado se determina una variante u otra del método. A continuación se nombran dichas variantes:
  - Asignación de cota inferior [42]
  - Ramificación y acotamiento [43]
  - Ramificación y corte [43]
  - Árbol de centro de k-grados [42]
- Programación dinámica. Se utilizan un numero fijo de n vehículos. En primer lugar se busca el costo mínimo que es posible alcanzar utilizando j vehículos ($j \leq n$) teniendo en cuenta la función costo en el recorrido de los vehículos. Por ultimo se encuentra el costo de todos los subconjuntos de recorridos con n vehículos [42].
- Programación entera y programación lineal. Este método garantiza la obtención de una solución óptima en un numero finito de pasos siempre y cuando se ejecute el algoritmo hasta finalizarlo. Existen tres técnicas de programación lineal y entera para abordar este tipo de problemas [42]:
  - Conjunto de particiones y generación de columnas.
  - Formulación de flujo de vehículos de tres índices
  - Formulación de flujo de vehículos de dos índices

4.4.2. Heurísticas

Las heurísticas son procedimientos que ofrecen soluciones de gran calidad a través de una exploración limitada del espacio de búsqueda. [31] [44] [45]
Clarke y Wright proponen en 1964 un algoritmo efectivo para resolver el VRP. [30] [46] [47] [48] [49]

Se pueden clasificar los métodos heurísticos en:

- Métodos constructivos. Están conformados por algoritmos de ahorro y heurísticas de inserción muy utilizados dentro de la investigación operativa. Dentro de los métodos constructivos se destacan el algoritmo de ahorro de Clarke y Wright, el algoritmo de los ahorros basados en matching y el algoritmo de ahorro mejorado.

- Métodos de fases. Están conformados por los métodos de asignación elemental, el algoritmo de ramificación y acotamiento truncados, el algoritmo de los pétalos, el método de rutear primero y asignar después y los procedimientos de búsqueda local.

4.4.3. Metaheurísticas

Las metaheuristicas surgen a finales de los 90 y están caracterizadas por realizar procedimientos de búsqueda para encontrar soluciones de buena calidad aplicando una serie de operadores independientes al dominio, modificando de esta forma soluciones intermedias empleando la función objetivo.

Las metaherusticas se pueden clasificar en 6 tipos fundamentales mencionados a continuación y describiendo los que resultan relevantes para el informe:

- Algoritmos genéticos. Como se menciona con mayor detalle en el capítulo 3 estos algoritmos están inspirados en la evolución darwiniana. Los algoritmos genéticos parten de una población inicial que presentan soluciones iniciales factibles pero no necesariamente óptimas. A través de operadores evolutivos que alteran a los individuos de la población el algoritmo va creando nuevos individuos para las siguientes generaciones de poblaciones. De esta forma y a través de una función de fitness se determina la calidad de los individuos.

- Redes neuronales.

- Recocido simulado.

- Algoritmos de hormigas. Estos algoritmos se basan en el funcionamiento habitual de las colonias de hormigas a la hora de conseguir alimento. Cuando una hormiga encuentra una ruta para tener un tipo de alimento esta almacena una feromona dependiendo de la longitud y la calidad del alimento. Las hormigas siguen el camino que tenga mayor cantidad de feromonas ya que es posible que lleve a una ruta mas rápida y hacia un alimento de mejor calidad [50] En el caso particular de los problemas de tipo VRP se inicializa el algoritmo colocando una hormiga en cada uno de los puntos. Si un nodo ya fue visitado la probabilidad de recorrer ese nodo por una hormiga es cero y distinto de cero en caso contrario. Se van actualizando las feromonas y se finaliza

el algoritmo si se llega a obtener una solución con un valor inferior a una cota preestablecida previamente. En caso de no encontrar solución se recalculan las probabilidades de recorrer nodos y las hormigas continúan construyendo nuevos caminos.

- Búsqueda tabú.
- Búsqueda de vecindades.

Capítulo 5

Implementación

El presente capítulo describe el algoritmo evolutivo implementado para resolver el problema propuesto en el capítulo 2. En la sección 5.1 se presenta la biblioteca y herramientas utilizadas para la implementación del algoritmo. En la sección 5.2 se describe detalladamente el proceso de implementación de la solución propuesta.

5.1. Biblioteca de desarrollo

En esta sección se presenta la biblioteca utilizada así como el lenguaje para la resolución del problema propuesto.

5.1.1. ECJ

El lenguaje utilizado para el desarrollo del algoritmo evolutivo fue JAVA, empleando la biblioteca ecj[51] la cual es una biblioteca de computación evolutiva. La librería fue diseñada para resolver problemas grandes y complejos orientados fuertemente hacia la programación genética. ECJ es 'free open-source' y cuenta con más de 15 años de antigüedad encontrandose a día de hoy en un estado maduro y estable con su ultima actualización en el año 2019. Gracias a su diseño es fácilmente mantenible para poder actualizarse sin problemas y adaptarse a las nuevas tecnologías. El sistema es ampliamente utilizado en la comunidad de programación genética.

ECJ fue diseñado para abordar grandes proyectos y resolverlos con facilidad aprovechando las ventajas que brinda la herramienta, aunque esto tiene como inconveniente la elevada curva de aprendizaje inicial. Por esta razón la propia librería incluye multitud de ejemplos con diferentes escenarios y problemas clásicos resueltos donde es posible entender el funcionamiento general de la mayor parte de componentes que ofrece ECJ.

Entre otras cosas, se ofrece herramientas para facilitar la impresión de estadísticas de las ejecuciones realizadas junto con un generador de números pseudoaleatorios.

5.2. AE para el problema propuesto

En esta sección se describen como son los datos de entrada y salida del algoritmo. Se describen también como se codifican las soluciones, la función de fitness utilizada, los operadores de selección, recombinación y mutación utilizados. Por último se presenta el mecanismo de inicialización de la población utilizado así como el operador correctivo de soluciones no factibles implementado.

5.2.1. Datos de entrada y salida

El algoritmo evolutivo implementado recibe como entrada, para poner en marcha su funcionamiento, un archivo 'in.txt' el cual contiene la información necesaria para comenzar a operar. Este archivo de entrada contiene la dirección de la ruta de salida para arrojar un archivo con el resultado de la ejecución y la información de los vehículos y los puntos que se desean utilizar en el problema. La información de los vehículos y puntos es almacenada en el archivo de entrada en formato JSON, uno para la información de vehículos y otro para la información de los puntos. A continuación se muestra un ejemplo del formato para el JSON de vehículos y puntos respectivamente.

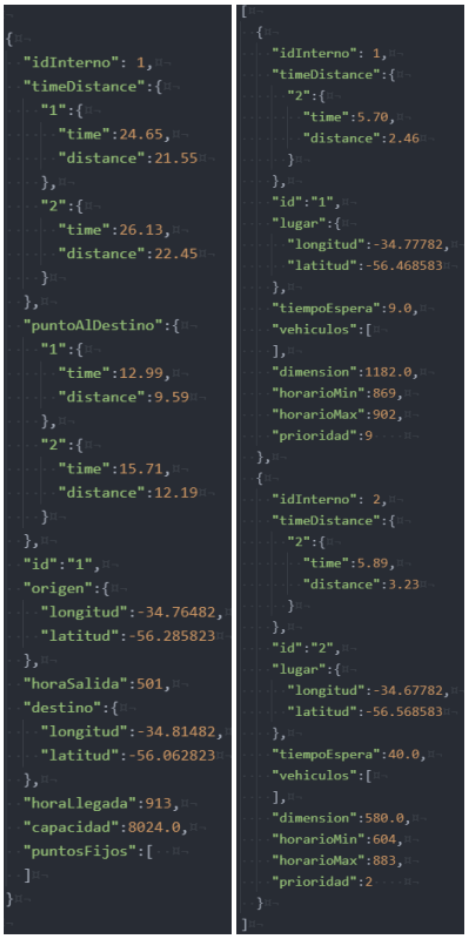

Figura 5.1: Ejemplo de JSON de entrada para vehículos y puntos.

Por lo tanto un archivo de entrada tiene el siguiente formato:

- Ruta de salida
- JSON vehiculos
- JSON puntos

El algoritmo evolutivo una vez finaliza, devuelve un archivo 'out.txt' en la ruta especificada por la dirección de salida contenida en el archivo de entrada. Este archivo de salida contiene la información de los puntos asociados a cada vehículo en orden de recogida. En caso de quedar puntos pendientes (puntos que no fue posible asignarlos a un vehículo) se devuelve una lista de puntos pendientes junto a la solución. La información de los puntos asociados a los vehículos y los puntos pendientes se almacena en el archivo en formato JSON. De esta forma el resultado puede ser eventualmente utilizado por algún sistema externo o procesado por algún middleware. A continuación se muestra un JSON de ejemplo válido para un archivo de salida.

Figura 5.2: Ejemplo de JSON de salida.

5.2.2. Codificación de las soluciones

Sea P el número de puntos.

Las soluciones del problema son representadas como tuplas de largo 3P-1 donde los primeros 2P-1 lugares representan valores con los puntos y valores separadores que cumplen la función de dividir puntos entre vehículos y poder distinguir de esta forma donde empiezan y terminan los puntos asignados a un vehículo. Los últimos P lugares (puede darse que cada punto requiera un vehículo) representan los valores con los identificadores del vehículo. Si quedan puntos sin recoger se utiliza el 0 del lado de los vehículos. Como separador entre puntos se utiliza el valor 0.

Los puntos se numeran con enteros a partir del número 1, pues el 0 está reservado para representar el separador. Los vehículos de la misma forma se numeran con enteros a partir del número 1, siendo este numero un identificador del vehículo. Por otro lado, el orden en el que aparecen los puntos en un vehículo hace referencia al orden de recogida de dichos puntos. Es decir:

```
|4|5|0|6|0|1|2|0|
|  puntos |vehíc|
```

Implica que el primer vehículo con identificador 1 primero pasa por el punto 4, luego por el 5 y por ultimo va al destino. Luego el vehículo con identificador 2 pasa por el punto 6 y se dirige al destino.

Un ejemplo para representar que el punto 4 y 5 son recogidos por un vehículo con identificador 1 y los puntos 6, 7 son recogidos por un vehículo con identificador 3 dejando el punto 8 pendiente seria de la siguiente forma:

```
|4|5|0|6|7|0|0|8|0|1|3|0|0|0|
|     puntos      |vehículos|
```

Como se puede observar, el punto 8 esta asignado a un vehículo con identificador 0, esto para el algoritmo implica que el punto está pendiente y no pudo ser asignado a un vehículo.

En este otro ejemplo se muestra una solución valida para un problema de 3 puntos donde cada punto es recogido por un vehículo diferente:

```
|4|0|5|0|6|1|5|7|
|  puntos |vehíc|
```

Una solución inválida sería, por ejemplo que existan más puntos en un vehículo que la capacidad que este permite. Sea el siguiente escenario donde el vehículo 1 (posición 9) tiene una capacidad de 1000L y cada punto asignado a el tiene un volumen de carga de 250L. En esta caso la capacidad de los puntos recogidos supera la capacidad del vehículo ($1250 > 1000$).

```
|4|5|6|7|8|0|0|0|0|1|2|3|4|5|
|     puntos      |vehículos|
```

Este tipo de soluciones no válidas son corregidas por el algoritmo de forma tal de generar siempre soluciones válidas. No se admiten soluciones no válidas.

5.2.3. Función de fitness

El objetivo del problema es minimizar el costo total, es decir, obtener una solución con la menor cantidad de puntos pendientes, la menor distancia recorrida y que pase por la mayor cantidad de puntos prioritarios cumpliendo todas las restricciones. Las soluciones con menor costo total serán las mas adecuadas para resolver el problema. Para transformar el problema de minimización de costo en un problema de maximización de fitness, se utiliza el inverso de la función de costo total de una solución (CT):

```
F = 1 / CT
```

5.2.4. Selección

Existen muchos operadores de selección [26]. Para este problema se utilizaron dos mecanismos de selección. Estos mecanismos fueron elegidos a partir de los resultados de la configuración parmétrica (Sección 6). De los resultados de la configuración paramétrica para este problema se obtuvieron como operadores de selección el operador de selección proporcional, normalmente conocido como selección por ruleta y una variante del mismo como segundo operador de selección.

La selección proporcional o selección por ruleta asigna a cada individuo de la población una probabilidad de selección dada por el cociente entre su valor de fitness y la suma del fitness del total de los individuos en la población. La variante empleada se conoce como Operador de selección Greedy, el cual funciona de la siguiente manera: Los individuos son ordenados y divididos en 2 grupos: 'fitter' y 'less fit'. En el grupo 'fitter', el cual corresponde al 25 % de la subpoblación, irán los de mayor valor de fitness. Con cierta probabilidad (40 %) los individuos del grupo 'fitter' serán seleccionados (usando selección proporcional). Sino, los individuos del grupo 'less fit' serán seleccionados (también usando selección proporcional).

5.2.5. Recombinación

Se utilizó una versión modificada del operador de Cruzamiento Basado en la Posición (Position Based Crossover - PBX)[52] el cual se describe a continuación:

1. Seleccionar aleatoriamente varias posiciones del padre 1.
2. generar parcialmente el hijo 1, copiando los valores (alelos) de las posiciones elegidas del padre 1.
3. Marcar los alelos del padre 2 que ya fueron seleccionados en el padre 1. Si en el padre 1 el valor 0 fue seleccionado, entonces en el padre 2 se marca el valor 0 la cantidad de veces que fue seleccionado en el padre 1, en caso de haber suficientes 0 en el padre 2.
4. Desde el inicio del padre 2, seleccionar secuencialmente el siguiente alelo que no haya sido marcado, y ponerlo en la primer posición libre del hijo 1, desde el comienzo.

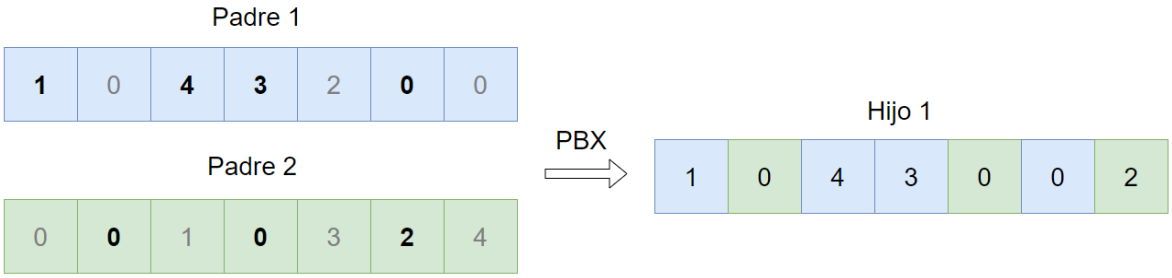

Figura 5.3: Ejemplo de cruzamiento PBX.

En el ejemplo anterior son seleccionados los valores 1, 4, 3 y 0 del padre 1. El padre 2 no podrá utilizar el primer 0, ni el valor 1, ni el valor 3 ni 4.

La variante implementada realiza este procedimiento para cada una de las partes de la representación. Esto puede dar lugar a la generación de soluciones invalidas, por lo que, posteriormente a la ejecución del operador de recombinación, se aplica el operador de corrección para estos casos.

5.2.6. Mutación

Se utilizo una variante del operador de Mutación por Intercambio EM(Exchange Mutation)[53]. El mismo sortea aleatoriamente dos posiciones en la solución y las intercambia. La variante implementada realiza este procedimiento para cada una de las partes de la representación. Es decir, sortea aleatoriamente dos posiciones para la parte que representa a los puntos y sortea de la misma manera dos posiciones para la parte de la representación que conforman los vehículos. Esto puede dar lugar a la generación de soluciones invalidas, por lo que, posteriormente a la ejecución del operador de mutación, se aplica el operador de corrección para estos casos.

Figura 5.4: Ejemplo de mutación EM.

5.2.7. Generación de la población inicial

Para generar la población inicial se utiliza un algoritmo Greedy. El algoritmo asigna vehículos en orden de mayor a menor capacidad. Se toma el primer vehículo disponible y se busca el punto más óptimo para ir a recoger hasta completar el vehículo (en cuyo caso se pasa al siguiente vehículo disponible) o hasta que se haya pasado por todos los puntos. Un punto se considera óptimo si:

- 1. Es un punto fijo del vehículo
- 2. Es un punto de la mayor prioridad posible, cumple las 5 restricciones del problema y no es un punto fijo de otro vehículo.

El punto 1 tiene preferencia sobre el punto 2 ya que es un requisito del problema pasar por todos los puntos fijos del vehículo. Por letra se asume que es posible pasar

por todos los puntos fijos del problema y cumplir las 5 restricciones. De forma tal que un vehículo que tenga x puntos fijos pasará por al menos x puntos para recoger, $x >= 0$

Algoritmo Greedy

El algoritmo asigna vehículos en orden de mejor a peor vehículo, es decir, primero se escoge el mejor vehículo disponible, en segundo lugar el segundo mejor vehículo y así sucesivamente hasta completar la cantidad de vehículos necesaria. Para el problema planteado no se tienen en cuenta el tipo de vehículo ni el consumo de combustible por lo que un vehículo se considera mejor que otro si tiene mayor capacidad. En caso de haber más lugares en el arreglo para vehículos que la cantidad de vehículos disponible se completa los lugares restante con 0.

Se toma el primer vehículo disponible y se busca el punto óptimo para ir a recoger. Este proceso se repite hasta completar el vehículo (en cuyo caso se pasa al siguiente vehículo disponible) o hasta que todos los puntos hayan sido recogidos.

Primero se recogen los puntos óptimos que sean fijos del vehículo. Puede suceder que para un vehículo con x puntos fijos donde $x > 1$ exista solamente una combinación válida (no puede ser cero por letra) para poder pasar a recoger todos los puntos fijos de ese vehículo y cumplir las restricciones del problema. Para estos casos el algoritmo busca la combinación válida de puntos fijos y los recoge en el orden asignado. Una vez se tienen los puntos óptimos correspondientes a los puntos fijos del vehículo se procede a buscar tantos puntos óptimos, en el resto de puntos, como sea posible para completar la capacidad del vehículo. Para ello se busca puntos óptimos que tengan la mayor prioridad posible, no sean puntos fijos de otro vehículo y cumplan las 5 restricciones del problema. Puede ocurrir que se encuentren varios puntos óptimos con la misma prioridad que no sean puntos fijos de otro vehículo y que cumplan todas las restricciones del problema. En estos casos se toma como punto óptimo el punto más cercano desde donde parte el vehículo.

```
Mientras vehiculosSinAsignar() hacer
     v = elegirMejorVehiculo({vehiculosRestantesPorAsignar});
     colocarVehiculo(v);
Fin Mientras
completarConCeroLugaresSobrantes();
volumen = 0;
VehiculoId = primerVehiculoAsignado();

 Mientras puntosSinAsignar() hacer
     si(VehiculoId == null) entonces //No quedan mas vehiculos, solo ceros
        agregarPunto(null, puntosRestantesPorAsignar().siguiente(), solucion)
     en caso contrario
         puntoOptimo = buscarPuntoOptimo(puntosRestantesPorAsignar())
         si puntoOptimo == null entonces
             //no hay mas puntos posibles para asignar al vehiculo
             //se agrega separador a la solucion (0)
             agregarPunto(null, 0, solucion)
             volumen = 0
```

```
                vehiculoId = siguienteVehiculoAsignado()
            en caso contario
                agregarPunto(vehiculoId, puntoOptimo, solucion)
                volumen += puntoOptimo.volumen()
            fin si
        fin si
    fin mientras
    retornar solución
```

5.2.8. Mecanismo de inicialización

El siguiente algoritmo describe el funcionamiento del mecanismo de inicialización de la población inicial. La población inicial se inicializa a partir del resultado del algoritmo Greedy. Siendo el primer individuo directamente el retornado por el dicho algoritmo. Posteriormente los restantes individuos, se inicializan realizando un número aleatorio de alteraciones sobre la solución del algoritmo Greedy. Cada alteración consiste en intercambiar dos valores del individuo tanto para la sección de puntos como para la sección de vehículos. La cantidad máxima de alteraciones utilizadas es un cuarto del tamaño del genoma (3 *cantidadPuntos - 1), ya que a mayor numero de alteraciones sobre el individuo mas posibilidades hay de que represente una solución inválida.

```
desde i = 1 a 3P-1 hacer
    poblacion[1][i] = Greedy[i];
fin desde
//Asigno el resultado del algoritmo
//Greedy al primer individuo

desde i = 2 a tamPoblacion hacer
    individuo = poblacion[i];
    desde j = 1 a 3P - 1 hacer
    individuo[j] = Greedy[j];
    alteraciones = aleatorio(1, (3p-1)/4);
    init = random(1, tamGenoma);
    desde k = 1 a alteraciones hacer
        //Seccion puntos
        pos0_1 = random(0, 2P - 2);
        pos0_2 = random(0, 2P - 2);

        /Seccion Vehiculos
        pos1_1 = random(2P-1, 3P - 2);
        pos1_2 = random(2P-1, 3P - 2);

        mientras pos0_1 == pos0_2 hacer
            pos0_2 = random(0, 2P - 2);
        fin mientras

        individuo.intercambiar(pos0_1, pos0_2);
```

```
        mientras pos1_1 == pos1_2 hacer
            pos0_2 = random(2P - 1, 3P - 2);
        fin mientras

        individuo.intercambiar(pos0_1, pos0_2);
    fin desde
    corregirIndividuo();
fin desde
```

Luego de inicializar la población, alguno de los individuos generados puede violar las restricciones del problema por lo que, posteriormente a la ejecución del algoritmo de inicialización, se aplica el operador de corrección para estos individuos.

5.2.9. Operador correctivo

El genoma se corrige en 5 partes de forma incremental. Cada una de las partes se encarga de corregir una de las restricciones del problema. Dejando al final de la ejecución de dicho operador un individuo factible. Cada vez que se ejecuta una de las correcciones, la misma se asegura de no romper las anteriores. Es decir, la corrección 3 no rompe el resultado de la corrección 2 ni el de la 1. De esta forma primero se corrige los puntos fijos de cada vehículo. Luego los vehículos permitidos en puntos, en tercer lugar la capacidad de los vehículos, la hora de llegada y por último la ventana de tiempo. Por lo tanto después de la primera corrección cada vehículo tendrá todos sus puntos fijos. Al finalizar la tercera corrección todos los vehículos tendrán una cantidad de puntos que haga que no se supere la capacidad de dicho vehículo, tendrán todos sus puntos fijos y pasarán por los puntos permitidos. Lo mismo ocurre con el resto de restricciones corregidas.

- **Puntos fijos:** Se recorre cada vehículo v_i y se chequea que tengan todos sus puntos fijos. En caso contrario, se busca el/los v_j, i<>j que los contenga. Para cada punto fijo de v_i en v_j, pf_i_j se agrega ese punto a v_i y se quita de v_j. Se repite para cada vehículo. Puede ocurrir que los puntos fijos de un vehículo no estén en el propio vehículo ni en el resto de vehículos de la solución, por lo que estos puntos fijos se encuentran como puntos pendientes. En estos casos se recorre la lista de puntos pendientes en busca de el/los puntos fijos del vehículo, una vez encontrados se asignan al vehículo correspondiente. Al final se obtienen todos los vehículos con sus puntos fijos cumpliendo de esta forma la restricción de los puntos fijos.

```
Para Vehiculo v : solucion.vehiculos() hacer
    Para pf : v.puntosFijos() hacer
        Si !solucion(v).contiene(pf)
        //si la lista de puntos asignados al vehiculo v
        //no contiene el punto fijo pf se busca
        punto, v2 = buscarPuntoEnRestoVehiculos(pf, solucion.vehiculos())
            Si punto == null entonces
                //el punto fijo esta en la lista de pendientes
```

```
            punto = buscarPuntoEnListaDePendientes(pf,solucion.pendientes())
                removerPunto(punto, solucion.pendientes())
              En caso contrario
                removerPunto(punto, v2)
              Fin Si
              agregarPuntoAVehiculo(punto, v, solucion)
          Fin Si
      Fin Para
  Fin Para
  retornar solucion
```

- **Vehículos permitidos en punto:** Para cada punto se verifica si por él pasa un vehículo para el cual el punto lo admite. Si no es así, se sortea entre los vehículos que el punto permite y se agrega dicho punto al vehículo correspondiente. Se repite para cada punto. Al final se obtienen vehículos que pasan por sus puntos fijos y por sus puntos admitidos. En este punto se cumplen 2 restricciones. Por letra se asume que un vehículo no puede tener un punto fijo para el cual dicho punto no permita que el vehículo pase por el.

```
Para Vehiculo v : solucion.vehiculos() hacer
   Para Punto p : v.puntos() hacer
       Si noAdmite(v,p)
           v1 = random(p.vehiculosPermitidos())
           agregarPuntoAVehiculo(p, v1, solucion)
           removerPuntoVehiculo(p, v, solucion)
       Fin Si
   Fin Para
Fin Para
retornar solucion
```

- **Capacidad vehículos:** Para cada vehículo que no cumpla la restricción de capacidad se sortean puntos no fijos del vehículo y se añaden a una lista de puntos pendientes. Esto se realiza hasta que el vehículo no incumpla la restricción de capacidad. Para cada punto de la lista de puntos pendientes obtenida del proceso anterior se intenta asignar cada punto a un vehículo que pueda cargarlo. Un vehículo v_i podrá cargar un punto p si al agregar el punto no se incumple la restricción de capacidad del vehículo v_i ni la restricción de vehículos permitidos en el punto p_j. En caso de no poder reasignar el punto a un vehículo queda como pendiente. Cada punto de la lista de pendientes no es fijo de ningún vehículo pues cuando se agregaron puntos a dicha lista se tuvo en cuenta que los puntos sorteados no fueran fijos de los vehículos.

```
Para Vehiculo v : solucion.vehiculos() hacer
    Mientras capacidad(solucion(v)) > v.capacidad() entonces
       puntoASacar = sortearNoFijo(solucion(v))
```

```
        pendientes.add(puntoASacar)
        removerPuntoVehiculo(puntoASacar, v, solucion)
    Fin Mientras
Fin Para

Para Punto p : pendientes hacer
    Para Vehiculo v : solucion.vehiculos() hacer
        Si capacidad(solucion(v)) + p.capacidad <= v.capacidad()
            and p.permiteVehiculo(v) entonces
            agregarPuntoAVehiculo(p, v, solucion)
            removerPunto(p, pendientes)
    Fin Para
Fin Para
agregarPendientes(solucion, pendientes)
retornar solucion
```

- **Hora llegada vehículos:** Para cada vehículo que no cumpla la restricción de hora de llegada, es decir, para cada vehículo cuyo tiempo de salida más tiempo de recogida de sus puntos sea superior a la hora de llegada establecida para ese vehículo se sortean puntos no fijos y se añaden a una lista de puntos pendientes. Esto se realiza hasta que el vehículo no incumpla la restricción de hora de llegada. Al estar quitando puntos no fijos no se incumple la restricción de puntos fijos y tampoco la restricción de capacidad ya que se libera carga en el vehículo. Para cada punto de la lista de puntos pendientes obtenida del proceso anterior se intenta asignar cada uno a un vehículo que pueda cargarlo. Un vehículo v_i podrá cargar un punto p si al agregar el punto en alguna posición del recorrido del vehículo no se incumple la restricción de hora de llegada, la restricción de capacidad del vehículo v_i ni la restricción de vehículos permitidos en el punto p_j. Para un punto pendiente p que se intente colocar en un vehículo v hay que comprobar las combinaciones de posiciones donde puede ir ese punto. Esto es así ya que colocar ese punto en una posición p_i del vehículo puede hacer que no se cumpla la condición de hora de llegada pero si se coloca el punto en una posición p_j, $p_i \neq p_j$, del recorrido puede dar lugar a que si se cumpla la condición por cuestiones de distancias entre un punto y otro. En caso de no poder reasignar el punto a un vehículo queda como pendiente.

```
Para Vehiculo v : solucion.vehiculos() hacer
Mientras v.horaSalida()+tiempoRecogida(solucion(v))> v.horaLLegada() entonces
       puntoASacar = sortearNoFijo(solucion(v))
       pendientes.add(puntoASacar)
       removerPuntoVehiculo(puntoASacar, v, solucion)
Fin Mientras
Fin Para

Para Punto p : pendientes hacer
    encontre = false
```

```
    Para Vehiculo v : solucion.vehiculos() hacer
        Si capacidad(solucion(v)) + p.capacidad <= v.capacidad()
            and p.permiteVehiculo(v) entonces
            Para pos : posicionRecorrido(v) hacer
                colocarPuntoEnPosicion(p, pos, v)
                Si  v.horaSalida() + tiempoRecogida(solucion(v), p)
                                <= v.horaLLegada() hacer
                        agregarPuntoAVehiculo(p, v, solucion)
                        removerPunto(p, pendientes)
                        encontre = true
                        break
                Fin Si
            Fin Para
            Si encontre entonces
                break
            Fin Si
    Fin Para
Fin Para
 agregarPendientes(solucion, pendientes)
retornar solucion
```

- **Ventana de tiempo:** Para cada vehículo se chequea que para cada uno de sus puntos, el vehículo pase por ese punto en el rango horario permitido. Si hay al menos un punto que no cumple, se empiezan a probar diferentes combinaciones de orden de pasada por esos puntos. Al combinar puntos no se varía la cantidad de puntos del vehículo por lo que no se incumplen las restricciones de puntos fijos, puntos permitidos y capacidad. Si se puede incumplir la restricción de hora de llegada. Si se encontró una combinación tal que el vehículo pase por sus puntos asociados cumpliendo en cada uno de ellos el rango horario permitido y se sigan respetando la condición de hora de llegada entonces la corrección terminó para este vehículo. En el caso que no haya una combinación posible de sus puntos asociados, entonces se procede a quitar, aleatoriamente y de a uno por vez, los puntos no fijos de los puntos asignados y se intenta encontrar una combinación para este subconjunto de puntos. Esto se repite hasta que el vehículo cumpla esta restricción. Luego, los puntos que se quitaron de los puntos asignados originalmente pasan a quedar como pendientes.

```
        Para Vehiculo v : solucion.vehiculos() hacer
            Si noCumpleVentanaTiempo(v, v.puntos())
                comb = encontrarCombinacion(v, v.puntos())
                Si comb != null
                    modificar(solucion, comb, v)
                En caso contrario
                    Mientras comb == null hacer
                        p1 = random(v.puntos())
                        removerPuntoVehiculo(p1, v, solucion)
                        agregarPendientes(solucion, p1)
```

```
                comb = encontrarCombinacion(v, v.puntos)
                Si comb != null
                   modificar(solucion, comb, v)
                Fin Si
            Fin Mientras
        Fin Si
    Fin Si
Fin Para
retornar solucion
```

Capítulo 6

Evaluación experimental

El presente capítulo describe la evaluación experimental del algoritmo evolutivo implementado para resolver el problema propuesto. La sección 6.1 describe el proceso para generar instancias para la ejecución de pruebas y evaluación de la eficiencia del algoritmo. La sección 6.2 describe las pruebas realizadas, ajuste paramétrico, comparativas con un algoritmo ávido y resultados numéricos durante la evaluación experimental del algoritmo implementado

6.1. Generación de instancias de prueba

Con el objetivo de realizar la evaluación experimental del algoritmo evolutivo se implementó un generador de instancias realistas del problema. La sección 6.1.1 describe el proceso y funcionamiento del generador de instancias.

6.1.1. Proceso de generación de instancias

El generador de instancias se desarrolló utilizando como lenguaje de programación Java. El programa recibe como entrada la cantidad de puntos y vehículos que se quiere generar. Para la generación de coordenadas para los puntos y lugares de partida y llegada de los vehículos se delimito, a efectos prácticos, un rectángulo que abarca la mayor parte del área de Montevideo.

Generación de puntos

Para la generación de puntos se comienza a generar las coordenadas del mismo verificando que pertenezcan al rectángulo delimitado mencionado anteriormente. El tiempo de espera de cada punto se elige de forma aleatoria entre un mínimo y un máximo configurables. Por defecto el tiempo de espera mínimo es de 1 minuto y el tiempo de espera máximo es de 60 minutos. Los vehículos permitidos para cada punto se seleccionan de forma aleatoria entre la cantidad de vehículos disponibles. Cada punto podrá tener desde 0 hasta cantidad de vehículos asignados a sus vehículos permitidos. El volumen de cada punto se elige de forma aleatoria entre un mínimo y un máximo configurables. Por defecto el volumen mínimo es de 500L y el volumen máximo es de 2500L. La ventana de tiempo de cada punto (rango horario donde el punto esta disponible) se elige de forma aleatoria entre un mínimo y

un máximo configurables. Por defecto la hora mínima donde el punto esta disponible es de 480 minutos (8am) y la hora máxima es de 1080 minutos (18pm). Para la ventana de tiempo se tienen en cuenta varios aspectos. La hora de disponibilidad mínima no puede ser superior a la hora de disponibilidad máxima. La espera en un punto no puede superar la ventana de tiempo disponible del punto. A efectos de generar instancias realistas se tiene en cuenta, además, que la diferencia entre la hora máxima disponible de un punto y la hora mínima no sea inferior a 30 minutos. La prioridad se elige de forma aleatoria entre un mínimo y un máximo configurables. Por defecto la prioridad mínima es de 10 (peor prioridad) y la prioridad máxima es de 1 (mejor prioridad).

Generación de vehículos

Para la generación de vehículos se comienza a generar las coordenadas para el origen y el destino, para las cuales se chequea que se encuentren al menos a 2km de distancia y que pertenezcan al rectángulo delimitado mencionado anteriormente. La hora de partida del vehículo y la hora de llegada se generar de forma análoga a como se genera la ventana de tiempo de cada punto. Para la hora de partida y llegada se tienen en cuenta algunos aspectos. La hora de salida debe ser menor a la hora de llegada. A efectos de generar instancias realista se requiere que la diferencia entre la hora de salida y la hora de llegada de un vehículo sea de al menos 180 minutos para tener un recorrido realista que hacer y se aceptan únicamente horas de salida que no superen un valor de 600 (10 am) y horas de llegada que tengan un valor de al menos 900 (15pm). La capacidad de cada vehículo se elige de forma aleatoria entre un mínimo y un máximo configurables. Por defecto la capacidad mínima es de 5000L y la capacidad máxima es de 10.000L. Los puntos fijos de cada vehículo se generan seleccionando una cantidad de puntos fijos aleatoria comprendidas entre un mínimo y un máximo configurables. Por defecto la cantidad de puntos fijos mínima es de 0 y la cantidad de puntos fijos máxima es de 20 puntos fijos por vehículo. Para la selección de puntos fijos se tienen en cuenta algunos aspectos. Los puntos fijos seleccionados para un vehículo tienen como vehículo permitido el vehículo al cual se le asigna el punto fijo o no tiene ninguna restricción en cuanto a vehículos permitidos. Todos los puntos fijos de un vehículo permiten realizar un recorrido para el cual se cumplan todas las restricciones del problema. Esto es así ya que por la realidad se asume que un vehículo debe poder pasar por todos sus puntos fijos y cumplir las restricciones al problema.

Tanto para la generación de puntos como de vehículos los valores mínimos y máximos utilizados son valores estimados previamente por la empresa (en algunos casos llevados al límite para ver el comportamiento del algoritmo) para los recorridos utilizados en el área de Montevideo.

6.2. Análisis experimental

6.2.1. Entorno de ejecución

Todas las ejecuciones del algoritmo se realizaron en cuatro máquinas con las siguientes características:

- Procesador: Intel(R) Xeon(R) Gold 5120 con frecuencia base 2.20GHz y 3.20GHz frecuencia turbo. Dicho procesador cuenta con 14 cores y 28 threads
- Memoria RAM: 16 GB

6.2.2. Configuración paramétrica

Dado que los algoritmos evolutivos son estocásticos, es necesario ajustar sus parámetros antes de realizar un análisis experimental. Para los experimentos de configuración pramétrica se utilizaron tres instancias del problema con las siguientes caracterısticas:

- Una instancia de 50 puntos y 10 vehículos.
- Una instancia de 50 puntos y 11 vehículos.
- Una instancia de 75 puntos y 13 vehículos.

El ajuste se enfoco en estudiar los siguientes parámetros:

- Tamaño de población
- Probabilidad de cruzamiento
- Probabilidad de mutación
- Primer operador de selección
- Segundo operador de selección

Para el tamaño de la población se utilizaron los siguientes valores:

- 50
- 100
- 200

Para la probabilidad de cruzamiento se utilizaron los siguientes valores:

- 0.5
- 0.75
- 1

Para la probabilidad de mutación se utilizaron los siguientes valores:

- 0.01
- 0.05
- 0.1

Para el primer y segundo operador de selección se utilizaron los siguientes valores:

- Selección por torneo (1)
- Selección proporcional (2)
- Selección greedy (3)

Se estudiaron todas las combinaciones de valores sobre las 3 instancias, realizando 50 ejecuciones independientes del algoritmo evolutivo en cada caso. Cada ejecución puede llegar a la generación 10.000 si el algoritmo sigue encontrando buenas soluciones. La condición de parada es llegar a la generación 10.000, no encontrar un individuo mejor durante 1000 generaciones consecutivas o superar el tiempo de ejecución máximo establecido (para este problema se configuro el tiempo máximo de ejecución en 5400 segundos). En total 50*243*3, 36450 ejecuciones. Habitualmente la cantidad de generaciones utilizadas para un algoritmo evolutivo oscilan entre 1000 y 5000. En este caso se optó por darle la posibilidad al algoritmo de llegar a la generación número 10.000 por tratarse de un problema más complejo de lo habitual dada la cantidad de restricciones que el mismo posee. De la misma forma se utilizó como segundo criterio de parada detener al algoritmo en caso de no mejorar durante 1000 generaciones consecutivas. El tiempo máximo de parada se estimo en 5400 segundos por condiciones de la realidad actual del problema. Actualmente el operador encargado de realizar la gestión manual de rutas para este problema tiene un tiempo de acción de al menos 5400 segundos antes de que los vehículos comiencen la jornada.

El estudio fue realizado con los valores de fitness. Como se menciona en el capítulo 5, el objetivo del problema es minimizar el costo total (CT) utilizando la función de fitness definida como

```
F = 1 / CT
```

Por esta razón aquellas configuraciones de parámetros que alcancen valores de fitness mayores serán preferidas, ya que a mayor fitness, menor costo.

Para analizar los resultados obtenidos se realizó el test de Shapiro Wilk [54] para establecer si los valores de fitness obtenidos seguían o no una distribución normal.

Los resultados obtenidos fueron los siguientes:

- La Instancia con 50 puntos y 10 vehículos tuvo 83/243 resultados donde $p - valor < 0{,}05$
- La Instancia con 50 puntos y 11 vehículos tuvo 84/243 resultados donde $p - valor < 0{,}05$
- La Instancia con 75 puntos y 13 vehículos tuvo 209/243 resultados donde $p - valor < 0{,}05$

Por lo tanto se optó por utilizar tests no paramétricos. En este caso se utilizó el test de rangos de Friedman [55].

A continuación se muestran los resultados de dicho test para las tres instancias. Dado que se evaluaron cinco parámetros y cada uno pudiendo contener tres posibles valores (243 combinaciones) se optó por mostrar los cinco mejores resultados

y los cinco peores para cada una de las instancias. En la columna tratamiento se muestra el nombre técnico de la instancia utilizada. El primer número representa la población, el segundo la probabilidad de cruzamiento, el tercero la probabilidad de mutación y los últimos dos números los operadores de selección. Por ejemplo 200-0.5-0.1-1-1 significa que la instancia utiliza como cantidad de población 200, probabilidad de cruzamiento 50 %, probabilidad de mutación 10 % y como ambos métodos de selección utiliza el 1 (selección por torneo)

Instancia	Tratamiento	Suma(Ranks)	Media(Ranks)	n
I50-10	200-0.75-0.01-2-1	11373	227,46	50
	200-0.5-0.01-3-2	11357	227,14	50
	100-0.5-0.01-2-1	11353	227,06	50
	200-0.5-0.01-2-3	11301	226,02	50
	200-0.5-0.01-2-2	11289	225,78	50
	50-0.75-0.1-1-1	1380	27,6	50
	50-1.0-0.1-1-1	1305	26,1	50
	50-1.0-0.1-3-1	1249	24,98	50
	50-0.5-0.1-1-1	1225	24,5	50
	50-1.0-0.1-2-1	1185	23,7	50

Figura 6.1: Instancia de calibración 1 (I5010)

Instancia	Tratamiento	Suma(Ranks)	Media(Ranks)	n
I50-11	200-0.5-0.01-2-3	11494	229,88	50
	200-0.5-0.01-2-1	11423	228,46	50
	200-0.5-0.01-2-2	11375	227,5	50
	200-0.5-0.01-3-2	11372	227,44	50
	100-0.5-0.01-2-3	11328	226,56	50
	50-0.5-0.1-1-2.	1317	26,34	50
	50-0.75-0.1-1-3	1273	25,46	50
	50-0.5-0.1-3-1	1247	24,94	50
	50-0.75-0.1-1-2	1181	23,62	50
	50-1.0-0.1-3-1	1129	22,58	50

Figura 6.2: Instancia de calibración 2 (I5011)

Instancia	Tratamiento	Suma(Ranks)	Media(Ranks)	n
I7513	200-0.5-0.01-2-1	11649	232,98	50
	200-0.5-0.01-2-2	11558	231,16	50
	200-0.5-0.01-2-3	11489	229,78	50
	100-0.5-0.01-2-1	11463	229,26	50
	200-0.75-0.01-2-2	11455	229,1	50
	50-1.0-0.1-1-2	1241,5	24,83	50
	50-0.5-0.1-2-2	1091	21,82	50
	100-0.75-0.1-3-1	1044,5	20,89	50
	100-1.0-0.1-2-1	887,5	17,75	50
	50-1.0-0.1-3-1	847	16,94	50

Figura 6.3: Instancia de calibración 3 (I7513)

Para los tres test se realizó un promedio de la suma de los ranks obtenidos en el test de rangos para los cinco mejores resultados de cada instancia. La configuración cuyo promedio es el mayor se consideró la mejor de las configuraciones.

Instancia	200-0.5-0.01-2-3	200-0.5-0.01-2-1	200-0.5-0.01-2-2	200-0.5-0.01-3-2	100-0.5-0.01-2-1	200-0.75-0.01-2-1	100-0.5-0.01-2-3	200-0.75-0.01-2-2
I5010	11301	11210	11289	11357	11353	11373	11041	11155
I5011	11494	11423	11375	11372	11264	11242,5	11328	11256
I7513	11489	11649	11558	11286	11463	11339	11041	11455
AVG	11428	11427,33333	11407,33333	11338,33333	11360	11318,16667	11136,66667	11288,66667

Figura 6.4: AVG

De esta forma se observa que los mejores resultados se obtienen con la siguiente configuración de parámetros (tratamiento 200-0.5-0.01-2-3):

- Tamaño de población: 200
- Probabilidad de cruzamiento: 0.5
- Probabilidad de mutación: 0.01
- Primer operador de selección: Selección proporcional
- Segundo operador de selección: Selección greedy

Esto es así, ya que, como se mencionó anteriormente, se evalúa el fitness. Por lo tanto a mayor fitness menor costo. Como se puede observar para las tres instancias la configuración elegida (Tratamiento 200-0.5-0.01-2-3) tuvo un mejor desempeño que el resto en promedio.

6.2.3. Comparación con algoritmo greedy

Un algoritmo greedy (o ávido) es un método de construcción de soluciones que toma una decisión localmente óptima en cada paso, con la idea de llegar a un óptimo global del problema. Con el propósito de comparar el algoritmo, se implementó un algoritmo Greedy para resolver el problema planteado en este informe. El algoritmo asigna vehículos de forma aleatoria para ir usándolos de forma ordenada para recoger puntos. Esta asignación aleatoria hace que este algoritmo sea no determinista.

En caso de haber más lugares en el arreglo para vehículos que la cantidad de vehículos disponible se completa los lugares restante con 0.

El funcionamiento a partir de este punto es análogo al descrito en la sección 5.2.7. Se toma el primer vehículo disponible y se busca el punto óptimo para ir a recoger. Este proceso se repite hasta completar el vehículo (en cuyo caso se pasa al siguiente vehículo disponible) o hasta que todos los puntos hayan sido recogidos.

Primero se recogen los puntos óptimos que sean fijos del vehículo. Puede suceder que para un vehículo con x puntos fijos donde $x > 1$ exista solamente una combinación válida (no puede ser 0 por letra) para poder pasar a recoger todos los puntos fijos de ese vehículo y cumplir las restricciones del problema. Para estos casos el algoritmo busca la combinación válida de puntos fijos y los recoge en el orden asignado. Una vez se tienen los puntos óptimos correspondientes a los puntos fijos del vehículo se procede a buscar tantos puntos óptimos, en el resto de puntos, como sea posible para completar la capacidad del vehículo. Para ello se busca puntos óptimos que tengan la mayor prioridad posible, no sean puntos fijos de otro vehículo y cumplan las 5 restricciones del problema. Puede ocurrir que se encuentren varios puntos óptimos con la misma prioridad que no sean puntos fijos de otro vehículo y que cumplan todas las restricciones del problema. En estos casos se toma como punto óptimo el punto más cercano desde donde parte el vehículo.

```
Mientras vehiculosSinAsignar() hacer
     v = elegirVehiculoAleatorio({vehiculosRestantesPorAsignar});
     colocarVehiculo(v);
Fin Mientras
completarConCeroLugaresSobrantes();
volumen = 0;
VehiculoId = primerVehiculoAsignado();

 Mientras puntosSinAsignar() hacer
     si(VehiculoId == null) entonces //No quedan mas vehiculos, solo ceros
        agregarPunto(null, puntosRestantesPorAsignar().siguiente(), solucion)
     en caso contrario
         puntoOptimo = buscarPuntoOptimo(puntosRestantesPorAsignar())
         si puntoOptimo == null entonces
             //no hay mas puntos posibles para asignar al vehiculo
             //se agrega separador a la solucion (0)
             agregarPunto(null, 0, solucion)
             volumen = 0
             vehiculoId = siguienteVehiculoAsignado()
         en caso contario
```

```
            agregarPunto(vehiculoId, puntoOptimo, solucion)
            volumen += puntoOptimo.volumen()
        fin si
    fin si
fin mientras
retornar solución
```

6.2.4. Evaluación del algoritmo evolutivo

El análisis experimental se enfoco en evaluar el desempeño del AE y la calidad de las soluciones alcanzadas. Se utilizaron 20 instancias del problema (diferentes a las utilizadas en los experimentos de calibración, para evitar sesgos en los resultados). Las instancias utilizadas tienen las siguientes características:

- 5 instancias de tamaño pequeño. Se considera que una instancia es de tamaño pequeño si contiene entre 10 y 60 puntos a ser recogidos.
  En particular se utilizaron las siguientes instancias pequeñas:
  - 1 instancia de 40 puntos y 6 vehículos - #1
  - 1 instancia de 45 puntos y 8 vehículos - #2
  - 1 instancia de 50 puntos y 8 vehículos - #3
  - 1 instancia de de 55 puntos y 9 vehículos - #4
  - 1 instancia de de 60 puntos y 10 vehículos - #5
- 10 instancias de tamaño medio. Se considera que una instancia es de tamaño medio si contiene entre 61 y 110 puntos a ser recogidos. En particular se utilizaron las siguientes instancias medias:
  - 1 instancia de de 65 puntos y 11 vehículos - #6
  - 1 instancia de de 70 puntos y 12 vehículos - #7
  - 1 instancia de de 75 puntos y 13 vehículos - #8
  - 1 instancia de de 80 puntos y 14 vehículos - #9
  - 1 instancia de de 85 puntos y 14 vehículos - #10
  - 1 instancia de de 90 puntos y 15 vehículos - #11
  - 1 instancia de de 95 puntos y 16 vehículos - #12
  - 1 instancia de de 100 puntos y 17 vehículos - #13
  - 1 instancia de de 105 puntos y 18 vehículos - #14
  - 1 instancia de de 110 puntos y 18 vehículos - #15
- 5 instancias de tamaño grande. Se considera que una instancia es de tamaño grande si contiene mas de 110 puntos a ser recogidos. En particular se utilizaron las siguientes instancias grandes:
  - 1 instancia de de 115 puntos y 19 vehículos - #16
  - 1 instancia de de 120 puntos y 20 vehículos - #17

- 1 instancia de de 125 puntos y 21 vehículos - #18
- 1 instancia de de 130 puntos y 22 vehículos - #19
- 1 instancia de de 135 puntos y 23 vehículos - #20

Resultados numéricos

Resultados numéricos AE

La tabla de la figura 6.5 reporta los resultados del AE para las 20 instancias del problema estudiadas. Las ejecuciones fueron realizadas con la configuración paramétrica determinada en los experimentos de calibración.

Se realizaron 50 ejecuciones independientes del AE para cada una de las instancias del problema, utilizando hasta diez mil generaciones para cada ejecución. Cada ejecución puede llegar a la generación 10.000 si el algoritmo sigue encontrando buenas soluciones. La condición de parada, de forma análoga a la utilizada en la configuración paramétrica, es llegar a la generación 10.000, no encontrar un individuo mejor durante 1000 generaciones consecutivas o superar el tiempo de ejecución máximo establecido (para este problema se configuro el tiempo máximo de ejecución en 5400 segundos).

Para analizar las soluciones obtenidas por el AE se aplicó el test de Shapiro-Wilk (S-W)[54] para establecer si los valores de fitness obtenidos seguían o no una distribución normal y poder comparar apropiadamente las medias/medianas.

La tabla de la figura 6.5 reporta los siguientes valores:

- Instancia utilizada con el formato #Puntos-#vehículos
- Mejor valor de fitness alcanzado.
- Promedio de los valores de fitness alcanzados.
- Peor valor de fitness alcanzado.
- Mediana
- Peor tiempo de ejecución (en segundos).
- Tiempo de ejecución promedio (en segundos).
- Mejor tiempo de ejecución (en segundos).
- P-valor del test de Shapiro Wilk sobre los valores de costo.

Los resultados numéricos reportados demuestran que para un nivel de significación de $\alpha = 0{,}05$ en el test de Shapiro-Wilk para normalidad sobre los valores de fitness, 12/20 ejecuciones cumplen $p - valor < \alpha$, de donde es posible concluir que puede descartarse la hipótesis nula del test de Shapiro-Wilk, que propone que los valores de fitness obtenidos siguen una distribución normal.

Instancia	Mejor Fitness	Fitness AVG	Peor Fitness	Mediana	T (MAX)	T (AVG)	T (MIN)	p-valor S-W
40-6	0,08726320943	0,08693112276	0,08636303198	0,0871141954	9	3,44898	2	3,041E-02
45-8	1,512045179	1,36739078	1,203532078	1,374000638	24	10,673469	4	0,0061467
50-8	0,04828472615	0,04804671058	0,04781353349	0,04806590443	592	201,306122	46	0,0724961
55-9	0,602489051	0,5684669992	0,5378316425	0,5666863779	53	18,816327	7	0,0396585
60-10	0,08755500589	0,08677305978	0,08592500771	0,086716113	305	87,979592	16	0,1781590
65-11	0,08640370184	0,0854727385	0,08469181187	0,08555315978	393	193,673469	73	0,2649220
70-12	0,8470093582	0,7964169058	0,7334243709	0,7947846865	694	261,122449	57	0,0488479
75-13	0,04650750453	0,04625073127	0,04601542962	0,04625337787	558	304,020408	60	0,1352080
80-14	0,7888987431	0,715513647	0,6418490593	0,7130768961	4693	1439,836735	196	0,4056230
85-14	0,5826502487	0,5509407028	0,5183074215	0,5501678232	2167	1075,693878	488	0,4026930
90-15	0,7123398055	0,6600939562	0,60005997	0,6572145198	5411	2640,22449	485	0,0973286
95-16	0,5530742494	0,5232158407	0,4864714006	0,5216051187	1392	297,510204	108	0,0252856
100-17	0,555603214	0,5219538563	0,4812271832	0,5251775087	4226	1612,77551	429	0,1456240
105-18	0,08700831893	0,08419043898	0,07849956497	0,08438565764	5463	5361,367347	4196	0,0001541
110-18	0,078420628	0,0773723416	0,0762413386	0,07739827914	5401	2.790	666	0,0384775
115-19	0,07992989125	0,07847648757	0,07495529797	0,0787249772	5558	5270,979592	2611	0,0006071
120-20	0,08378848144	0,08261295837	0,08037789959	0,08275500823	5448	5299,428571	3348	0,0022687
125-21	0,07927813705	0,0780622544	0,07685963403	0,07803566684	5400	1259,285714	486	0,0434530
130-22	0,08356662908	0,07963366368	0,07666003923	0,0797219458	5559	5414,938776	5400	0,0192641
135-23	0,3227055843	0,2662855482	0,07031852965	0,2701043021	6389	5451,959184	5400	3,877E-02

Figura 6.5: Reporte Numérico AE

Resultados numéricos Greedy

La tabla de la figura 6.6 reporta los resultados del algoritmo Greedy (descrito en la sección 6.2.3) para las 20 instancias del problema estudiadas. Para el algoritmo Greedy se realizaron 50 ejecuciones independientes para cada una de las instancias del problema. Para analizar las soluciones obtenidas por el algoritmo Greedy se aplicó el test de Shapiro-Wilk (S-W) para establecer si los valores de costo obtenidos seguían o no una distribución normal.

La tabla de la figura 6.6 reporta los siguientes valores:

- Instancia utilizada con el formato #Puntos-#vehículos
- Mejor valor de fitness alcanzado.
- Promedio de los valores de fitness alcanzados.
- Peor valor de fitness alcanzado.
- Mediana.
- P-valor del test de Shapiro Wilk sobre los valores de costo encontrados por el algoritmo.

Los tiempos de ejecución para el algoritmo Greedy son despreciables, por lo que no se incluyeron en la tabla. Los resultados numéricos reportados demuestran que para un nivel de significación de $\alpha = 0{,}05$ en el test de Shapiro-Wilk para normalidad sobre los valores de fitness obtenidos por el Greedy, todas las ejecuciones cumplen $p - valor < \alpha$, de donde es posible concluir que puede descartarse la hipótesis nula del test de Shapiro-Wilk, que propone que los valores de costo obtenidos siguen una distribución normal, ya que como se menciono en la sección 6.2.3 el algoritmo presenta características que lo hacen no determinista al ser variable la asignación de la sección correspondiente a los vehículos y permite realizar este tipo de análisis.

Instancia	Mejor Fitness	Fitness AVG	Peor Fitness	Mediana	p-valor S-W
40-6	0,04571566544	0,03194230875	0,01938062901	0,03139469619	1,5962600E-02
45-8	0,5114074808	0,1202249997	0,03160041992	0,08400931005	2,5921500E-07
50-8	0,01942982616	0,01926230547	0,01911715758	0,01925501507	4,6189600E-02
55-9	0,08179547079	0,0527799935	0,04429615804	0,04484128946	2,4558600E-06
60-10	0,04458495582	0,02944928462	0,01593055641	0,03051296673	4,9506500E-06
65-11	0,02346618718	0,01774255786	0,01209995179	0,01604857367	4,3370200E-05
70-12	0,04488467211	0,02855163755	0,01901432872	0,03058554855	4,5718200E-06
75-13	0,04400277955	0,03176907717	0,02307422794	0,03026517471	2,6441600E-02
80-14	0,340381963	0,07544394982	0,03002593649	0,04311157406	5,7223100E-07
85-14	0,01882027946	0,01123692698	0,007504741222	0,01075104386	4,8731600E-04
90-15	0,07478356682	0,03403729263	0,01861653859	0,03011400941	5,9246700E-02
95-16	0,07108338796	0,05138586352	0,02239304423	0,04148142342	5,1926400E-02
100-17	0,2329823408	0,08686291563	0,02906729993	0,06852053872	2,0443700E-04
105-18	0,07276848403	0,03692903104	0,02258102406	0,02963904079	5,9532400E-03
110-18	0,04055097107	0,02164871288	0,0118879274	0,01838788638	5,9674900E-02
115-19	0,02930721802	0,02410353341	0,01824358239	0,0225258563	6,1003200E-02
120-20	0,06665521352	0,03001236514	0,01330149086	0,02809953944	1,5608000E-05
125-21	0,06657649812	0,04848059228	0,01823745432	0,03984974812	9,4036900E-02
130-22	0,02264364511	0,01410612099	0,01058364321	0,01347852347	7,9737900E-04
135-23	0,03841823299	0,0229266169	0,0132308122	0,02167113222	2,4978800E-06

Figura 6.6: Reporte Numérico Greedy

Comparación contra algoritmo Greedy - Reporte

En esta sección se reporta el análisis comparativo entre los resultados calculados por el AE y el algoritmo Greedy mencionado en 6.2.3 (el cual presenta, como se menciono, características no deterministas), para las 20 instancias del problema estudiadas.

La Tabla de la figura 6.7 reporta el análisis comparativo entre los resultados calculados por el AE y el algoritmo Greedy, para las 20 instancias de la evaluación experimental. Para analizar la significancia estadıstica de la comparación, se aplico el test U de Mann-Whitney [56] para establecer si los resultados obtenidos con el AE tienen una diferencia significativa con los calculados por el algoritmo Greedy. La columna p-valor MW en la tabla de la figura 6.7 reporta el p-valor del test de Mann-Whitney sobre los valores de fitness calculados por el AE y el algoritmo Greedy.

Las columna Mejora(MAX) reporta el porcentaje de mejora máximo del AE frente al Greedy en cuanto al mejor fitness del algoritmo evolutivo frente al peor fitness del algoritmo Greedy, es decir, $((MejorFitnessAE - PeorFitnessGreedy) * 100) / MejorFitnessAE$ La columna Mejora(AVG) reporta el porcentaje de mejora promedio del algoritmo evolutivo frente al Greedy en cuanto al fitness promedio de ambos algoritmos, es decir, $((AVGFitnessAE - AVGFitnessGreedy) * 100) / AVGFitnessAE$.

Las restantes columnas corresponden a las explicadas en las tablas de las figuras 6.5 y 6.6 respectivamente. Todos los tiempos reportados se miden en segundos.

El test U de Mann-Whitney aplicado sobre las distribuciones de resultados muestra que existe una diferencia significativa entre las soluciones obtenidas por el AE y

aquellas alcanzadas por el algoritmo Greedy.

Instancia	Algoritmo	Mejor Fitness	Fitness AVG	Peor Fitness	Mediana	T (MAX)	T (AVG)	T (MIN)	Mejora(% MAX)	Mejora(% AVG)	p-valor M-W
40-6	AE	0,08726320943	0,08693112276	0,08636303198	0,0871141954	9	3,44898	2	77,79	63,26	4.44089e-16
	AGreedy	0,04571566544	0,03194230875	0,01938062901	0,03139469619	0	0	0	-	-	
45-8	AE	1,512045179	1,36739078	1,203532078	1,374000638	24	10,673469	4	97,91008758	91,2077073	4.44089e-16
	AGreedy	0,5114074808	0,1202249997	0,03160041992	0,08400931005	0	0	0	-	-	
50-8	AE	0,04828472615	0,04804671058	0,04781353349	0,04806590443	592	201,306122	46	60,40744329	59,90921077	4.44089e-16
	AGreedy	0,01942982616	0,01926230547	0,01911715758	0,01925501507	0	0	0	-	-	
55-9	AE	0,602489051	0,5684669992	0,5378316425	0,5666863779	53	18,816327	7	92,64780697	90,71538127	2.22045e-16
	AGreedy	0,08179547079	0,0527799935	0,04429615804	0,04484128946	0	0	0	-	-	
60-10	AE	0,08755500589	0,08677305978	0,08592500771	0,086716113	305	87,979592	16	81,80508784	66,06171928	2.22045e-16
	AGreedy	0,04458495582	0,02944928462	0,01593055641	0,03051296673	0	0	0	-	-	
65-11	AE	0,08640370184	0,0854727385	0,08469181187	0,08555315978	393	193,673469	73	85,99602618	79,24185164	2.22045e-16
	AGreedy	0,02346618718	0,01774255786	0,01209995179	0,01604857367	0	0	0	-	-	
70-12	AE	0,8470093582	0,7964169058	0,7334243709	0,7947846865	694	261,122449	57	97,75512177	96,41498851	2.22045e-16
	AGreedy	0,04488467211	0,02855163755	0,01901432872	0,03058554855	0	0	0	-	-	
75-13	AE	0,04650750453	0,04625073127	0,04601542962	0,04625337787	558	304,020408	60	50,38601153	31,31118949	2.22045e-16
	AGreedy	0,04400277955	0,03176907717	0,02307422794	0,03026517471	0	0	0	-	-	
80-14	AE	0,7888987431	0,715513647	0,6418490593	0,7130768961	4693	1439,836735	196	96,19394292	89,45597332	2.22045e-16
	AGreedy	0,340381963	0,07544394982	0,03002593649	0,04311157406	0	0	0	-	-	
85-14	AE	0,5826502487	0,5509407028	0,5183074215	0,5501678232	2167	1075,693878	488	98,71196464	97,96041082	2.22045e-16,
	AGreedy	0,01882027946	0,01123692698	0,007504741222	0,01075104386	0	0	0	-	-	
90-15	AE	0,7123398055	0,6600939562	0,60005997	0,6572145198	5411	2640,22449	485	97,38656489	94,84356851	2.22045e-16,
	AGreedy	0,07478356682	0,03403729263	0,01861653859	0,03011400941	0	0	0	-	-	
95-16	AE	0,5530742494	0,5232158407	0,4864714006	0,5216051187	1392	297,510204	108	95,95116854	90,17884025	2.22045e-16
	AGreedy	0,07108338796	0,05138586352	0,02239304423	0,04148142342	0	0	0	-	-	
100-17	AE	0,555603214	0,5219538563	0,4812271832	0,5251775087	4226	1612,77551	429	94,76833481	83,3581236	2.22045e-16
	AGreedy	0,2329823408	0,08686291563	0,02906729993	0,06852053872	0	0	0	-	-	
105-18	AE	0,08700831893	0,08419043898	0,07849956497	0,08438565764	5463	5361,367347	4196	74,04728153	56,13631252	2.22045e-16
	AGreedy	0,07276848403	0,03692903104	0,02258102406	0,02963904079	0	0	0	-	-	
110-18	AE	0,078420628	0,0773723416	0,0762413386	0,07739827914	5401	2.790	666	84,84081587	72,02008827	2.22045e-16
	AGreedy	0,04055097107	0,02164871288	0,0118879274	0,01838788638	0	0	0	-	-	
115-19	AE	0,07992989125	0,07847648757	0,07495529797	0,0787249772	5558	5270,979592	2611	77,17551956	69,28566229	2.22045e-16
	AGreedy	0,02930721802	0,02410353341	0,01824358239	0,0225258563	0	0	0	-	-	
120-20	AE	0,08378848144	0,08261295837	0,08037789959	0,08275500823	5448	5299,428571	3348	84,12491714	63,67111682	2.22045e-16
	AGreedy	0,06665521352	0,03001236514	0,01330149086	0,02809953944	0	0	0	-	-	
125-21	AE	0,07927813705	0,0780622544	0,07685963403	0,07803566684	5400	1259,285714	486	76,99560686	37,89496261	2.22045e-16
	AGreedy	0,06657649812	0,04848059228	0,01823745432	0,03984974812	0	0	0	-	-	
130-22	AE	0,08356662908	0,07963366368	0,07666003923	0,0797219458	5559	5414,938776	5400	87,3350842	82,28623381	2.22045e-16
	AGreedy	0,02264364511	0,01410612099	0,01058364321	0,01347852347	0	0	0	-	-	
135-23	AE	0,3227055843	0,2662855482	0,07031852965	0,2701043021	6389	5451,959184	5400	95,90003618	91,3902136	2.22045e-16
	AGreedy	0,03841823299	0,0229266169	0,0132308122	0,02167113222	0	0	0	-	-	

Figura 6.7: Comparación de fitness AE y Greedy

La siguientes figuras muestran de forma gráfica la comparación del mejor fitness (MAX), fitness promedio (AVG), peor fitness (MIN) y mediana (MEDIANA) obtenidos entre el AE y el algoritmo Greedy para las 20 instancias utilizadas.

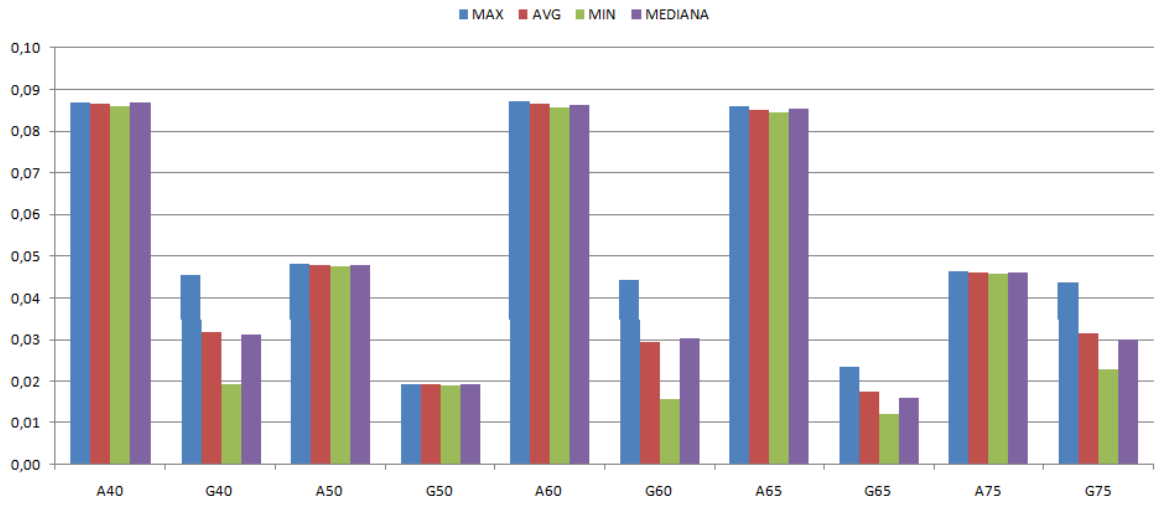

Figura 6.8: Gráfica 1. Comparación de fitness AE y Greedy

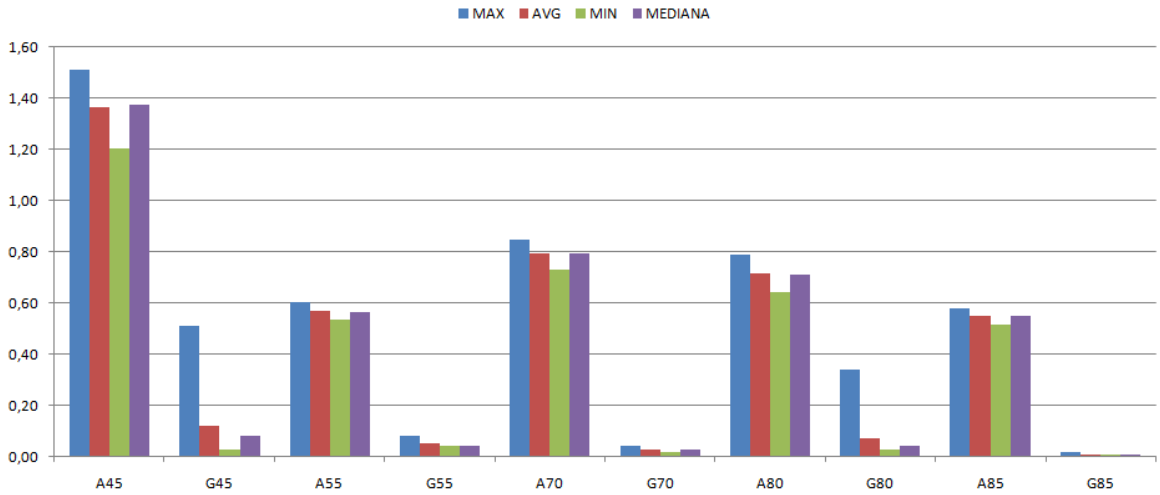

Figura 6.9: Gráfica 2. Comparación de fitness AE y Greedy

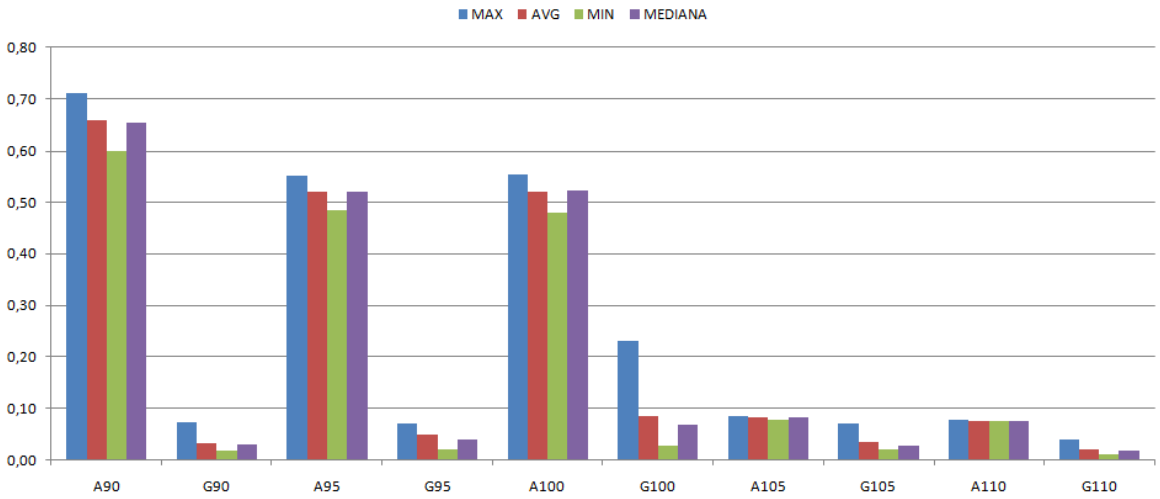

Figura 6.10: Gráfica 3. Comparación de fitness AE y Greedy

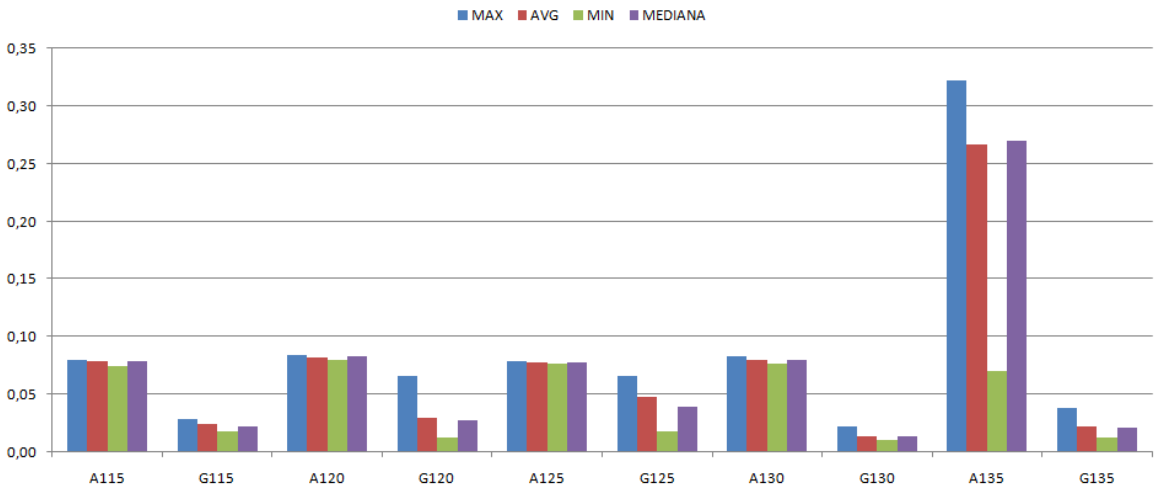

Figura 6.11: Gráfica 4. Comparación de fitness AE y Greedy

En términos generales se tiene una mejora promedio (promedio de mejora promedio) de un 75,33 % frente al algoritmo greedy. Se podría esperar que para instancias más pequeñas o medias $(<= 110 puntos)$ el algoritmo evolutivo tendría un mejor desempeño que el algoritmo greedy frente a instancias grandes (dada la complejidad computacional). Sin embargo, vemos que depende de cada instancia, es decir, varios factores determinan que tan bueno es el AE frente al Greedy. La cantidad de puntos, la ubicación de cada punto así como la cantidad de vehículos disponibles y restricciones como la ventana de tiempo o puntos fijos influyen en la eficiencia del AE. Por eso se observa que el AE puede tener un mejor margen de mejora incluso si se compara con instancias mas grandes, por ejemplo, si se analiza la instancia 75-13 el AE obtuvo una mejora promedio de 31,3 % con respecto al Greedy. Sin embargo, para la instancia 135-23 el AE obtuvo una mejora promedio de 91,39 % frente al Greedy.

Por otra parte es posible observar, como se muestra en las siguientes figuras, que a mayor tamaño de instancia mayor es el tiempo de ejecución del algoritmo evolutivo, en términos generales, llegando a tiempos de 5400 segundos. Esto es debido, en parte, a que al aumentar la cantidad de puntos y vehículos las posibles combinaciones de rutes aumentan de forma exponencial.

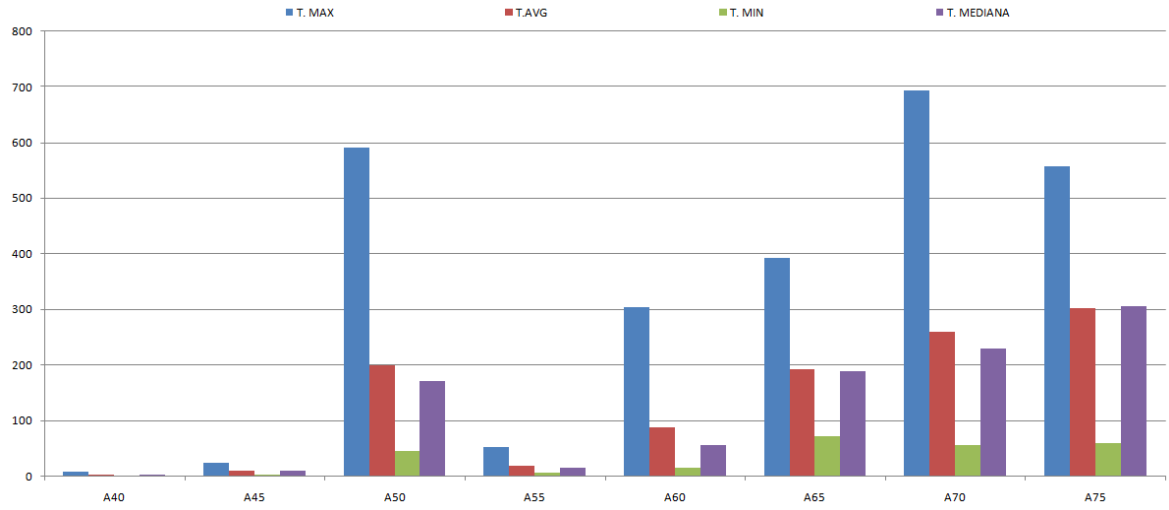

Figura 6.12: Gráfica 1. Tiempos de ejecución del AE

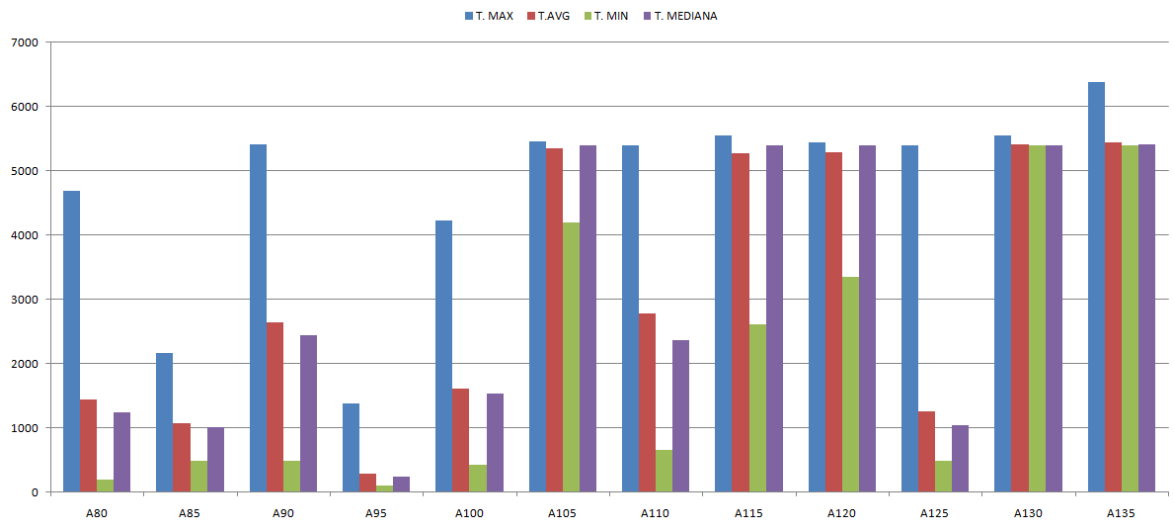

Figura 6.13: Gráfica 2. Tiempos de ejecución del AE

Se espera que un algoritmo evolutivo de el mejor resultado en el menor tiempo posible y es posible que para diversos escenarios estos tiempos de respuesta no sean considerados aceptables.

Como se menciono anteriormente el algoritmo cuenta con tres condiciones de parada:

- Llegar a 10.000 mil generaciones
- No encontrar una mejor solución durante 1000 generaciones consecutivas
- Superar el tiempo máximo de ejecución establecido (5400 por defecto)

Para intentar mejorar los tiempos del algoritmo evolutivo y evitar perder la calidad en las soluciones obtenidas se realizó una prueba para comprobar si el algoritmo evolutivo implementado encuentra para una cierta instancia, en un tiempo

razonable X, una solución con una calidad similar a la que devuelve para una ejecución en la que tardó mas de X segundos.

Se utilizó como tiempo de ejecución X el valor 300. Por lo tanto se modificó la tercera condición de parada, modificando el tiempo máximo de ejecución de 5400 segundos a 300 segundos.

Para comprobar esto se realizaron 50 ejecuciones independientes del AE para cada una de las instancias del problema utilizadas en la evaluación experimental donde el algoritmo evolutivo tuvo un tiempo de ejecución promedio mayor a 300 segundos. Se utilizaron las mismas condiciones que para la evaluación experimental original, bajando únicamente la condición de parada del tiempo máximo de ejecución A 300 segundos. Se utilizo el mismo hardware para evitar sesgos en los resultados.

La tabla de la figura 6.14 reporta los siguientes valores:

- Instancia evaluada.
- Algoritmo, que indica si se trata de la ejecución con tiempos $>$ 300 segundos (AE) o la nueva ejecución hasta 300 segundos (AC).
- Mejor valor de fitness alcanzado.
- Promedio de los valores de fitness alcanzados.
- Peor valor de fitness alcanzado.
- Mediana.
- Peor tiempo de ejecución (en segundos).
- Tiempo de ejecución promedio (en segundos).
- Mejor tiempo de ejecución (en segundos).
- Porcentaje de mejora promedio

La columna Mejora(AVG) reportan el porcentaje de mejora del AE cuyas instancias ejecutaron por más de 300 segundos frente al AE cuyas instancias ejecutaron hasta 300 segundos en cuanto al mejor fitness y fitness promedio. Las restantes columnas corresponden a las ya explicadas anteriormente. Todos los tiempos reportados se miden en segundos.

Instancia	Algoritmo	Mejor Fitness	Fitness AVG	Peor Fitness	Mediana	T (MAX)	T (AVG)	T (MIN)	Mejora(% AVG)
	AE	0,7888987431	0,715513647	0,6418490593	0,7130768961	4693	1439,836735	196,0000000	0,1633178546
80-14	AC	0,8137044759	0,7143450854	0,6218169965	0,7120911511	300	254,5	144,0000000	-
	AE	0,5826502487	0,5509407028	0,5183074215	0,5501678232	2167	1075,693878	488,0000000	2,313875697
85-14	AC	0,5727891912	0,5381926198	0,503537857	0,5345071681	300	300	300,0000000	-
	AE	0,7123398055	0,6600939562	0,60005997	0,6572145198	5411	2640,22449	485,0000000	10,54780805
90-15	AC	0,6810345432	0,5904685127	0,08731368558	0,6079248722	300	300	300,0000000	-
	AE	0,5530742494	0,5232158407	0,4864714006	0,5216051187	1.392	298	108,0000000	0,6908172513
95-16	AC	0,5609630209	0,5196013754	0,4693049678	0,522445131	300	108	54,0000000	-
	AE	0,555603214	0,5219538563	0,4812271832	0,5251775087	4226	1612,77551	429,0000000	3,966928107
100-17	AC	0,5450669236	0,5012483221	0,4359050452	0,503299759	300	300	300,0000000	-
	AE	0,08700831893	0,08419043898	0,07849956497	0,08438565764	5463	5361,367347	4.196,0000000	6,368045366
105-18	AC	0,08256409058	0,07882915363	0,07506290666	0,07877291709	300	300	3,000E+02	-
	AE	0,078420628	0,0773723416	0,0762413386	0,07739827914	5401	2.790	666	-0,07799322941
110-18	AC	0,07823185677	0,07743268679	0,07653181274	0,07749385313	300	300	300	-
	AE	0,07992989125	0,07847648757	0,07495529797	0,0787249772	5558	5270,979592	2611	2,868604943
115-19	AC	0,07854979915	0,07622530716	0,07460120286	0,07607854827	300	300	300	-
	AE	0,08378848144	0,08261295837	0,08037789959	0,08275500823	5448	5299,428571	3348	7,297617931
120-20	AC	0,08113074582	0,07658418031	0,04359972363	0,07862676407	300	300	300	-
	AE	0,07927813705	0,0780622544	0,07685963403	0,07803566684	5400	1259,285714	486	0,05032960173
125-21	AC	0,07943957974	0,07802296598	0,07657594169	0,07805701661	300	300	300	-
	AE	0,08356662908	0,07963366368	0,07666003923	0,0797219458	5559	5414,938776	5400	33,56320408
130-22	AC	0,07733169642	0,05290605463	0,02978039475	0,04297190332	300	300	300	-
	AE	0,3227055843	0,2662855482	0,07031852965	0,2701043021	6389	5451,959184	5400	13,93264172
135-23	AC	0,2777810186	0,2291849368	0,04146471674	0,2480928359	300	300	300	-

Figura 6.14: Gráfica 4. Comparativa de instancias

La siguientes figuras muestran de forma gráfica la comparación del mejor fitness (MAX), fitness promedio (AVG), peor fitness (MIN) y mediana (MEDIANA) obtenidos entre el AE cuyas instancias tuvieron un tiempo de ejecución mayor a 300 segundos y el AE cuyas instancias tuvieron un tiempo de ejecución hasta 300 segundos.

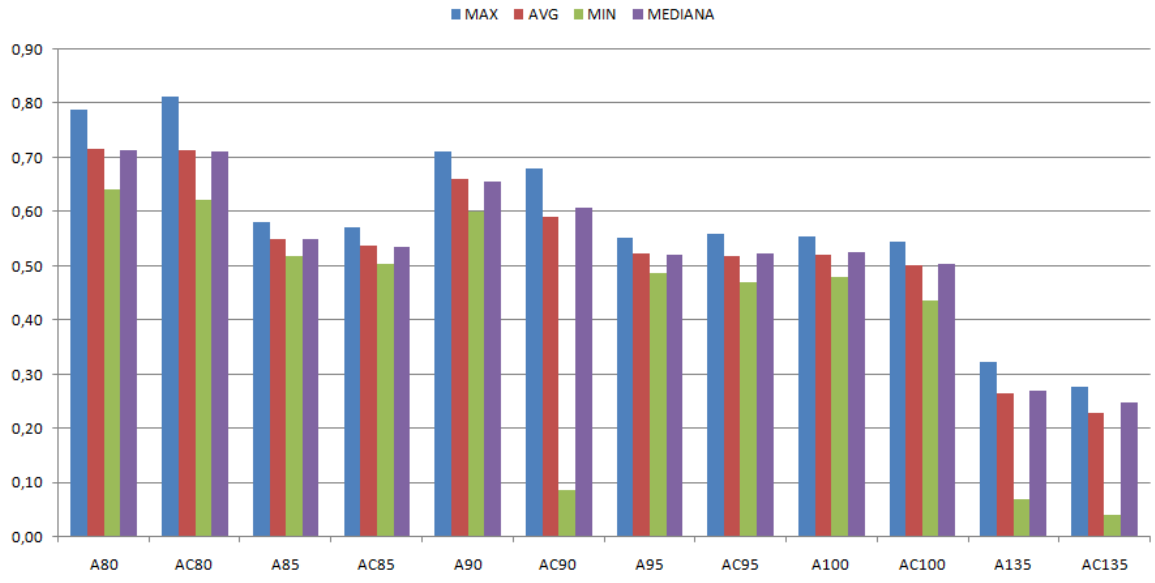

Figura 6.15: Gráfica 5. Comparativa de instancias 1

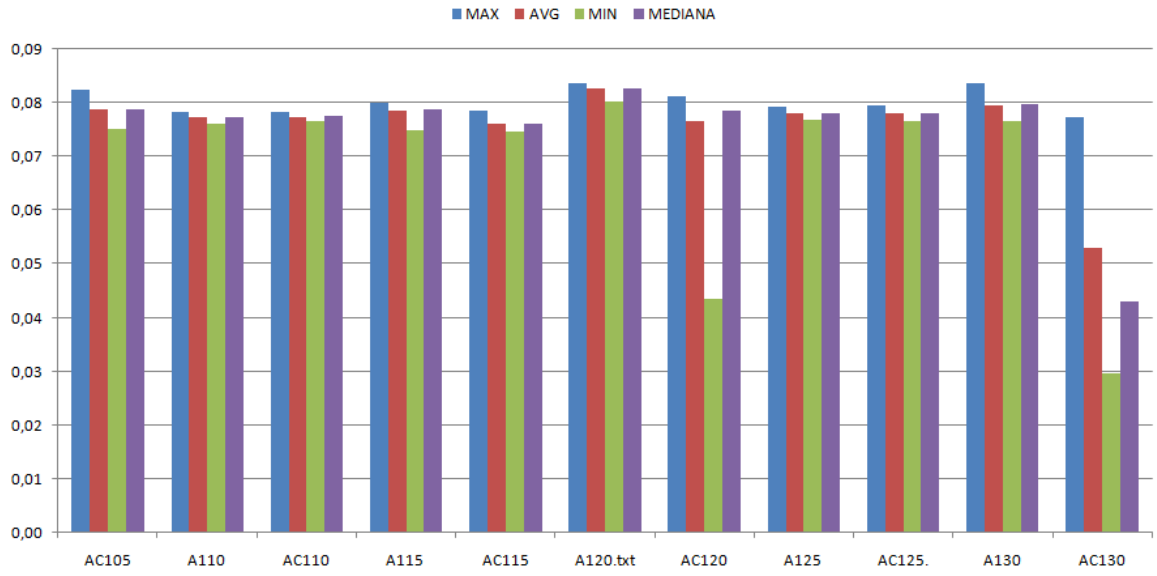

Figura 6.16: Gráfica 6. Comparativa de instancias 2

Observando los resultados se observa que para 10/12 de las instancias la mejora promedio del algoritmo cuyo tiempo de ejecución fue mayor a 300 segundos no supera el 10 % con respecto al algoritmo cuyo tiempo de ejecución fue hasta 300 segundos.

Solamente en dos de las instancias esta mejora fue mayor al 10 %. En particular para la instancia 130-22 la mejora fue de un 33,56 % y para la instancia 135-23 la mejora fue de un 13,93 % respectivamente.

Ambas instancias tuvieron un tiempo promedio de ejecución de 5400 segundos en el primer caso y un tiempo promedio de 300 segundos en el segundo.

Por lo tanto se puede ver que en general dejar el AE ejecutando por un tiempo elevado no garantiza un aumento sustancial en la calidad de las soluciones para las instancias utilizadas. El algoritmo encuentra mucho antes una solución la cual no puede mejorar.

Capítulo 7

Api para el manejo efectivo del Algoritmo Evolutivo

7.1. Descripción general

En esta sección se describe la api que oficia de servidor para atender las solicitudes.

Se implementó un servidor capaz de manejar diferentes solicitudes de el/los potenciales clientes que quieran utilizar de manera concurrente y organizada el algoritmo evolutivo. Se espera que cada sucursal de correos pueda planificar su recorrido utilizando para ello el algoritmo implementado. Por esta razón surge la necesidad de desarrollar un sistema que se encargue de gestionar los accesos al algoritmo de forma tal que dos clientes no puedan cargar puntos y vehículos a la vez, sino que cada cliente pueda encolar una petición con puntos y vehículos y una vez que el algoritmo pueda atenderla le retorne el resultado obtenido. De esta forma se garantiza que cuando un cliente solicite el algoritmo evolutivo no quede a la espera de ser atendido, si no que pueda enviar su petición y obtener una respuesta de forma asincrónica cuando el mismo lo atienda. De la misma manera puede ocurrir que un cliente envíe una petición de puntos y vehículos para calcular una ruta para su zona y posteriormente actualice esa lista que envió previamente agregándole más puntos. Para estos casos el servidor gestiona una sesión con el cliente persistiendo los puntos del mismo de forma tal que si en un futuro el cliente solicita calcular más puntos, se actualiza su estado y se vuelve a calcular una ruta.

Todas las comunicaciones con el servidor son a través de métodos REST. Existen varios tipos de solicitudes a realizar al servidor. Primeramente el cliente que quiera hacer uso del servidor deberá 'presentarse' ante el servidor, para ello consume un web service **solicitando un número de sesión e indicando en qué url se quiere que el servidor entregue la solución una vez finalice el algoritmo.** Por su parte, el servidor, persiste en base los datos del cliente y devuelve un identificador único para el cliente el cual se usa para identificar las peticiones futuras del mismo.

Una vez el cliente tenga la sesión podrá realizar más peticiones al servidor, como **agregar puntos y vehículos** a su espacio de solución, esto es un conjunto o subconjunto de los mismos que tenga reflejados en su realidad. El servidor con estos datos, va calculando la matriz de distancia-tiempo, esto es, las distancias y tiempos que hay de cada punto a otro y de cada origen y destino de los vehículos a todos los

puntos. Esta información queda almacenada en base de datos. La idea en este paso es que a la hora de ejecutar el algoritmo todos estos cálculos ya se encuentren realizados ahorrando tiempo. Un cliente puede realizar tantas veces como quiera esta petición. El servidor, por cada petición, hará los cálculos de los datos enviados hasta el momento por el cliente, es decir los que se envían en la petición actual y todos los datos que haya enviado el cliente hasta el momento.

Una vez que el cliente haya enviado sus puntos y vehículos en la/las peticiones anteriores puede realizar la petición de **calcular su solución**. En este punto, el servidor se asegura que todas sus solicitudes anteriores se hayan procesado y procede a la ejecución del algoritmo. Dicha ejecución se efectúa como se explica en secciones anteriores. Cuando finaliza la ejecución de este, la solución es encolada para ser enviada a través del web service que el cliente especificó al solicitar sesión.

A continuación, en la figura 7.1, se puede observar el funcionamiento del mismo explicado anteriormente.

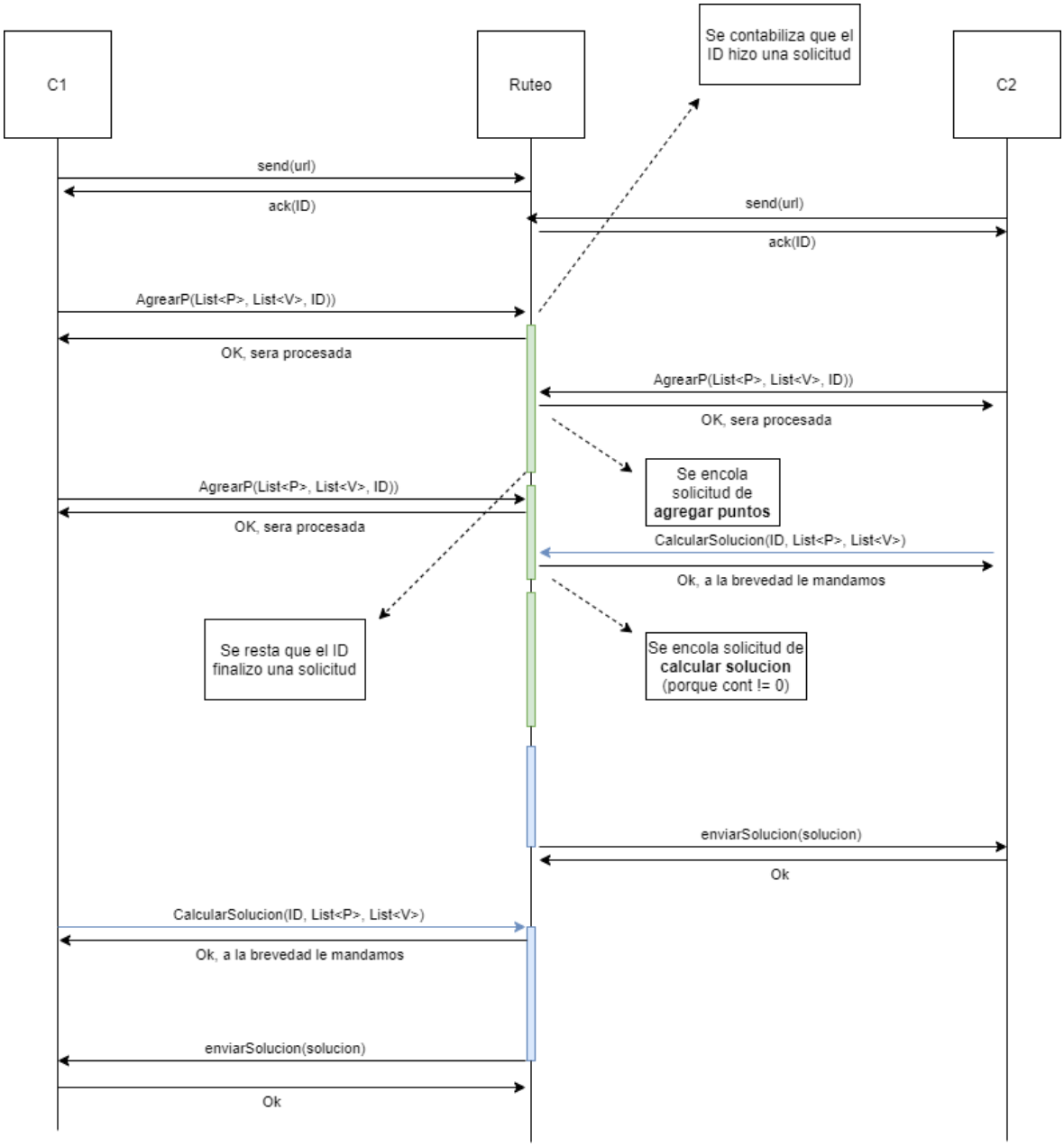

Figura 7.1: Ejemplo de funcionamiento de la API con 2 clientes.

7.2. Arquitectura del sistema y tecnologías utilizadas

El servidor fue desarrollado en Java. Para manejar la concurrencia de las diferentes solicitudes se utilizaron varias colas de mensajes JMS [57] [58]. Una cola se encarga de atender las solicitudes de 'agregar puntos y vehículos' que realizan los clientes. Se decidió atender una solicitud a la vez de manera de que el tiempo de respuesta individual no dependiera de otras solicitudes. También se lleva un control de las peticiones realizadas por cada cliente para saber en determinado momento si se procesaron todas las peticiones realizadas por un cliente específico.

Por otro lado se tiene otra cola JMS que atiende las solicitudes de 'calcular solución'. Esta petición no se procesa hasta que no se hayan procesado todas las peticiones anteriores del cliente solicitante. Cuando se cumplan las condiciones de ejecutar esta petición, se procede a la ejecución del algoritmo evolutivo.

Por ultimo, se tiene una cola JMS que maneja el envió de las soluciones hacia los clientes. Esto es, cuando finaliza la ejecución del algoritmo evolutivo, se inserta la solución en dicha cola para que posteriormente se procese el envió, que como se mencionó anteriormente, se realiza mediante un servicio REST que provee el cliente en una primera etapa. Se decidió implementar esto con una cola JMS ya que el envió de la solución puede fallar (el host de destino puede no estar disponible, etc.). A su vez se implementó una política de reintento para los casos donde falle el envió. La misma consiste en esperar un tiempo dinámico (empezando por un tiempo pequeño que crece a medida que siga fallando) y encolar nuevamente para ser procesada. Además, se tiene un tope de cantidad de reintentos.

La idea del uso de estas colas JMS es conseguir la mayor concurrencia posible de manera de que el servidor tenga un buen desempeño.

Por otra parte, para la base de datos se utilizó Postgresql [59].

7.3. Ejemplo de consumo de las api

A continuación se describe cómo es el uso en el caso típico.

Como se ha mencionado anteriormente, el primer paso a realizar es consumir el web service GET del servidor para solicitar una sesión. Donde también se indica en qué URL se quiere recibir la solución.

```
http://osrm-prod:8080/rest/operaciones/sesion?url=
http://app123:8580/app/rest/recibirSolucion
```

Luego de este paso, el cliente está habilitado a realizar las otras peticiones. El siguiente paso es enviar datos de los puntos y vehículos al servidor, para esto es necesario consumir un servicio web POST:

```
http://osrm-prod:8080/rest/operaciones/agrergarDatos
```

En el body de dicho servicio debe ir un JSON el cual identifique al cliente y contenga una lista de puntos y otra lista con vehículos con sus datos correspondientes. Este servicio puede consumirse tantas veces como el cliente necesite.

Una vez el cliente ya envió todos los datos, lo siguiente es solicitar correr el algoritmo, para eso se consume otro web service POST:

`http://osrm-prod:8080/rest/operaciones/calcularSolucion`

En este caso, como en el anterior, se debe enviar un JSON que contenga la sesión del cliente y una lista de puntos y vehículos. Estas listas deben ser un subconjunto de los puntos y vehículos ya enviados anteriormente por este cliente. Este subconjunto son los datos que el cliente quiere que el algoritmo trabaje.

Por ultimo, una vez termine la ejecución del algoritmo, el servidor le enviará la solución a la url que el cliente especificó inicialmente, en este caso de ejemplo es:

`http://app123:8580/app/rest/recibirSolucion`

Capítulo 8

Conclusiones

Esta sección presenta las conclusiones del trabajo realizado en el marco del proyecto y las principales líneas de trabajo futuro.

8.1. Conclusiones

En este documento se presenta una revisión actual, hasta donde se pudo investigar, de la literatura sobre el problema atacado. Dada la gran cantidad de material bibliográfico presente se decidió escoger una muestra representativa de la bibliografía utilizada.

La mayoría de artículos, informes, tesis y material bibliográfico leído y analizado se hace referencia a la complejidad que este tipo de problema posee, Np-hard. La mayoría de artículos analizados no se ponen de acuerdo en cual es la mejor forma para resolver este tipo de problemas. Por lo que se opto por desarrollar una forma propia tomando los conocimientos obtenidos en los diversos artículos.

Se destaca dentro del estudio bibliográfico realizado, que para problemas de este estilo grandes, las soluciones exactas no son viables tanto por recursos informáticos como en los tiempos de resolución. Incluso en investigaciones con resoluciones exactas [60] se proponen modificaciones a dichas resoluciones para llevarlas a heurísticas y así resolver problemas concretos de gran escala.

Si bien el problema de ruteo con ventanas de tiempo es una variante del problema clásico VRP, este sigue siendo un problema muy amplio y complejo de resolver existiendo multitud de variantes donde cada una se enfoca en diferentes aspectos de la realidad. Por la complejidad de estos problemas resulta complejo abarcar todas las posibilidades y generalizarlas a un único modelo.

Este proyecto ha presentado el diseño e implementación de un algoritmo evolutivo para la resolución del problema de recoger una cantidad variable de puntos desde múltiples orígenes hacia múltiples destinos con diversas restricciones como ventanas de tiempo, vehículos permitidos por los puntos, etc. Esto empleando un numero variable de vehículos con diversas características como la capacidad, puntos fijos, etc. El algoritmo evolutivo propuesto conforma una eficiente solución para el problema abordado. El análisis experimental realizado usando un conjunto de instancias del problema construidas en base a coordenadas generadas de forma aleatoria pero reales, es decir, pertenecientes a puntos válidos de Montevideo, mostró que los resultados del algoritmo evolutivo permiten mejorar significativa-

mente los resultados sobre los encontrados empleando un algoritmo Greedy que simula el comportamiento más natural de una empresa u entidad que se enfrente a un problema de este estilo. Concretamente se obtuvo una mejora promedio (promedio de mejora promedio de los resultados) de un 75,33 % frente al algoritmo greedy utilizado. En una primera instancia los resultados mostraron que a medida que la cantidad de puntos aumenta el tiempo de ejecución se eleva considerablemente, pudiendo alcanzar los 5400 segundos de tiempo máximo establecido como tiempo de parada para el algoritmo. Se realizó una prueba la cual comparaba las instancias las cuales tardaron más de 300 segundos en finalizar frente a las mismas instancias pero ejecutando hasta 300 segundos como tiempo máximo para evaluar si con el tiempo de ejecución extra se obtenía una mejora sustancial en la soluciones. Los resultados indicaron que no se llega a perder calidad en las soluciones ejecutando el algoritmo hasta 300 segundos. De las 12 instancias evaluadas diez tuvieron una mejora de soluciones promedio inferior al 10 % frente al algoritmo que ejecutó las mismas instancias pero con un tiempo máximo de 300 segundos. Cabe destacar que estas pruebas fueron realizadas con los mismos recursos computacionales y en igualdad de condiciones para evitar sesgos en la comparación.

Por lo tanto se puede ver que el algoritmo encuentra en 300 segundos o menos una solución considerablemente superior a la que encuentra el algoritmo Greedy y mantiene la calidad de las soluciones que encontraría si se ejecutara el algoritmo por más de 300 segundos.

Por otra parte se desarrolló la arquitectura e implementación de un sistema que permite utilizar el algoritmo evolutivo de forma óptima. Estos servicios fueron implementados en JAVA y se basan en REST. El sistema implementado permite gestionar las diferentes solicitudes para calcular rutas provenientes de potenciales clientes. Cada cliente puede obtener una sesion con la que podrá calcular sus rutas de forma independiente y asincrónica, obteniendo el resultado del algoritmo cuando el mismo finalice, sin esperas en el lado del cliente.

8.2. Trabajo a futuro

Como trabajo a futuro se busca mejorar el desempeño del algoritmo evolutivo encontrando una mejor función de evaluación, mejorando los operadores de corrección o probar diferentes enfoques como pueden ser penalizar soluciones o descartarlas. De igual forma se busca analizar estadísticamente el desempeño del mismo utilizando instancias realistas de algún organismo que comience a utilizar este algoritmo, actualmente la Administración Nacional de Correos. El proyecto puede ser de gran utilidad, sin embargo, a todo aquel organismo que realice un trabajo de logística y necesite mejorar sus servicios o comenzar a utilizar un servicio de calculo de rutas. Por lo tanto sería idóneo extrapolar el proyecto a uno más general que se adapte a las necesidades de cada organismo. Otro punto a enfocarse como trabajo a futuro es el desarrollo de una aplicación móvil y/o página web que facilite el ingreso de puntos y vehículos para que el AE ejecute. Dicha aplicación y/o página web debe contener un mapa en el cual se puede agregar puntos de recogida en el mismo, las restricciones correspondientes a cada punto y asignar los orígenes y destinos para la ejecución. De la misma manera poder asignar los vehículos y sus

restricciones asociadas. De esta forma se podría utilizar más fácilmente los servicios REST presentados en el capítulo 7 y mejorar la experiencia de los usuarios. Por último, es de interés estudiar la aplicabilidad de los algoritmos desarrollados en este proyecto a escenarios distintos. Interesa a su vez estudiar variantes del problema, como pueden ser la carga y descarga en los puntos y no solo la carga como aborda el problema estudiado. Muchos de los problemas atacados en este proyecto pueden facilitar el desarrollo de algoritmos para estas u otras variantes del problema.

Bibliografía

[1] Suresh Nanda Kumar, Ramasamy Panneerselvam. *A Survey on the Vehicle Routing Problem and Its Variants*. [Vehicle Routing Problem.] 2012. URL: https://www.scirp.org/pdf/IIM20120300003_68227741.pdf.

[2] *Francis, P., Smilowitz, K., y Tzur, M. (2008). The period vehicle routing problem and its extensions. Operations Research/ Computer Science Interfaces Series, 43 , 73-102.*

[3] Weisstein, Eric W. *MathWorld*. [NP-Hard Problem.] 2013. URL: http://mathworld.wolfram.com/NP-HardProblem.html.

[4] Jun Jiang y col. "Vehicle routing problem with a heterogeneous fleet and time windows". En: *Expert Systems with Applications* 41.8 (2014), págs. 3748-3760.

[5] *Francis,Tummel, C., Franzen, C., Hauck, E., y Jeschke, S. (2013). The multi-depot heterogeneous fleet vehicle routing problem with time windows and 91 assignment restrictions (mvrptwar).*

[6] Paulo Henrique Ribeiro Gabriel y Alexandre Cláudio Botazzo Delbem. *Fundamentos de algoritmos evolutivos*. 2008.

[7] *ANC Administración Nacional de Correos*. URL: https://www.correo.com.uy/web/guest/nosotros.

[8] Barreto Delgado, Ibeth Zaneynne y col. "Diseño de un modelo de ruteo de vehículos-VRP para la distribución de llantas aplicando programación dinámica". En: ().

[9] Carlos A Coello Coello y Araceli Yánez López. "El algoritmo genético como alternativa a la programación dinámica". En: *Actas del VIII Simposio Internacional en Aplicaciones de Informática*. 1994, págs. 151-157.

[10] David de Frutos Escrig. "Alan Turing: Una aproximación personal a su obra". En: *La Gaceta de la Real Sociedad Matemática Española* 15.4 (2012), págs. 675-696.

[11] Cristiano Castelfranchi. "Alan Turing's "Computing machinery and intelligence"". En: *Topoi* 32.2 (2013), págs. 293-299.

[12] Joanos Neumann, Arthur W Burks y col. *Theory of self-reproducing automata*. Vol. 1102024. University of Illinois press Urbana, 1966.

[13] David B Fogel. "Nils Barricelli-artificial life, coevolution, self-adaptation". En: *IEEE Computational Intelligence Magazine* 1.1 (2006), págs. 41-45.

[14] Woodrow Wilson Bledsoe e Iben Browning. "Pattern recognition and reading by machine". En: *Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference*. 1959, págs. 225-232.

[15] Hans J Bremermann y col. "Optimization through evolution and recombination". En: *Self-organizing systems* 93 (1962), pág. 106.

[16] Allen Newell, John C Shaw y Herbert A Simon. "Report on a general problem solving program". En: *IFIP congress*. Vol. 256. Pittsburgh, PA. 1959, pág. 64.

[17] Ingo Rechenberg. "Evolutionsstrategien". En: *Simulationsmethoden in der Medizin und Biologie*. Springer, 1978, págs. 83-114.

[18] Holland John. "Holland. genetic algorithms". En: *Scientific american* 267.1 (1992), págs. 44-50.

[19] Mason Anders. *DNA, Genes, and Chromosomes*. CAPSTONE PR, 2017. ISBN: 978-1515772606.

[20] Darrell Whitley. "A genetic algorithm tutorial". En: *Statistics and computing* 4.2 (1994), págs. 65-85.

[21] Sergio Nesmachnow. *Algoritmos Evolutivos*. URL: `https://eva.fing.edu.uy/course/view.php?id=1049`.

[22] MANUEL; VICIANO DELIBANO. *Cromosoma 8*. Plaza y Janes; 001 edición, 2008. ISBN: 978-8401336935.

[23] *G. Kendall and G. Whitwell. An evolutionary approach for the tuning of a chess evaluation function using population dynamics. In Proceedings of the 2001 Congress on Evolutionary Computation, pages 995– 1002. IEEE Press, World Trade Center, Seoul, Korea, 2001.*

[24] *Shah, S.M.; Thaker, C. S.; Singh, D.; Multimedia based fitness function optimization through evolutionary game learning, Emerging Trends in Networks and Computer Communications (ETNCC), 2011 International Conference on Publication Year: 2011, page(s): 164- 168.*

[25] Chetan Chudasama, SM Shah y Mahesh Panchal. "Comparison of parents selection methods of genetic algorithm for TSP". En: *International Conference on Computer Communication and Networks CSI-COMNET-2011, Proceedings*. 2011, págs. 85-87.

[26] T. Back, D. B. Fogel, y Z. Michalewicz, editores. En: *Handbook of Evolutionary Computation. IOP Publishing Ltd., Bristol, UK, primera edición.* 1997.

[27] Pui Wah Poon y Jonathan Neil Carter. "Genetic algorithm crossover operators for ordering applications". En: *Computers & Operations Research* 22.1 (1995), págs. 135-147.

[28] Kusum Deep y Hadush Mebrahtu. "Combined mutation operators of genetic algorithm for the travelling salesman problem". En: *International Journal of Combinatorial Optimization Problems and Informatics* 2.3 (2011), págs. 1-23.

[29] *G. B. Dantzig, a., y J. H. Ramser, a. (1959). The truck dispatching problem. Management Science(1), 80.*

[30] *Paolo Toth y Daniele Vigo, "The Vehicle Routing Problem". Society of Industrial and Applied Mathematics (SIAM) monographs on discrete mathematics and applications, Philadelphia, USA, 2002, pp 1-23, 109-149.*

[31] *Alfredo Olivera, "Heurísticas para problemas de ruteo de vehículos", reporte de investigación, Instituto de Computación – Facultad de Ingeniería, Universidad de la República, Montevideo, Uruguay, 2004, disponible en http://www.fing.edu.uy/inco/pedeciba/bibliote/reptec/TR0408.pdf.*

[32] *Bruce Golden, S. Raghavan y Edward Wasil, "The vehicle routing problem: latest advances and new challenges". Springer, New York, 2008, pp 3-122.*

[33] *Leonora Bianchi, Mauro Birattari, Marco Chiarandini, Max Manfrin y Monaldo Mastrolilli, "Metaheuristics for the Vehicle Routing Problem with Stochastic Demands", Lecture Notes in Computer Science, Vol 3242, 2004, pp 450-460.*

[34] *The VRP Web, Collaboration between AUREN and the Languages and Computation Sciences department of the University of Málaga by Bernabé Dorronsoro Díaz, última actualización: marzo de 2007, consultada en abril de 2010, disponible en http://neo.lcc.uma.es/radiaeb/WebVRP/.*

[35] *Nabila Azi, Michel Gendreau y Jean-Yves Potvin, "An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles", European Journal of Operational Research, Vol. 202, No. 3, 2010, pp 756-763.*

[36] *María Battarraa, M. Monaci y Daniele Vigo, "An adaptive guidance approach for the heuristic solution of a minimum multiple trip vehicle routing problem", Computers and Operations Research, Vol. 36, 2009, pp 3041-3050.*

[37] *D.J. Guan y Xuding Zhu, "Multiple capacity vehicle routing on paths", Siam J. Discrete math, Vol. 11, No. 4, 1998, pp 590-602.*

[38] *Rego, C., Gamboa, D., Glover, F., y Osterman, C. (2011). Invited review: Traveling salesman problem heuristics: Leading methods, implementations and latest advances. European Journal of Operational Research, 211 , 427 - 441.*

[39] *Hoffman, K. L., Padberg, M., y Rinaldi, G. (2013). Traveling salesman problem. En S. I. Gass y M. C. Fu (Eds.), Encyclopedia of operations research and management science (pp. 1573–1578).*

[40] *Toth, P., y Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. Discrete Applied Mathematics, 123 (1-3), 487-512.*

[41] *Dionisio Pérez Brito, José Andrés Moreno Pérez y Carlos Gustavo García González, "Búsqueda por entornos variables: Desarrollo y Aplicaciones en localización" En: Avances en localización de servicios y sus aplicaciones por Blas Pelegrín Pelegrín. 1ª Edición, Servicio de publicaciones – Universidad de Murcia, Murcia, España, 2004, pp 349-374.*

[42] *Gilbert Laporte, 'The Vehicle Routing Problem: An overview of exact and approximate algorithms', European Journal of Operational Research, Vol. 59, 1991, pp 345-358.*

[43] *Víctor Yepes Piqueras, "Optimización heurística económica aplicada a las redes de transporte del tipo VRPTW", tesis doctoral, Departamento de Ingeniería de la Construcción y Proyectos de Ingeniería Civil, Escuela Técnica Superior de Ingenieros de Caminos Canales y Puertos – Universidad Politécnica de Valencia, Valencia, España, 2002.*

[44] *Alexander Ayala Rodríguez y Edgar González Butrón, "Asignación de rutas de vehículos para un sistema de recolección de residuos sólidos en la acera", Revista de Ingeniería - Universidad de Los Andes, No. 13, 2001, pp 5-11.*

[45] *Wee-Kit Ho, Juay Chin Ang y Andrew Lim, "A hybrid search algorithm for the vehicle routing problem with time windows", International Journal on Artificial Intelligence Tools, Vol. 10, N0.3, 2001, pp 431-449.*

[46] *M. L. Balinzki y R. E. Quandt, "On an Integer Program for a Delivery Problem", Operational Research, Vol. 12, No. 2, 1964, pp 300-304. Mencionado por Prawda, J. (2002).*

[47] *W. W. Garvin, H. W. Crandall, J.B. John y R. A. Spellman, "Aplications of Linear Programming in the Oil Industry", Management Science, Vol. 3, 1957, pp 407. Mencionado por Prawda, J. (2002).*

[48] *Gilbert Laporte, Michel Gendreau y Alain Hertz, "An aproximation algorithm for the traveling salesman problem with time windows", Institute for Operation Research and de Management Science – Operations Research, Vol. 45, No. 4, 1998, pp 639-641.*

[49] *Claudio Andrés Contardo Vera, "Formulación y solución de un problema de ruteo de vehículos con demanda variable en tiempo real, trasbordos y ventanas de tiempo", Memoria para optar al título de ingeniero civil matemático, Departamento de Ingeniería Matemática, Universidad de Chile, Santiago de Chile, Chile, 2005.*

[50] *Karl Doerner et all. "Savings Ants for the Vehicle Routing Problem", Lecture Notes in Computer Science – Applications of Evolutionary Computing, Vol. 2279, 2001, pp 73-109.*

[51] Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Jeff Bassett, Robert Hubley, A Chircop. *A java-based evolutionary computation research system*. [ECJ]. URL: https://cs.gmu.edu/~eclab/projects/ecj/.

[52] P.W. Poon y J.N. Carter. "Genetic algorithm crossover operators for ordering applications". En: *Computers and Operations Research* 22.1 (1995). Genetic Algorithms, págs. 135-147. ISSN: 0305-0548. DOI: https://doi.org/10.1016/0305-0548(93)E0024-N. URL: http://www.sciencedirect.com/science/article/pii/0305054893E0024N.

[53] W. Banzhaf. En: *The "molecular" traveling salesman, Biological Cybernetics, vol. 64, no. 1, pp. 7-14*. 1990.

[54] S. S. Shapiro y M. B. Wilk. En: *An analysis of variance test for normality (complete samples). Biometrika, 52(3-4):591–611*. 1965.

[55] Donald W Zimmerman y Bruno D Zumbo. "Relative power of the Wilcoxon test, the Friedman test, and repeated-measures ANOVA on ranks". En: *The Journal of Experimental Education* 62.1 (1993), págs. 75-86.

[56] H. Mann and D. En: *On a test of whether one of two random variables is stochastically larger than the other, The Annals of Mathematical Statistics, vol. 18, no. 1, pp. 50–60.* 1947.

[57] Kim Haase. "Java Message Service API Tutorial". En: *Sun Microsystems, Inc* (2002), págs. 1-278.

[58] Qusay H. Mahmoud. *Getting Started with Java Message Service (JMS)*. Accessed: 22-6-2020. Nov. de 2004. URL: https://www.oracle.com/technical-resources/articles/java/intro-java-message-service.html.

[59] Eighth Edition. C. J. Date. ISBN 0-321-19784-4. Addison-Wesley. 2003.

[60] *Francis, P., Smilowitz, K., y Tzur, M. (2006). The period vehicle routing problem with service choice. Transportation Science, 40 (4), 439-454.*

[61] Wikipedia contributors. *Algoritmo evolutivo*. [accessed 28-september-2019]. 2013. URL: https://es.wikipedia.org/wiki/Algoritmo_evolutivo.

[62] Jaroslav Hájek, Zbyněk Šidák y Pranab K. Sen. "Chapter 4 - Selected rank tests, Kolmogorov-Smirnov". En: *Theory of Rank Tests (Second Edition)*. Ed. por Jaroslav Hájek, Zbyněk Šidák y Pranab K. Sen. Second Edition. Probability and Mathematical Statistics. San Diego: Academic Press, 1999, págs. 94-164. DOI: https://doi.org/10.1016/B978-012642350-1/50022-9. URL: http://www.sciencedirect.com/science/article/pii/B9780126423501500229.

[63] M. Haklay y P. Weber. "OpenStreetMap: User-Generated Street Maps". En: *IEEE Pervasive Computing* 7.4 (oct. de 2008), págs. 12-18. ISSN: 1558-2590. DOI: 10.1109/MPRV.2008.80.

[64] Danilo Alvez, Juan Pablo Chalupa y Diego Correa. "Problema de ruteo con múltiples ventanas de tiempo para la recolección de leche". En: (2019).

[65] NIH. *National Human Genome Research*. URL: https://www.genome.gov/es/genetics-glossary/Alelo#:~:text=Un%20alelo%20es%20cada%20una,es%20homocigoto%20para%20este%20gen..