Un algoritmo colonia de hormigas para el problema de ruteo de vehículos eléctricos con función de carga parcial y ventanas de tiempo

Juan Felipe Avellaneda Gelvez, Rafael Eduardo Estévez Landazábal

Trabajo de Grado para optar por el título de Ingeniero Industrial

Director:

Karín Julieth Aguilar Imitola

MSc en Ingeniería Industrial

Codirector:

Henry Lamos Díaz

Ph. D en Física, Matemática

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Estudios Industriales y Empresariales

Bucaramanga, Santander

2019

Dedicatoria

Éste logro está dedicado a:

A mis padres Henry y Elsa con los que estaré eternamente en deuda por hacerme quien soy hoy, por ser mi apoyo en todos los aspectos de mi vida y darme la oportunidad de estudiar en esta prestigiosa universidad.

A mis hermanas Juanca, Diego, Jorge, Daniel, Henry y Cesar quienes me impulsaron durante toda mi vida a apuntar siempre a la cima.

A mi prima Natalia Gelvez, quien es mi modelo a seguir y a quien admiro con todo mi corazón, esto no hubiera sido posible sin su amor y colaboración.

A mis amigos, quienes me acompañaron incondicionalmente desde el colegio y en cada una de las etapas siguientes, y a quienes conocí en esta etapa universitaria, los cuales me demostraron que no importa el tiempo, cualquier persona puede convertirse en familia sin importar las circunstancias, gracias a todas estas personas por hacer de mi etapa universitaria como la más feliz de mi vida, por hacerme amar la vida y compartir nuestros logros de la mano, a todos ellos los llevaré siempre en mi corazón.

A Angie Arias, por ser mi compañera incondicional, brindarme todo su apoyo y amor, su paciencia y permitirme soñar juntos.

Por último, a mi compañero de proyecto Rafael Estévez, que más que un compañero de clase desde el primer día, ha sido un gran amigo y a quien mi familia estima mucho, gracias por nunca rendirse y por darme la fuerza necesaria para seguir cuando las cosas parecían estar en su peor momento.

Juan Felipe Avellaneda Gelvez

Dedicatoria

Éste logro está dedicado a:

A mis padres Ramiro y Nancy, por darme la excelente formación que tuve, por ser mis modelos a seguir, por tantos consejos, historias y recuerdos, gracias a ustedes hoy soy quien soy.

A mis hermanas Laura, Silvia y Mafe por ser un apoyo moral inquebrantable y siempre estar presentes.

A todos mis tíos y primos, por ser mis referentes y ejemplos a seguir más cercanos después de mis padres.

A mis amigos, con todos los que compartí dentro y fuera de las aulas. A aquellos amigos del colegio que se convirtieron en amigos de vida; y a los que pronto serán mis colegas, sin importar si comenzaron la carrera conmigo o si nos encontramos en el camino, especialmente a todos los que hicieron parte de la familia posgrados, por ser más que solo compañeros de trabajo y estudio, por ser realmente una segunda familia para mí, gracias por todo el apoyo y la alegría a lo largo de éstos años.

Por último pero lejos de ser lo menos importante, a Felipe Avellaneda, por ser mi compañero en éste último viaje, por confiar en mí desde el principio de esta historia y ser el causante de esta idea, no podría haber pedido un mejor compañero de proyecto.

Rafael Estévez Landazábal

Agradecimientos:

Agradecemos a nuestros padres, por darnos el orgullo y el privilegio de ser sus hijos, por todo el amor, sacrificio y trabajo puestos para darnos siempre lo mejor.

A nuestra directora Karin Aguilar por guiarnos a lo largo de todo éste proceso, por su confianza, paciencia y apoyo hacia nosotros.

A Iván Londoño, quien nos brindó su mano en el momento más difícil del proceso para seguir adelante.

A Javier Chacón, por su paciencia y colaboración con nosotros.

A la Universidad Industrial de Santander y a la Escuela de Estudios Industriales y Empresariales por habernos brindado tantas oportunidades para crecer como profesionales y como seres humanos.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de Apéndices

(Apéndices disponibles en CD adjunto – Disponible en Biblioteca UIS)

Resumen

Título: Un algoritmo colonia de hormigas para el problema de ruteo de vehículos eléctricos con función de carga parcial y ventanas de tiempo[1*]

Autores: Juan Felipe Avellaneda Gelvez; Rafael Eduardo Estévez Landazábal[**]

Palabras clave: VRP, EVRP, EVRPTW, EVRPTW-PR, ACO, Ant Colony, Partial Recharge

Descripción: Los vehículos eléctricos comerciales han contribuido significativamente en el mejoramiento de la movilidad en los países altamente desarrollados, por lo que las compañías han invertido para causar el mismo efecto en el área logística. El problema de ruteo de vehículos eléctricos con ventanas de tiempo y carga parcial (EVRPTW-PR) es una extensión del clásico problema de ruteo de vehículos (VRP) en el cual se cuenta con una flota de vehículos eléctricos, que debido a su rango de conducción limitado, requerirá visitar estaciones de carga mientras realiza el recorrido; Las recargas pueden ser realizadas en cualquier momento del recorrido y cualquier nivel de batería, además, gracias a los cortos tiempos de estas, se permiten recargas parciales para asemejar el modelo aún más a la vida real. En este documento, presentamos un modelo matemático para el problema de ruteo de vehículos eléctricos en cuestión y un algoritmo colonia de hormigas para resolverlo eficientemente, aplicando diversos métodos basados en planteamientos y sugerencias de la literatura reciente. Los resultados son presentados como el total de distancia recorrida y el número de vehículos necesarios para resolver el problema, estos son evaluados en instancias de literatura y comparados con otros métodos de solución para problemas afines, determinando mejoras potenciales.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Estudios Industriales y Empresariales. Director: Karín Julieth Aguilar Imitola, MSc en Ingeniería Industrial. Codirector: Henry Lamos Díaz, Ph. D en Física, Matemática.

Abstract

Title: An ant colony algorithm for the electric vehicle routing problem with time windows and partial recharging[*]

Authors: Juan Felipe Avellaneda Gelvez; Rafael Eduardo Estévez Landazábal[**]

Keywords: VRP, EVRP, EVRPTW, EVRPTW-PR, ACO, Ant Colony, Partial Recharge

Description: Electric commercial vehicles have significantly contributed to the improvement of mobility in highly developed countries, reason why the companies have invested to cause the same effect in the logistical area. The electric vehicle routing problem with time windows and partial recharging is an extension of the classic vehicle routing problem, in which it counts with fleet of electric vehicles, who, due to its limited driving range, requires visiting charging stations while making a trip. The recharges can be done at whatever moment and battery level, and thanks to the short duration of these recharges, it allows partial recharges to resemble to the model of real life. In this document, we present a mathematical model for the electric vehicle routing problem in question and an ant colony algorithm to solve it efficiently, applying diverse methods based on recent literature. The results are presented as the total traveled distance and the total number of vehicles required on the model evaluated in benchmark instances and compared to other solution methods for related problems to determine potential improvements.

[*] Bachelor Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Estudios Industriales y Empresariales. Director: Karín Julieth Aguilar Imitola, MSc en Ingeniería Industrial. Codirector: Henry Lamos Díaz, Ph. D en Física, Matemática.

Introducción

El Desarrollo de un país ha estado ligado normalmente al aumento del bienestar tanto personal como colectivo de sus habitantes, sin embargo en los últimos años ha entrado en rigor otro aspecto a tratar, haciéndose popular el término "Desarrollo sostenible", definido como aquel que garantiza la satisfacción de las necesidades del presente, sin comprometer las posibilidades de generaciones futuras para satisfacer sus propias necesidades (ONU, s.f.), siendo éste uno de los pilares más fuertes para el diseño de las nuevas de estrategias nacionales consecuentes con el compromiso de reducir las emisiones de Gases de Efecto Invernadero en un 20% para el año 2030 (MinAmbiente, 2015), adquirido por Colombia al firmar el acuerdo de París de 2015 en el COP21 de la convención Marco de las Naciones Unidas sobre el Cambio Climático, el Gobierno Nacional avanza en el fortalecimiento de medidas encaminadas a la adaptación y mitigación para reforzar la respuesta mundial a ésta amenaza.

Como muestra de ello, el gobierno de Colombia ha incluido en el Plan Nacional de Desarrollo la implementación de la Política de Cambio Climático en seis sectores, entre ellos el de transporte, impulsado por el ministerio de Medio Ambiente y desarrollo sostenible se han apoyado las estrategias de Movilidad eléctrica y medios de transporte alternativos en aras de garantizar una mejor calidad del aire para los colombianos y en consecuencia una mejor calidad de vida (MinAmbiente, 2018), ya que los vehículos eléctricos juegan un rol bastante importante en cuanto a brindar una alternativa más económica y eco-sostenible frente a los medios de transporte tradicionales impulsados por motores de combustión; sin embargo, el poco desarrollo en baterías con la capacidad necesaria para el uso diario de éstos vehículos hace que sea fundamental tener

una ruta planeada para que los usuarios puedan utilizar el servicio de transporte sin inconvenientes causados por el requerimiento de paradas ocasionales destinadas a recargar energía eléctrica de los puntos establecidos en las ciudades, convirtiéndose en una gran oportunidad por parte de la academia para contribuir con ésta integración, y transición del sistema tradicional a la movilidad eléctrica en el país.

En el presente proyecto se llevará a cabo un planteamiento teórico del problema de ruteo de vehículos eléctricos con ventanas de tiempo (EVRPTW Por sus siglas en inglés) en donde los vehículos podrán recibir una carga parcial en las ventanas de tiempo establecidas, creando una situación con mayor semejanza a la utilización real de los puntos de carga, dado el requerimiento de cumplir con las demandas de tiempo de los usuario, sea desde el ámbito comercial de la logística, como en los desplazamientos regulares de los usuarios en su vida cotidiana, finalmente será diseñado un Algoritmo Colonia de Hormigas (ACO por sus siglas en inglés) para resolver el problema en un tiempo computacional razonable.

Cumplimiento de Objetivos

Tabla 1.
Cumplimiento de objetivos.

Objetivo	Apartado Relacionado
1. Realizar una revisión de literatura sobre el problema general de vehículos eléctricos (EVRP) y su extensión con función de carga parcial y ventanas de tiempo (EVRPTW-PR).	Capítulo 2
2. Formular un modelo matemático para el EVRPTW-PR.	Capítulo 4.2
3. Construir un algoritmo colonia de hormigas para la solución del EVRP con carga parcial y ventanas de tiempo en el lenguaje de programación Python.	Capítulo 4.3
4. Verificar la eficiencia del algoritmo planteado mediante instancias de la literatura.	Capítulo 5 y 6
5. Elaborar un artículo publicable a partir de la investigación realizada.	Apéndice H.

1. Generalidades de la investigación

1.1. Objetivos

1.1.1 Objetivo General.

Desarrollar un algoritmo colonia de hormigas para el problema de ruteo de vehículos eléctricos con función de carga parcial y ventanas de tiempo**.**

1.1.2 Objetivos Específicos.

- Realizar una revisión de literatura sobre el problema general de vehículos eléctricos (EVRP) y su extensión con función de carga parcial y ventanas de tiempo (EVRPTW-PR).
- Formular un modelo matemático para el EVRPTW-PR.
- Construir un algoritmo colonia de hormigas para la solución del EVRP con carga parcial y ventanas de tiempo en el lenguaje de programación Python.
- Verificar la eficiencia del algoritmo planteado mediante instancias de la literatura.
- Elaborar un artículo publicable a partir de la investigación realizada.

1.2. Metodología

El trabajo de investigación se desarrollará en cinco etapas, en las cuales cada una contará con una serie de actividades para lograr el cumplimiento del proyecto.

1.2.1. Etapa I: Búsqueda y revisión de la literatura.

- Realizar una lectura preliminar de artículos científicos en diferentes bases de datos con el fin de definir el tema del proyecto.
- Definición de palabras claves y ecuación para la búsqueda en las bases de Datos de la Universidad Industrial de Santander y otras fuentes externas.
- Revisión de la literatura sobre las aplicaciones y estado real de la investigación para obtener una valoración del tema investigado.
- Selección de artículos existentes en las bases de datos que permitan analizar diferentes aplicaciones del problema de investigación.

1.2.2. Etapa II: Definición y estudio de las características del problema.

- De acuerdo con lo encontrado en el desarrollo de la revisión de literatura, se identifican y definen las características del problema presentado, estableciendo así las restricciones, las variables, los parámetros y los supuestos.
- Estudiar las características del algoritmo colonia de hormigas para su adaptación al EVRP.
- Estudiar las bases del lenguaje de programación Python.

1.2.3. Etapa III: Formulación y programación.

- Formulación del modelo matemático para el EVRPTW-PR.
- Diseño del algoritmo colonia de hormigas para el EVRPTW-PR.
- Programación del algoritmo colonia de hormigas establecido para el modelo en el lenguaje de programación Python y posteriormente la implementación en la solución del EVRPTW-PR.

1.2.4. Etapa IV: Experimentación numérica.

- Experimentación numérica con la finalidad de comprobar la efectividad del algoritmo colonia de hormigas para validar la solución del EVRPTW-PR utilizando las instancias de literatura existentes para este problema para verificar y comparar la eficiencia del método de solución planteado en el proyecto.

1.2.5. Etapa V: Documentación.

- Concluir acerca de los resultados obtenidos utilizando el algoritmo colonia de hormigas y plantear recomendaciones para futuras investigaciones en temas afines.
- Elaboración del Libro en donde se consolide los resultados del trabajo de investigación.
- Realización de un artículo publicable con los resultados obtenidos.

2. Revisión de la literatura

Las consecuencias que ha sufrido el planeta a causa de los cambios climáticos acelerados producto de un estilo de vida ambientalmente irresponsable, han estimulado la generación de investigaciones frente a la temática de energías renovables que ofrecen soluciones a la sociedad, estas deben ser planeadas teniendo en cuenta las diferentes perspectivas sociales, políticas, económicas y ambientales, pues lo que se busca es generar un beneficio disminuyendo al máximo los costos y las los impactos negativos para el planeta.

El EVRP es un área de investigación dentro del problema de ruteo de vehículos, con la característica distintiva de manejar una flota de vehículos eléctricos, es decir, el EVRP puede verse como una variante del GVRP (Green vehicle routing problem), propuesto por Erdoğan & Miller-Hooks (2012), en donde consideran el uso de combustibles o vehículos alternativos para el problema de ruteo de vehículos como solución a la problemática del exceso de contaminación ambiental, siendo por definición, una extensión al VRP.

Por lo anterior, en esta revisión preliminar de la literatura se incluyen estudios sobre el ruteo de vehículos convencional, ya que es considerado como uno de los primeros y más importantes resultados sobre la optimización logística de transporte y de sus categorías derivadas como lo es el VRP y en el caso de esta investigación, el EVRP. Este comenzó con el análisis y solución del problema del vendedor viajero planteada por Dantzig & Ramser (1959) donde se buscaba la asignación optima de estaciones de gasolina para camiones transportadores de combustible, minimizando la totalidad de distancia recorrida y cumpliendo con la demanda programada de cada estación. Aunque los investigadores no realizaron una aplicación práctica del modelo, lograron

abrir paso para formular los primeros VRPs, que consisten en la consideración de rutas representadas gráficamente por una serie de arcos y vértices correspondientes a los costos, tiempos y distancias de los recorridos en la búsqueda de optimizar las rutas seleccionadas para después ser asignadas a una cantidad calculada o establecida de vehículos. Para adaptar el modelo a casos más reales, con una mayor cantidad de condiciones, se comenzó a plantear extensiones del modelo como el descrito por Pullen & Webb (1967), siendo este uno de los primeros planteamientos sobre el problema de ruteo de vehículos con ventanas de tiempo (VRPTW), el cual estudia la programación horaria para el cumplimiento de un servicio de transporte, en este caso, el de entrega de correo por parte de conductores de furgonetas en el centro de Londres, Inglaterra, con el fin de ajustar el horario de cada uno de los conductores de tal forma en que se redujera el tiempo de inactividad en el trabajo. Como lo plantean Schneider, Stenger, & Goeke (2014), las ventanas de tiempo aparecieron para lograr cumplir con la demanda solicitada en un límite de tiempo marcado por la llegada del cliente al nodo, definiendo el momento de inicio del servicio de tal manera que se pueden generar esperas en la ruta dependiendo de la ventana de tiempo de cada nodo.

Zheng & Liu (2006) también trabajan el VRPTW, enfocándose en la minimización de la distancia recorrida por los vehículos y ampliando aún más las condiciones del problema, considerando los tiempos de viaje como variables difusas y formulando un modelo de optimización difuso, resolviendo el problema mediante un algoritmo genético, y uno inteligente hibrido compuesto por una simulación.

Taş, Jabali, & Van Woensel (2014) plantearon una variante del VRPTW en donde se permite que los vehículos no cumplan en su totalidad con las ventanas de tiempo del cliente bajo una tolerancia definida, en aras de buscar una reducción en el costo del transporte sin tener una restricción tan estricta; a esta instancia la llamaron el problema de ruteo de vehículos con ventanas

de tiempo suaves (VRPFSTW), solucionándolo por medio del algoritmo de búsqueda tabú y la heurística del vecino más cercano.

Un avance importante para el problema de ruteo de vehículos, fue la incursión de autores que lograran combinar el VRP con las nuevas tecnologías, comenzando así investigaciones como el artículo “Electric vehicle routing problem” (Lin, Zhou, & Wolfson, 2016) y “Electric vehicle routing problem with recharging stations for minimizing energy consumption” (Zhang, Gajpal, Appadoo, & Abdulkader, 2018), artículos en los cuales aparecen las restricciones características de los vehículos eléctricos, como el tiempo de carga que dificultan el problema y limitan las opciones, además, no se obliga a que la batería sea cargada totalmente antes de poder continuar con el recorrido, permitiendo que los vehículos salgan de las estaciones de recarga antes de concluir con el tiempo de carga total.

Otro tipo de profundización en las diferentes restricciones que presenta el EVRP, puede verse en el artículo “Solving the battery swap station location-routing problem with capacitated electric vehicles using an AVNS algorithm for vehicle-routing problems with intermediate stops” (Hof, Schneider, & Goeke, 2017), donde se propone que en las estaciones de carga se haga un intercambio entre la batería usada y una totalmente cargada para optimizar el tiempo de funcionamiento de los vehículos a lo largo del día; es decir, que el tiempo en el que los vehículos eléctricos circulen en su ruta definida va a ser significativamente más alto que si tuvieran que cargarse, disminuyendo así los costos de transporte desde un punto de vista del tiempo de holgura que tendrían los vehículos si tuvieras que cargarse.

En los artículos “Partial recharge strategies for the electric vehicle routing problem with time Windows” (Merve & Çatay, 2016) y “The electric location routing problem with time windows

and partial recharging" (Schiffer & Walther, 2017), se combina la investigación del ruteo de vehículos con ventanas de tiempo con las nuevas investigaciones del EVRP, se desecha la consideración del nivel de carga de batería como función lineal del tiempo y se resuelven por medio del algoritmo de búsqueda de vecindario "Adaptive Large Neighborhood Search" (ALNS), uno de los métodos de solución más utilizados junto al VNS, el algoritmo genético y el ACO, cuyos resultados arrojaron soluciones de alta calidad.

Merve & Çatay (2016) utilizaron en su artículo "Electric Vehicle Routing Problem with Recharging Stations for Minimizing Energy Consumption" un algoritmo de optimización llamado colonia de hormigas (ACO), para resolver el problema de exceso de visitas a estaciones de carga en un operador logístico y minimizar el consumo de energía por causa de la limitada capacidad de las baterías para vehículos eléctricos; Para esto, se basan en meta-heurísticas de solución de EVRP's y evalúan la efectividad de la solución por medio de instancias, ilustrando el beneficio de usar una función objetivo basada en la minimización del consumo de energía, y no de la distancia recorrida.

Los resultados obtenidos demostraron una alta efectividad gracias a su método de solución, ya que mejoraron en un 16.44% el resultado de las instancias, venciendo en términos de resultados al algoritmo ALNS en instancias con gran cantidad de datos y en tiempo de computación en general, demostrando que el hecho de recorrer menor cantidad de distancia, no necesariamente significa minimizar el consumo de energía.

Para resumir, podemos decir que el ruteo de vehículos eléctricos (EVRP) es un tema de estudio reciente, ya que se han venido publicando artículos sobre éste en su mayoría desde el año 2013 (hace 5 años) y generalmente en países desarrollados e industrializados, siendo China, Alemania

y Estados Unidos los líderes en cuanto a número de publicaciones sobre éste; El EVRP es una extensión del Problema de Ruteo de vehículos (VRP), el cual se ha venido desarrollando desde su forma más simple como lo es el Problema del Vendedor Viajero (TSP) planteado por Dantzig & Ramser (1959) hasta sus formas actuales, en donde hoy en día se toma un enfoque sostenible y con restricciones cada vez más realistas para la vida cotidiana como lo es el caso del EVRPTW, buscando formas de resolverlos con Meta-heurísticas y algoritmos, siendo el ANLS, la optimización por colonia de hormigas y el algoritmo genético los métodos de solución más recurrentes en la literatura.

3. Marco teórico

3.1. Optimización Combinatoria

Un problema de optimización combinatoria usualmente consiste en una estructura discreta, como una red o una familia de grupos, unidos a un conjunto de números, el cual puede representar el costo o la capacidad asignada a cada uno de los elementos de la primera estructura (Cook, Cunningham, Pulleyblank, & Schrijver, 1997) siendo así, una optimización combinatoria puede definirse como el proceso de búsqueda del valor máximo (o mínimo, dependiendo del caso) de una función objetivo F cuyo dominio es discreto pero mantiene un gran espacio de configuración (contrario a un espacio continuo N-dimensional).

3.2. Problema del Vendedor Viajero (TSP)

El problema puede ser planteado de la siguiente manera: un vendedor ambulante desea visitar exactamente una vez, cada una de las n ciudades (donde el costo de viajar de la ciudad i a la ciudad j es cij) y luego regresar a la ciudad local (Hoffman, Padberg, & Rinaldi, 2013), buscando minimizar el costo total del recorrido, el TSP es la base del tipo de problemas de optimización combinatoria como lo es el Problema de Ruteo de Vehículos (VRP).

3.3. Problema de Ruteo de Vehículos (VRP)

El problema de ruteo de vehículos (VRP, Vehicle Routing Problem) consiste en un conjunto de clientes con demandas específicas, un depósito y una flota de vehículos con una capacidad determinada, con un costo asignado a cada uno de los desplazamientos entre la localización geográfica de los clientes, influyendo el sentido en el cual se realice, de ésta forma que se pretende encontrar una ruta que minimice costos, cuyo inicio y fin sea el mismo punto (el depósito) (Gelves T., Mora M., & Lamos D., 2015); en este caso, la capacidad de los vehículos y las demandas son factores determinísticos, la soluciones a éste problema buscan minimizar los costos de transporte que se presentan en la logística de la cadena de suministro.

3.4. Problema de Ruteo de Vehículos Eléctricos (EVRP)

Los Vehículos Eléctricos (EV´s por sus siglas en inglés) son mucho más eficientes energéticamente que los vehículos convencionales impulsados por combustibles fósiles y por lo tanto se espera que el aumento en el uso de éstos proporcione un ahorro considerable en cuanto al requerimiento energético de las ciudades. Sin embargo, comparados con los vehículos convencionales, los EV´s poseen un rango de acción menor, normalmente entre 40 y 100 millas.

Antes de ser recargados nuevamente. Debido a éste límite en el rango de acción, los EV´s pueden tener la necesidad de visitar una estación de carga entre las entregas de su operación diaria, ésta nueva adición hace que el problema de ruteo de vehículos eléctricos (EVRP) sea tratado de manera distinta al VRP tradicional (Lin et al., 2016).

A continuación, se presentan los diferentes casos estudiados para el Problema de Ruteo de vehículos eléctricos.

Ruteo de vehículos eléctricos con carga lineal y no lineal

Generalmente las funciones de carga no son lineales, porque existen varios factores que fluctúan durante el proceso de carga, como el voltaje y la corriente eléctrica del terminal.

El proceso de carga se divide en dos fases, en la primera, la corriente de carga se mantiene constante y así el nivel de la batería se incrementa de manera lineal con el tiempo, éste proceso continuo hasta que el voltaje del terminal de la batería llega a un “valor máximo” específico; en la segunda fase, la corriente disminuye de manera exponencial y el voltaje de la terminal se mantiene constante para evitar daños en la batería, el nivel de carga aumenta con forma cóncava en el tiempo.

A pesar de que la forma de las funciones de carga es conocida, plantear expresiones analíticas para modelarlas es complejo debido a que dependen de factores como la corriente, el voltaje, la auto recuperación y la temperatura (Montoya, Guéret, Mendoza, & Villegas, 2017).

El nivel de carga de la batería puede ser descrito usando ecuaciones diferenciales, sin embargo, éstas ecuaciones son difíciles de incorporar en modelos de E-VRP, los investigadores confían en las aproximaciones de las funciones de carga actuales a funciones de carga con expresiones lineales para sus modelos, por ejemplo, Bruglieri et al. (2014) usa una aproximación lineal que considera

solamente el segmento lineal de la función de carga ente 0 y el 80%(aproximadamente) de la capacidad de carga de total de la batería, denominando ésta Q, ésta aproximación evita tratar el segmento no lineal de la función de carga (aproximadamente entre 0.8Q y Q), y otros autores aproximan la función de carga completa a una expresión lineal (Montoya et al., 2017).

Ruteo de vehículos eléctricos con ventanas de tiempo (EVRPTW)

Las labores de distribución de las compañías logísticas generalmente son representadas como problemas de ruteo de vehículos (VRP´s), en los cuales se busca minimizar el costo total de transporte al visitar un conjunto de clientes por medio de múltiples rutas que comienzan y terminan en el almacén, sin embargo, se han venido planteando nuevas variedades y extensiones del problema con el fin de incorporar condiciones y restricciones que se presentan a la hora de llevar el problema al mundo real.

La utilización de vehículos eléctricos en funciones comerciales debe tener presente la restricción práctica más importante del servicio de logística, las ventanas de tiempo (TW por sus siglas en inglés), en donde se debe alcanzar a los clientes dentro de un intervalo de tiempo especificado, primero deben tenerse en cuenta la restricción de capacidad del vehículo para planificar sus labores y segundo, muchas compañías en el sector de la logística de última milla, poseen un gran porcentaje de pedidos con tiempos de entrega definidos, lo que hace la integración de la ventana de tiempo establecida por los clientes una necesidad en el planteamiento del modelo. El segundo aspecto es bastante interesante ya que los tiempos de recarga de los EV no pueden asumirse como fijos, ya que éstos dependen del nivel de batería con la que el vehículo llega a la estación de carga, junto al factor tiempo, ya que la operación de recarga toma una cantidad significativa de tiempo, especialmente si es comparado con los tiempos de servicio al cliente, (por

ejemplo al recoger paquetes pequeños), un factor que claramente afecta la planeación del ruteo (Schneider et al., 2014).

Ruteo de vehículos eléctricos capacitados (CEVRP)

Como se mencionó anteriormente, las restricciones aplicadas a los problemas cada vez son más realistas, ya sean factores proporcionados por los clientes, como son las ventanas de tiempo, o limitaciones propias del proveedor del servicio logístico, como lo es la capacidad de carga de los vehículos, ya que ésta, junto con las ventanas de tiempo, son las características que más se presentan en los problemas de Ruteo de Vehículos(VRP) en la literatura de los últimos años (Braekers, Ramaekers, & Van Nieuwenhuyse, 2016).

Ruteo de vehículos eléctricos con cargas parciales (PR-EVRP)

La literatura de problemas de EVRP puede dividirse en dos grupos en términos de criterios de carga: estudios que asumen una recarga total y quienes asumen recarga parcial.

Como su nombre lo indica, en las políticas de recarga total, la capacidad de la batería es restaurada en su totalidad cada vez que el vehículo llega a una estación de carga (CS, por sus siglas en inglés) y en las políticas de recarga parcial, el tiempo invertido en cada CS depende del nivel de energía presente en la batería una vez el EV llega y la tasa de recarga (constante) de la CS. En las políticas de recarga parcial, el nivel de carga (junto con el tiempo invertido en cada CS) representan una decisión variable. Hasta el momento, todos los modelos existentes de E-VRP con carga parcial consideran la recarga del vehículo como aproximaciones de funciones lineales (Montoya et al., 2017).

Meta-Heurísticas de solución para problemas de ruteo de vehículos.

Las técnicas denominadas Heurísticas elaboran una exploración relativamente limitada en el espacio de búsqueda y normalmente presentan soluciones de buena calidad dentro de un tiempo de cómputo razonable.

Sin embargo, hacer énfasis en las técnicas heurísticas avanzadas (meta-heurísticas) permite realizar una exploración profunda de las regiones más prometedoras del espacio solución. La calidad de las soluciones producidas por estos métodos es mucho mejor que la obtenida por las heurísticas clásicas, aunque su tiempo de cómputo es por lo general mucho mayor.

Existen dos tipos principales de familias de técnicas heurísticas avanzadas, las basadas en métodos de búsqueda local (o por vecindario),en donde se parte del hecho que hay una solución inicial realizable, y a ésta se le realizarán una serie de modificaciones locales, siempre y cuando, dichas modificaciones se acerquen más al objetivo del problema en estudio y las que exploran las poblaciones de soluciones (Algoritmos genéticos y meméticos), en donde se emplea un conjunto de soluciones (población) en cada iteración del algoritmo (Cardozo O., 2013).

Adaptive Large Neighborhood search (ALNS)

El ALNS es una estructura de búsqueda local en donde cierto número de algoritmos simples compiten por modificar la solución actual. En cada iteración un algoritmo es escogido para destruir la solución establecida hasta el momento, y un algoritmo para repararla. La nueva solución solamente es aceptada si ésta satisface los criterios definidos por el marco de búsqueda local aplicada en el nivel maestro, sin embargo. En vez de verse la heurística ALNS como una secuencia de operaciones de destruir y reparar, otra alternativa puede ser verlo como como una secuencia de operaciones de arreglo y optimización.

En cada iteración del ciclo principal, se escoge un vecindario para destruir y otro para reparar, una capa adaptativa controla estocásticamente cual vecindario escoger, de acuerdo al desempeño que ha tenido en el pasado (puntaje). Mientras más un vecindario haya contribuido al proceso de solución, mayor será el puntaje obtenido, y, por lo tanto, tendrá mayor posibilidad de ser escogido.

La estructura ALNS es una extensión del LNS (Large Neighborhood Search), en donde un gran conjunto de variables es modificado en cada iteración, en el ALNS, los Vecindarios son buscados por heurísticas simples y rápidas, en cada iteración la solución dada hasta el momento es destruida parcialmente y reparada usando heurísticas, el ALNS también presenta similitudes con el VLSN (Very Large Scale Neighborhood). Pues en éste, el algoritmo opera en un vecindario muy grande, escogido de tal manera que los algoritmos pueden seguir trabajando en la búsqueda de soluciones de manera eficiente (Pisinger & Ropke, 2007).

Algoritmo Genético (Genetic Algorithm)

Éste algoritmo forma parte un conjunto de técnicas basadas en el proceso evolutivo de las especies agrupadas bajo el nombre de “computación evolucionaria”.

Este algoritmo opera con una población de individuos en donde cada uno de ellos representa un punto de búsqueda en el espacio de las soluciones potenciales para el problema; el desempeño de cada individuo se evalúa utilizando una función de adaptación, ésta permite ordenar los individuos de la población del mejor al peor en un continuo de grados de adaptación (Valencia, 1997). De ésta forma, la población inicial evoluciona continuamente dirigiéndose a mejores resultados al moverse dentro del espacio de búsqueda mediante procesos probabilísticos al seleccionar a los individuos más adaptados en la población (a mayor grado de adaptación mayor probabilidad de

dejar descendencia) y modificando por recombinación y/o mutación a los individuos seleccionados.

Búsqueda Tabú (Tabu Search).

Este método de búsqueda utiliza cierta memoria o conocimiento previo acerca del dominio para encontrar una buena solución al problema que se pretende resolver, o una solución, al menos, cercana a la óptima.

Tomando X como el dominio de búsqueda relacionado con algún problema y sea “x” (perteneciente al conjunto X) una posible solución. Para buscar puntos mejores se define de alguna manera conveniente, la vecindad de un punto cualquiera en X. De esta manera es posible buscar soluciones mejores que x entre sus vecinos, a los que se denotará con N(x). Conforme pasa el tiempo y se va explorando más el dominio, será conveniente excluir algunos puntos que se sabe no son buenos (para no buscar en vano). De manera que la vecindad de un punto x cambia con el tiempo, depende de la historia (H) del proceso de búsqueda. Así que es conveniente denotar con N(x) sólo a la vecindad inicial de x, y con N(H,x) la vecindad de x como función de la historia. Algunos puntos que estaban en N(X) y que, al paso del tiempo, resulten no ser buenos candidatos de solución, serán marcados como “tabú” y no podrán ser alcanzados desde x, es decir, no serán elementos de N(H,x).

Al principio de la ejecución del algoritmo es deseable identificar regiones donde se obtienen buenas soluciones y regiones donde se obtienen malas. Esto es posible si se posee suficiente diversidad, es decir, si se procura la exploración. En etapas tardías del algoritmo es mejor hacer una búsqueda más intensiva dentro de las regiones buenas para aproximarse al óptimo. Esto es, en general, válido para varias heurísticas. Cuando es costoso determinar qué elementos están en

N(H,x) se selecciona una muestra representativa de la vecindad NC(H,x) Í N(H,x). En la búsqueda tabú la aleatoriedad es, si no ignorada, por lo menos minimizada (Kuri, 2000).

Algoritmo Colonia de Hormigas (Ant Colony Algorithm)

La optimización por Colonia de Hormigas es parte del gran campo de la Inteligencia de enjambre (swarm intelligence) en donde los científicos estudian los patrones de comportamiento de abejas, termitas, hormigas y otros insectos sociales con el fin de simular sus procedimientos. Los algoritmos de inteligencia artificial como lo es la optimización por Colonia de hormigas son aplicados a grandes problemas de optimización combinatoria y usados para crear métodos de auto-organización para éstos. Cabe aclarar que el ACO es una Meta heurística de exploración poblacional (Cardozo O., 2013).

El algoritmo Colonia de Hormigas (ACO por sus siglas en inglés) es una técnica meta-heurística que utiliza hormigas artificiales con el fin de encontrar solución a problemas de optimización combinatoria. El ACO está basado en el comportamiento de hormigas reales y posee ventajas como la memoria de acciones pasadas y el conocimiento de la distancia a otras locaciones. En la naturaleza, una hormiga individualmente es incapaz de comunicarse o conseguir comida de manera efectiva, pero como grupo, las hormigas poseen la habilidad de resolver problemas complejos, encontrar y recolectar con éxito comida para su colonia.

Las hormigas se comunican a través de sustancias químicas llamadas feromonas, cuando una hormiga viaja, deposita una cantidad constante de feromonas que otra hormiga puede seguir, cada hormiga se mueve de una manera algo aleatoria, pero cuando una de ellas encuentra un camino de feromonas, debe decidir si seguirlo o no, si ésta sigue el camino, las feromonas de ésta nueva hormiga reforzará el camino ya existente y el aumento en feromonas incrementará la probabilidad

de que una nueva hormiga también decida seguirlo, por lo tanto, mientras más hormigas viajen en un determinado camino, este será más atractivo para futuras hormigas, adicionalmente una hormiga que usa un camino corto para ir a donde se encuentra la fuente de alimento, regresará al hormiguero más rápido y por lo tanto el camino será reforzado por ella misma antes de que otras hormigas regresen. Esto influencia directamente la probabilidad de selección de la próxima hormiga que salga del hormiguero, con el tiempo, mientras más hormigas sean capaces de completar el camino más corto, las feromonas se acumularán más rápido en este y los caminos largos no serán reforzados.

La evaporación de las feromonas también hace las rutas menos deseables más difíciles de detectar y disminuye su utilización. Sin embargo, la continua selección aleatoria de caminos por parte de hormigas individuales ayuda a la colonia a descubrir rutas alternas y asegura una navegación exitosa alrededor de los obstáculos que interrumpen la ruta. La selección de caminos por hormigas es un proceso proporcional pseudo-aleatorio y es un elemento clave del algoritmo de simulación de optimización colonia de hormigas (Bell & McMullen, 2004).

4. Diseño del modelo

Teniendo en cuenta la información obtenida en la revisión de la literatura, se plantea un modelo matemático, adaptado del formulado en el artículo “The electric location routing problem with time windows and partial recharging” (Schiffer & Walther, 2017). En este artículo, los autores buscan minimizar 5 funciones objetivo diferentes, donde no solamente se tienen en cuenta los

costos totales, sino también individualmente la cantidad de vehículos utilizados, la distancia total recorrida, el número total de estaciones de carga ubicadas y la relación entre vehículos y estaciones de carga utilizadas; con el fin de comprobar cuál es el parámetro que más incide en el costo total del recorrido. Los autores hacen uso de más de 50 restricciones diferentes adaptadas a cada función objetivo, asumiendo la existencia de 3 tipos de costos diferentes que representan los costos fijos de adquisición de un vehículo y de una estación de carga, y el costo operacional que depende de la distancia total recorrida. Sin embargo, los autores no consideran el uso de tecnologías múltiples en las estaciones de carga como se hace en las instancias del GVRP; por lo tanto, utilizaron una modificación de las instancias de Solomón planteadas en el artículo "The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations" (Schneider, Stenger & Goeke, 2017), las cuales fueron desarrolladas por los autores para problemas de ruteo de vehículos eléctricos con ventanas de tiempo. Teniendo en cuenta estas instancias, y removiendo las restricciones de localización del modelo guía, se realizaron modificaciones a todo el modelo matemático para que pueda ser formulado con las condiciones que se listan a continuación en el presente documento.

4.1. Descripción del Problema

El problema de ruteo de vehículos eléctricos (EVRP) está representado por medio de un grafo $G = (v, A)$, que contiene el conjunto de todos los nodos y arcos para plasmar el problema, donde $v = (C \cup R, S)$, es un conjunto compuesto por los nodos de clientes $C = \{1, .., n\}$ con una demanda D_i (unidades) y un tiempo de servicio s_i (horas), $R = \{1, .., m\}$, los nodos de las estaciones de carga existentes, además del depósito y, en el cual está ubicada una estación de carga que funciona para cargar los vehículos en las noches. El grafo presenta una serie de arcos (i, j) contenidos en A, que tienen una distancia d_{ij} (kilómetros) y un tiempo de viaje desde i hasta j

denotado como t_{ij} (horas). Con el fin de permitir múltiples visitas a los nodos y visitas simultaneas entre vehículos, se define el uso de sets de nodos ficticios como se propone en "Transportation Research Part E: Logistics and Transportation" (Felipe et al., 2014), los cuales están contenidos en S, con copias de C y de R que coinciden en coordenadas y parámetros originales con los nodos base, lo cual hace posible que cada vehículo pueda salir y entrar por medio de una copia diferente tanto del depósito, como del resto de nodos.

Para representar las ventanas de tiempo (horas) se incluyen los parámetros e_i, como el tiempo mínimo de llegada permitido para llegar a i, y l_i, como el tiempo máximo de llegada permitido para i. Adicionalmente, las capacidades máximas de carga de mercancía y de carga de batería son definidas por los escalares F (unidades) y Q (Kwh) respectivamente, con el fin de limitarlas a valores reales para obtener una solución funcional.

El tiempo de carga es el producto entre la cantidad de energía cargada en el nodo i, denotada como w_i (Kwh), y la tasa de recarga de batería representada por el escalar r (hora/Kwh), a su vez, la cantidad de energía consumida está descrita como la relación lineal entre la distancia entre los nodos (d_{ij}) y la tasa de consumo de batería o (Kwh/km). Para definir el trayecto de los vehículos es necesario hacer uso de una variable de decisión binaria que nos indique que el arco (i, j) es usado, para esto se hará uso de la variable x_{ij}, la cual tomara el valor de $x_{ij} = 1$ si el arco es viajado, o $x_{ij} = 0$ en el caso contrario, en el que no se viaje ese arco.

Para simplificar el planteamiento del problema, se siguió el proceso de rastreo inverso descrito en "The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations" (Schneider, Stenger & Goeke, 2017), el cual tiene como objetivo eliminar el subíndice que representa los vehículos en el problema, estimando el número de estos, basado en la información

del número de arcos usados, aprovechando el hecho de que no puede haber arcos que realicen el mismo recorrido de ida y vuelta. El tiempo de llegada al nodo i está definido por el parámetro τ_i (horas), la cantidad de carga restante de bateria en el nodo i y la cantidad de carga de mercancia en el nodo i están definidas como q_i y f_i respectivamente.

El proceso de carga de batería puede ser llevado a cabo en cualquier momento y están permitidas las cargas parciales, pueden realizarse de manera simultánea y el desgaste de la batería se desprecia. Adicionalmente, se establecen 2 costos unitarios: C^v (€) que representa el costo de adquisición de un vehículo y C^d ($€/km$), que es el costo por kilómetro recorrido, el cual afectará la función según la cantidad de veces en las que se haga uso de las estaciones de carga y el depósito para recargar las baterías de los vehículos.

Dadas estas condiciones, se considerarán factibles las rutas que inicien y finalicen en el depósito sin exceder el límite de tiempo estipulado, cumplan los requisitos de capacidades y las restricciones planteadas en el modelo matemático a continuación. El número máximo de vehículos a utilizar está definido por el escalar N.

4.2. Modelo Matemático

Las variables de decisión del modelo serán:

- $x_{ij} = \{0,1\}$ es una variable binaria que toma el valor de 1 si el vehículo recorre el arco (i, j) y 0 si no lo hace.
- $\tau_i \geq 0$ tiempo de llegada al nodo i.
- $w_i \geq 0$ cantidad de energía cargada en la batería en el nodo i.
- $f_i \geq 0$ cantidad de carga de mercancía en el vehículo en el nodo i.
- $s_i \geq 0$ tiempo de servicio en el nodo i.

- $q_i \geq 0$ cantidad de energía en la batería del vehículo en i.

Min:
$$z = C^v \sum_{j \in v \setminus y} x_{yj} + C^d \sum_{i,j \in v, i \neq j} d_{ij} * x_{ij} \quad (1)$$

S.a.

$$\sum_{j \in v, i \neq j} x_{ij} = 1 \qquad \forall i \in C \quad (2)$$

$$\sum_{j \in v, i \neq j} x_{ij} \leq 1 \qquad \forall i \in R \quad (3)$$

$$\sum_{j \in v, i \neq j} x_{ji} - \sum_{j \in v, i \neq j} x_{ij} = 0 \qquad \forall j \in v \quad (4)$$

$$\sum_{j \in v} x_{yj} = \sum_{j \in v} x_{jy} \leq 1 \qquad \forall j \in v \quad (5)$$

$$\tau_j \geq \tau_i + (t_{ij} + s_i)x_{ij} - l_y(1 - x_{ij}) \qquad \forall i \in C, \forall j \in v, i \neq j \quad (6)$$

$$\tau_j \geq \tau_i + t_{ij}\, x_{ij} + r\, w_i - (l_y + rQ)(1 - x_{ij}) \qquad \forall i \in R, \forall j \in v, i \neq j \quad (7)$$

$$e_i \leq \tau_i \leq l_i \qquad \forall i \in v \quad (8)$$

$$0 \leq f_j \leq f_i - D_i\, x_{ij} + F(1 - x_{ij}) \qquad \forall i \in C, \forall j \in v, i \neq j \quad (9)$$

$$0 \leq f_i \leq F \qquad \forall i \in v \quad (10)$$

$$q_j \leq q_i - o\, d_{ij}\, x_{ij} + Q(1 - x_{ij}) \qquad \forall i, j \in v, i \neq j \quad (11)$$

$$0 \leq q_j \leq q_i + w_i - o\, d_{ij}\, x_{ij} + Q(1 - x_{ij}) \qquad \forall i \in R, \forall j \in v, i \neq j \quad (12)$$

$$0 \leq q_i \leq Q \qquad \forall i \in v \qquad (13)$$

$$0 \leq q_i + w_i \leq Q \qquad \forall i \in R \qquad (14)$$

$$x_{ij} \in \{0.1\}; t_{ij}, d_{ij}, \tau_i, e_i, l_i, w_i, q_i, f_i, s_i, D_i \geq 0 \qquad \forall i, j \qquad (15)$$

$$e_i + s_i + t_{ij} + r\, w_j + t_{jk} > l_k \qquad i \in y \cup C, j \in R, k \in v \qquad (16)$$

$$e_i + s_i + t_{ij} > l_j \qquad i \in y \cup C, j \in v \qquad (17)$$

$$D_i + D_j > F \qquad i, j \in C \qquad (18)$$

$$o\ d_{ij} > Q \qquad i, j \in v \qquad (19)$$

La función objetivo (1) buscar minimizar el costo total, específicamente el factor de inversión por la compra de los vehículos y el costo de la distancia total recorrida por cada uno de ellos. La restricción (2) especifica que todos los clientes deben ser visitados al menos una vez, en (3) se define que las estaciones de carga y sus copias pueden o no ser visitadas, con (4) se asegura de que se puedan realizar recorridos en cualquier dirección, por lo que el número de arcos entrantes y salientes en cada nodo son iguales, la ecuación (6) establece que las rutas deben iniciar y finalizar en el depósito, además de que cada una debe realizarse por medio de un único vehículo.

Las ecuaciones (7)-(9) definen las restricciones de tiempo, la (6) hace referencia al tiempo de llegada al nodo j teniendo en cuenta el tiempo de viaje entre (i, j) y el tiempo de servicio en el nodo i, en (7) se vuelve a evaluar el mismo caso que en (6) pero ahora agregando el tiempo de recarga como el producto entre la cantidad de energía recargada y el ritmo de carga de la estación, y finalmente para (8) se garantiza que se cumplan las ventanas de tiempo planteadas por cada uno de los clientes.

Las restricciones (9) – (10). Hacen referencia a la capacidad de carga de mercancía en cada vehículo, en (9) se define que la cantidad de carga disponible en j va a ser reducida debido a la demanda del vértice i si el arco (i, j) es utilizado, mientras que en (10) se definen los límites de carga, de tal manera que no supere la cantidad máxima F y que se hagan entregas completas en cada visita a un nodo cliente.

En la restricciones (11) – (14) Se hace referencia a la capacidad de la batería en los vehículos, en (11), la energía con la que el vehículo llega al nodo j debe ser menor que aquella con la que salió de su punto de partida i dado el consumo de energía al recorrer el arco, en (12) se recrea el caso de (11) pero esta vez se tiene en cuenta la cantidad de energía que se recargo en i. En (13) se plantea que la carga de energía del vehículo al salir del depósito y las estaciones, debe estar siempre entre 0 y la capacidad Q de la batería, y en la ecuación (14) se garantiza que la suma de la batería del vehículo y la cantidad de energía cargada en i debe estar entre 0 y la capacidad Q de la batería.

La ecuación (15) define las variables y el dominio de cada una.

Para reducir el margen de error computacional al momento de resolver el problema, se añadieron 4 restricciones (16) -(19) cuyo objetivo es identificar arcos inviables para descartarlos inmediatamente, es decir, si el arco cumple con la restricción, será catalogado como un arco inviable.

Las restricciones (16) y (17) descartan los arcos que violen con las ventanas de tiempo, la ecuación (18) descarta un arco en el cual sea imposible satisfacer la demanda debido a la insuficiencia de la capacidad de carga de mercancía F, y finalmente en (19) se descartan los arcos en los que las distancias d_{ij} sean demasiado largas y la capacidad de la batería no sea suficiente para finalizar el recorrido.

4.3. Algoritmo colonia de hormigas

En esta sección se presenta el algoritmo colonia de hormigas modificado propuesto para la solución del EVRPTW-PR, el cual está basado en el sistema de colonia de hormigas propuesto originalmente por (Dorigo, 1996), y las variantes del ACO desarrolladas y aplicadas en los problemas de ruteo de vehículos en la literatura reciente, teniendo en cuenta las restricciones principales del EVRP como lo son la carga eléctrica del vehículo, la carga de mercancía del vehículo y las ventanas de tiempo en las cuales la hormiga puede visitar los nodos de clientes.

El algoritmo colonia de hormigas es una meta-heurística de la familia de inteligencia de enjambres usada para solucionar problemas computacionales de optimización, el cual simula el proceso de toma de decisiones que realizan las hormigas al momento de buscar alimento en lugares cercanos a su ubicación, por lo que dejan un rastro de feromonas que se refuerzan y se evaporan de tal manera en que la distancia total de la ruta se vaya reduciendo cada vez más.

Para describir el problema, es necesario aclarar el paralelo realizado entre el procedimiento de la colonia de hormigas y el VRP clásico. La entrada del nido de la colonia de hormigas será tomada como el nodo en donde se encuentra ubicado el depósito de la mercancía a entregar, la ubicación de los alimentos será tomada como la de los nodos en donde se encuentran los clientes, y finalmente, las hormigas que realizan las rutas son equivalentes a los vehículos que transportan la mercancía a entregar para luego retornar al depósito.

El procedimiento comienza una vez que la primera hormiga abandona el nido con una elección aleatoria entre las N ubicaciones de alimento más cercanas, donde $N = 1/n$, siendo n el número total de nodos de clientes del problema como sugieren los autores del artículo “Electric Vehicle Routing Problem with Recharging Stations for Minimizing Energy Consumption” (Zhang, Gajpal,

Appadoo, & Abdulkader, 2018). Esta hormiga realiza un recorrido desde el nido hasta el primer nodo de cliente para abastecer su demanda, pero en vez de regresar al depósito inmediatamente, sigue su camino hacia un próximo cliente o estación de carga, teniendo en cuenta que la elección se dará entre la N cantidad de nodos factibles de clientes más cercanos al nodo actual y las restricciones de tiempo, carga de mercancía y carga de energía.

Se realiza la misma operación hasta el momento en el que no se pueda satisfacer a ningún otro cliente debido a la falta de mercancía demandada, de batería, o la violación a las ventanas de tiempo, en éste punto la hormiga toma la decisión de finalizar el recorrido y retornar al depósito, reiniciando las condiciones iniciales del vehículo y del tiempo, agregando una unidad al contador de vehículos utilizados en el problema y registrando el total de distancia recorrida.

En ese instante, la hormiga vuelve a salir del depósito, realizando la misma operación que en el primer recorrido, pero con la diferencia de que ésta vez, no se realizará una elección entre todos los nodos de clientes, sino solamente entre los nodos que aún no han sido visitados; Ésta realiza su ruta con las mismas condiciones del recorrido anterior y regresa al depósito, esto se repite una y otra vez hasta que no existan más nodos de clientes a satisfacer, cerrando así una sub ruta, que se toma como la hormiga número 1.

Lo anterior, contando el número de hormigas que salieron del nido hasta completar todas las visitas, cuenta como la primera iteración, el primer resultado, y el primer conjunto de rutas de solución al problema; sin embargo, no se puede garantizar que este sea el resultado óptimo para el problema, se deberá repetir éste proceso realizando más iteraciones, con una cantidad calculada de hormigas.

Hay que tener en cuenta el hecho de que nuestra hormiga no representa un vehículo impulsado por combustibles fósiles, sino uno impulsado con energía renovable; por lo tanto, los desplazamientos entre nodos representan un gasto considerable de energía eléctrica, lo que conlleva a la necesidad de desplazarse a puntos de recarga entre las visitas a los respectivos nodos. Esto causa un incremento a la distancia total recorrida y el tiempo necesario para ejecutar completamente la ruta, afectando también los tiempos de entrega de mercancía a los clientes, reflejado en el cumplimento de las ventanas de tiempo, obligando al vehículo a considerar varios aspectos antes de elegir visitar a un cliente cualquiera.

Dentro de las consideraciones mencionadas anteriormente, se debe resaltar el instrumento de apoyo fundamental para garantizar que el comportamiento del algoritmo sea como el de las hormigas en la naturaleza, este es el uso de la técnica biológica de las hormigas para elegir el camino óptimo mediante un esfuerzo colectivo, usando feromonas como rastros de cada camino recorrido individualmente, con el fin de indicar cuantas veces ha sido utilizada una ruta anteriormente por hormigas de la colonia, provocando un aumento en la probabilidad de que una hormiga decida tomar el mismo camino que la anterior, evitando tomar rutas que puedan ser menos eficientes que las anteriores. Una vez obtenido un camino optimo local, este se marca y fortalece con una feromona más fuerte y duradera mediante una hormiga élite, quien cada vez que encuentra una mejor ruta, esparce nuevamente su feromona sobre ésta.

Como se mencionó anteriormente, el objetivo principal de esta investigación es construir una red de rutas factibles con el menor costo posible, en el caso del ACO, cada hormiga representa a un vehículo, y concentra su esfuerzo en la forma de escoger el próximo nodo a visitar; Esta selección está basada en la cantidad de feromona depositada en cada uno de los arcos recorridos y en una función de probabilidad para la toma de decisiones. Esto lleva a dos acciones posibles, que

la hormiga elija entre recorrer una ruta marcada como la mejor (menor costo) a causa de las feromonas, o que escoja un nuevo nodo candidato factible según una función de probabilidad basada en la cantidad de feromonas de una ruta. Para esto, es necesario expresar matemáticamente la cantidad de feromona depositada por una hormiga en cada recorrido, teniendo en cuenta que la cantidad será mayor entre más corta sea la ruta, es decir, la distancia recorrida es inversamente proporcional a la cantidad de feromona depositada como se puede apreciar en la siguiente ecuación(20):

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{1}{d_{ij}} \\ 0 \end{cases} \quad (20)$$

Donde $\Delta\tau_{ij}^{k}$ representa la cantidad de feromona depositada por la hormiga k en el arco ij, tomando el valor de $\frac{1}{d_{ij}}$ donde, como fue planteado en el modelo matemático, d_{ij} es el valor de la distancia recorrida en el arco ij, esto si el arco es recorrido, 0 en el caso contrario.

Para calcular y actualizar el valor de la cantidad de feromonas en cada uno de los arcos con el fin de que un arco no se vuelva extremadamente dominante sobre los otros transformándose en un obstáculo potencial para encontrar una solución óptima, se añade una sumatoria con el objetivo de crear una memoria de datos presentada con la siguiente ecuación (21):

$$\tau_{ij}^{k} = (1-\rho)\tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + \Delta\tau_{ij}^{e} \quad (21)$$

Donde τ_{ij}^{k} es la nueva cantidad de feromona en el arco ij, la cual es calculada como la suma entre los términos:

- $(1-\rho)\tau_{ij}$, que representa la cantidad de feromona evaporada en el arco como el producto de τ_{ij}(la cantidad de feromonas actual en el arco) y $[(1-\rho)]$(el complemento de la tasa de evaporación ρ, que toma un valor de 0 a 1 dependiendo del problema).
- $\sum_{k=1}^{m} \Delta\tau_{ij}^{k}$, que es la sumatoria de la cantidad de feromona que debe depositar cada una de la m cantidad de hormigas en cada uno de los arcos.
- $\Delta\tau_{ij}^{e}$, que define la cantidad de feromona depositada por la hormiga elite, la cual es actualizada cada vez que se encuentra un mejor resultado local por medio de la ecuación $\Delta\tau_{ij}^{e} = \frac{1}{L^{e}}$, donde L^{e} representa la distancia total de recorrido del mejor resultado local encontrado hasta ese momento.

Para hacer uso de los valores de feromona calculados anteriormente en la toma de decisiones, se simula el comportamiento de las hormigas ante los niveles de estas por medio de la siguiente función de probabilidad (22):

$$P_{ij} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum((\tau_{ij})^{\alpha}(\eta_{ij})^{\beta})} \quad (22)$$

En la cual se establece la probabilidad que tiene la hormiga k de elegir viajar desde el nodo i hacia el nodo j según la distribución de probabilidad entre todos los nodos factibles a visitar, obedeciendo la regla de proporción aleatoria (Fountas & Vlachos, 2005), que favorece los caminos de menor distancia por medio del alto nivel de feromona depositado en ese arco.

En esta ecuación, τ_{ij} representa la cantidad de feromona en el arco, y su nivel de importancia está definida por el escalar α; η_{ij} es un valor de importancia que se le da a una ruta corta como

decisión de la heurística, esta se calcula con $\eta_{ij} = \frac{1}{d_{ij}}$, y su nivel de importancia se define con el escalar β.

α y β son números mayores o iguales a 0 que representan la influencia relativa entre sí, y tienen como objetivo definir si en el algoritmo se le va a dar más importancia a la toma de decisiones respecto al nivel de feromona depositada o a la acción natural de la heurística de darle prioridad a viajar al nodo más cercano, por lo tanto si alguno de estos toma el valor de 0 se espera el siguiente resultado:

- $\alpha = 0$: La probabilidad de escoger el siguiente nodo a visitar se basa únicamente en la heurística.
- $\beta = 0$: La probabilidad de escoger el siguiente nodo a visitar se basa únicamente en la concentración de feromonas en el arco ij.

Para la solución de este problema se tomaron valores típicos de estos parámetros como se menciona en Bolaños, Correa, & Echeverri (2009), tomando el valor de α como 1 y 3 el de β, el cual se encuentra en el punto medio del rango sugerido entre 2 y 5.

4.4. Diseño del algoritmo

Para la elaboración del código se utilizó Python en su versión 3.6.5, se escogió como el lenguaje de programación ideal para esta investigación debido a que éste maneja una sintaxis bastante simple en su escritura, facilitando el uso para el usuario y aumentando su popularidad entre desarrolladores, razón por la cual cuenta con una gran cantidad de librerías de código abierto en la red. Como muestra, se usó Anaconda, una colección de librerías para análisis científico, y la escritura del código se hizo mediante el entorno de programación dinámica Jupyter Notebook, el

cual viene incluido en Anaconda, éste cuenta con una interfaz que permite crear, compartir y probar códigos en distintos lenguajes de programación, incluyendo Python entre las posibles opciones.

En cuanto al tipo de programación del algoritmo se usó el paradigma de la programación orientada a objetos ya que se realiza un simulación con objetos, llamados hormigas, los cuales cuentan con atributos como carga de mercancía, carga eléctrica, distancia recorrida y tiempo acumulado, los cuales presentan cambios dependiendo de los movimientos realizados , por lo cual el orden de ejecución de las acciones y los cambios de estado presentados son de gran importancia para la solución del problema. El algoritmo está descrito de la siguiente manera, primero son presentadas las suposiciones realizadas, seguidos por las clases[2] utilizadas para resolver el EVRPTW-PR.

Suposiciones:

- Los vehículos (hormigas) reciben carga de mercancía por parte del depósito, pero nunca se realiza la acción en sentido contrario.
- Cada hormiga cuenta con un atributo llamado carga de mercancía, cuyo valor es actualizado cada vez que un nodo es visitado.
- Un nodo de tipo cliente solo puede ser visitado por la hormiga si ésta cuenta con la carga de mercancía necesaria para satisfacer la demanda de éste.
- No hay gasto energético mientras transcurre la entrega de mercancía a los clientes, ya que la hormiga se encuentra estática

[2] En Programación Orientada a Objetos hace referencia a una plantilla de código de programa extensible para crear objetos, que proporciona valores iniciales para estado e implementaciones de comportamiento (como las funciones)

- No hay prioridad entre los nodos clientes que no han sido visitados.
- Los nodos que ya han sido visitados anteriormente son ignorados por la hormiga.
- Si la hormiga no contiene la suficiente carga de mercancía para satisfacer algún nodo cliente, ésta se devuelve al depósito a recargar su capacidad máxima.
- Una vez todos los nodos clientes han sido visitados la hormiga se devuelve automáticamente al depósito
- El punto de partida de la hormiga es el depósito.
- Los rastros de feromonas en cada arco son actualizados cada vez que es transitado.
- La hormiga élite no es contada como una de las hormigas que buscan rutas.

Aquí son presentadas 5 clases en el algoritmo con sus respectivas funciones, explicando en detalle su rol dentro del algoritmo

Clase Nodo:

Esta clase se debe inicializar dando las condiciones de cada uno de los nodos del problema, estableciendo las coordenadas en donde se encuentran, el tipo de nodo al que corresponde (depósito, estación de recarga o cliente); en caso de ser un nodo cliente, también se debe especificar la demanda de mercancía, las ventanas de tiempo entre las cuales debe ser visitado, y el tiempo de servicio requerido para que la demanda sea entregada, éstos datos son importados de las instancias del problema de EVRPTW de Schneider.

```
Clase nodo:
    def inicializar(coord, tipo, demanda, ventana, tServ):
        self.coord = coord
        self.flagV = False
        self.tipo = tipo
        Si self.tipo == 'c':
            self.demanda = demanda
            self.ventana = ventana
            self.tServicio = tServ
        Fin Si
        Si self.tipo == 'e':
            self.demanda = 0
            self.ventana = [0, infinito]
            self.tServicio = 0
        Fin Si

    def getCoord(self):
        retornar self.coord

    def getFlag(self):
        retornar self.flagV

    def setFlag(boolVal):
        self.flagV = boolVal

    def toString(self):
        retornar [self.coord, self.flagV]
```

Figura 1. Pseudocódigo Clase nodo.

Clase Mundo:

En esta clase se ingresan los valores de números de los nodos, la carga eléctrica de la batería del vehículo, la tasa de consumo, tasa de recarga y la velocidad de los vehículos, las estaciones, la demanda de los clientes, el costo de adquisición los vehículos a utilizar y el costo unitario de la distancia recorrida. Respecto a la longitud de los arcos, ésta es obtenida mediante una función llamada “función l” la cual calcula la distancia euclidiana entre dos puntos usando sus coordenadas, y lo retorna como una arista o arco entre ellos.

Con el fin de tener las condiciones ideales para dar inicio al recorrido de la primera sub-ruta, el nivel de feromona en todas las aristas es reiniciado con un valor predeterminado para evitar que se

presente alguna preferencia entre caminos, pues ningún arco ha sido recorrido en el momento inicial.

```
Clase mundo:
    def inicializar(nods, lfunc, cargE, tazCon, tazRec, vel, estacs, dem&&s, cosVeh, cosDis):
        self.nods = nods
        self.lfunc = lfunc
        self.estacs = estacs
        self.aristas = self.crearAristas()
        self.cargE = cargE
        self.tazRec = tazRec
        self.tazCon = tazCon
        self.vel = vel
        self.dem&&s = dem&&s
        self.cosVeh = cosVeh
        self.cosDis = cosDis

    def crearAristas(self):
        aristas = {}
        nods = self.nods + self.estacs
        Para m en nods:
            Para n en nods:
                a = m.getCoord()
                b = n.getCoord()
                Si a != b:
                    arista = arista(a, b, self.lfunc(a, b))
                    aristas[m, n] = arista
        Fin Para

        retornar aristas

    def reiniciarFeromona(nivel = 0.01):
        Para arista en self.aristas:
            arista.feromona = nivel
        Fin Para
```

Figura 2. Pseudocódigo Clase mundo.

Clase Arista:

Dado que las aristas como tal no cuentan con un sentido en el cual deben ser transitadas, es necesario especificar el sentido en el cual cada camino fue recorrido por el objeto hormiga, para esto un nodo es asignado como inicio y otro como fin, ésta clase también es inicializada con la longitud de la arista y el nivel de feromona que ésta presenta.

```
class arista:
    def inicializar(inicio, fin, longitud, feromona):
        self.inicio = inicio
        self.fin = fin
        self.longitud = longitud
        self.feromona = feromona
```

Figura 3. Pseudocódigo Clase arista.

Clase Solucionador:

Se ingresan los parámetros que se utilizarán para resolver las instancias, siendo éstos:

- alfa: el cual representa el factor de importancia relativa dado por la hormiga a la feromona al momento de escoger qué camino tomar,
- beta: representa el factor de importancia relativa dado por la hormiga a la longitud de las aristas (Importancia heurística),
- rho: el porcentaje de evaporación que presenta la feromona con respecto al tiempo.
- q: la cantidad de feromona depositada por una hormiga
- limit: el número de iteraciones a realizar.
- ant_count: número de hormigas a utilizar en cada iteración.

Debido a que el tiempo computacional es un factor importante en la solución del problema, se utiliza la librería de python “PyTicToc” para calcular el tiempo de computación que se utiliza en la búsqueda de la solución de cada instancia.

Los parámetros de las instancias a resolver son ingresados de forma manual:

- La carga máxima de mercancía que puede transportar un vehículo
- La carga máxima de energía eléctrica de la batería
- Las tasas de recarga y consumo de energía eléctrica
- La velocidad promedio del vehículo, el costo unitario asociado a la distancia recorrida
- El costo de adquisición por vehículo
- El número y nombre de la instancia a resolver es ingresado de forma manual

Las coordenadas de los nodos cliente, sus respectivas ventanas de tiempo y tiempos de servicio, son leídos de forma automática de un documento de Excel en donde se encuentran las instancias.

En ésta clase son registradas 11 funciones, excluyendo la función inicializar “_init_” (incluida en la clase solver), en donde se registran los atributos iniciales. La ejecución de la función “solve” resume el funcionamiento de ésta clase, esta comienza reiniciando la cantidad de feromona en todas las aristas, provocando que vuelvan al nivel predeterminado de esta como se mencionó anteriormente, a continuación se asigna un valor nulo al mejorGlobal, se crea la colonia de hormigas con el número de hormigas ingresado en “ant count” y para un valor de iteración “i” que se encuentre entre 1 y el valor límite de iteraciones , posteriormente se llama a la función que reinicia los datos de las hormigas en la colonia , y utilizando la función “antColonyOpt” las hormigas proceden a realizar la búsqueda de la mejor ruta de manera natural, estableciendo un valor para el “mejor local” de la iteración, luego de completar cada iteración se utiliza un condicional para evaluar si la distancia de la ruta asignada como mejor local es menor al valor establecido como “mejorGlobal”, en caso de que sea verdadero, el valor del mejor local pasa a ser asignado como mejorGlobal, y la hormiga élite procede a recorrer ésta ruta por medio de feromonas para intensificar el camino optimo, cabe aclarar que la ruta dada por el recorrido de la primera iteración es tomado automáticamente como mejor global dado que éste mejor local es el primer resultado, el cual se compara con un valor nulo; en caso de ser falso, se mantiene el valor del mejorGlobal de la iteración anterior.

```
def solve(mundo):

    mundo.reiniciar_feromona(self.t0)
    mejorGlobal = Nulo
    colonia = self.crearColonia(mundo)
    para i en rango(self.limite):
        self.reiniciarColonia(colonia)
        mejorLocal = self.antColonyOpt(colonia)
        Si mejorGlobal is Nulo or mejorLocal.distancia < mejorGlobal.distancia:
            mejorGlobal = mejorLocal
        Fin Si
        self.elite(mejorGlobal)
    Fin para
    retornar mejorGlobal
```

Figura 4. Pseudocódigo Función Solve de la Clase Solucionador.

- **Función Crear Colonia:** Si el valor del número de hormigas ingresado es menor a 1, es decir, no es establecido inicialmente, se utiliza un número de hormigas igual al número de nodos cliente en el problema, en caso de ser mayor, se utiliza el valor ingresado.
- **Función Reiniciar Colonia:** Luego de terminar cada iteración, es decir cada vez que todas las hormigas de la colonia hayan concluido su respectiva búsqueda de la mejor ruta posible, todas las hormigas son reiniciadas, sus atributos vuelven a sus valores preestablecidos.
- **Función AntColonyOpt:** Retorna la ruta óptima de la función soluciones colonia
- **Función Hormigas Alrededor:** Organiza a todas las hormigas en el lugar establecido como punto de origen
- **Función Hormigas Aleatorias.** Establece que el número de Hormigas debe ser igual al número de nodos del tipo cliente en el problema y les asigna ubicaciones aleatorias.
- **Función Soluciones (Hormigas).** Ésta función es utilizada con el objetivo de saber cuántos nodos han sido visitados por la hormiga, o en caso contrario saber cuántos no fueron visitados, utilizando dos contadores en un While establecido después de marcar todos los nodos con la bandera de No visitados cada vez que una hormiga comienza una búsqueda.

- **Función Act_local.** Actualiza el nivel de feromona presente en una arista, si ésta es viajada.
- **Función Act_Global.** Reinicia el nivel de feromona presente en todas las aristas del mundo al terminar una iteración
- **Función Elite Hormiga.** Remarca el mejor camino cada vez que es establecido un óptimo local, utilizando un rastro de feromona con un valor más alto que el normal.

Clase Hormiga:

Dentro de ésta clase se plasma la mayoría de las restricciones a tener en cuenta para resolver el problema, especificando que el objeto hormiga es considerado como un vehículo que recorrerá todos los nodos que requieren ser visitados, ateniéndose a las restricciones del modelo matemático planteado en esta investigación.

El objeto “Hormiga” es inicializado con sus atributos y valores especificados con los que se intentará resolver el problema, los valores de alfa y beta, que representan la prioridad que asigna la hormiga a la longitud de un arco y la prioridad que toma el rastro de feromona dejado anteriormente por otras hormigas en los arcos que ya han sido recorridos respectivamente, tomando el valor de 1 para alfa y 3 para beta como se mencionó anteriormente. Los atributos mercancía y energía son los principales para el correcto desarrollo de las condiciones del problema a tratar junto con los tiempos de llegada a los nodos de clientes.

```
class hormiga:
    def inicializar(alfa, beta, merc, energia, cosVeh, cosDis, inst):
        self.mundo = Nulo
        self.alfa = alfa
        self.beta = beta
        self.inicio = Nulo
        self.nodo2 = Nulo
        self.nodo3 = Nulo
        self.distancia = 0
        self.visitados = []
        self.noVisitados = []
        self.arcos = []
        self.merc = merc
        self.mercMax = merc
        self.energia = energia
        self.cosVeh = cosVeh
        self.cosDis = cosDis
        self.tiempo = 0
        self.cont = 0
```

Figura 5. Pseudocódigo de la función inicializar de la clase hormiga.

Luego de haber sido creado el objeto hormiga con sus respectivos atributos, se establecen las funciones que le permitirán cumplir con las restricciones que presenta el EVRPTW-PR.

Con el fin de reducir el tiempo de computación se utiliza la función “Selec14” la cual limita las opciones entre los nodos a los cuales la hormiga puede viajar, a una cuarta parte del número total de nodos tipo cliente, evitando malgastar tiempo evaluando opciones que no están próximas a su ubicación en el grafo.

Luego de especificar el espacio de los nodos más cercanos que aún no han sido visitados utilizando la función Selec14, se plantean las principales restricciones del problema, la carga de mercancía, la carga eléctrica del vehículo y las limitaciones de las ventanas de tiempo en las cuales los clientes pueden ser visitados.

Función Evaluar tiempo

Con el fin de cumplir las restricciones relacionadas con la factibilidad por tiempo del modelo matemático, se utiliza la función “EvalTiempo” la cual retorna 4 banderas con un valor booleano (verdadero o falso) teniendo en cuenta el tiempo requerido en el siguiente orden: primeramente, retorna:

- “FlagOntiempo”, la cual indica si la hormiga está a tiempo para visitar al siguiente cliente.
- “FlagSup” indica si el tiempo de llegada que tendría la hormiga al nodo es superior a la ventana superior de este.
- “FlagDep” señala si el vehículo debe ir de vuelta al depósito.
- “FlagN3” avisa si un siguiente cliente puede ser visitado después de la elección actual.

Para ésta función es necesario hacer un llamado a nodo2, distancia al nodo 2, nodo 3, el conjunto de todos los nodos y el tiempo de la hormiga, aparte de éstos también se usa el valor de la velocidad del vehículo “v” y se da un valor inicial a “tiempo” de 0, se pregunta si el nodo 2 es de tipo lista, si la respuesta es verdadera se extrae la ventana de tiempo inferior(vent20), la ventana de tiempo superior(vent21) y el tiempo de servicio(tServ2) del espacio ordenado de la lista, si se presenta el caso contrario y no es de tipo lista, se extraen éstos 3 datos y se asignan directamente.

```
def evalTiempo(nodo2, dist2, nodo3, nodos, tiempohormiga):
    v = self.mundo.velocity
    flagOntiempo = False
    flagSup = False
    flagDep = False
    flagN3 = False
    tiempo = 0
    Si tipo(nodo2) == lista:
        vent20 = nodo2[0].ventana[0]
        vent21 = nodo2[0].ventana[1]
        tServ2 = nodo2[0].tServicio
    Caso Contrario:
        vent20 = nodo2.ventana[0]
        vent21 = nodo2.ventana[1]
        tServ2 = nodo2.tServicio
    Fin Si
```

Figura 6. Pseudocódigo de la función EvalTiempo.

Se realiza el mismo procedimiento con el nodo 3, asegurándose antes, de que el valor del nodo 3 no se nulo, y se calcula la distancia euclidiana entre las coordenadas del nodo 2 y el nodo 3, con el fin de tenerla presente para futuras operaciones en donde sea necesaria, en caso de que el viaje entre los nodos 2 y 3 sea realizado.

```
Si nodo3 != Nulo:
    flagN3 = True
    Si tipo(nodo3) == lista:
        vent30 = nodo3[0].ventana[0]
        vent31 = nodo3[0].ventana[1]
        tServ3 = nodo3[0].tServicio
        Si tipo(nodo2) == lista:
            euc3 = euclidean(nodo2[0].getCoord(), nodo3[0].getCoord())
            dist3 = (self.mundo.cosVeh) + (self.mundo.cosDis * euc3)
        Caso Contrario:
            euc2 = euclidean(nodo2.getCoord(), nodo3[0].getCoord())
            dist3 = (self.mundo.cosVeh) + (self.mundo.cosDis * euc3)
        Fin Si
```

Figura 7. Continuación Pseudocódigo de la función EvalTiempo

Con respecto a las restricciones de tiempo, éstas son evaluadas teniendo en cuenta el tiempo “actual” de la hormiga y el tiempo necesario para realizar los desplazamientos entre nodos, este es calculado usando la distancia del desplazamiento y la velocidad promedio, se valora mediante un condicional con la ventana de tiempo inferior del nodo2, en caso de ser afirmativo, el vehículo espera en el nodo actual hasta que falte el tiempo necesario para realizar el viaje y llegar tan pronto sea posible a entregar la mercancía, la bandera FlagOntiempo toma un valor verdadero y el tiempo tomará el valor de la ventana inferior del nodo una vez se realice el viaje, donde posteriormente se sumará a éste el tiempo de servicio del nodo, y si el valor de flagN3 es verdadero, se realiza el mismo proceso para el nodo 3.

```
Si tiempo + (dist2/v) < vent20:
    tiempo += vent20 - (dist2/v) - tiempo
    flagOntiempo = True
    tiempo += tServ2
    Si flagN3:
        Si tiempo + (dist3/v) < vent30:
            tiempo += vent30 - (dist3/v) - tiempo
            flagN3 = True
            tiempo += tServ3
        Fin Si
```

Figura 8. Continuación Pseudocódigo de la función Evaltiempo.

De ésta misma manera se evalúa si la hormiga puede llegar a los clientes dentro de la ventana de tiempo propia de cada cliente.

```
Si tiempo + (dist2/v) > vent20 && tiempo + (dist2/v) < vent21:
```

Figura 9. Continuación Pseudocódigo de la función Evaltiempo.

Dado el caso de que el tiempo de la hormiga sumado con el tiempo necesario para realizar el desplazamiento hacia el nodo sea mayor a la ventana de tiempo de éste, la bandera FlagSup toma un valor verdadero y se pone a prueba si el tiempo de la hormiga ya es superior al tiempo máximo de llegada de la ventana de tiempo más tardía de la instancia a resolver, si éste condicional es verdadero, la bandera FlagDep toma un valor verdadero y la única opción de la hormiga es volver al depósito.

```
Si tiempo + (dist2/v) > vent21:
    flagSup = True
    flagOntiempo = False
    flagN3 = False
    tiempoMax = self.nodoMaxVent(nodos).ventana[1]
    Si tiempo > tiempoMax:
        flagDep = True
    Fin Si
Fin Si
```

Figura 10. Continuación Pseudocódigo de la función Evaltiempo.

Función recargar

Ésta función es llamada dentro de la función “EnergiaRst” y específicamente se encarga de evaluar si la cantidad de energía que presenta el vehículo es menor a la necesaria para ir al próximo nodo y a su respectiva estación de carga más cercana, si es esto es cierto, el auto debe recargar, la diferencia de energías inicialmente es tomada como el nivel máximo de la capacidad de la batería, si la energía necesaria para ir de la ubicación actual al siguiente nodo y su estación más cercana es menor al valor máximo, procede a preguntar si la energía necesaria para ir al próximo nodo, luego al siguiente de ese y a su respectiva estación de carga más cercana es menor al valor máximo, si esto es verdadero, procede a cargar ésta cantidad, en caso contrario carga solamente lo necesario para ir al nodo siguiente y a su respectiva estación de carga , en todos los casos es agregada una unidad extra de energía para evitar un posible desfase entre la energía del vehículo y la energía necesaria para realizar el recorrido que ocasione estancamiento del vehículo

```
def recargar(eA, distNAEst1, distN2, distN2Est2, distN3, distN3Est3):
    cRate = self.mundo.conRate
    Si eA < cRate * (distN2 + distN2Est2):
        dif = self.mundo.cargaE
        Si cRate * (distN2 + distN2Est2) < dif:
            Si cRate * (distN2 + distN3 + distN3Est3) < dif:
                eA = (cRate * (distN2 + distN3 + distN3Est3)) + 1
            Caso Contrario:
                eA = (cRate * (distN2 + distN2Est2)) + 1
            Fin Si
        Fin Si

        Si cRate * distNAEst1 < dif:
            eA = (cRate * distNAEst1) + 1
        Fin Si
    Fin Si

    retornar eA
```

Figura 11. Pseudocódigo función recargar.

Éstas recargas parciales aseguran que solamente sea cargado el nivel de energía necesario para que el vehículo ejecute los próximos movimientos de la forma más eficiente posible, esto implica permanecer menos tiempo en la estación de carga, lo que influye de manera positiva en la posibilidad de alcanzar a los clientes dentro de las ventanas de tiempo establecidas.

Función energía Restante:

La función Energía restante (energiaRst) evita que la hormiga se estanque a mitad del recorrido, al asegurar que el vehículo tenga la suficiente energía para entrar y salir de un nodo, es decir, llegar al cliente y luego ir a la estación de carga más cercana; utilizando los valores de la energía actual de la hormiga, la tasa de consumo por distancia del vehículo (cRate) y las distancias entre el nodo actual, los posibles nodos siguientes y las respectivas estaciones de carga más cercanas.

Asigna un valor de 0 al tiempo y, utiliza la bandera FlagEner para expresar si el vehículo tiene la energía suficiente para alcanzar el próximo nodo sin correr riesgo de quedar estancado, FlagN3 para denotar que el vehículo puede ir a los nodos 2 y 3 sin estancarse por cuestiones de energía y la FlagEst, la cual indica si el nivel de energía actual en el vehículo es mayor o menor al necesario para ir a la estación más cercana al nodo actual.

Dentro de ésta función se especifica que, si el tipo de nodo en el que se encuentra el vehículo es tipo “e” o “estación de carga “, es llamada la función recargar y el valor de la energía actual del vehículo aumenta, ésta función retorna las 3 banderas mencionadas anteriormente, los nodos 2 y 3, la energía actual del vehículo, el tiempo usado en la recarga y el vector de nodos cercanos obtenido de utilizar selec14 en el nodo2.

```
tiempo = 0
flagEner = False
flagN3 = False
flagEst = False

Si nodo1.tipo == 'e':
    eA = self.recargar(eA, distNAEst1, distN2, distN2Est2, distN3, distN3Est3)
Fin Si

Si eA > eNodo2:
    flagEner = True
    Si nodo3 != Nulo:
        Si eA > (distN2 * cRate) + eNE3:
            flagN3 = True
        Caso Contrario:
            flagN3 = False
        Fin Si
    Caso Contrario:
        flagN3 = False
    Fin Si
Fin Si

Si eA > eEst1:
    flagEst = True
Fin Si

retornar [flagEner, flagN3, flagEst, nodo2, nodo3, eA, tiempo, vec]
```

Figura 12. Pseudocódigo Energía Restante

Función mercancía restante:

Evalúa si la mercancía actual es suficiente para satisfacer la demanda del nodo siguiente a visitar, retornando el valor de la bandera carga (FlagCarga), como verdadero si es factible seguir haciendo el recorrido por mercancía o Falso si la mercancía es insuficiente.

```
def mercRest(nodoSig):
    Si self.merc >= nodoSig.demanda:
        flagCarga = True
    Caso Contrario:
        flagCarga = False
Fin Si

retornar flagCarga
```

Figura 13. Pseudocódigo Función Mercancía Restante.

Función Nodo Factible:

Se obtienen las coordenadas tanto del nodo 1 como del nodo 2, se revisa que el nodo 2 tenga un valor distinto a nulo y que no se encuentre entre la lista de los visitados. Una vez que se haya comprobado que se tiene la suficiente carga de mercancía mediante el valor de la bandera FlagCarga, se obtiene el valor de la distancia entre los nodos 1 y 2, se ubican las estaciones de carga más cercanas a éstas, utilizando la función lookcharge, es declarado el valor de EnergiaR, el cual es el resultado de la función energiaRst para el nodo 2 , “energia” toma el valor de la casilla número 5 del vector retornado de la función energiaRst, especificado como energía actual (eA), el valor de la casilla 6 es sumado al tiempo representando el lapso de recarga, el nodo 3 es establecido por el valor 4, nodos cerca viene de la casilla 7, en donde se encuentran todos los nodos cercanos sin visitar dado por la función Selec14, el tiempo es tomado de la función evalTiempo y es sumado el 4to valor del vector resultado de la función evalTiempo.

```
v = float(self.mundo.velocity)
recRate = float(self.mundo.recRate)
conRate = float(self.mundo.conRate)
cargaE = float(self.mundo.cargaE)

Si tipo(self.nodo()) == lista:
    coord1 = self.nodo()[0].getCoord()
Caso Contrario:
    coord1 = self.nodo().getCoord()
Fin Si

Si tipo(nodo2) == lista:
    coord2 = nodo2[0].getCoord()
Caso Contrario:
    coord2 = nodo2.getCoord()
Fin Si
tiempo = cop.deepcopy(self.tiempo)
energia = cop.deepcopy(self.energia)
flagGeneral = False
flagDep = False
flagEst = False
flagN3 = False

Si nodo2 != Nulo:
    Si tipo(nodo2) == lista:
        flagCarga = self.mercRest(nodo2[0])
    Caso Contrario:
        flagCarga = self.mercRest(nodo2)
    Fin Si
Caso Contrario:
    flagCarga = False
Fin Si
```

Figura 14. Pseudocódigo Nodo Factible.

Después de esto es evaluado si el recorrido del vehículo se encuentra dentro de las ventanas de tiempo establecidas, utilizando los valores del vector que retorna la función energiaRst con el fin de conocer los valores que tomarán las banderas que son declaradas al principio de la función, (FlagEst, FlagGeneral, FlagN3 y Flag Depot). Se establecen varias combinaciones entre los posibles valores de tiempoR[] para simular las posibles situaciones, si es posible realizar el siguiente movimiento, si el vehiculo solo puede viajar al nodo 2, o a los nodos 2 y 3 o quizás simplemente a la estación de carga más cercana.

```
Si flagCarga:
    dist1 = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coord2))
    estNodo1 = self.lookCharge(coord1)
    estNodo2 = self.lookCharge(coord2)
    energiaR = self.energiaRst(nodo2)
    energia = energiaR[5]
    tiempo += energiaR[6]
    nodo3 = energiaR[4]
    nodosCerca = energiaR[7]
    tiempoR = self.evalTiempo(nodo2, dist1, nodo3, nodosCerca, tiempo)
    tiempo += tiempoR[4]
    Si energiaR[0] && energiaR[1] && not energiaR[2]:
        flagEst = False
        Si tiempoR[0] && tiempoR[1] && not tiempoR[2] && not tiempoR[3]:
            flagGeneral = True
            flagN3 = True
            flagDep = False
        Fin Si
        Si tiempoR[0] && not tiempoR[1] && not tiempoR[2] && not tiempoR[3]:


            flagGeneral = True
            flagN3 = False
            flagDep = False
        Fin Si
        Si not tiempoR[0] && not tiempoR[1] && tiempoR[2] && not tiempoR[3]:
            flagGeneral = False
            flagN3 = False
            flagDep = False
        Fin Si
        Si not tiempoR[0] && not tiempoR[1] && tiempoR[2] && tiempoR[3]:
            flagGeneral = False
            flagN3 = False
            flagDep = True
        Fin Si
  Fin Si
```

Figura 15. Continuación Figura Nodo Factible.

De ésta función se obtienen los valores del vector Fact, los cuales son utilizados en la función move; éste vector consta de 8 casillas, siendo éstas:

- Fact [0] , Flag General, la cual señala si puede ejecutarse el viaje al nodo 2.
- Fact [1], flagN3, indica si puede realizar el viaje al nodo 2 y en seguida ir al nodo 3.
- Fact [2], flagEst, indica si debe pasar por una estación a recargar energía.
- Fact [3], flagDep, muestra si el vehículo debe volver al depósito por mercancía.
- Fact [4] tiempo.
- Fact [5] energía.
- Fact [6],estNodo1, la estación más cercana al nodo 1.
- Fact [7] nodo3.

```
retornar [flagGeneral, flagN3, flagEst, flagDep, tiempo, energia, estNodo1, nodo3]
```

Figura 16. Pseudocódigo vector retornado de la función Nodo factible

Es declarado el vector “EnergiaR”, el cual está compuesto por los valores retorno de la función EnergiaRst, aquí se revisa que el próximo nodo no se encuentre entre la lista de visitados.

Función Can Move:

Se inicia definiendo las banderas que informan si es necesario visitar una estación de carga y si es necesario ir al depósito a recargar mercancía (FlagEst y FlagMerc) ambas tomando un valor falso, la estación más cercana toma un valor Nulo y se declaran 3 vectores vacíos que representarán respectivamente la lista de nodos no factibles(NotF), la lista de nodos que no han sido visitados y se encuentran cerca (Selec) y la lista de todos los nodos que no han sido visitados hasta el momento(selec2).

```
def can_move():
    flagEst = False
    flagmerc = False
    estC = Nulo
    notF = []
    notF.vaciar()
    selec = []
    selec.vaciar()
    selec2 = []
    selec2.vaciar()
```

Figura 17. Pseudocódigo función Can Move parte 1.

Se evalúa si la carga de mercancía en el momento es mayor o igual al valor mínimo de la demanda de los clientes que no han sido visitados, si esto es verdadero, la bandera FlagMerc toma un valor verdadero , la lista de nodos factibles y cercanos es llenada con la función selec14 para el nodo evaluado y selec2 es establecido como una copia de selec, si la longitud del vector selec es mayor a 0, se crea el vector sel en donde se guardarán los nodos factibles de ésta evaluación, pero ésta estará inicialmente vacía.

```
Si self.merc >= min(self.noVisitados, key = lambda x : x.demanda).demanda:
    flagmerc = True
    selec = self.Selec14(self.nodo())
    selec2 = cop.deepcopy(selec)
    Si longitud(selec) > 0:
        sel = []
        sel.vaciar()
```

Figura 18. Pseudocódigo función Can Move parte 2.

Se evalúa si los nodos han sido visitados asignando una bandera (flagV) a los nodos pertenecientes a selec y se toman los valores retornados por el vector de la función NodoFactible en Fact.

```
Para i en rango(longitud(selec)):
    nod = selec[i]
    Si tipo(nod) == lista:
        fl = nod[0].flagV
    Caso Contrario:
        fl = nod.flagV
    Fin Si

    fact1 = self.nodoFactible(selec[i])
```

Figura 19. Pseudocódigo función Can Move parte 3.

Dependiendo de si cumplen las condiciones o no, los elementos de selec irán siendo agregados a alguno de los dos vectores, si es un valor factible será agregado a “sel” y si no es factible se agregará a “notF”. Se realizan varias pruebas y finalmente se retorna un vector con la lista de nodos factibles, la lista de no factibles, la bandera que indica si es necesario visitar una estación de carga, la bandera indicando la mercancía, la estación de carga más cercana y la lista de los nodos que faltan por visitar sin importar si son factibles o no.

```
retornar [sel, notF, flagEst, flagmerc, estC, selec2]
```

Figura 20. Pseudocódigo vector retornado por la función Can Move.

En caso tal que ningún nodo se factible se tornará “sel” como un espacio vacío.

Función Choose Move:

Si la longitud del vector de decisiones es asignada como 0, la función retorna un valor nulo, pero si ésta es asignada como 1, éste retorna una lista de todas las decisiones que se han tomado.

```
def choose_move(choices):
    Si longitud(choices) == 0:
        retornar Nulo
    Fin Si
    Si longitud(choices) == 1:
        Si tipo(choices[0]) == lista:
            retornar choices[0][0]
        Caso Contrario:
            retornar choices[0]
        Fin Si
    Fin Si
```

Figura 21. Pseudocódigo función choose move parte 1.

Con respecto a la toma de decisiones entre qué arista elegir, ésta se da teniendo en cuenta la función de probabilidad planteada anteriormente, la cual se pondera respecto al “peso” que maneja cada arista debido a la feromona, y la decisión de la heurística de buscar los nodos más cercanos.

Como podemos observar al final de la toma de decisión, a pesar de los valores asignados al peso en las aristas, la elección final de qué nodo visitar maneja un componente aleatorio proporcionado por la hormiga, aunque siempre evaluado entre las opciones factibles dadas por la función “move”.

```
weights = []
Para move en choices:
    Si tipo(self.nodo()) == lista:
        Si tipo(move) == lista:
            arista = self.mundo.aristas[self.nodo()[0], move[0]]
        Caso Contrario:
            arista = self.mundo.aristas[self.nodo()[0], move]
        Fin Si
    Caso Contrario:
        Si tipo(move) == lista:
            arista = self.mundo.aristas[self.nodo(), move[0]]
        Caso Contrario:
            arista = self.mundo.aristas[self.nodo(), move]
        Fin Si
    Fin Si
    weights.agregar(self.weigh(arista))
Fin Para

total = sum(weights)
cumdist = lista(itertools.accumulate(weights)) + [total]
choice = choices[bisect.bisect(cumdist, random.random() * total)]

retornar choice
```

Figura 22. Pseudocódigo función choose move parte 2.

Función Move:

Se inicializa con el resultado de la función “can_move”, teniendo presente la lista de factibles, aquí también se realizan combinaciones entre los valores del vector fact con el fin de establecer cuál arista se recorrerá y por consiguiente cuál cliente y qué cantidad de mercancía deberá retirarse del atributo mercancía debido a que cada cliente tiene una demanda específica de producto y si debe realizar viajes a estaciones cercanas o no, ésta función retorna una modificación en el tiempo.

```
def move(self):
    remaining = self.can_move()

choice = self.choose_move(remaining[0])
Si choice != Nulo:
    fact = self.nodoFactible(choice)

Fin Si
Si longitud(remaining[1]) > 0 && remaining[3]:
    choice = self.choose_move(remaining[1])
    fact = self.nodoFactible(choice)
Caso Contrario:
    aristaDep = self.insert_depot()
    retornar self
Fin Si

Si fact[0] && fact[1] && not fact[2] && not fact[3]:
    self.energia = fact[5]
    aristaN2 = self.make_move(choice)
    Si tipo(choice) == lista:
        self.merc -= choice[0].demanda
    Caso Contrario:
        self.merc -= choice.demanda
    Fin Si

    aristaN3 = self.make_move(fact[7])

    Si tipo(fact[7]) == lista:
        self.merc -= fact[7][0].demanda
    Caso Contrario:
        self.merc -= fact[7].demanda
    Fin Si
    retornar self
Fin Si

Si fact[0] && not fact[1] && not fact[2] && not fact[3]:
    self.energia = fact[5]
    aristaN2 = self.make_move(choice)
    Si tipo(choice) == lista:
        self.merc -= choice[0].demanda
    Caso Contrario:
        self.merc -= choice.demanda
```

Figura 23. Pseudocódigo función move.

[Continuación Figura 23]

```
    Fin Si
    retornar self
Fin Si

Si not fact[0] && not fact[1] && fact[2] && not fact[3]:
    self.energia = fact[5]
    aristaEst = self.make_move(fact[6])
    retornar self
Fin Si
Si not fact[0] && fact[2] && fact[3]:
    self.energia = fact[5]
    aristaEst = self.make_move(fact[6])
    retornar self
Fin Si
Si not fact[0] && not fact[2] && fact[3]:
    self.energia = fact[5]
    aristaDep = self.insert_depot()
    retornar self
Fin Si

self.tiempo = fact[4]
```

Figura 23. Pseudocódigo función move.

Función LookCharge:

Ésta función nos proporciona la estación de carga más cercana a la coordenada solicitada, las distancias de la coordenada a cada estación son agregadas al vector veDist y posteriormente se toma el índice de la estación con la menor distancia.

```
def lookCharge(coord):
    veDist = []
    Para i en rango(longitud(self.mundo.estaciones)):
        Si coord != self.mundo.estaciones[i].getCoord():
            veDist.agregar(euclidean(coord, self.mundo.estaciones[i].getCoord()))
        Fin Si
    Fin Para
    menor = min(veDist)
    minInd = veDist.index(menor)
    retornar [self.mundo.estaciones[minInd], menor]
```

Figura 24. Pseudocódigo función Lookcharge.

Función insertar depósito:

La función insert_depot cumple la función de enviar a la hormiga de vuelta al depósito una vez esto sea necesario, se empieza asignándole un valor de 0 al tiempo, se obtienen las coordenadas del depósito y del nodo número 1 usando la función getCoord, con éstas se calcula la distancia entre el nodo y el depósito con el fin de obtener el valor de la energía necesaria para que el vehículo vuelva al depósito sin problemas.

Si el nodo es tipo estación de carga, se evalúa si el nivel de energía es menor a la necesaria para ir al depósito y si ésta energía también es menor a la carga máxima de batería del problema, en caso de ser verdad, se calcula la diferencia entre la energía necesaria para ir al depósito enerDep y la energía actual, con el fin de recargar este faltante y agregar éste tiempo de carga al tiempo del recorrido , en caso de que sea contrario y la energía necesaria para ir al depósito sea mayor a la carga máxima de la batería, el vehículo debe buscar la estación de carga más cercana , utilizando la función look charge , dirigirse a la coordenada de la estación más cercana para recargar ahí y así poder volver a evaluar el nivel de carga del vehículo después de ejecutar ese desplazamiento.

```
Si nodo1.tipo == 'e':

    Si self.energia <= enerDep && enerDep <= self.mundo.cargaE:
        dSi = enerDep - self.energia
        tiempo += dif/self.mundo.recRate
        self.energia = enerDep
        self.make_move(Nulo)
    Fin Si

    Si self.energia <= enerDep && enerDep > self.mundo.cargaE:
        Est = self.look_charge(coord1)
        nodost = self.mundo.estaciones[Est[0]]
        coordEst = nodost.getCoord()
        distNE = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coordEst))
        enerEst = distNE * self.mundo.conRate
        Si self.energia <= enerEst && enerEst <= self.mundo.cargaE:
            dSi = enerEst - self.energia
            tiempo += dif/self.mundo.recRate
            self.energia = enerEst
            self.make_move(nodost)
            self.insert_depot()
        Fin Si
        Si self.energia >= enerEst:
            self.make_move(nodost)
            self.insert_depot()
```

Figura 25. Pseudocódigo función insertar depósito parte 1.

[Continuación Figura 25]

```
            Fin Si
        Fin Si
        Si self.energia >= enerDep:
            self.make_move(Nulo)
        Fin Si
    Fin Si
```

Figura 25. Pseudocódigo función insertar depósito parte 1.

En caso de encontrarse en un nodo tipo cliente , debe preguntarse si la energía actual es menor a la necesaria para llegar al depósito, si esto es afirmativo debe buscar la estación de carga más cercana al nodo, desplazarse hasta ella teniendo en cuenta la energía necesaria para alcanzar el punto de carga más cercano (enerEst), luego de realizar el desplazamiento, la hormiga vuelve a preguntarse la diferencia entre su energía actual y la energía necesaria para llegar al depósito , si ésta es mayor desde el principio, ejecuta el movimiento para ir directamente al depósito, pero si no lo es, recarga, suma el tiempo de recarga y luego ejecuta el movimiento hacia el depósito.

```
Si nodo1.tipo == 'c':

    Si self.energia <= enerDep:
        Est = self.look_charge(coord1)
        nodost = self.mundo.estaciones[Est[0]]
        coordEst = nodost.getCoord()
        distNE = (self.mundo.cosVeh) + (self.mundo.cosDis * euclidean(coord1, coordEst))
        enerEst = distNE * self.mundo.conRate
        Si self.energia <= enerEst && enerEst <= self.mundo.cargaE:
            dSi = enerEst - self.energia
            tiempo += dif/self.mundo.recRate
            self.energia = enerEst
            self.make_move(nodost)
            self.insert_depot()
        Fin Si
        Si self.energia >= enerEst:
            self.make_move(nodost)
            self.insert_depot()
        Fin Si
    Fin Si
    Si self.energia >= enerDep:
        self.make_move(Nulo)
    Fin Si
Fin Si

retornar self
```

Figura 26. Pseudocódigo insertar depósito parte 2.

Función movimientos Restantes:

Devuelve el número de clientes no visitados.

```
def movimientosRest(self):
    retornar self.noVisitados
```

Figura 27. Pseudocódigo función Movimientos restantes

Función Make Move:

La función empieza estableciendo un nodo como origen, si el destino es nulo se establece un destino como inicio, y éste destino se agrega a los nodos visitados, el tiempo se inicia en 0, le energía inicial es la carga máxima del vehículo y la mercancía también es la cantidad máxima de mercancía que el vehículo es capaz de transportar, si el origen es de tipo lista y el destino también, siempre y cuando estos dos sean distintos, se crea la arista, la cual va del punto origen al punto destino, en caso contrario que no se dé esto, la arista será nula, pues el vehículo no realizaría viaje alguno, si el valor de la arista es distinto al nulo y el tipo de arista es arista y diferente a hormiga, la arista es agregada a arcos y se retorna a sí misma.

En caso de que el destino sea de tipo lista, el destino es agregado a “visitados” y éste es marcado con la bandera de visitado (FlagV = True).

Si el origen y el destino son de tipo lista, las coordenadas del origen son asignadas como coordenadas 1 y las del destino son asignadas como coordenadas 2, si estas dos son diferentes, la arista es agregada al mundo(self.mundo.aristas [ori, dest] y a la lista de arcos, la distancia euclidiana entre estos dos puntos es dada por la función “euc” , la cual calcula el valor de ésta distancia utilizando sólo sus coordenadas, ésta distancia está presente en la función “dist” en donde es multiplicada por el costo unitario de la distancia y sumada con el costo de adquisición de los vehículos de forma local. De igual manera, ésta ecuación se encuentra presente en varias partes

del código y posteriormente todas éstas distancias serán agregadas a la distancia acumulada dada como self.distancia;

Fuera de usarse solamente en la función distancia, éste valor “dist” también es utilizado en el cálculo de la energía requerida para los desplazamientos en las aristas y por lo tanto representa una disminución en la cantidad de energía que presenta el vehículo, ésta cálculo de la energía es obtenido multiplicando éste valor de distancia por la tasa de consumo del vehículo.

Función Weight:

Cada arista presenta un peso relativo para le elección que realiza la hormiga, antes de comenzar la búsqueda, todas las aristas cuentan con un mismo peso, el peso pre está establecido como 1/la longitud de la arista, mientras se realiza la búsqueda de la mejor ruta, el peso de las aristas va cambiando dependiendo de las hormigas que transiten por la arista y por ende depositen feromona en ella, este es el peso post. la feromona depositada se rige por las ecuaciones planteadas al inicio de éste capítulo.

```
def weight(arista):
    pre = 1 / (arista.longitud)
    post = arista.feromona
    retornar post ** self.alfa * pre ** self.beta
```

Figura 28. Pseudocódigo función weight.

5. Diseño de experimentos

5.1. Definición de los parámetros del algoritmo colonia de hormigas

El desempeño del algoritmo de la metaheurística ACO depende de los siguientes factores:

- Número de hormigas (m)

- Tasa de evaporación de feromona (ρ)
- Número de iteraciones ($\lfloor$)

Estos serán evaluados en las instancias planteadas en el artículo “The Electric Vehicle Routing Problem with Time Windows and Recharging Stations” (Schneider, Stenger y Goeke; 2014), las cuales están divididas en 2 grupos: instancias pequeñas e instancias grandes, presentadas en el apéndice C, con la siguiente estructura:

- Instancias pequeñas: Poseen entre 5 y 15 nodos de clientes con un horizonte de planeación corto; es decir, las ventanas de tiempo son altamente restrictivas y en algunos casos de corto rango, entre la apertura y el cierre.
- Instancias grandes: Poseen 100 nodos de clientes con un horizonte de planeación corto.

Las anteriores instancias están divididas en 3 diferentes tipos según la distribución geográfica de los nodos para garantizar que el comportamiento en cada uno de ellos sea verificado apropiadamente, donde:

- Cluster (c): Es una distribución en la cual los nodos se encuentran agrupados entre si estratégicamente.
- Random (r): Es una distribución en la cual los nodos fueron ubicados aleatoriamente en el grafo.
- Random-Cluster (rc): Es una combinación de las anteriores distribuciones en la cual existen agrupaciones de nodos que son ubicadas aleatoriamente en el grafo.

Los valores de la tasa de evaporación de feromona a utilizar en cada uno de estos son establecidos de acuerdo al límite superior e inferior planteados en el diseño de experimentos

presentados por Michalis Mavrovouniotis y Shengxiang Yang (2013) en su artículo donde se toman como recomendación los valores de 0,2 y 0,8 para ρ. Por otro lado, para establecer los valores del número de hormigas y de iteraciones, se tiene un limitante que es el tiempo de computación, ya que, si se toma un valor muy elevado de hormigas o de iteraciones, el tiempo de procesamiento para cada una de las instancias se eleva en gran medida, impidiendo tomar múltiples resultados para realizar un diseño de experimentos efectivo. Siendo así, se optó por tomar dos valores factibles en tiempo computacional para los parámetros m y ρ, asignando 10 como un nivel bajo y 100 como un nivel alto teniendo en cuenta que cuando hay más de 100 hormigas con 100 iteraciones en las instancias grandes la solución deja de tener un tiempo computacional aceptable.

Para el caso de las instancias pequeñas se realizó una corrida preliminar para determinar si se conservaban las mismas condiciones que en el caso de las grandes. Para esto, se compararon los resultados obtenidos en 6 instancias aleatorias al utilizar 100 hormigas con 100 iteraciones y 1000 hormigas con 1000 iteraciones como se puede ver en la tabla 1, estos resultados se replicaron 5 veces y pueden ser consultados en el apéndice D.

Tabla 2.
Corridas preliminares de las instancias pequeñas.

Corridas	**Instancia**					
	c106-15	**r102-15**	**r103-10**	**r105-5**	**rc105-5**	**c103-15**
	Resultados (Vehículos-Distancia total recorrida en Km)					
m=100; ⌊=100; ρ=0,2	- 204	3 - 315	4 - 216	3 - 175	4 - 185	4 - 344
m=1000; ⌊=1000; ρ=0,2	3 - 204	3 - 315	4 - 216	3 - 175	4 - 185	4 - 330
m=100; ⌊=100; ρ=0,8	3 - 213	3 - 315	4 - 216	3 - 175	4 - 186	4 - 330
m=1000; ⌊=1000; ρ=0,8	3 - 213	3 - 315	4 - 216	3 - 175	4 - 185	4 - 330

Se puede evidenciar que, aunque se aumentara considerablemente el número de hormigas e iteraciones, la variación en los resultados es mínima o nula, por lo tanto, se decidió utilizar los niveles más bajos de los parámetros para realizar el diseño de experimentos de las instancias

pequeñas con el objetivo de reducir el tiempo computacional aprovechando que los niveles más altos no obtuvieron cambios relevantes en los resultados.

Tabla 3.
Configuración de los valores de los niveles de cada factor.

Factor	Nivel Bajo	Nivel Alto
Número de hormigas (m)	10	100
Tasa de evaporación (ρ)	0,2	0,8
Número de iteraciones (⌊)	10	100

5.2. Definición del diseño experimental

Para el desarrollo del diseño experimental se tomaron aleatoriamente 3 instancias de cada tipo de distribución geográfica para los grupos de instancias grandes y pequeñas, esto se debe a que el comportamiento de las clases cluster, random y random-cluster poseen un funcionamiento característico con propiedades inherentes, las cuales permiten realizar un estudio de comportamiento de cada una sin necesidad de evaluar la totalidad de instancias. En ese orden de ideas, se seleccionaron las siguientes:

Tabla 4.
Instancias a evaluar.

Dist. Geográfica	Instancias pequeñas
c	c101-5; c104-10; c106-15
r	r105-5; r103-10; r105-15
rc	rc108-5; rc108-10; rc103-15
c	c104; c102; c103
r	r107; r104; r102
rc	rc101; rc104; rc107

Para cada una de estas instancias, se realiza un diseño de experimentos factorial 2^3. Los 8 tratamientos a evaluar, fueron obtenidos usando el software estadístico Statgraphics Centurion y se presentan en la tabla 4.

Tabla 5.
Tratamientos a utilizar en el diseño experimental para las instancias pequeñas.

Número de hormigas (m)	**Tasa de evaporación (ρ)**	**Número de iteraciones (ļ)**
-1	-1	-1
1	-1	-1
-1	1	-1
1	1	-1
-1	-1	1
1	-1	1
-1	1	1
1	1	1

Debido al inaceptable tiempo de computación para las instancias grandes con más de 100 hormigas y 100 iteraciones, se decidió excluir los 2 tratamientos con ese valor de parámetros ya que dificultaban en gran medida el desarrollo del diseño de experimentos. Teniendo en cuenta que el kernel[3] de Python presentó problemas a la hora de procesar tanta información, sumándole la pérdida de información a causa de estos y los extensos tiempos de espera, se optó por realizar un diseño factorial 2^3 fraccionado excluyendo estos 2 tratamientos y tomando los otros 6 que presentan características viables para realizar un diseño de experimentos efectivo. La tabla 5 presenta la organización de tratamientos para las instancias grandes:

[3] Kernel: Motor computacional para los lenguajes de programación que funciona como núcleo informático para ejecutar los códigos contenidos en las celdas de las librerías.

Tabla 6.
Tratamientos a utilizar en el diseño experimental para las instancias grandes.

Número de hormigas (m)	Tasa de evaporación (ρ)	Número de iteraciones (⌊)
1	-1	-1
1	1	-1
-1	-1	-1
-1	1	-1
-1	-1	1
-1	1	1

5.3. Resultado del diseño experimental

Es importante mencionar que cada uno de los valores obtenidos para evaluar en los tratamientos fueron replicados 5 veces para lograr alcanzar el mejor valor de la distancia total recorrida y el número de vehículos utilizados, aumentando la efectividad y veracidad del diseño.

Para definir cuál fue el mejor resultado obtenido en cada instancia, se le da prioridad al número de vehículos utilizados y después a la distancia total recorrida como factor de desempate, ya que el costo de los vehículos es mucho más elevado que el costo de kilómetros recorridos. Para lograr este resultado, es necesario realizar dos diseños experimentales diferentes, uno para el número de vehículos, y otro para la distancia total recorrida respectivamente.

Los resultados de las 5 réplicas para cada uno de los datos para el diseño experimental pueden ser consultados en los apéndice E.

A continuación, se presentan las características del equipo de cómputo mediante el cual se ejecutó el programa y los respectivos resultados obtenidos.

Tabla 7.
Características del equipo de cómputo.

Procesador	Intel Core i5 – 3.4[GHz]
Memoria RAM	8 GB
Sistema operativo	Windows 10 - 64 bits
Versión de Python	Python 3.7.1

Tabla 8.
Resultado del diseño experimental para las instancias grandes (Distancia total recorrida).

Corridas			**Valor promedio de la función objetivo en kilómetros (Distancia total recorrida)**								
m	**ρ**	**l**		**c**			**r**			**rc**	
			c102	**c103**	**c104**	**r102**	**r107**	**r104**	**rc101**	**rc104**	**rc107**
1	-1	-1	1221	1220	1269	1174	1392	1334	1754	1654	1780
1	1	-1	1477	1251	1267	**1122**	1386	1346	1782	1706	1784
-1	-1	-1	1658	1520	1510	1557	1543	1492	1879	1872	2103
-1	1	-1	1710	1643	1597	1435	1657	1501	1916	1891	1998
-1	-1	1	**1109**	**1145**	1175	1151	**1311**	1256	1643	1671	**1699**
-1	1	1	1187	1207	**978**	1442	1486	**1254**	**1609**	**1586**	1705

Tabla 9.
Resultado del diseño experimental para las instancias grandes (Vehículos utilizados).

Corridas			**Valor promedio de la función objetivo en vehículos (número de vehículos)**								
m	**ρ**	**l**		**c**			**r**			**rc**	
			c102	**c103**	**c104**	**r102**	**r107**	**r104**	**rc101**	**rc104**	**rc107**
1	-1	-1	13	15	12	18	14	16	17	**12**	18
1	1	-1	13	15	14	**17**	**12**	**11**	17	13	13
-1	-1	-1	15	16	15	20	20	18	19	15	16
-1	1	-1	17	17	15	20	21	18	21	17	17
-1	-1	1	**11**	**14**	**8**	19	15	14	**15**	**12**	**11**
-1	1	1	13	15	14	**17**	18	12	16	14	12

Tabla 10.
Resultado del diseño experimental para las instancias pequeñas (Distancia total recorrida).

Corridas			**Valor promedio de la función objetivo en kilómetros (Distancia total recorrida)**								
m	**ρ**	**l**		**c**			**r**			**rc**	
			c101-5	**c104-10**	**c106-15**	**r105-5**	**r103-10**	**r105-15**	**rc108-5**	**rc108-10**	**rc103-15**
1	-1	-1	**198**	262	225	**175**	218	296	342	**322**	251
1	1	-1	**198**	262	232	177	222	304	342	**322**	255
-1	-1	-1	**198**	270	233	177	232	312	345	350	317
-1	1	-1	**198**	263	244	177	218	312	345	350	322
-1	-1	1	**198**	262	219	177	**216**	303	342	**322**	**218**
-1	1	1	**198**	263	211	**175**	**216**	304	342	**322**	267
1	1	1	**198**	**260**	213	**175**	**216**	**295**	**341**	344	240
1	-1	1	**198**	**260**	**204**	**175**	**216**	**295**	**341**	344	232

Tabla 11.
Resultado del diseño experimental para las instancias pequeñas (Vehículos utilizados).

Corridas			**Valor promedio de la función objetivo en vehículos (número de vehículos)**								
m	**ρ**	**l**	**c**			**r**			**rc**		
			c101-5	**c104-10**	**c106-15**	**r105-5**	**r103-10**	**r105-15**	**rc108-5**	**rc108-10**	**rc103-15**
1	-1	-1	**3**	**3**	**3**	**3**	**4**	4	**4**	4	4
1	1	-1	**3**	**3**	**3**	**3**	**4**	**3**	**4**	4	3
-1	-1	-1	**3**	**3**	**3**	**3**	**4**	4	**4**	4	5
-1	1	-1	**3**	**3**	**3**	**3**	**4**	**3**	**4**	4	5
-1	-1	1	**3**	**3**	**3**	**3**	**4**	4	**4**	**3**	**2**
-1	1	1	**3**	**3**	**3**	**3**	**4**	**3**	**4**	4	4
1	1	1	**3**	**3**	**3**	**3**	**4**	4	**4**	**3**	3
1	-1	1	**3**	**3**	**3**	**3**	**4**	4	**4**	**3**	**2**

En las anteriores tablas se presentan los resultados obtenidos para cada instancia con cada uno de los tratamientos planteados. Es así como para cada una de éstas, existe una combinación de parámetros que logra ofrecer el menor valor para la minimización de la función objetivo; estos resaltados se encuentran resaltados con el color rojo.

5.4. Análisis del diseño experimental

El modelo de regresión correspondiente al diseño de experimentos se expresa a continuación:

$$y_1 = \beta_0 + \beta_1 x_1 + \beta_1 x_2 + \beta_1 x_3 + \varepsilon$$
$$y_2 = \beta_0 + \beta_1 x_1 + \beta_1 x_2 + \beta_1 x_3 + \varepsilon$$

Donde:

y_1 Es la distancia total recorrida esperada.

y_2 Es número de vehículos a utilizar esperado.

β_0 Es el efecto medio global.

$\beta_1 x_1$ Es el efecto producido por el parámetro A, es decir el número de hormigas.

$\beta_1 x_2$ Es el efecto producido por el parámetro B, es decir la tasa de evaporación.

$\beta_1 x_3$ Es el efecto producido por el parámetro C, es decir el número de iteraciones.

ε Es el error aleatorio.

Teniendo los resultados del diseño experimental, en el cual se realizaron ocho tratamientos para las instancias pequeñas, 6 tratamientos para las grandes, y adicionalmente cinco réplicas para los obtener los datos de cada uno de estos, se logró identificar los efectos de cada parámetro sobre la función objetivo de cada una de las instancias.

Tabla 12.
Efectos estimados para la función objetivo en instancias grandes (y_1).

Factores	**Efecto estimado (Instancias grandes - Distancia total)**								
	c			**r**			**rc**		
	c102	**c103**	**c104**	**r102**	**r107**	**r104**	**rc101**	**rc104**	**rc108**
Número de hormigas (m)	-15075	-15570	-12848	-15660	-4995	-7042,5	-5827,5	-9067,5	-12083
Tasa de evaporación (ρ)	38,6	21,6	-11,2	11,7	20,3	-18,1	3,1	-1,4	-9,5
Número de iteraciones (l)	-24120	-18248	-21465	-8977,5	-7267,5	-15368	-12218	-11385	-15683

Tabla 13.
Efectos estimados para la función objetivo en instancias grandes (y_2).

Factores	Efecto estimado (Instancias grandes - Número de vehículos utilizados)								
	c			**r**			**rc**		
	c102	**c103**	**c104**	**r102**	**r107**	**r104**	**rc101**	**rc104**	**rc108**
Número de hormigas (m)	-135	-67,5	-90	-112,5	-337,5	-202,5	-135	-157,5	-45
Tasa de evaporación (ρ)	0,4	0,2	0,8	-0,3	0,2	-0,7	0,3	0,5	-0,3
Número de iteraciones (l)	-180	-90	-180	-90	-180	-225	-202,5	-135	-225

Tabla 14.
Efectos estimados para la función objetivo en instancias pequeñas (y_1).

Factores	Efecto estimado (Instancias pequeñas - Distancia total)								
	c			**r**			**rc**		
	c101-5	**c104-10**	**c106-15**	**r105-5**	**r103-10**	**r105-15**	**rc108-5**	**rc108-10**	**rc103-15**
Número de hormigas (m)	0	-3,5	-8,25	-1	-2,5	-10,25	-2	-3	-36,5
Tasa de evaporación (ρ)	0	-1	4,75	0	-2,5	2,25	0	0	16,5
Número de iteraciones (l)	0	-3	-21,75	-1	-6,5	-6,75	-2	-3	-47

Tabla 15.
Efectos estimados para la función objetivo en instancias pequeñas (y_2).

Factores	Efecto estimado (Instancias pequeñas - Número de vehículos utilizados)								
	c			**r**			**rc**		
	c101-5	**c104-10**	**c106-15**	**r105-5**	**r103-10**	**r105-15**	**rc108-5**	**rc108-10**	**rc103-15**
Número de hormigas (m)	0	0	0	0	0	0,25	0	-0,25	-1
Tasa de evaporación (ρ)	0	0	0	0	0	-0,75	0	0,25	0,5
Número de iteraciones (l)	0	0	0	0	0	0,25	0	-0,75	-1,5

A partir de estos resultados, debe tenerse presente que si el valor del efecto es positivo significa que, al pasar del nivel bajo al nivel alto, el valor de la función objetivo va a aumentar, lo que quiere decir que la distancia total del recorrido, o el número de vehículos necesarios para completar el viaje va a ser mayor. Es así como, dependiendo de la instancia a estudiar, se busca establecer qué factores deben estar en nivel bajo y qué factores en nivel alto.

Así mismo, a continuación, se presentan las gráficas de Pareto estandarizadas con el objetivo de identificar los parámetros que tienen una mayor influencia sobre la función objetivo.

5.5. Diagramas de Pareto para las instancias grandes (Distancia total)

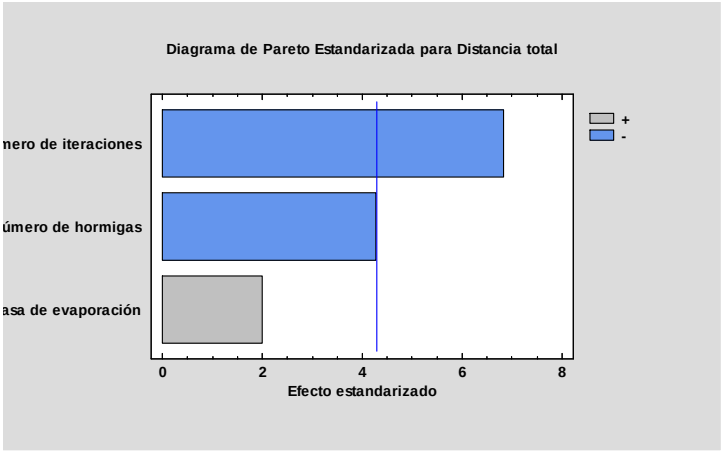


Figura 29. Diagrama de Pareto para la instancia c102 (Distancia total).

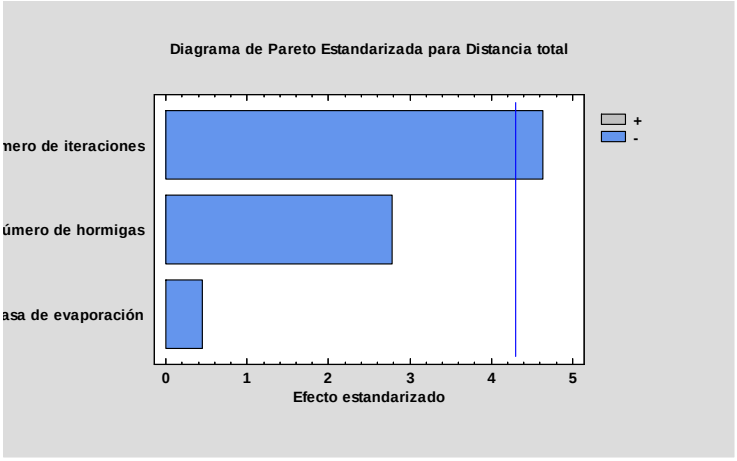


Figura 30. Diagrama de Pareto para la instancia c104 (Distancia total).

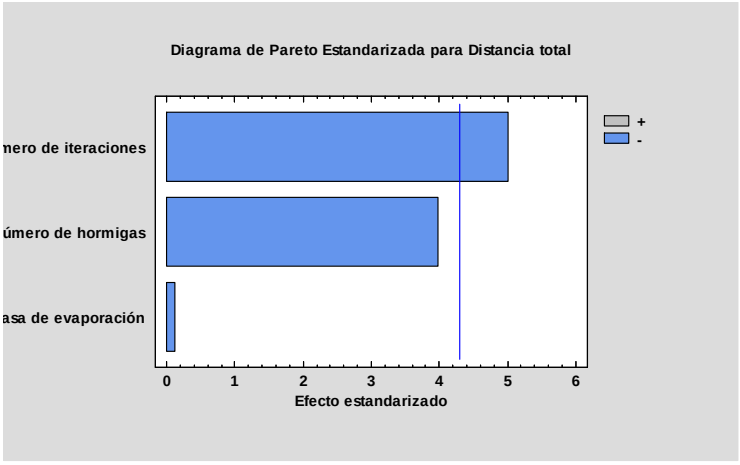


Figura 31. Diagrama de Pareto para la instancia rc104 (Distancia total).

Como se indica las figuras 29, 30 y 31, en las instancias c102, c104 y rc104 se tiene que el parámetro que ejerce un efecto significativo es el número de iteraciones, el cual genera una influencia negativa lo que indica que debe ubicarse en el nivel alto.

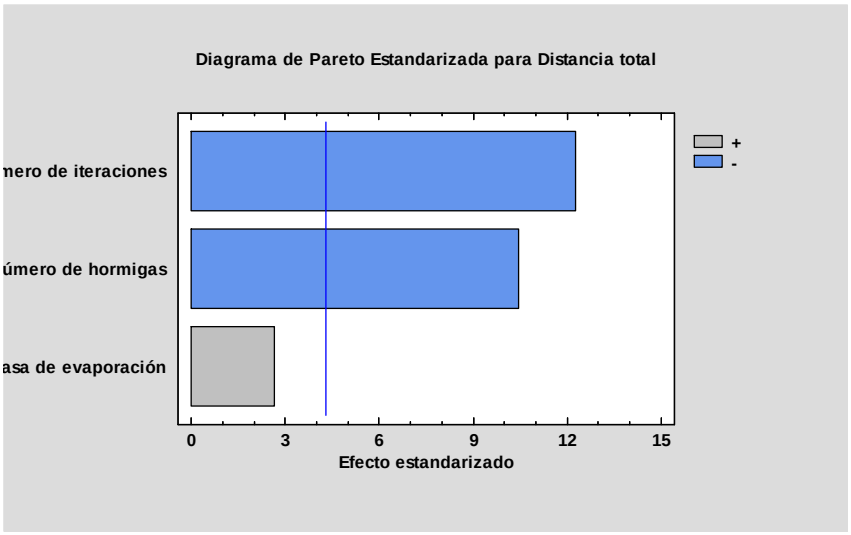


Figura 32. Diagrama de Pareto para la instancia c103 (Distancia total).

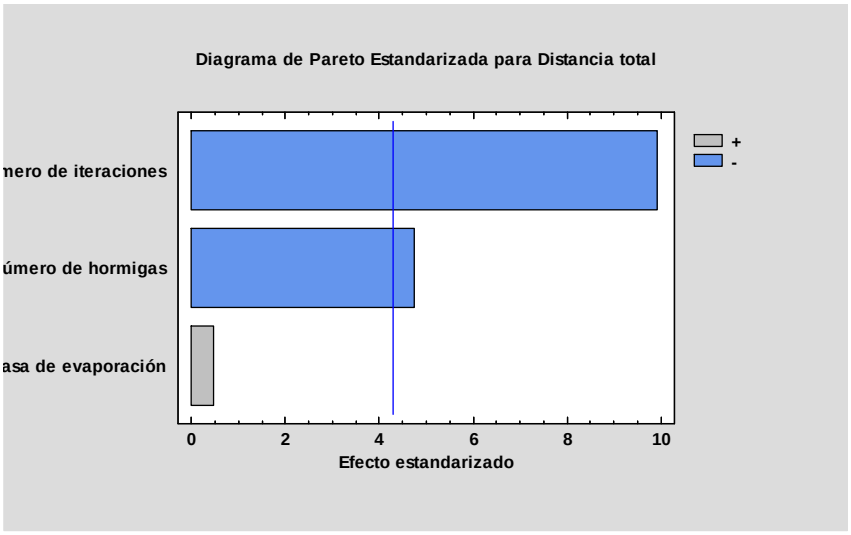


Figura 33. Diagrama de Pareto para la instancia rc101 (Distancia total)

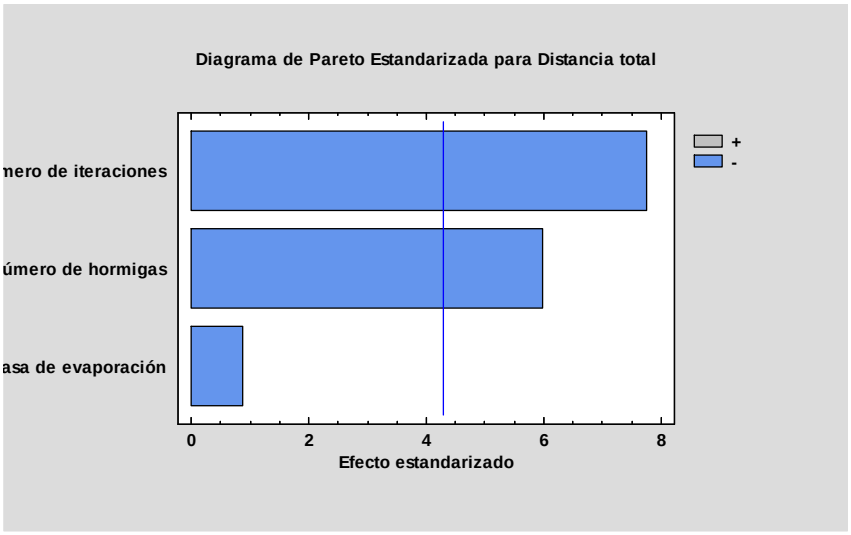


Figura 34. Diagrama de Pareto para la instancia rc107 (Distancia total).

Para las instancias c103, rc101 y rc107 existen 2 parámetros que ejercen un efecto significativo en la variable respuesta como se ve en las figuras 32, 33 y 34, estos son el número de iteraciones y el número de hormigas, los cuales generan una influencia negativa, lo que indica que deben ubicarse en el nivel alto.

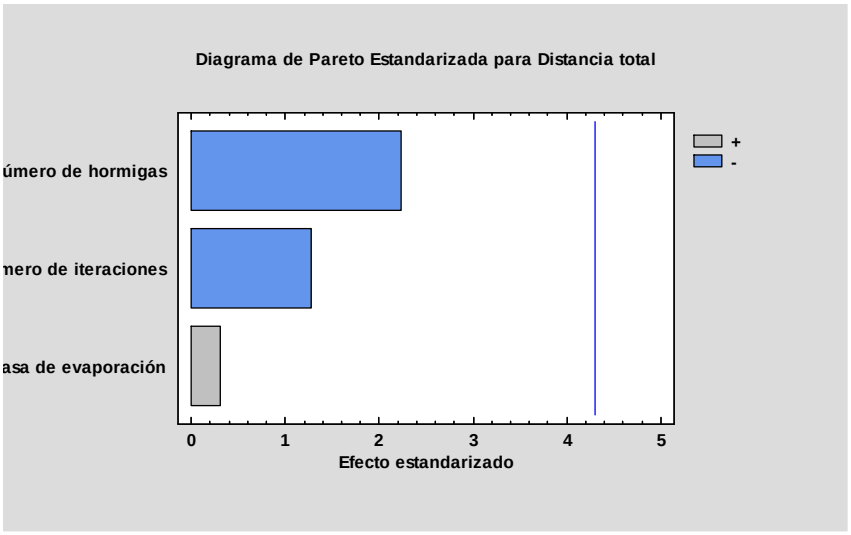


Figura 35. Diagrama de Pareto para la instancia r102 (Distancia total).

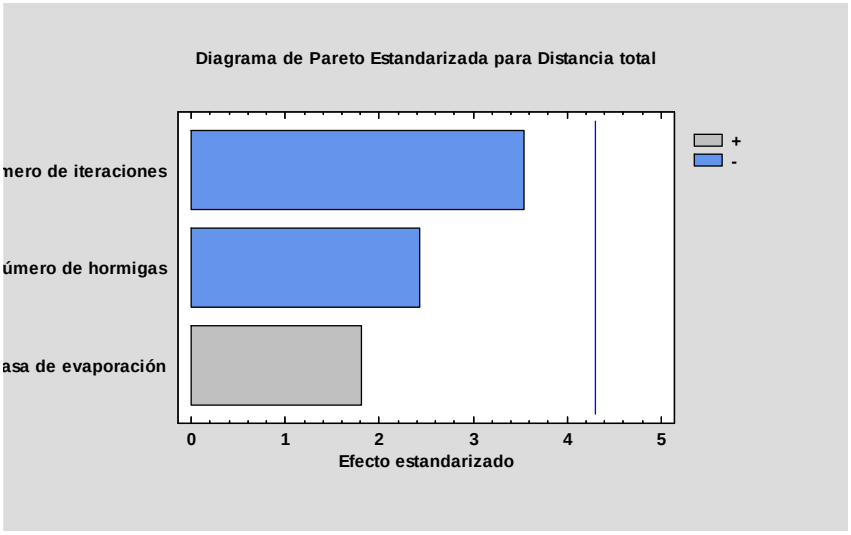


Figura 36. Diagrama de Pareto para la instancia r107 (Distancia total).

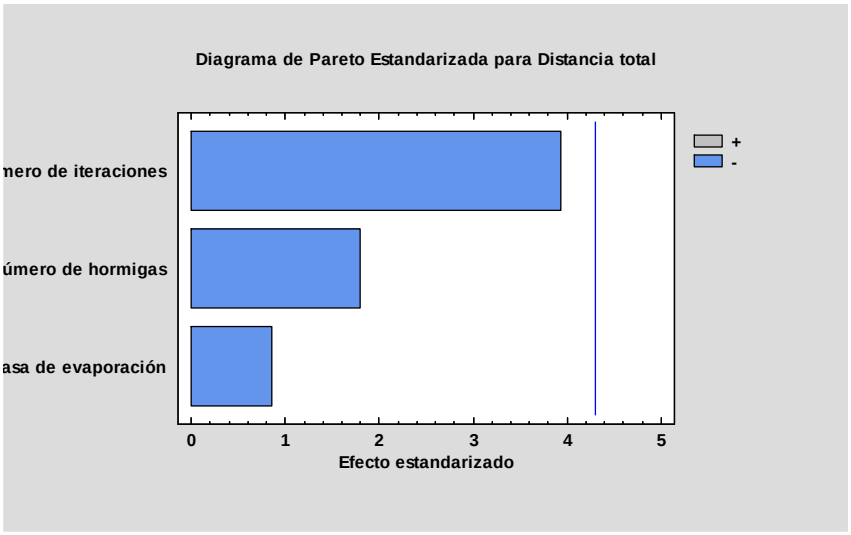


Figura 37. Diagrama de Pareto para la instancia r104 (Distancia total).

De las figuras 35, 36 y 37, se puede concluir que ningún parámetro incide en la variable de respuesta para las instancias r102, r107 y r104, ya que el valor de significancia para todos estos está por encima de 0.05.

5.6. Diagramas de Pareto para las instancias grandes (Vehículos utilizados)

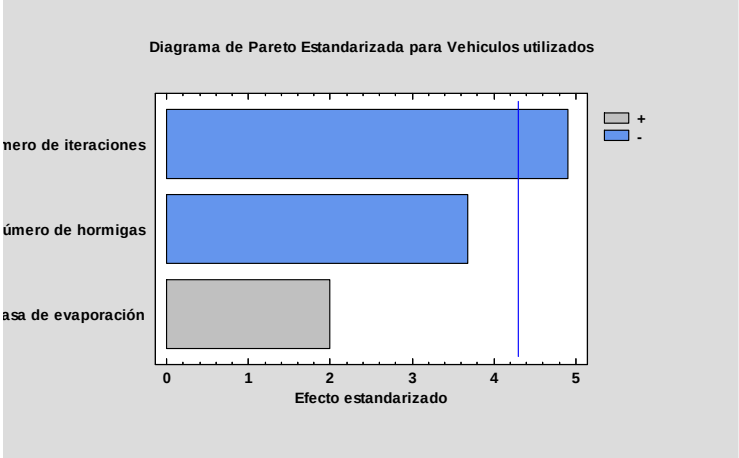


Figura 38. Diagrama de Pareto para la instancia c102 (Vehículos utilizados).

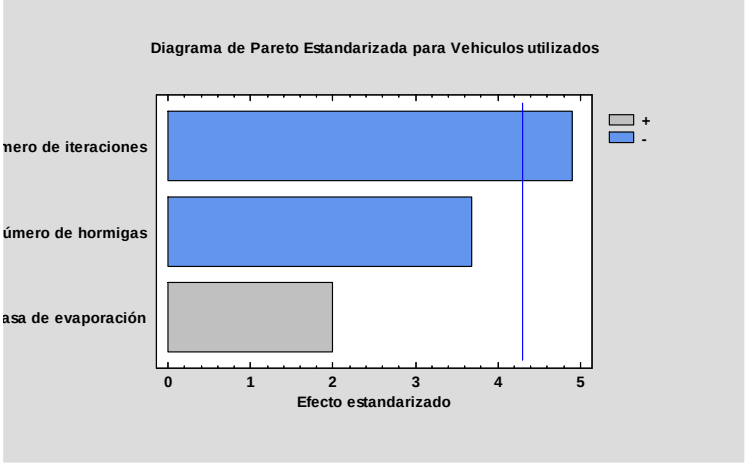


Figura 39. Diagrama de Pareto para la instancia c103 (Vehículos utilizados).

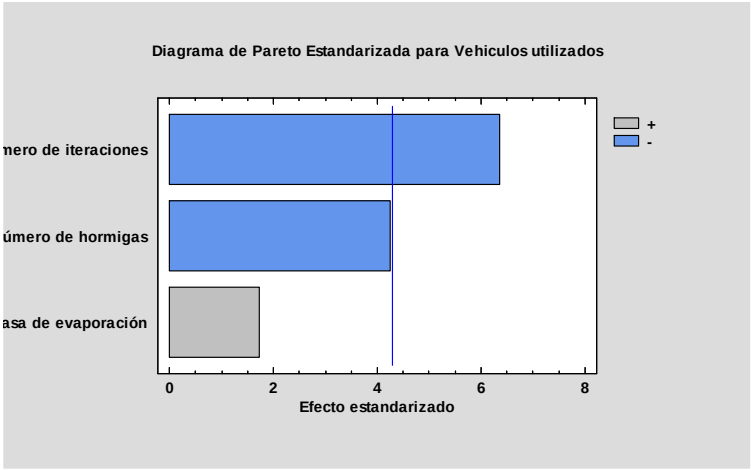


Figura 40. Diagrama de Pareto para la instancia rc101 (Vehículos utilizados).

Para las instancias c102, c103 y rc101 se tiene que el número de iteraciones es el parámetro que ejerce un efecto significativo en la variable de respuesta, generando una influencia negativa como se expresa en las figuras 38, 39, 40, lo que indica que debe ubicarse en el nivel alto.

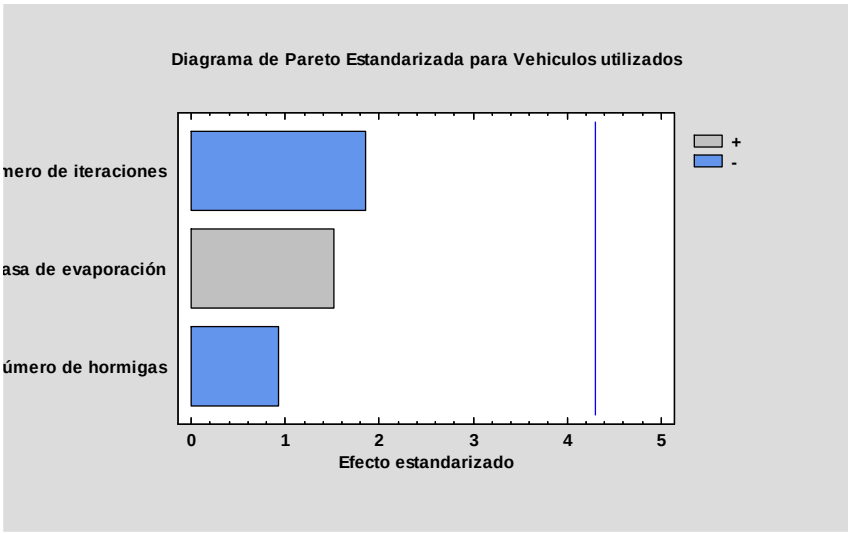


Figura 41. Diagrama de Pareto para la instancia c104 (Vehículos utilizados).

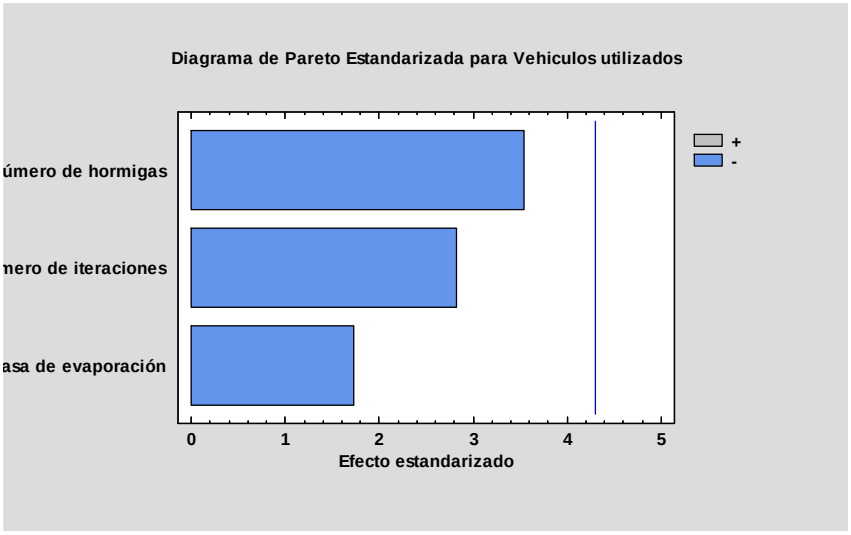


Figura 42. Diagrama de Pareto para la instancia r102 (Vehículos utilizados).

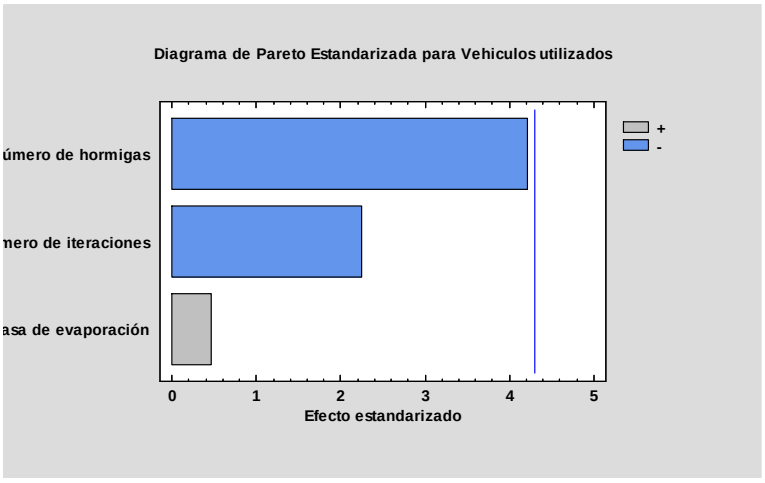


Figura 43. Diagrama de Pareto para la instancia r107 (Vehículos utilizados).

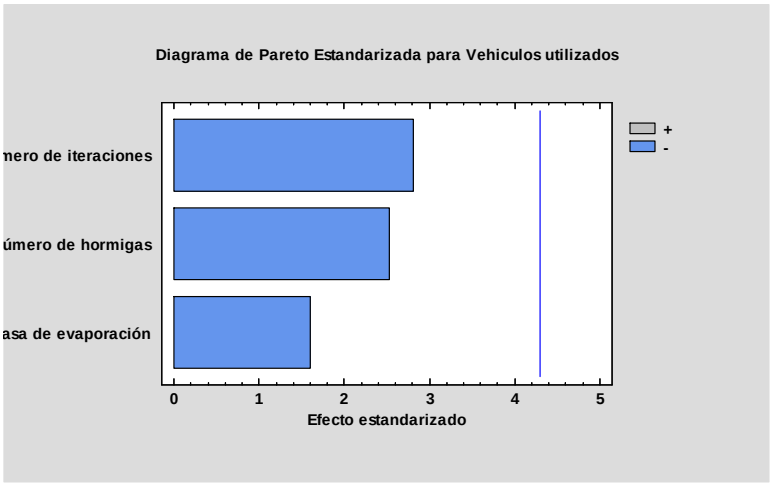


Figura 44. Diagrama de Pareto para la instancia r104 (Vehículos utilizados).

Figura 45. Diagrama de Pareto para la instancia rc107 (Vehículos utilizados).

En base a las figuras 41, 42, 43, 44, y 45 se concluye que para las instancias c104, r102, r107, r104 y rc107 los parámetros planteados no inciden en la variable de respuesta, ya que el valor de significancia para todos está por encima de 0.05.

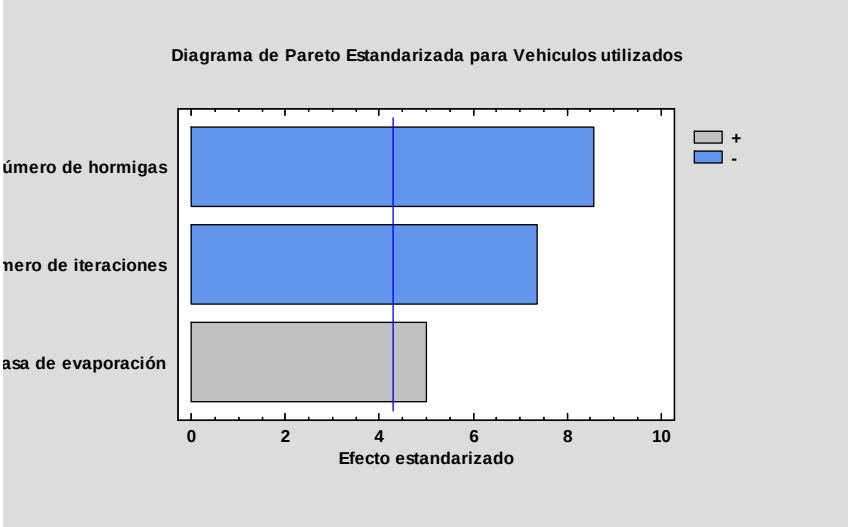


Figura 46. Diagrama de Pareto para la instancia rc104 (Vehículos utilizados).

Para la instancia rc104 se tiene que todos los parámetros ejercen un efecto significativo en la minimización de los vehículos a utilizar, donde el número de hormigas y el número de iteraciones generan una influencia negativa, indicando así que deben ser ubicados en el nivel alto, caso contrario de la tasa de evaporación, la cual genera un efecto positivo y debe ser ubicada en el nivel bajo.

5.7. Diagramas de Pareto para las instancias pequeñas (Distancia total)

Para la instancia c105-5 fue imposible realizar un análisis de parámetros por medio de un diagrama de Pareto, ya que en los 8 tratamientos realizados todas las respuestas fueron exactamente iguales, por lo tanto, es imposible determinar cuál de los parámetros ejerció una mayor influencia en la obtención del resultado.

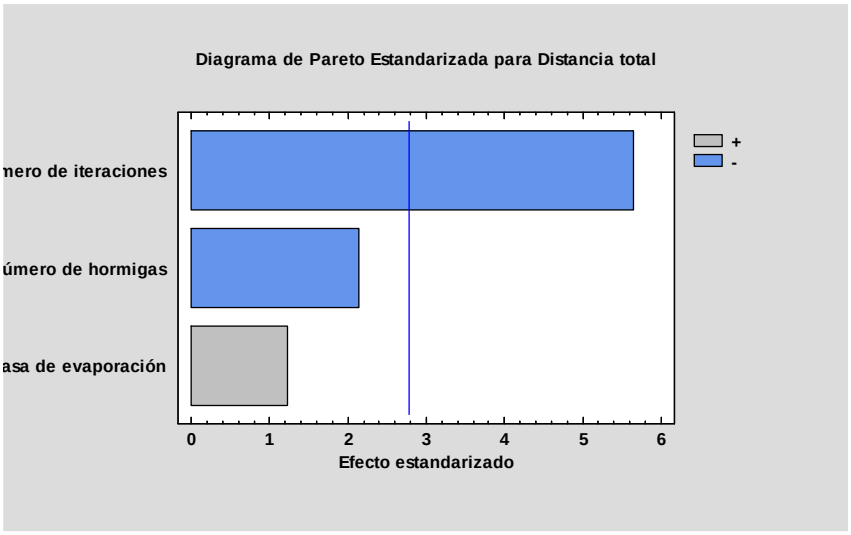


Figura 47. Diagrama de Pareto para la instancia c106-15 (Distancia total).

Para la instancia c106-15 se tiene que el número de iteraciones tiene una influencia significativa en la variable de respuesta, generando una influencia negativa lo que indica que debe ubicarse en el nivel alto.

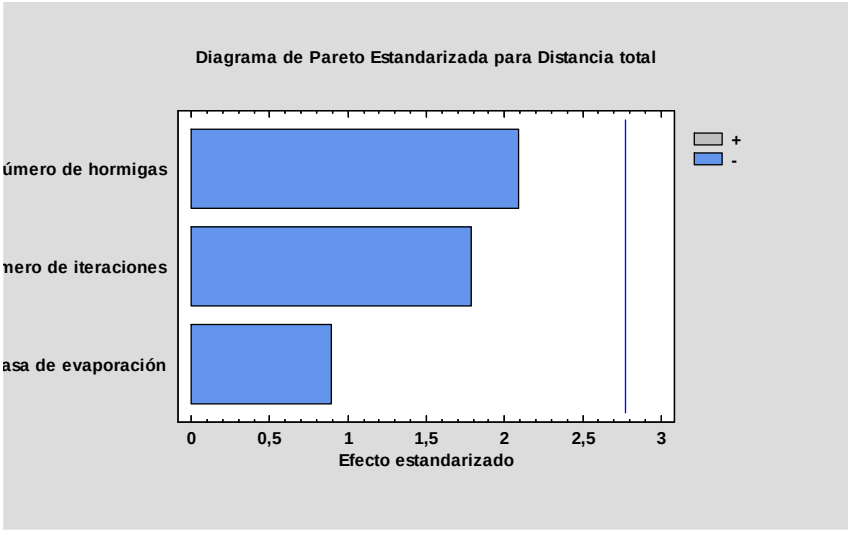


Figura 48. Diagrama de Pareto para la instancia c104-10 (Distancia total).

Figura 49. Diagrama de Pareto para la instancia r105-5 (Distancia total).

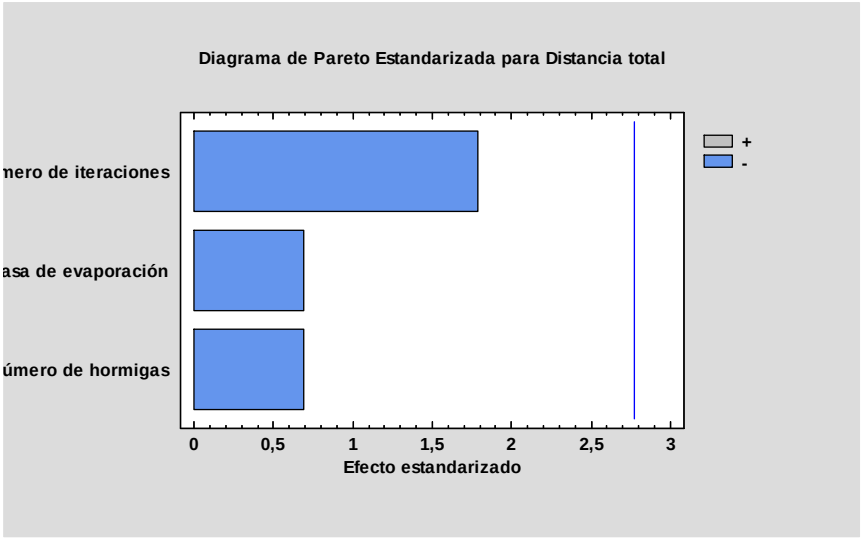
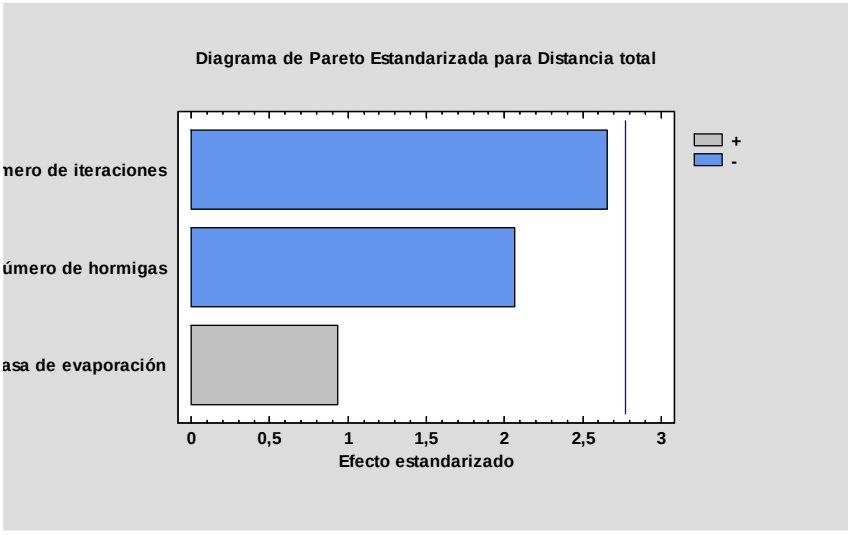


Figura 50. Diagrama de Pareto para la instancia rc108-10 (Distancia total).

Figura 51. Diagrama de Pareto para la instancia rc103-15 (Distancia total).



Para las instancias c104-10, r105-5, r103-10, rc108-10 y rc103-15 no se evidencian parámetros que ejerzan una influencia significativa en la variable de respuesta.

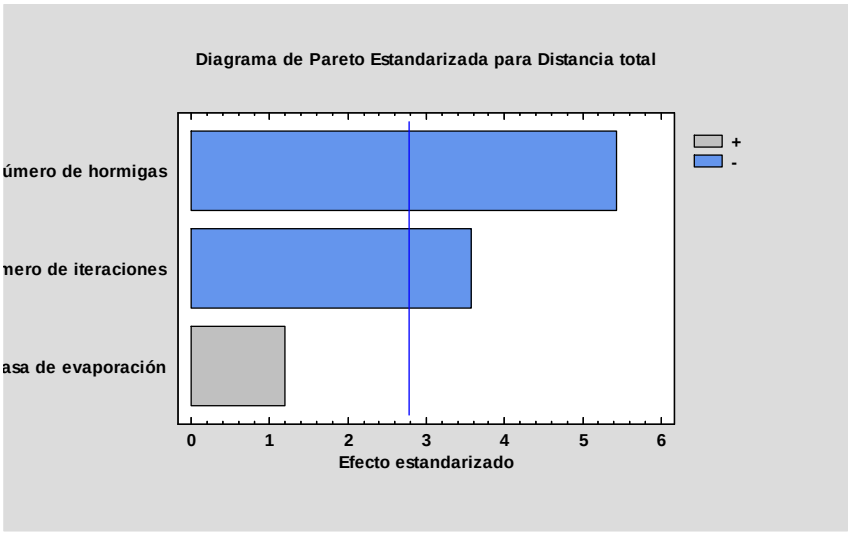


Figura 52. Diagrama de Pareto para la instancia r105-15 (Distancia total).

Para la instancia r105-15 se tiene que el parámetro más influyente en la variable de respuesta es el número de hormigas, y después de este es el número de iteraciones. Ambos parámetros generan una influencia negativa, lo que indica que deben ubicarse en el nivel alto.

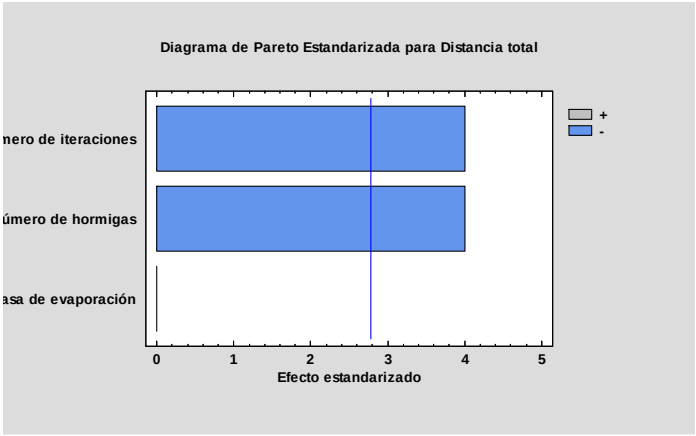


Figura 53. Diagrama de Pareto para la instancia rc108-5 (Distancia total).

Para la instancia rc108-5 se tiene que los parámetros más influyentes en la variable de respuesta son el número de iteraciones, y el número de hormigas. Ambos parámetros generan una influencia en igual medida y de carácter negativa, lo que indica que deben ubicarse en el nivel alto.

5.8. Diagramas de Pareto para las instancias pequeñas (Vehículos utilizados)

Para la instancia c105-5, c104-10, c106-15, r105-5, r103-10 y rc108-5 fue imposible realizar un análisis de parámetros por medio de un diagrama de Pareto, ya que en los 8 tratamientos realizados todas las respuestas fueron exactamente iguales, por lo tanto, es imposible determinar cuál de los parámetros ejerció una mayor influencia en la obtención del resultado.

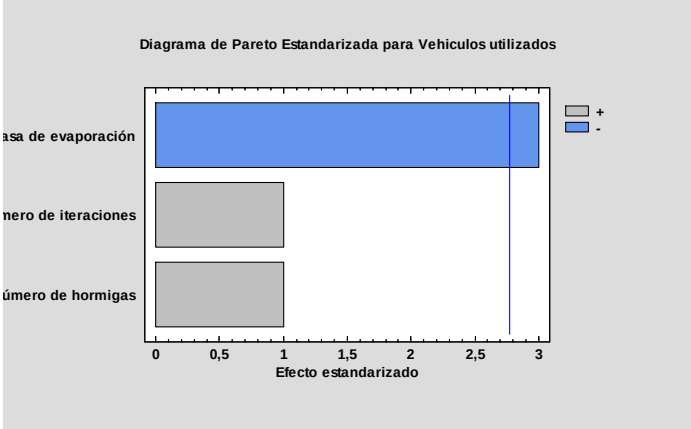


Figura 54. Diagrama de Pareto para la instancia r105-15 (Vehículos utilizados)

Para la instancia r105-15 se tiene que el parámetro que ejerce un efecto significativo en la función objetivo es la tasa de evaporación, la cual genera una influencia negativa, lo que indica que debe ubicarse en el nivel alto.

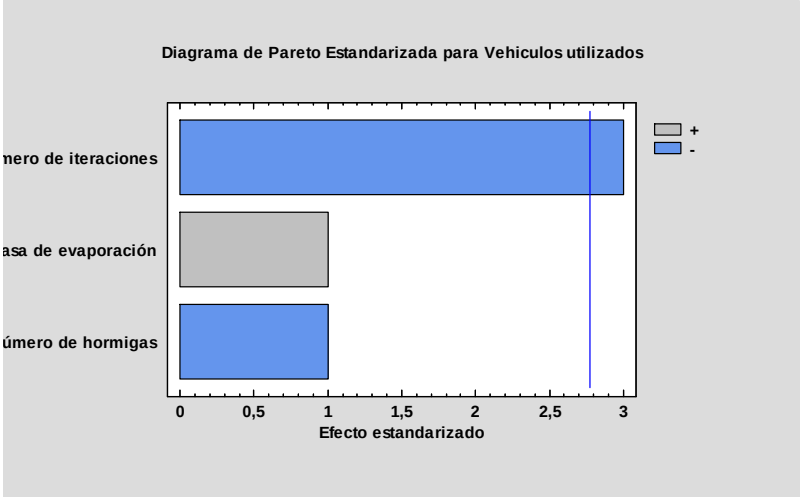


Figura 55. Diagrama de Pareto para la instancia rc108-10 (Vehículos utilizados).

Para la instancia rc108-10 se tiene que el parámetro que ejerce un efecto significativo en la variable de respuesta es el número de iteraciones, el cual genera una influencia negativa lo que indica que debe ubicarse en el nivel alto.

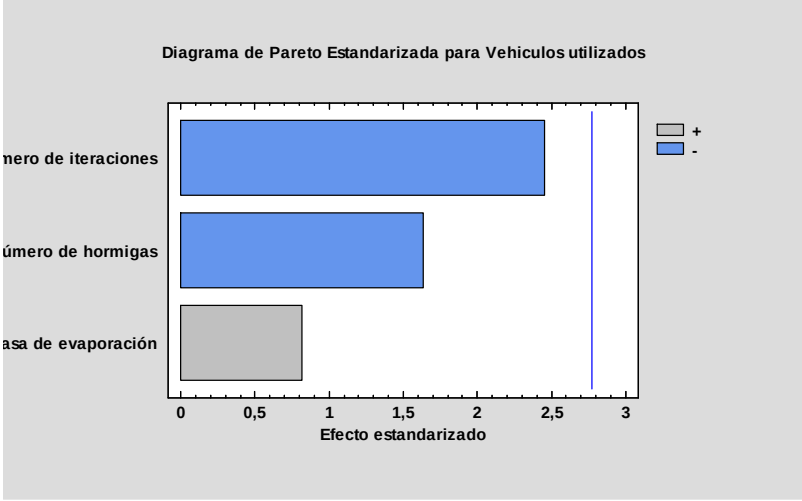


Figura 56. Diagrama de Pareto para la instancia rc103-15 (Vehículos utilizados).

De la figura 56 se puede concluir que ningún parámetro altera suficientemente la función objetivo como para considerarse significativo en la instancia rc103-15.

5.9. Conclusiones del diseño experimental

Teniendo en cuenta los resultados obtenidos y evidenciados en los diagramas de Pareto, se procede a construir la combinación de parámetros más adecuada para correr las instancias y solucionar el problema planteado en esta investigación según los resultados del diseño experimental, para esto, se toma como prioridad seguir el modelo que más se adecue a minimizar la cantidad de vehículos a utilizar, ya que el objetivo del problema es minimizar la función objetivo de costos, siendo los costos de adquisición de vehículos los que más inciden en esta.

Los resultados del diseño experimental arrojaron la siguiente sugerencia para minimizar la cantidad de vehículos utilizados en los problemas en cada grupo de instancias y sus respectivas distribuciones geográficas como se ve en la tabla 15.

Tabla 16.
Parámetros sugeridos según el diseño de experimentos.

Factores	Parámetros utilizados para resolver y validar las instancias grandes		
	c	**r**	**rc**
Número de hormigas (m	Indiferente	Indiferente	**100**
Tasa de evaporación (ρ)	Indiferente	Indiferente	**0,2**
Número de iteraciones (|)	**100**	Indiferente	**100**
Factores	Parámetros utilizados para resolver y validar las instancias pequeñas		
	c	**r**	**rc**
Número de hormigas (m)	Indiferente	Indiferente	Indiferente
Tasa de evaporación (ρ)	Indiferente	**0,8**	Indiferente
Número de iteraciones (|)	Indiferente	Indiferente	**100**

Debido a que en el caso de las instancias grandes con distribución geográfica Random-cluster se sugiere una combinación que fue retirada de los tratamientos a causa de los extensos tiempos computacionales, y que muchos de los parámetros a utilizar se dejan a disposición de los investigadores porque no afectan significativamente a la función objetivo, se opta por tomar las combinaciones de parámetros presentados en las tablas 8 y 10 que presentaron los mejores

resultados para cada distribución geográfica, dejando fijos los parámetros sugeridos por el diseño de experimentos, exceptuando el de color rojo, que fue retirado a causa del tiempo computacional inaceptable que este implica, dando lugar a las tablas 16 y 17 presentadas en el siguiente capítulo, donde se establecen los parámetros definitivos a utilizar para resolver y validar las instancias grandes y pequeñas respectivamente.

6. Validación del algoritmo

Para evaluar el desempeño del algoritmo Colonia de Hormigas se utilizaron dos conjuntos de instancias del benchmarking de referencia para el EVRP-TW mostradas en los apéndices C, D, E y F. El primer conjunto de instancias incluye 18 instancias divididas en 3 subconjuntos con 5, 10 y 15 clientes respectivamente, y un segundo conjunto mejor adaptado al tamaño que tendría un problema de carácter realista, tratando 21 instancias de 100 clientes. Ambos conjuntos poseen un horizonte de planeación corto y están repartidos en 3 distribuciones geográficas: Cluster, Random y Random-cluster.

Todas estas instancias fueron planteadas y diseñadas por Schneider, Stenger y Goeke, y fueron generadas basándose en las instancias de Solomón (1987).

Se toma la decisión de elegir éstos problemas por ser los mejores adaptados con respecto a la situación a tratar en cuanto al campo de estudio escogido para el problema (ventanas de tiempo y estaciones de carga). En adición a esto, las instancias que consideran combustibles alternativos, un rango de manejo limitado y la posibilidad de hacer recargas son bastante escasas o nulas en la

literatura como bien lo mencionaron Schneider, Stenger y Goeke, tanto así que se vieron obligados a diseñar sus propias instancias para poder evaluar su algoritmo.

Los parámetros usados para resolver las instancias se pueden ver en las tablas 15 y 16, los cuales fueron obtenidos al realizar el diseño de experimentos con el fin de garantizar el mejor funcionamiento del algoritmo para cada una de las diferentes clases de instancias y obtener el mejor resultado posible. Se ejecutó el programa 10 veces por cada instancia con las posibles combinaciones de los parámetros, tomando únicamente la mejor solución teniendo en cuenta primero la minimización de los vehículos utilizados y de segundo lugar como desempate la distancia total recorrida. Los resultados pueden ser consultados en el apéndice F.

El lenguaje de programación utilizado para validar el algoritmo fue Python 3.6.5.

Tabla 17.
Parámetros utilizados para resolver y validar las instancias grandes.

Factores	Parámetros utilizados para resolver y validar las instancias grandes		
	c	**r**	**rc**
Número de hormigas (m)	10	100	10
Tasa de evaporación (ρ)	0,2	0,8	**0,2**
Número de iteraciones (⌋)	**100**	10	**100**

Tabla 18.
Parámetros utilizados para resolver y validar las instancias pequeñas.

Factores	Parámetros utilizados para resolver y validar las instancias pequeñas		
	c	**r**	**rc**
Número de hormigas (m)	100	10	10
Tasa de evaporación (ρ)	0,2	**0,8**	0,2
Número de iteraciones (⌋)	100	100	**100**

Se llevó a cabo una comparación entre el ACO y el VNS/TS propuesto por Schneider, Stenger y Goeke en las mismas instancias del benchmarking; los resultados entre los dos métodos se muestran en las tablas 17 y 18. Cabe aclarar que se usaron métodos diferentes para resolver los problemas, por ende, no es posible comparar los parámetros utilizados en cada uno de estos.

Se compara la mejor solución obtenida con el ACO propuesto y la mejor solución conocida para cada problema, calculando el porcentaje de diferencia entre las soluciones, el cual es negativo y está resaltado en rojo si tuvo una desmejora, es positivo y verde si tuvo una mejora, o es de color negro si no presentó ningún cambio respecto a la otra.

Basándonos en los resultados presentados por los autores mencionados anteriormente, estudiamos la efectividad del método de solución ACO comparando nuestros resultados.

6.1. Resultados de la validación de las instancias

La notación usada en las tablas de resultados se describe a continuación:

- n: Número de vehículos utilizados.
- d: Distancia total recorrida para resolver el problema.
- Δn(%): Porcentaje de diferencia del número de vehículos utilizados entre los dos métodos de solución.
- Δd(%): Porcentaje de diferencia de la distancia total recorrida para resolver el problema entre los dos métodos de solución.

Tabla 19.
Comparación de resultados de las instancias pequeñas.

	VNS/TS		**ACO**			
Instancia	**n**	**d**	**n**	**d**	**Δn(%)**	**Δd(%)**
c101-5	2	257	3	198	**-50**	**22,96**
c103-5	1	176	2	150	**-100**	**14,77**
r104-5	2	137	2	146	**0**	**-6,57**
r105-5	2	156	3	175	**-50**	**-12,18**
rc105-5	2	241	4	185	**-100**	**23,24**
rc108-5	1	254	3	342	**-200**	**-34,65**
c101-10	3	394	4	267	**-33**	**32,23**
c104-10	2	274	3	260	**-50**	**5,11**
r102-10	3	249	4	214	**-33**	**14,06**

	VNS/TS		ACO			
Instancia	**n**	**d**	**n**	**d**	**Δn(%)**	**Δd(%)**
c101-5	2	257	3	198	**-50**	**22,96**
rc102-10	4	423	4	396	**0**	**6,38**
rc108-10	3	346	3	344	**0**	**0,58**
c103-15	3	385	2	357	**33**	**7,27**
c106-15	3	275	3	204	**0**	**25,82**
r102-15	5	413	3	315	**40**	**23,73**
r105-15	4	336	3	304	**25**	**9,52**
rc103-15	4	398	2	218	**50**	**45,23**
rc108-15	3	370	4	395	**-33**	**-6,76**

Comparando los resultados del ACO y el VNS/TS para las instancias pequeñas, se observa que el algoritmo colonia de hormigas nunca iguala el mejor valor de resultados conocido, sin embargo, en algunas ocasiones mejora los resultados generales y en otras optimiza la distancia más no el número de vehículos utilizados. El umbral máximo de diferencia para el valor de n para las instancias entre 5 y 15 clientes es de 2 vehículos, presentando falencias a la hora de reducir al máximo la cantidad de vehículos necesarios para satisfacer la demanda de todos los clientes, pero sin presentar un aumento irracional en esta.

Para las instancias de 5 clientes no se presenta un comportamiento uniforme, no obstante, se puede ver un patrón respecto a la distribución geográfica, ya que para las instancias tipo:

- Cluster, no pudo alcanzarse la mejor cantidad de vehículos conocida, pero sí pudo reducirse la distancia total recorrida hasta en un 22,96%, resultado que puede ser más ventajoso si la flota de vehículos disponible es fija y solo se busca minimizar los costos de recarga.
- Random, no pudo lograrse ninguna mejora en el resultado, ya que fue requerida una mayor cantidad de vehículos y de kilómetros recorridos para satisfacer a los clientes, presentando un aumento de hasta el 12,18% de distancia total recorrida y de 1 vehículo requerido para cumplir con las demandas del recorrido.

- Random-cluster, no pudo alcanzarse la cantidad de vehículos conocida para ninguna de las dos instancias, sin embargo, se optimizo el valor de la distancia total recorrida para la instancia rc105-5 en un 23,24%, mientras que la instancia rc108-5 tuvo una desmejora del 34,65% en su distancia total.

Para las instancias de 10 clientes se evidencia una mejora en la distancia total recorrida de hasta un 32,23% para todas las instancias excepto la r103-10, la cual presentó un 4,35% de kilómetros adicionales recorridos respecto a la mejor solución conocida, siendo este un valor pequeño aceptable y considerado como un buen resultado. Para el caso de las instancias Random-cluster de 10 clientes, no solamente se logró mejorar el resultado de la distancia total recorrida, sino también se igualó el mejor valor conocido de la cantidad de vehículos utilizados en las instancias rc102-10 y rc108-10, mostrando una alta efectividad del algoritmo para problemas rc de 10 clientes.

Para las instancias de 15 clientes se presentan los mejores resultados del grupo de instancias pequeñas, donde 5 de las 6 instancias logran optimizarse considerablemente convirtiéndose en nuevos mejores resultados conocidos, ya que 4 de ellas logran reducir tanto la cantidad de vehículos utilizados como la distancia total recorrida en hasta en 2 y un 45,23% respectivamente, y en la quinta se optimiza la distancia total recorrida en un 25,82% mientras que iguala el mejor resultado conocido de la cantidad de vehículos utilizados.

Finalmente, se obtiene un único resultado negativo donde se exceden los vehículos necesarios en una unidad, y se ve un aumento del 6,76% de kilómetros recorridos respecto a la mejor solución conocida, resultado que, aunque sea negativo, no se considera obsoleto.

Tabla 20.
Comparación de resultados de las instancias grandes.

	VNS/TS		ACO			
Instancia	**n**	**d**	**n**	**d**	**Δn(%)**	**Δd(%)**
c102	11	1056	11	1109	**0**	**-5,02**
c103	10	1041	14	1145	**-40**	**-9,99**
c104	10	979	8	1175	**20**	**-20,02**
c105	11	1075	11	1072	**0**	**0,28**
c106	11	1057	10	1265	**9**	**-19,68**
c107	11	1031	11	1035	**0**	**-0,39**
c108	10	1100	10	1078	**0**	**2**
r101	18	1670	18	1710	**0**	**-2,40**
r102	16	1495	17	1122	**-6**	**24,95**
r103	13	1299	13	1301	**0**	**-0,15**
r104	11	1088	11	1346	**0**	**-23,71**
r105	14	1461	13	1475	**7**	**-0,96**
r106	13	1344	12	1438	**8**	**-6,99**
r107	12	1154	12	1386	**0**	**-20,10**
rc101	16	1731	15	1643	**6**	**5,08**
rc102	15	1554	14	1633	**7**	**-5,08**
rc103	13	1351	13	1349	**0**	**0,15**
rc104	11	1238	12	1654	**-9**	**-33,60**
rc105	14	1475	15	1512	**-7**	**-2,51**
rc106	13	1437	12	1434	**8**	**0,21**
rc107	12	1275	11	1699	**8**	**-33,25**

Comparando los resultados del ACO y el VNS/TS para las instancias grandes, se observan grandes diferencias respecto a los resultados de las instancias pequeñas, esta vez el algoritmo colonia de hormigas iguala el mejor valor conocido de vehículos utilizados en el 42,85% de las instancias y lo mejora en el 38,1% de las instancias, obteniendo un resultado óptimo para la cantidad de vehículos en el 80,95% de las instancias evaluadas.

Sin embargo, se sigue presentando una irregularidad en la mejora de los resultados generales, ya que en el 71,42% de las instancias se obtienen resultados de distancia total recorrida por encima del mejor conocido, lo que puede ser causado debido a la misma reducción en la cantidad de vehículos necesarios, ya que al disminuir estos, se espera un efecto de aumento en la distancia total

recorrida para resolver el problema. El umbral máximo de diferencia para el valor de n para las instancias de 100 clientes es de 4 vehículos, pero al contrario del caso de las instancias pequeñas, y como se mencionó anteriormente, se presentan falencias a la hora de reducir la distancia total del recorrido aun para las instancias en las que el número de vehículos utilizados son iguales a la del mejor resultado conocido, presentando un aumento de máximo el 33,6%.

Para las instancias de 100 clientes no se presenta un comportamiento uniforme en la mejora de resultados ni siquiera según la distribución geográfica, no obstante, se puede afirmar que exceptuando las instancias c103, rc104 y rc105, donde se presentan resultados totalmente por debajo del mejor conocido, el algoritmo tiene un buen efecto sobre las instancias en las que fue evaluado, teniendo en cuenta que se le da una mayor prioridad a la reducción de la cantidad de vehículos utilizados debido a su alto costo de adquisición.

7. Resultados y Análisis

El algoritmo colonia de hormigas propuesto se ejecutó para solucionar el EVRPTW-PR con los parámetros ajustados por el diseño de experimentos para encontrar la solución con el menor costo total en 6 instancias del benchmarking adaptadas al problema en cuestión.

Se tomaron las instancias c101-10, r104-5, rc108-15, c102, r-104y rc101 para probar el algoritmo con unas condiciones más realistas, tomando precios reales del mercado de vehículos eléctricos para transporte logístico.

A continuación, se presentan los costos a tener en cuenta para el desarrollo del problema:

Tabla 21.
Costos referentes a vehículos eléctricos de carga de mercancía.

Concepto	Costo ()
Vehículo	€ 29.500
Carga	€ 0,02/km

Fue realizado un benchmarking de un modelo de camión eléctrico para transporte logístico en ciudad, cuyas características se asemejaran lo más posible a las condiciones establecidas por las instancias para la solución del problema, seleccionando de ésta manera el vehículo italiano "ALKE XT320E PC Cabinato", presentando una autonomía de 75 hasta 150 km, una capacidad de carga de 620 kilogramos de mercancía y una batería de litio con un costo de recarga completa de 1,5 euros y un costo de adquisición de 29.500 euros por vehículo (Alke, n.d.-b) (Alke, n.d.-a).

Para variar las aplicaciones del problema se asumen dos situaciones:

- 3 problemas de 100 clientes donde no se cuenta con una flota de vehículos, por lo que esta será comprada con base a los resultados de optimización del algoritmo el cual minimizará la totalidad de los costos.
- 3 problemas de entre 5 y 15 clientes donde se cuenta con una flota de vehículos existente, por lo que se minimizarán únicamente los costos operacionales de la ruta logística.

En la tabla 21 se muestran los resultados encontrados por el ACO para los problemas planteados, las rutas tomadas por cada una de estas soluciones pueden ser consultadas en el apéndice G.

Se revela que el método propuesto supera el rendimiento del algoritmo VNS/TS en una de las situaciones planteadas demostrando la competitividad de este en ciertos escenarios de los problemas pequeños, mientras que, para las situaciones de 100 clientes, se logró reducir los costos

en solo una ocasión y en las otras dos se mantuvo un costo poco elevado hasta de un 9,09% extra respecto al otro método de solución, estableciendo al ACO planteado como un método de solución aceptable para los problemas grandes.

Tabla 22.
Resultados del EVRPTW-PR para la instancia c101-10.

Algoritmo	**c101-10**		
	Costo por vehículo	**Costo por distancia recorrida**	**Costo total**
VNS/TS	€ 88.500,00	€ 7,88	€ 88.507,88
ACO	€ 118.000,00	€ 5,34	€ 118.005,34

Tabla 23.
Resultados del EVRPTW-PR para la instancia r104-5.

Algoritmo	**r104-5**		
	Costo por vehículo	**Costo por distancia recorrida**	**Costo total**
VNS/TS	€ 59.000,00	€ 2,74	€ 59.002,74
ACO	€ 59.000,00	€ 2,92	€ 59.002,92

Tabla 24.
Resultados del EVRPTW-PR para la instancia rc108-15.

Algoritmo	**rc108-15**		
	Costo por vehiculo	**Costo por distancia recorrida**	**Costo total**
VNS/TS	€ 88.500,00	€ 7,40	€ 88.507,40
ACO	€ 118.000,00	€ 7,90	€ 118.007,90

Tabla 25.
Resultados del EVRPTW-PR para la instancia c102.

Algoritmo	**c102**		
	Costo por vehículo	**Costo por distancia recorrida**	**Costo total**
VNS/TS	€ 324.500,00	€ 21,12	€ 324.521,12
ACO	€ 324.500,00	€ 22,18	€ 324.522,18

Tabla 26.
Resultados del EVRPTW-PR para la instancia r104.

Algoritmo	**r104**			
	Costo por vehículo	**Costo por distancia recorrida**		**Costo total**
VNS/TS	€ 324.500,00	€	24,76	€ 324.524,76
ACO	€ 354.000,00	€	33,08	€ 354.033,08

Tabla 27.
Resultados del EVRPTW-PR para la instancia rc101.

Algoritmo	**rc101**			
	Costo por vehículo	**Costo por distancia recorrida**		**Costo total**
VNS/TS	€ 472.000,00	€	34,62	€ 472.034,62
ACO	€ 442.500,00	€	32,86	€ 442.532,86

8. Conclusiones

Los objetivos planteados para esta investigación se cumplieron en su totalidad. En base a la revisión de la literatura, se evidencia que la cantidad de artículos respecto a problemas de ruteo de vehículos eléctricos es aún muy escasa, siendo Europa y China líderes en la investigación de estos gracias a su infraestructura y condiciones políticas que incentivan el desarrollo y aplicación de algoritmos para la solución de estos problemas a diferencia de países menos desarrollados como Colombia.

Con base en los resultados obtenidos para las diferentes instancias y su respectiva comparación con el método VNS/TS, se valida que el algoritmo colonia de hormigas es un método viable para la solución del EVRPTW-PR, demostrando ser una metaheurística competitiva para problemas

pequeños y aceptable para los grandes ante otros algoritmos presentados en la literatura.

Se pudo comprobar a través del diseño de experimentos que el algoritmo se comporta de manera muy particular según las características del problema, ya que factores como el tamaño de los conjuntos del grafo y la distribución geográfica de los clientes afectan directamente el valor de la variedad de parámetros usados para obtener la solución, haciendo que en algunos casos sea más conveniente utilizar una combinación diferente de estos que en otros.

En cuanto a los efectos de los parámetros, se puede observar que el número de iteraciones es el parámetro con mayor influencia sobre los resultados ya que presenta un efecto significativo en 13 de las instancias evaluadas en el diseño de experimentos. Así mismo, la tasa de evaporación es el parámetro con menor influencia, siendo significativo para solo una instancia. Esto puede deberse a que los valores de α y β fueron fijados, haciendo que el nivel de importancia en la toma de decisiones de la hormiga a causa de las feromonas sea menor que el nivel que le da la metaheurística a tomar los caminos más cortos.

Se observó que el algoritmo posee una mayor capacidad para resolver problemas pequeños debido al tiempo computacional requerido para procesar la solución, y la posibilidad de utilizar niveles de parámetros mucho más altos que los usados en las instancias grandes, lo cual se ve reflejado en la reducción de los costos operacionales en el 72% de las instancias pequeñas evaluadas, mientras que en las grandes este valor no supera el 25%.

Se encontró que, aunque el algoritmo no lograra alcanzar el mejor resultado conocido en las instancias evaluadas respecto a los costos totales, en algunas ocasiones logró optimizar el costo operacional de las rutas, minimizando la distancia total de tal manera en la que para problemas donde se tenga una flota de vehículos fija, el algoritmo colonia de hormigas sea más eficiente que

otros; y en otros casos, se logra optimizar la cantidad de vehículos a utilizar para resolver el problema, lo cual es más conveniente para problemas en los que la flota de vehículos sea muy limitada.

9. Recomendaciones

El código de la programación del algoritmo colonia de hormigas queda a disposición de los estudiantes de la Universidad Industrial de Santander o de los investigadores interesados en utilizar este algoritmo ya sea para modificarlo, compararlo o para usarlo como base para el desarrollo de uno más eficiente por medio de diferentes métodos, con el principal reto de reducir el extenso tiempo computacional para el cálculo de soluciones, y la mejora en la minimización de los resultados.

Incentivar el uso y aprendizaje de diferentes lenguajes de programación en los estudiantes de ingeniería industrial con el objetivo de adquirir las competencias necesarias al momento de desarrollar investigaciones de este tipo, facilitando el proceso de construcción de los algoritmos.

Desarrollar softwares aplicados para la industria bumanguesa usando el algoritmo propuesto como una herramienta de optimización para la asignación de operaciones.

Incentivar la ejecución de proyectos de grado que aborden investigaciones relacionadas con el ruteo de vehículos eléctricos y sus derivados utilizando diferentes métodos de solución que busquen llevar el problema a un plano realista.

Revisar y modificar el código del algoritmo para que de alguna manera se logre reducir el

tiempo computacional que toma su ejecución.

Determinar si el lenguaje de programación del algoritmo tiene incidencia en la calidad de la respuesta y el tiempo de procesamiento del problema, de tal forma en la que al usar otro, el tiempo de ejecución pueda reducirse mientras se mantenga la calidad de la solución o viceversa.

Referencias Bibliográficas

Alke. (n.d.-a). Alkè XT XT320E PC Cabinato (N1). Ficha técnica. Retrieved from https://www.automoto.it/listino/alke/xt/xt320e-pc-cabinato--n1/85520

Alke. (n.d.-b). Electric Vehicles - Delivering Solutions. Catálogo. Retrieved from https://www.alke.com/doc/alke-atx-electric-vehicles-catalog-eng.pdf

Bell, J. E., & McMullen, P. R. (2004). Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, *18*(1), 41–48. https://doi.org/10.1016/J.AEI.2004.07.001

Bolaños, R. A., Correa, C. A., & Echeverri, M. G. (2009). Optimización por colonia de hormigas aplicada al problema de planeamiento de la transmisión. Revista de Ingeniería. Retrieved from http://www.scielo.org.co/pdf/ring/n30/n30a5.pdf

Braekers, K., Ramaekers, K., & Van Nieuwenhuyse, I. (2016). The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, *99*, 300–313. https://doi.org/10.1016/J.CIE.2015.12.007

Cardozo O., J. (2013). *Solución al problema de ruteo de vehículos con capacidad limitada "CVRP" a través de la heurística de barrido y la implementación del algoritmo genético de Chu-Beasley*. Pereira. Retrieved from http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/4000/5196O75.pdf?sequence=1

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., & Schrijver, A. (1997). *Combinatorial*

Optimization. Recuperado de http://profs.sci.univr.it/~rrizzi/classes/Algoritmi/dispense/combOpt/cobook_select.pdf

Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, *6*(1), 80–91. https://doi.org/10.1287/mnsc.6.1.80

Erdoğan, S., & Miller-Hooks, E. (2012). A Green Vehicle Routing Problem. *Transportation Research Part E: Logistics and Transportation Review*, *48*(1), 100–114. https://doi.org/10.1016/J.TRE.2011.08.001

Fountas, C., & Vlachos, A. (2005). Ant colonies optimization (ACO) for the solution of the vehicle routing problem (VRP). Journal of Information and Optimization Sciences, 1(26), 135–142.

Gelves T., N., Mora M., R., & Lamos D., H. (2015). Solución del problema de ruteo de vehículos con demandas estocásticas mediante la optimización por espiral. *Scielo*. https://doi.org/10.19053/01211129.4626

Hof, J., Schneider, M., & Goeke, D. (2017). *Transportation research Part B, Methodological*. *Transportation Research Part B: Methodological* (Vol. 97). Elsevier Science. Recuperado de https://trid.trb.org/view/1459142

Hoffman, K. L., Padberg, M., & Rinaldi, G. (2013). Traveling Salesman Problem. In *Encyclopedia of Operations Research and Management Science* (pp. 1573–1578). Boston, MA: Springer US. https://doi.org/10.1007/978-1-4419-1153-7_1068

Kuri, A. (2000). *Algoritmos Genéticos*. Retrieved from http://cursos.itam.mx/akuri/PUBLICA.CNS/2000/AGS.PDF

Lin, J., Zhou, W., & Wolfson, O. (2016). Electric Vehicle Routing Problem. *Transportation Research Procedia*, *12*, 508–521. https://doi.org/10.1016/J.TRPRO.2016.02.007

Mavrovouniotis, M., & Yang, S. (2013). Adapting the Pheromone Evaporation Rate in Dynamic Routing Problems (pp. 606–615). Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-37192-9_61

Merve, K., & Çatay, B. (2016). *Transportation research. Part C, Emerging technologies. Transportation Research Part C: Emerging Technologies* (Vol. 65). Pergamon Press. Recuperado de https://trid.trb.org/view/1399943

MinAmbiente. (2015). EL ABC DE LOS COMPROMISOS ABC de los compromisos de Colombia para la COP21. Ministerio de Ambiente y Desarrollo Sostenible. Recuperado de www.wwf.org.co

MinAmbiente. (2018). Colombia inicia el desarrollo de la Estrategia Nacional de Movilidad Eléctrica. Recuperado de http://www.minambiente.gov.co/index.php/noticias/3675-colombia-inicia-el-desarrollo-de-la-estrategia-nacional-de-movilidad-electrica

Montoya, A., Guéret, C., Mendoza, J. E., & Villegas, J. G. (2017). The electric vehicle routing problem with nonlinear charging function. *Transportation Research Part B: Methodological*, *103*, 87–110. https://doi.org/10.1016/J.TRB.2017.02.004

Morales Q., B. (2014). *Modelo de masificación de vehículos eléctricos en Bogotá D.C.* Universidad Nacional de Colombia. Recuperado de http://www.bdigital.unal.edu.co/48580/1/73575424.2015.pdf

ONU. (n.d.). *Report of the World Commission on Environment and Development: Our Common Future*. Recuperado de http://www.un-documents.net/our-common-future.pdf

Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, *34*(8), 2403–2435. https://doi.org/10.1016/J.COR.2005.09.012

PNUD. (2016). IDEAM y PNUD presentan Inventario Nacional de Emisiones de Gases de Efecto Invernadero. Recuperado de http://www.co.undp.org/content/colombia/es/home/presscenter/articles/2016/11/03/ideam-y-pnud-presentan-inventario-nacional-de-emisiones-de-gases-de-efecto-invernadero.html

Pullen, H. G. M., & Webb, M. H. J. (1967). A computer application to a transport scheduling problem. *The Computer Journal*, *10*(1), 10–13. https://doi.org/10.1093/comjnl/10.1.10

Schiffer, M., & Walther, G. (2017). The electric location routing problem with time windows and partial recharging. *European Journal of Operational Research*, *260*(3), 995–1013. https://doi.org/10.1016/J.EJOR.2017.01.011

Schneider, M., Stenger, A., & Goeke, D. (2014). The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations. *Transportation Science*, *48*(4), 500–520. https://doi.org/10.1287/trsc.2013.0490

Taş, D., Jabali, O., & Van Woensel, T. (2014). A Vehicle Routing Problem with Flexible Time Windows. *Computers & Operations Research*, *52*, 39–54. https://doi.org/10.1016/J.COR.2014.07.005

Tolosa Barón, J. L. (2005). *Colonia de hormigas, Fundamentación teórica y aplicación en la*

optimización de sistemas logísticos de ruteo con intervalos de recepción y tiempos de atención máxima. Universidad Industrial de Santander.

Valencia, E. (1997). Optimización mediante algoritmos genéticos. *Anales Del Instituto de Ingenieros de Chile*, *109*(2), 83–92.

Zhang, S., Gajpal, Y., Appadoo, S. S., & Abdulkader, M. M. S. (2018). Electric vehicle routing problem with recharging stations for minimizing energy consumption. *International Journal of Production Economics*, *203*, 404–413. https://doi.org/10.1016/j.ijpe.2018.07.016

Zheng, Y., & Liu, B. (2006). Fuzzy vehicle routing model with credibility measure and its hybrid intelligent algorithm. *Applied Mathematics and Computation*, *176*(2), 673–683. https://doi.org/10.1016/J.AMC.2005.10.013