PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

ALGORITMO COLONIA DE HORMIGAS (ANT COLONY) PARA EL ABASTECIMIENTO DE MEDICINAS ENTRE HOSPITALES REGIONALES EN EL CONTEXTO DE PANDEMIA DE COVID-19

Tesis para obtener el título profesional de Ingeniero Informático

AUTOR:

Luis Denilson Ramirez Osorio

ASESORES:

Manuel Tupia Anticona

Rony Cueva Moscoso

Lima, febrero, 2024

Informe de Similitud

Yo, …Rony Cueva Moscoso……..……,

docente de la Facultad de ……Ciencias e Ingeniería…… de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado

…… ALGORITMO COLONIA DE HORMIGAS (ANT COLONY) PARA EL ABASTECIMIENTO DE MEDICINAS ENTRE HOSPITALES REGIONALES EN EL CONTEXTO DE PANDEMIA DE COVID-19……..,

del/de la autor(a)/ de los(as) autores(as)

……. Luis Denilson Ramirez Osorio……,

…..……,

…..……,

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 22%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 28/01/2024.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: …Lima, 28 de enero del 2024……

Apellidos y nombres del asesor / de la asesora: Cueva Moscoso Rony	
DNI: 09942265	Firma
ORCID: 0000-0003-4861-571X	

Resumen

En el último año, los casos de personas contagiadas por COVID-19 se incrementó de manera alarmante dado a las distintas variantes que han ido surgiendo. Es por este motivo que es importante brindar atención oportuna de tal manera que los puestos de salud en las distintas regiones del Perú cuenten con el equipo y medicinas necesarias para tratar los contagios. De lo contrario, la falta de estos suministros puede ocasionar que no se satisfaga la demanda y agravar la salud de las personas que lo necesitan. Por ello, es importante contar con un plan de rutas para el abastecimiento de suministros a los centros de salud que cumpla con ciertos criterios y restricciones.

En este sentido, elaborar un plan de rutas de forma eficiente y que sea óptima es una tarea complicada dado de que se trata de un problema algorítmicamente compleja que forma parte de problemas del tipo NP difícil al ser una variante del problema del vendedor viajero. Por esta razón, la solución a este tipo de problemas no puede ser resueltas por algoritmos exactos, sino que se prefiere emplear algoritmos metaheurísticos, dado que estos son más eficientes en cuanto a los tiempos de ejecución y eficaces al encontrar soluciones de tal manera que se respeten las restricciones del problema.

El presente proyecto de tesis tiene como objetivo solucionar el problema presentado diseñando e implementando un algoritmo colonia de hormigas para dar solución al problema de ruteo de vehículos con capacidades en situaciones de emergencia puesto que está evidenciado que este algoritmo es perfecto para problemas de optimización enfocado a los problemas de ruteo. De este modo, se podrá definir el mejor plan de distribución y abastecimiento orientado a la realidad peruana durante la pandemia de COVID-19. Asimismo, se desarrolla el algoritmo voraz primero el mejor para tener una línea base sobre el cual poder comparar los resultados obtenidos y poder establecer la optimalidad del algoritmo colonia de hormigas.

Finalmente, se analiza y se pone a prueba la implementación del algoritmo en una región del Perú con el fin de poder determinar el mejor plan de rutas para el abastecimiento de medicinas a los centros de salud.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

Capítulo 1.Generalidades

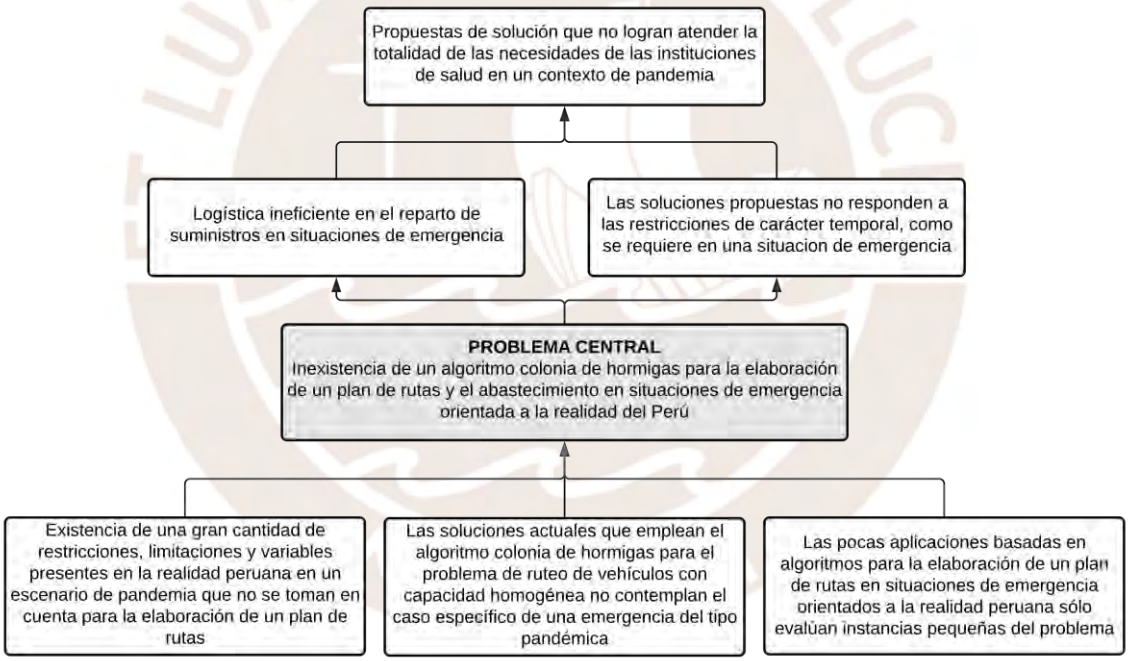
1.1 Problemática

En esta sección del capítulo de generalidades se abordará la problemática, la cual será presentada como un árbol de problemas que luego será descrita a detalle y, por último, se especificará el problema central a la cual se pretende encontrar una solución.

1.1.1 Árbol de Problemas

La forma en la que se estructuraron los problemas del presente proyecto se realizó con la técnica del Árbol de problemas tal y como se muestra en la Figura 1. En este árbol se muestra los problemas causas, el problema central y los problemas efecto.

Figura 1. Árbol de problemas



1.1.2 Descripción

El tema de encontrar una ruta óptima desde un punto a otro siempre ha sido un tema muy estudiado desde antes de la aparición del computador. Uno de los problemas más famosos sobre este tema es la del vendedor viajero (TSP por sus siglas en inglés), en el cual este vendedor busca encontrar la ruta de menor distancia de tal manera que recorra una serie de ciudades, pasar por ellas solo una vez y regresar al punto de partida (Dantzig & Ramser, 1959). Sin embargo, el problema puede complicarse aún

más si se quiere llevar a una situación real. Este problema puede ser generalizado a la generación de rutas a través de una red de carreteras para vehículos y, a este tipo de problema, se le conoce como el problema de ruteo de vehículos o VRP por sus siglas en inglés (Labadie et al., 2016). Si al caso anterior, también, se toma en cuenta la capacidad de los vehículos, es decir, qué tanto pueden transportar estos vehículos, y el nivel de demanda de los distintos puntos, la situación pasa a ser un problema de ruteo de vehículos con capacidades diferenciadas (CVRP por sus siglas en inglés) (Irnich et al., 2014).

Esta variante es un tipo de problema de optimización combinatoria y tiene una alta complejidad algorítmica, lo que significa que es complicado encontrar una buena solución en un tiempo razonable por medio de métodos tradicionales como los algoritmos que se basan en programación lineal. Este caso particular, es un problema recurrente en el área de la logística Industrial y empresarial, pues lo que se busca con la solución al problema es la optimización de su cadena de suministros y con ello la reducción de costos (Hasle & Kloster, 2007).

Sin embargo, en casos de emergencia, como un desastre natural, se requiere una rápida planificación para el rescate o envío de suministros a las zonas afectadas y así asegurar el bienestar de las personas (Li & Chung, 2019).

Actualmente, el inconveniente de esta situación es que existen muy pocas soluciones algorítmicas a este problema y en todos los casos se enfocan en los desastres naturales. Durante la revisión del estado del arte uno de los estudios plantea que la logística humanitaria del sistema de cadena de suministros de ayuda es un aspecto crítico dado que los desastres naturales son impredecibles (Bozorgi-Amiri et al., 2012). Las soluciones de los estudios que se encontraron para el problema plantean que es necesario actuar con rapidez luego de un desastre natural como un terremoto, un huracán o la erupción de un volcán (Batmetan et al., 2017; Li & Chung, 2019). Por ello, es necesario encontrar soluciones al problema de ruteo de vehículos de manera rápida y que estas sean lo más cercanas a las soluciones óptimas para atender oportunamente las necesidades.

Muy aparte de los desastres naturales, existen situaciones de emergencia como las pandemias. En el siglo XXI han ocurrido una serie de pandemias que ha afectado a gran parte de la población y las cuales ha costado la vida de millones de personas (Hidalgo García, 2020) y este año, el 11 de marzo de 2020, la Organización Mundial de

la Salud caracterizó al COVID-19 como una nueva pandemia que afecta a varios países y continentes (OPS, 2020).

En el caso del Perú, a pesar de ser uno de los primeros países en tomar medidas preventivas, es uno de los países más afectados por esta nueva pandemia y esto puede verse reflejado en el gran número de contagiados que presenta, pues es el segundo país con más contagiados de Sudamérica (Castillo, 2020). Esta crisis ha afectado a todos los peruanos y una de las consecuencias más notorias de este problema es que en distintas regiones del Perú las medicinas empezaron a escasear dada la alta demanda de estas. Según una entrevista realizada por el diario Gestión a la sociedad de Comercio Exterior del Perú (ComexPerú) se menciona que “el sector público, que es el mayor comprador de medicamentos en el país (70% del total) tiene un sistema deficiente de gestión y logística que origina un desabastecimiento crónico” (Diario Gestión, 2020). Asimismo, es sabido que las carreteras del Perú no son las mejores pues el 27% de la red vial nacional se encuentra en mal estado y el 8% en un estado regular (MTC, 2017). Esto genera que el transporte de los medicamentos a las distintas partes de una región sea lento.

Por ello, la falta de una buena gestión y logística de rutas para la distribución y abastecimiento de medicinas a las distintas zonas del Perú sumado con el mal estado de las carreteras y el difícil acceso a algunas ciudades debido a la accidentada geografía ocasiona que los medicamentos lleguen luego de mucho tiempo y con ello, muchas personas no llegan a tener acceso a las medicinas. Asimismo, muchas de las decisiones que se toman para el envío de suministros a las distintas zonas del país son realizadas de manera improvisada (Lossio-Ventura J.A., 2016). A pesar de que existe un plan de distribución de suministros elaborado por el CENARES (Centro Nacional de Abastecimiento de Recursos Estratégicos en Salud), este no está certificado y solo está basado en buenas prácticas (Salomón, 2017). De igual manera, la distribución realizada por CENARES llega hasta las direcciones regionales de salud (Diresas), las cuales son las encargadas de realizar la distribución a nivel de su región. Sin embargo, en una entrevista realizada por el diario El Comercio a un especialista en descentralización y políticas públicas “sostuvo que en el Perú no existen sistemas logísticos diseñados para una adecuada distribución de medicamentos e insumos en todo el país” (Arango, 2019). En otras palabras, al no tener un sistema, el plan no contempla el uso de un algoritmo para la generación de rutas sino que todo se realiza de manera manual y, por ende, las soluciones propuestas van a ser ineficientes,

inexactas e ineficaces en el sentido de que se va a incrementar el tiempo o el costo de los envíos, pues al realizarse de manera manual y guiarse solo por intuición no es posible contemplar todos los estados de posibles soluciones por lo que, generalmente, se terminaría eligiendo una solución subóptima (Almossawi, 2017).

A pesar de que existen soluciones basadas en algoritmos para el problema de ruteo de vehículos con capacidad limitada en situaciones de desastres naturales, en la revisión sistemática de la literatura no existen soluciones para el problema en situaciones de emergencia como lo es una pandemia. Aunque ambos son escenarios que representan una emergencia, son situaciones distintas. Durante una pandemia se tienen que considerar la variable del nivel de contagios que una ciudad pueda tener por lo que resulta conveniente repartir los suministros dándoles prioridad a las ciudades que tengan un alto nivel de contagios. Esta es una variable que en un desastre natural como los mencionados anteriormente no se toma en cuenta para el abastecimiento de medicinas a los hospitales de manera oportuna.

Por otra parte, en Perú existen escasas aplicaciones para el problema. Uno de estos estudios, tal y como se mencionó párrafos antes, está enfocado en situaciones de desastres naturales como lo es un terremoto, en este se busca una ruta óptima a nivel de solo Lima para disminuir los costos que implica repartir la ayuda humanitaria (Lossio-Ventura J.A., 2016). Otro de estos estudios, también se enfoca en el envío de ayuda humanitaria en diversas situaciones de desastres naturales como inundaciones, aluviones o terremotos pero este, a diferencia del anterior, considera diversos puntos alrededor del Perú (Aduviri, 2018).

Debido a estas variables como la distancia entre los puntos de entrega, el estado de las carreteras y el nivel de contagio, restricciones como el tiempo de entrega o la capacidad de los vehículos y limitantes como la cantidad de vehículos disponibles, es necesario plantear un algoritmo de acuerdo con ello. Al ser un problema que forma parte del tipo NP (Tiempo polinomial no determinista) pues forma parte del conjunto de problemas derivadas del problema del vendedor viajero, no puede ser resuelta mediante algoritmos exactos, por el hecho de que estos son computacionalmente costosos y no responden a las restricciones de carácter temporal para instancias grandes del problema (Irnich et al., 2014) Asimismo, existen algoritmos no exactos como lo son los heurísticos y entre ellos se encuentran algoritmos voraces como primero el mejor o GRASP, las cuales optan por la mejor solución a corto plazo sin considerar que esta puede ser una solución subóptima (Pinninghoff et al., 2014). Sin

embargo, estos tipos de algoritmos para problemas complejos con varias restricciones suelen quedarse atrapados en óptimos locales. Por otro lado, se encuentran los algoritmos metaheurísticos que, a diferencia de los heurísticos, estos logran salir de óptimos locales gracias a ciertas estrategias propias de este tipo de algoritmos y de acuerdo con D. Fogel este tipo de algoritmos han demostrado ser mejores que los algoritmos heurísticos pues combinan elementos del dominio del problema y repiten los procesos miles de veces (Fogel, 1995). Por ello, es importante emplear algoritmos que sean de rápida ejecución y que brinde buenas soluciones al problema como lo son los algoritmos metaheurísticos.

Una gran parte de estos algoritmos metaheurísticos hacen uso de los conceptos de inteligencia artificial y para el problema de ruteo de vehículos por lo revisado en el estado del arte, se emplean principalmente algoritmos de inteligencia de enjambre y bioinspirado. Entre los diversos estudios encontrados, los que más resaltan por los buenos resultados que brindan para resolver el problema son los algoritmos genéticos (Kamphukaew et al., 2018; Kurniawan et al., 2015; Pinninghoff et al., 2014), el algoritmo colonia de abejas (Brajevic, 2011; Ng et al., 2017; Punriboon et al., 2019; D. Zhang et al., 2018), el algoritmo enjambre de partículas (Akhand, Zahrul Jannat Peya, et al., 2016; Kao et al., 2012) y el algoritmo colonia de hormigas (Bouhafs et al., 2004; Giovanni Calabrò et al., 2020a; Mutar et al., 2020). La principal diferencia entre estos algoritmos es que los algoritmos genéticos están basados en la competencia y evolución, mientras que los siguientes tres están basados en la cooperación de agentes inteligentes. No obstante, inicialmente el algoritmo de colonia de abejas y el enjambre de partículas fueron planteadas para problemas con espacios de estados continuos, mientras que la colonia de hormigas para problemas discretos. Por ello, al estar optimizado para estados discretos, este algoritmo inspirado en las hormigas es perfecto para este problema pues toma en cuenta las distintas variables y restricciones del problema para encontrar una solución óptima. Asimismo, como parte del trabajo el algoritmo colonia de hormigas va a ser comparado con el algoritmo primero el mejor (Best first search o BFS por sus siglas en inglés) dado que este servirá como una línea base a partir de la cual se pueda conocer la optimalidad de los resultados obtenidos y verificar que realmente este no se queda atrapado en óptimos locales.

1.1.3 Problema seleccionado

Por todo lo expuesto en la sección anterior, el problema central que ha sido seleccionado para el presente proyecto es la inexistencia de un algoritmo colonia de

hormigas para la definición de una ruta de abastecimiento en situaciones de emergencia orientada a la realidad peruana.

1.2 Objetivos

En esta sección se presentarán los objetivos del presente proyecto de tesis, la cual abarca el objetivo general y los objetivos específicos de la tesis que se pretenden alcanzar.

1.2.1 Objetivo general

Implementar el algoritmo colonia de hormigas para dar solución al problema de ruteo de vehículos con capacidades en situaciones de emergencia con el fin de definir una ruta de abastecimiento orientada a la realidad del Perú.

1.2.2 Objetivos específicos

O1. Identificar y definir las estructuras de datos que soporten las variables y restricciones que necesita el algoritmo.

O2. Diseñar y codificar un algoritmo de colonia de hormigas para resolver el problema de ruteo de vehículos capacitados orientado a situaciones de emergencia.

O3. Comparar el desempeño del algoritmo colonia de hormigas contra un algoritmo de ruteo voraz.

1.2.3 Resultados esperados

O1. Resultados asociados al primer objetivo.

R1. Diseño y definición de la estructura de datos para los nodos, aristas y el grafo que representan los puntos a repartir, las rutas y el área de distribución.

O2. Resultados asociados al segundo objetivo.

R2. Diseño de la función objetivo a tomar en cuenta para trazar las rutas de distribución.

R3. Diseño de las funciones que emplean las feromonas para el algoritmo colonia de hormigas.

R4. Diseño de las reglas de transición para el problema.

R5. Diseño y codificación del algoritmo colonia de hormigas.

R6. Interfaz para la ejecución del algoritmo colonia de hormigas.

O3. Resultados asociados al tercer objetivo.

R7. Implementación del algoritmo primero el mejor.

R8. Calibración de parámetros de los algoritmos.

R9. Experimentación numérica de evaluación de los algoritmos primero el mejor y colonia de hormigas.

R10. Ejecución del algoritmo colonia de hormigas en una instancia real del problema.

1.2.4 Mapeo de objetivos, resultados y verificación

Tabla 1. Mapeo de objetivos, resultados y verificación del primer objetivo

Objetivo: Identificar y definir las estructuras de datos que soporten las variables y restricciones que necesita el algoritmo.		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**
R1. Diseño y definición de la estructura de datos para los nodos, aristas y el grafo que representan los puntos a repartir, las rutas y el área de distribución.	Documento de definición de las estructuras de datos.	Validado al 100% por un especialista en algoritmia.
	Archivos con el código de la implementación de las estructuras de datos	

Tabla 2. Mapeo de objetivos, resultados y verificación del segundo objetivo

Objetivo: Diseñar y codificar un algoritmo de colonia de hormigas para resolver el problema de ruteo de vehículos capacitados orientado a situaciones de emergencia.		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**
R2. Definición y diseño de la función objetivo a tomar en cuenta para trazar las rutas de distribución.	Documento de definición de la función objetivo.	Validado al 100% por un especialista en algoritmia.
	Documento con el pseudocódigo de la implementación de la función objetivo**.**	
R3. Diseño de las funciones que emplean las feromonas para el	Documento del diseño de la función de evaporación, depósito	Validado al 100% por un especialista en algoritmia.

algoritmo colonia de hormigas.	y de probabilidad de transición.	
	Documento con el pseudocódigo de la implementación de las funciones.	
R4. Diseño de las reglas de transición para el problema.	Documento del diseño de las reglas de transición	Validado al 100% por un especialista en algoritmia.
R5. Diseño y codificación del algoritmo colonia de hormigas.	Documento con el pseudocódigo del algoritmo colonia de hormigas validado por un especialista en algoritmos.	Verificación del 100% del código del algoritmo mediante pruebas de caja blanca.
	Archivos con el código de la implementación del algoritmo colonia de hormigas.	
	Documento con el resultado de las pruebas realizadas empleando las estructuras de datos definidas anteriormente.	
R6. Interfaz simple para la ejecución del algoritmo colonia de hormigas.	Archivos con el código de la implementación de la interfaz.	Pruebas de integración realizadas al 100% de la interfaz con el algoritmo.
	Documento con las pruebas unitarias de la interfaz desarrollada.	

Tabla 3. Mapeo de objetivos, resultados y verificación del tercer objetivo

Objetivo: Comparar el desempeño del algoritmo colonia de hormigas contra un algoritmo de ruteo voraz.		
Resultado	**Medio de verificación**	**Indicador objetivamente verificable**

R7. Implementación del algoritmo primero el mejor.	Documento con el pseudocódigo del algoritmo primero el mejor. Archivos con el código del algoritmo primero el mejor.	Validación al 100% con pruebas de caja blanca
R8. Calibración de parámetros de los algoritmos.	Documento con los resultados de la calibración.	Verificación del documento al 100% de la calibración por un especialista en algoritmia.
R9. Experimentación numérica de evaluación de los algoritmos primero el mejor y colonia de hormigas.	Informe con las pruebas realizadas para la experimentación numérica, prueba de dos tratamientos.	Verificación al 100% de los resultados de la experimentación numérica, prueba de dos tratamientos.

1.3 Métodos y Procedimientos

En esta sección se presentan para cada resultado una herramienta o método, si aplican, para conseguir dicho resultado. También se describe de modo resumido las herramientas o métodos y cómo se aplicarán en el presente proyecto.

Tabla 4. Herramientas, métodos y procedimientos

Objetivo específico	Resultado esperado	Herramientas	Métodos y procedimientos
O1	**R1** Diseño y definición de la estructura de datos para los nodos, aristas y el grafo que representan los puntos a repartir, las rutas y las áreas de distribución	- Pseudocódigo - Java - IDE IntelliJ IDEA - Git	- No aplica
O2	**R2** Diseño de la función objetivo a tomar en cuenta para trazar las rutas de distribución		- Formulación matemática
	R3 Diseño de la función de evaporación de la feromona para el algoritmo colonia de hormigas		- Formulación matemática
	R4 Diseño de las reglas de transición para el problema		- Formulación matemática
	R5 Diseño y codificación del algoritmo colonia de hormigas		- Scrum - Pruebas unitarias
	R6 Interfaz para la ejecución del		- Scrum - Pruebas unitarias

	algoritmo colonia de hormigas		
O3	**R7** Implementación del algoritmo primero el mejor		- Scrum - Pruebas unitarias
	R8 Documento de experimentación numérica de evaluación de los algoritmos primero el mejor y colonia de hormigas	- Python - Jupyter Notebooks - Git	- Experimentación con dos tratamientos y un solo factor.

1.3.1 Herramientas

Pseudocódigo

El pseudocódigo es una forma estructurada de describir un algoritmo a alto nivel con la finalidad de que una persona pueda entender qué y cómo se va a ejecutar un algoritmo de tal manera que su entendimiento sea sencillo (Vazquez G, 2012). En este proyecto se emplea esta herramienta para el diseño de los algoritmos y funciones.

Java

Java es un lenguaje de programación orientado a objetos e imperativo que se usa para distintos tipos de aplicaciones. Además, este lenguaje de programación es fácil de usar e incorpora lo mejor de otros lenguajes de programación para mejorar la experiencia del desarrollo de programas (Oracle, n.d.). Por ello, para el presente proyecto de tesis se ha considerado que el lenguaje de programación Java es la mejor opción para la codificación del algoritmo.

IntelliJ IDEA

IntelliJ IDEA es un entorno de desarrollo integrado para el desarrollo de programas en distintos lenguajes de programación y fue desarrollada por la compañía JetBrains. Este IDE fue desarrollado para maximizar la productividad pues cuenta con distintas herramientas como el autocompletado inteligente, asistencia sobre la documentación de los lenguajes de programación o su integración con herramientas de control de

versiones (JetBrains, n.d.). Estas características hacen que este entorno de desarrollo sea el ideal junto con Java para el desarrollo del algoritmo del presente proyecto.

Git

Git es una herramienta de código abierto para el control de versiones de código fuente diseñado para manejar desde proyectos pequeños hasta muy grandes con rapidez y eficiencia (Chacon & Straub, 2014). Dado que el diseño del algoritmo y sus diversas funciones se trabajarán por etapas es conveniente trabajar con esta herramienta.

Python

Python es un lenguaje de programación multiparadigma de código abierto multipropósito. Este cuenta con una gran cantidad de librerías para diversas aplicaciones, desde la realización de interfaces gráficas hasta el aprendizaje máquina. Entre todas estas librerías se encuentran las librerías para la manipulación de datos y el análisis estadístico el cual cuenta con distintas funciones para realizar estos tipos de análisis y cálculos (Python Software Foundation, n.d.). Este lenguaje de programación es el ideal para realizar los cálculos necesarios de la experimentación numérica que se va a realizar en el presente proyecto.

Jupyter Notebooks

Jupyter Notebook es un entorno de desarrollo integrado basado en el web especialmente diseñado para la ciencia de datos, computación científica y el aprendizaje máquina empleando el lenguaje de programación Python. Este cuenta con herramientas de autocompletado, depuración y herramientas de visualización de datos (Project Jupyter, n.d.). Por ello, este entorno de desarrollo es perfecto para la elaboración de la experimentación numérica del presente proyecto de tesis.

1.3.2 Métodos y procedimientos

Árbol de decisiones

Un árbol de decisiones es un modelo jerárquico de decisiones y sus consecuencias, las cuales son usadas para definir reglas que se van a llevar a cabo bajo ciertas condiciones (Rokach & Maimon, 2007). En el presente trabajo, se va a elaborar un modelo de árbol de decisiones para definir las reglas de transición que van a ser empleados para resolver el problema.

Scrum

Scrum es un marco de trabajo para desarrollar, entregar y mantener productos complejos, este consiste en una serie de iteraciones incrementales con ciertas actividades a realizarse que agregan valor al producto (Schwaber & Sutherland, 2017). Además, esta metodología de trabajo es flexible y se puede adaptar a casi cualquier proyecto. Es por estas características que se van a emplear ciertas prácticas de esta metodología para el desarrollo de los algoritmos de este trabajo de tesis.

Pruebas unitarias

La metodología de pruebas unitarias consiste en elaborar pruebas a una unidad separada del programa en el caso estas unidades presentan cierta independencia, de no ser así, las pruebas se realizan al conjunto relacionado de unidades, las cuales pueden ser funciones o procedimiento (Lindberg & Strandberg, 2006). Este tipo de pruebas ayudan a localizar posibles errores y a evitar problemas de integración con las distintas unidades del programa (Lindberg & Strandberg, 2006). Por ello, para el presente trabajo se va a emplear esta metodología para probar las distintas funciones y procedimientos del algoritmo.

Experimentación con dos tratamientos y un solo factor

Este procedimiento de experimentación numérica busca comparar parámetros como la media o la varianza de dos procesos o tratamientos para conocer si es que existe diferencia alguna o no. Esta, generalmente, se realiza bajo ciertos criterios y suposiciones dependiendo de los datos que se tengan (Pulido & Salazar, 2008).

Un factor es aquella característica del proceso que tiene un impacto sobre el producto final, las cuales pueden clasificarse como factores controlables, no controlables y estudiados. (Pulido & Salazar, 2008). Estos factores tienen un rango de valores denominados niveles de un factor, es decir, un factor puede tomar distintos valores de acuerdo a un rango y con ello se puede elaborar un plan de experimentación para conocer qué tanto influye sobre el producto final (Pulido & Salazar, 2008).

En el caso del presente proyecto se van a comparar dos algoritmos y se va a considerar un factor por lo que emplear este procedimiento es útil para conocer qué algoritmo es mejor en cuanto a la optimización de la función objetivo.

ET LUX IN TENEBRIS LUCET
MCMXVII

Capítulo 2.Marco Conceptual

2.1 Introducción

El presente capítulo tiene como principal objetivo definir los conceptos relacionados al tema de tesis para poder entender la problemática que este abarca. Por ello, se presentará, describirá y se ejemplificaran los conceptos acerca del tipo de problema, el problema principal con la variante que es el tema de estudio y conceptos sobre la propuesta de solución. De esta manera, se busca lograr un entendimiento sobre los conceptos clave que se van a tratar en el proyecto de tesis.

2.2 Desarrollo del marco

2.2.1 Situaciones de desastres

Los desastres naturales son aquellos eventos que ocurren en la naturaleza como lo son los terremotos, huracanes, erupciones volcánicas, tsunamis, etc. que causan daños materiales y pérdidas vidas humanas (World Banks & United Nations, 2010).

Por ejemplo, en el estudio realizado por Wei Yi y Arun Kumar busca brindar ayuda ante un desastre natural como un terremoto y para ello hacen uso del algoritmo colonia de hormigas para plantear rutas de evacuación (Yi & Kumar, 2007). Otro claro ejemplo de este tipo de situaciones es el desastre causado por un huracán como se menciona en el estudio realizado por Yinglei Li y Sung Hoon Chung, en la cual proponen brindar rutas de abastecimiento de suministros (Li & Chung, 2019).

En el presente proyecto se revisarán los distintos estudios que abordan el problema de ruteo de vehículos en situaciones de desastres para su posterior análisis sobre cómo afrontar el problema y qué solución se propone.

2.2.2 Situaciones de pandemias

De acuerdo a la Organización mundial de la salud se conoce a una situación de pandemia como “la propagación mundial de una nueva enfermedad” (Organización Mundial de la Salud, 2010). De acuerdo con la definición de la Real Academia Española, una pandemia es “una enfermedad que se expande de forma intensa e indiscriminada a muchos países o que ataca a casi todos los individuos de una localidad o región” (Real Academia Española, n.d.).

Por ejemplo la gripe de Hong Kong, que surgió durante 1968, se expandió alrededor del mundo y dejó a un millón de víctimas; la gripe española, que surgió en 1914, llegó a todo el mundo y dejó alrededor de 20 millones de personas muertas (Pané, 2020).

En el presente trabajo, el problema a resolver será bajo un contexto de pandemia como lo es el causado por el COVID-19, en el cual se tomará en cuenta las variables que surgen durante este escenario de emergencia.

2.2.3 Métodos exactos y metaheurísticos

Por un lado, los métodos exactos para resolver problemas de optimización combinatoria se basan, fundamentalmente, en la programación lineal, en el cual se emplean funciones e inecuaciones lineales que deben ser resueltas para dar solución a los problemas (Toth & Vigo, 2001). Por ejemplo, el algoritmo de ramificación y poda emplea este método para recorrer el dominio del problema y así encontrar la solución óptima (Baldacci et al., 2012).

Por otro lado, los métodos metaheurísticos son empleados para resolver problemas de alta complejidad que requieren una exploración exhaustiva y guiada a través de los distintos estados del problema con un cierto grado de aleatoriedad para encontrar una solución óptima o cercana a ella (Luke, 2013). Entre los algoritmos que emplean este método está las colonias de abeja (Brajevic, 2011; Punriboon et al., 2019; D. Zhang et al., 2018; S. Z. Zhang & Lee, 2016a), las colonias de hormiga (Bouhafs et al., 2004; G Calabrò et al., 2020; Mingping Xia, 2009; Mutar et al., 2020; Stodola et al., 2014; Yamina et al., 2013) o el recocido simulado (Wei et al., 2018).

2.2.4 Inteligencia de enjambre

La definición de inteligencia de enjambre fue propuesta por primera vez en el libro de Inteligencia de enjambre en sistemas robóticos celulares, en el cual se definía como “una serie de agentes que evidenciaban un comportamiento colectivo capaces de producir patrones ordenados específicos” (Beni & Wang, 1993). Esta definición fue afianzándose a lo largo de los años y en el 2014 propone como una primera definición general a un “enjambre de agentes, ya sea biológica o artificial que, sin un control central y de forma colectiva, llevan a cabo tareas que normalmente requiere alguna forma de inteligencia” (Beni, 2014). Esta forma de organizarse sin un control centralizado es una gran ventaja sobre sistemas centralizados, pues los agentes, al ser similares unos a otros, son sencillos, intercambiables y desechables, con ello logran adaptarse para resolver problemas de alta complejidad (Beni, 2014).

La inteligencia de enjambre posee una gran variedad de aplicaciones, una de ellas es la resolución de problemas de optimización combinatoria. Por ejemplo, algunos enjambres que se encuentran en la naturaleza y que cumplen con la definición de inteligencia de enjambre son las colonias de hormigas (Dorigo & Stützle, 2004), las bandadas de aves (Akhand, Peya, et al., 2016) o los enjambres de abeja (Brajevic, 2011).

El proyecto en cuestión emplea el algoritmo colonia de hormigas, la cual es un tipo de inteligencia de enjambre para dar solución al problema de ruteo de vehículos con capacidad limitada en situaciones de emergencia.

2.2.5 Agentes en inteligencia artificial

Un agente es aquel que puede percibir su entorno a través de sensores como cámaras infrarrojas, sensores de temperatura, etc. y actúa sobre su entorno a través de actuadores como brazos mecánicos, pistones, parlantes, etc. (Russell & Norvig, 2009). Un agente inteligente, emplea lo percibido por sus sensores para poder tomar decisiones racionales sobre las acciones que va a realizar para poder alcanzar un objetivo, es decir, las acciones que tiene planeado realizar sean las adecuadas para lograr su objetivo (Russell & Norvig, 2009).

Un ejemplo de agente inteligente es un sistema clasificador de frutas, el cual tiene como sensor una cámara y como actuador un brazo mecánico. El sistema percibe imágenes, que luego son procesadas para identificar el estado de la fruta. Con dicha información puede tomar la decisión de separarla y colocarla en la caja de frutas de mal estado o dejarla pasar. Este es un ejemplo sencillo de un agente, pero para problemas más complejos en el que el agente no conozca su entorno y que tenga que aprender sobre la marcha, tomar decisiones de forma racional se vuelve más complicado la labor de estos agentes.

En el proyecto, el algoritmo colonia de hormigas emplea como agentes simples a las hormigas, las cuales en conjunto forman un tipo de agente inteligente para poder solucionar problemas.

2.2.6 Algoritmo colonia de hormigas

El algoritmo colonia de hormigas es un algoritmo metaheurístico de inteligencia colectiva inspirado en el comportamiento de las hormigas, las cuales son capaces de encontrar el camino más corto entre su comida y la colonia mediante el uso de

feromonas (Du & Swamy, 2016). La idea de emplear agentes simples como las hormigas para poder resolver el problema de encontrar la mejor ruta desde un punto a otro, fue introducida por primera vez en 1991 por M. Dorigo, V. Maniezzo y A. Colorni (Dorigo et al., 1991).

La manera en la que estos insectos ciegos logren encontrar una ruta óptima parte del hecho de que no solo actúan por sí solas, sino que comparten la información con las otras hormigas dejando feromonas sobre el camino que han encontrado y con dicha información sus compañeras incrementan o disminuyen sus probabilidades de ir por ese camino (Dorigo & Stützle, 2004).

Los algoritmos de colonia de hormigas son empleados, en su gran mayoría, para resolver problemas de optimización combinatoria (Pavón Mariño, 2015). Algunos ejemplos de casos de estudios que emplean este algoritmo son los siguientes:

- Improving inbound logistic planning for large-scale real-world routing problems: a novel ant-colony simulation-based optimization (Giovanni Calabrò et al., 2020b).
- A hybrid ant colony system approach for the capacitated vehicle routing problem (Bouhafs et al., 2004).

El tema de estudio de este proyecto de tesis es el algoritmo colonia de hormigas para resolver el problema de ruteo de vehículos con capacidad limitada.

2.2.7 Complejidad algorítmica

La complejidad algorítmica está definida como la cantidad de recursos computacional como tiempo y espacio que requiere un algoritmo para resolver un problema, lo cual es importante tomar en cuenta pues, por ejemplo, tener un algoritmo que tarde un año en dar una solución o que ocupe toda la memoria para dar solución a un problema no es de gran ayuda (Weiss, 1994).

2.2.8 Problema de optimización combinatoria

Los problemas optimización combinatoria en el área de las ciencias de la computación son aquellos problemas de gran complejidad, las cuales poseen un espacio de soluciones muy grande y encontrar la solución con métodos tradicionales (algoritmos exactos) es muy complicado o hasta imposible por el alto grado de costo computacional que este implica (Xue, 2001).

Este campo de estudio está altamente relacionado con los problemas que comúnmente se presentan, por ejemplo, en temas de ruteo de vehículos, asignación de tareas u optimización de tiempos, en las cuales se busca minimizar, maximizar o equilibrar algunas de las variables de este problema (Cook et al., 1997).

El presente proyecto busca plantear y resolver uno de estos problemas de optimización combinatoria: el problema de ruteo de vehículos con capacidad limitada en situaciones de emergencia mediante el uso del algoritmo colonia de hormigas.

2.2.9 Problemas de complejidad P y NP

Los problemas de complejidad computacional presentan varias clases de complejidad, entre ellas están los problemas de complejidad tipo P, la cual requiere un tiempo de computación de tipo polinomial. Otro tipo de problemas son de complejidad tipo NP este requiere un tiempo de computación polinomial no determinista, es decir, no se puede reducir con certeza a un problema polinomial (Arvind, 2014; Papadimitriou & Steiglitz, 1982).

Algunos ejemplos de problemas de complejidad P son los problemas de programación lineal o el problema de determinar si un número es primo. Mientras que problemas como el problema del vendedor viajero o el problema de la mochila son de complejidad NP.

En el presente estudio el problema presentado tiene una complejidad de tipo NP, pues es un derivado con más restricciones del problema del vendedor viajero.

2.2.10 Problema de ruteo de vehículos

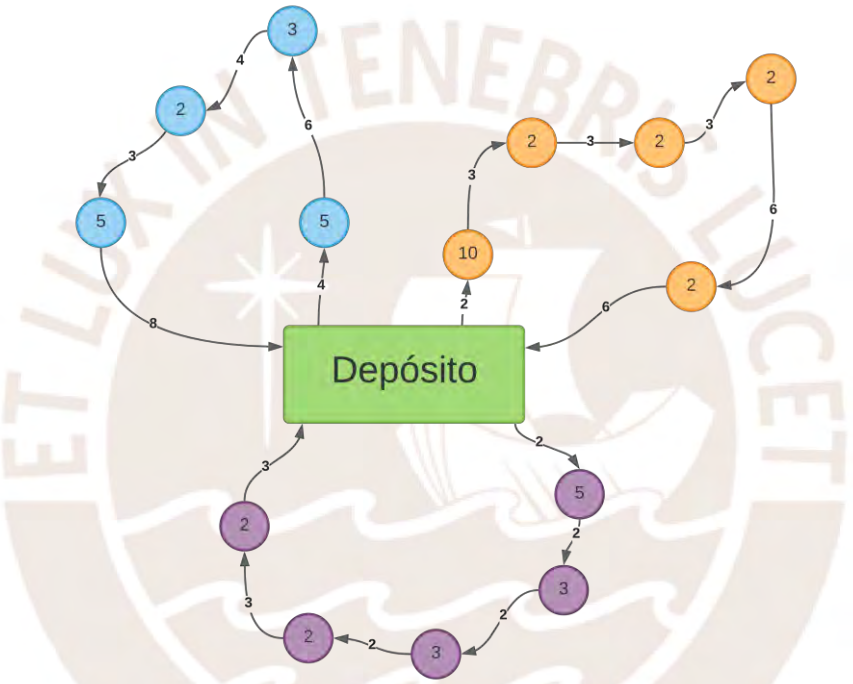
El problema de ruteo de vehículos (VRP por sus siglas en inglés) consiste en encontrar la ruta óptima que debe realizar un vehículo o una flota de estas para poder servir a un grupo de clientes (Irnich et al., 2014). Este es uno de los problemas de optimización combinatorio más importantes y estudiados, pues tiene una gran cantidad de aplicaciones en el mundo real, como el que fue planteado por primera vez en 1959 por Dantzig y Ramser para el transporte de combustible a distintas gasolineras (Dantzig & Ramser, 1959).

Asimismo, el problema de ruteo de vehículos para adecuarse a ciertas situaciones reales se tuvo que agregar al problema restricciones, limitaciones o condiciones y todo ello afecta a la función objetivo a optimizar. De esta manera, surgieron las variantes al

problema de ruteo de vehículos y muchas de estas son aún más complejas que el problema inicial (Pereira & Tavares, 2009).

Una de estas variantes es el problema de ruteo de vehículos capacitados o CVRP por sus siglas en inglés. Este problema trata de realizar entregas a ciertos puntos que poseen una determinada demanda tomando en cuenta la capacidad de los vehículos que se tiene y que, además, se debe tomar en cuenta el costo de la ruta (Carwalo et al., 2017; Irnich et al., 2014).

Figura 2. Ejemplo de solución a un CVRP



En la Figura 2 se puede observar un ejemplo de una solución de un problema de ruteo de vehículos capacitados, en el cual se presentan tres rutas representadas por nodos de diferentes colores. Cada ruta tiene asignada un costo de recorrido, la cual se ve en las aristas y una determinada demanda que se encuentra en el nodo.

Asimismo, una de las aplicaciones del problema de ruteo de vehículos de capacidades homogéneas es para el alivio de desastres naturales o situaciones de emergencia. Esta aplicación en particular tiene como principal objetivo el bienestar de las personas durante una situación de emergencia y para ello se deben ejecutar planes de rescate o de envío de suministros como comida y medicinas a las zonas afectadas (Irnich et al., 2014).

El presente proyecto está enfocado a brindar una solución al problema de ruteo de vehículos con capacidades homogéneas en situaciones de emergencia como lo es una pandemia.

Capítulo 3.Estado del Arte

3.1 Introducción

En este capítulo se presenta el tipo de revisión literaria y los resultados obtenidos. La metodología de revisión empleada es la revisión sistemática propuesta por Kitchenham y Charters. Este tipo de revisión sugiere seguir tres etapas: el planeamiento de la revisión, la ejecución de la revisión y, por último, reportar los resultados obtenidos (Kitchenham, 2007). Este proceso se realiza para poder identificar, evaluar e interpretar las investigaciones que sean relevantes para el proyecto de tesis.

3.2 Objetivos de revisión

Para esta revisión los tipos de revisión que se realizará es del estado del arte, teórico y empírico. A continuación, se presentan los objetivos que se esperan lograr al finalizar la revisión de la literatura.

- Conocer y entender los conceptos relacionados al problema de ruteo de vehículos con diferentes capacidades y el algoritmo inteligencia de enjambre, haciendo hincapié en los algoritmos colonia de hormigas para resolver problemas de optimización combinatoria de ruteo y reparto.
- Determinar los estudios relevantes sobre el problema de ruteo y reparto con vehículos de diferentes características y sus variantes, para casos de situaciones de emergencia o catástrofe, que empleen inteligencia de enjambre.
- Identificar las ventajas y desventajas del empleo del uso de los algoritmos de la familia inteligencia de enjambre frente a otros algoritmos para el problema de ruteo y reparto con vehículos de diferentes características.

3.3 Preguntas de revisión

El objetivo principal de la revisión sistemática es orientar metódicamente la búsqueda hacia el tema del proyecto de tesis. Por tal motivo, se elaboraron una serie de preguntas bajo la metodología PICOC. Esta manera de realizar las preguntas plantea que está compuesta por cinco componentes importantes. De esta manera, se plantea la siguiente tabla para poder estructurar las preguntas.

Tabla 5. Elementos PICOC

Criterio	Responde a	Descripción
Población	¿Quién?	Problemas de ruteo de vehículos con capacidades.
Intervención	¿Qué? o ¿Cómo?	Algoritmo colonia de hormigas para resolver problemas de optimización combinatoria de ruteo.
Comparación	¿Comparado con qué?	Comparar los métodos de solución de las distintas propuestas de formulación del algoritmo colonia de hormigas.
Resultados	¿Qué se está tratando de lograr o mejorar?	Algoritmos que brinden soluciones al problema de optimización combinatorio de ruteo de vehículos con capacidades homogéneas para situaciones de emergencia.
Contexto	¿En qué tipo de organización/ circunstancia?	Académico Industrial Empresarial

Con la ayuda de la tabla se formularon las siguientes preguntas:

- P1. ¿Qué tipos de algoritmos están siendo usados para resolver el problema de ruteo de vehículos con diferentes capacidades y cuáles son sus características?
- P2. ¿De qué manera son aplicados los algoritmos de inteligencia colectiva o bioinspirado para resolver el problema de ruteo con vehículos?
- P3. ¿De qué manera los algoritmos colonia de hormigas pueden resolver de manera óptima el problema de ruteo y reparto con vehículos de diferentes capacidades?
- P4. ¿De qué manera se han aplicado las soluciones al problema de ruteo de vehículos para situaciones de emergencia?

3.4 Estrategia de búsqueda

3.4.1 Motores de búsqueda a usar

Los motores de búsqueda que se emplearán para realizar la revisión de la literatura, por recomendación de los asesores son los siguientes:

- SCOPUS
- IEEE XPLORE
- ACM DL

3.4.2 Cadenas de búsqueda a usar

Parte de la metodología PICOC es identificar las palabras relacionadas que se van a usar en las cadenas de búsqueda (Ver Tabla 6).

Tabla 6. Palabras relacionadas a los elementos PICOC

Valor	Palabras relacionadas
Población	• CVRP • C-VRP • Capacitated vehicle routing problem
Intervención	• Ant colony optimization • Ant colony • ACO • Ant systems • Disaster relief • Disaster response • Emergency relief • Emergency response
Resultados	• Algorithm

Las cadenas de búsquedas que se plantean por pregunta y por el de motor de búsqueda que se va a emplear. Así, para cada pregunta se presentarán tres cadenas de búsquedas en las tablas 7, 8, 9 y 10.

Tabla 7. Cadenas de búsqueda para la primera pregunta

Motor de búsqueda	**P1. ¿Qué tipos de algoritmos están siendo usados para resolver el problema de ruteo de vehículos con diferentes capacidades?**
SCOPUS	TITLE-ABS-KEY ("algorithm" AND ("cvrp" OR "c-vrp" OR "CVRP" OR "C-VRP" OR "capacitated vehicle routing problem"))
IEEE XPLORE	"algorithm" AND ("cvrp" OR "c-vrp" OR "CVRP" OR "C-VRP" OR "capacitated vehicle routing problem")
ACM DL	[All: "algorithm"] AND [[All: "cvrp"] OR [All: "c-vrp"] OR [All: "cvrp"] OR [All: "c-vrp"] OR [All: "capacitated vehicle routing problem"]]

Tabla 8. Cadenas de búsqueda para la segunda pregunta

Motor de búsqueda	**P2. ¿De qué manera son aplicados los algoritmos de inteligencia colectiva o bioinspirado para resolver el problema de ruteo con vehículos?**
SCOPUS	TITLE-ABS-KEY (("swarm intelligence" OR "collective intelligence" OR "bio-inspired" OR "bioinspired") AND ("cvrp" OR "c-vrp" OR "CVRP" OR "C-VRP" OR "capacitated vehicle routing problem"))
IEEE XPLORE	("swarm intelligence" OR "collective intelligence" OR "bio-inspired" OR "bioinspired") AND ("cvrp" OR "c-vrp" OR "CVRP" OR "C-VRP" OR "capacitated vehicle routing problem")
ACM DL	[All: "swarm intelligence"] AND [[All: "cvrp"] OR [All: "c-vrp"] OR [All: "cvrp"] OR [All: "c-vrp"] OR [All: "capacitated vehicle routing problem"]]

Tabla 9. Cadenas de búsqueda para la tercera pregunta

Motor de búsqueda	**P3. ¿De qué manera los algoritmos colonia de hormigas pueden resolver de manera óptima el problema de ruteo y reparto con vehículos de diferentes capacidades?**
SCOPUS	TITLE-ABS-KEY (("ant colony" OR "ant colony optimization" OR "ACO") AND ("cvrp" OR "c-vrp" OR "CVRP" OR "C-VRP" OR "capacitated vehicle routing problem"))
IEEE XPLORE	("ant colony" OR "ant colony optimization" OR "ACO") AND ("cvrp" OR "c-vrp" OR "CVRP" OR "C-VRP" OR "capacitated vehicle routing problem")

ACM DL	[[All: "ant colony"] OR [All: "ant colony optimization"] OR [All: "aco"]] AND [[All: "cvrp"] OR [All: "c-vrp"] OR [All: "cvrp"] OR [All: "c-vrp"] OR [All: "capacitated vehicle routing problem"]]

Tabla 10. Cadenas de búsqueda para la cuarta pregunta

Motor de búsqueda	**P4. ¿De qué manera se han aplicado las soluciones al problema de ruteo de vehículos para situaciones de emergencia?**
SCOPUS	TITLE-ABS-KEY (("bioinspired" OR "collective intelligence" OR "swarm intelligence" OR "ant colony" OR "ant colony optimization" OR "ACO") AND ("disaster relief" OR "emergency relief"))
IEEE XPLORE	("bioinspired" OR "collective intelligence " OR "swarm intelligence" OR "ant colony" OR "ant colony optimization" OR "ACO") AND ("disaster relief" OR " emergency relief ")
ACM DL	[[All: "bioinspired"] OR [All: "collective intelligence "] OR [All: "swarm intelligence"] OR [All: "ant colony"] OR [All: "ant colony optimization"] OR [All: "aco"]] AND [[All: "disaster relief"] OR [All: " emergency relief "]]

3.4.3 Documentos encontrados

Luego de realizar la búsqueda con las cadenas propuestas se obtuvieron los resultados que se muestran en la tabla 11.

Tabla 11. Número de resultados de las búsquedas

Motor	P1	P2	P3	P4
SCOPUS	898	33	101	49
IEEE XPLORE	151	8	35	8
ACM DL	94	13	27	43

Luego, se importaron todos los estudios encontrados a Mendeley, la cual es una herramienta de gestión de referencias. Esta herramienta se empleó para filtrar los estudios duplicados por cada pregunta y los resultados obtenidos se muestran en la tabla 12.

Tabla 12 Número de duplicados por pregunta de las búsquedas

	P1	P2	P3	P4
Número de duplicados	52	20	8	0

Después se descartaron los estudios de acuerdo con los criterios de inclusión y exclusión listados en el punto 3.4.4. En este punto se encontraron distintos estudios de optimización de los algoritmos en hardware, estudios de optimización de los parámetros de los algoritmos, métodos de benchmarking para los algoritmos, entre otros temas que fueron excluidos. Los resultados de aplicar los criterios de inclusión/exclusión se muestran en la tabla 13.

Tabla 13. Número de referencias excluidas por los criterios de exclusión e inclusión

	P1	P2	P3	P4
Número de excluidos	774	43	68	94
Quedan	207	60	92	6

Durante dicha revisión se encontraron algunos estudios duplicados entre las preguntas, es decir, algunos resultados obtenidos de la pregunta 1 contenía resultados de la pregunta 2 pues la primera es una pregunta general que contiene a la pregunta 2. Lo mismo ocurrió con la pregunta 3 y 4, por lo que se procedió a descartarlos y se terminó con 14 documentos para la pregunta 1, 16 para la pregunta 2, 11 para la pregunta 3 y 5 para la pregunta 4.

En la tabla 14 se muestran los estudios encontrados y seleccionados.

Tabla 14 Documentos relevantes seleccionados

ID	Referencia (formato APA)
E000	Gomes, L. D. C. T., Von Zuben, F. J., de CT Gomes, L., & Von Zuben, F. J. (2002). A neuro-fuzzy approach to the capacitated vehicle routing problem. *Proceedings of the International Joint Conference on Neural Networks*, *2*, 1930–1935. https://doi.org/10.1109/IJCNN.2002.1007814
E001	Khebbache, S., Prins, C., Yalaoui, A., & Reghioui, M. (2009). Memetic Algorithm for two-dimensional loading capacitated vehicle routing problem

	with time windows. *2009 International Conference on Computers Industrial Engineering*, 1110–1113. https://doi.org/10.1109/ICCIE.2009.5223786
E002	Calabrò, G., Torrisi, V., Inturri, G., & Ignaccolo, M. (2020). Improving inbound logistic planning for large-scale real-world routing problems: a novel ant-colony simulation-based optimization. *European Transport Research Review*, *12*(1). https://doi.org/10.1186/s12544-020-00409-7
E003	Ren, C. Y. (2013). Tabu search algorithm for for capacitated vehicle routing problem. *Advanced Materials Research*, *753–755*, 3060–3063. https://doi.org/10.4028/www.scientific.net/AMR.753-755.3060
E004	Pinninghoff, M. A., Contreras, R., & Pantoja, C. (2014). A comparison of methods for the vehicle routing problem. In D. A. Ezzatti P. (Ed.), *Proceedings of the 2014 Latin American Computing Conference, CLEI 2014*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/CLEI.2014.6965167
E005	Oliveira, R. A. de C., & Delgado, K. V. (2015). Capacitated Vehicle Routing System Applying Monte Carlo Methods. *Proceedings of the Annual Conference on Brazilian Symposium on Information Systems: Information Systems: A Computer Socio-Technical Perspective - Volume 1*, 1–8.
E006	Kurniawan, R., Sulistiyo, M. D., & Wulandari, G. S. (2016). Genetic Algorithm for Capacitated Vehicle Routing Problem with considering traffic density. *2015 International Conference on Information Technology Systems and Innovation, ICITSI 2015 - Proceedings*. https://doi.org/10.1109/ICITSI.2015.7437695
E007	Yu, V. F., Redi, A. A. N. P., Yang, C.-L., Ruskartina, E., & Santosa, B. (2017). Symbiotic organisms search and two solution representations for solving the capacitated vehicle routing problem. *Applied Soft Computing Journal*, *52*, 657–672. https://doi.org/10.1016/j.asoc.2016.10.006
E008	Khebbache, S., Prins, C., Yalaoui, A., & Reghioui, M. (2009). Memetic Algorithm for two-dimensional loading capacitated vehicle routing problem with time windows. *2009 International Conference on Computers Industrial Engineering*, 1110–1113. https://doi.org/10.1109/ICCIE.2009.5223786
E009	Nucamendi-Guillén, S., Angel-Bello, F., Martínez-Salazar, I., & Cordero-Franco, A. E. (2018). The cumulative capacitated vehicle routing problem: New formulations and iterated greedy algorithms. *Expert Systems with Applications*, *113*, 315–327. https://doi.org/10.1016/j.eswa.2018.07.025

E010	Kamphukaew, R., Sethanan, K., Jamrus, T., & Wang, H.-K. (2018). Differential evolution algorithms with local search for the multi-products capacitated vehicle routing problem with time windows: A case study of the ice industry. *Engineering and Applied Science Research*, *45*(4), 273–281. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85061856489&partnerID=40&md5=217fe71ac3cd8dc66c981b5810b21030
E011	Wei, L., Zhang, Z., Zhang, D., & Leung, S. C. H. (2018). A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, *265*(3), 843–859. https://doi.org/10.1016/j.ejor.2017.08.035
E012	Uy, C. H., Charoenlarpkul, N., Sarttra, T., & Rajsiri, S. (2019). An efficient algorithm applied to capacitated vehicle routing problem with consideration of time windows by using ranking-based concept and dynamic programming. *ACM International Conference Proceeding Series*, 267–274. https://doi.org/10.1145/3335550.3335588
E013	Niazy, N., El-Sawy, A., & Gadallah, M. (2020). A hybrid chicken swarm optimization with tabu search algorithm for solving capacitated vehicle routing problem. *International Journal of Intelligent Engineering and Systems*, *13*(4), 237–247. https://doi.org/10.22266/IJIES2020.0831.21
E014	Wang, Z., Zhou, M., Li, J., Fan, J., Zhengchu Wang, Jun Li, Muxun Zhou, & Jian Fan. (2009). Research in capacitated vehicle routing problem based on modified hybrid particle swarm optimization. *Proceedings - 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems, ICIS 2009*, *3*, 289–293. https://doi.org/10.1109/ICICISYS.2009.5358182
E015	Brajevic, I. (2011). Artificial bee colony algorithm for the capacitated vehicle routing problem. *Proceedings of the European Computing Conference, ECC '11*, 239–244. https://www.scopus.com/inward/record.uri?eid=2-s2.0-80053140490&partnerID=40&md5=9476fbe990520ba39930a42731709eac
E016	Kao, Y., Chen, M.-H., & Huang, Y.-T. (2012). A hybrid algorithm based on ACO and PSO for capacitated vehicle routing problems. *Mathematical Problems in Engineering*, *2012*. https://doi.org/10.1155/2012/726564
E017	Shan, Q., & Wang, J. (2013). Solve capacitated vehicle routing problem using hybrid chaotic particle swarm optimization. *Proceedings - 6th International Symposium on Computational Intelligence and Design, ISCID 2013*, *2*, 422–427. https://doi.org/10.1109/ISCID.2013.218
E018	Marinakis, Y., & Marinaki, M. (2013). Combinatorial Expanding Neighborhood Topology Particle Swarm Optimization for the Vehicle Routing Problem with Stochastic Demands. *Proceedings of the 15th Annual Conference on*

	Genetic and Evolutionary Computation, 49–56. https://doi.org/10.1145/2463372.2463375
E019	Korayem, L., Khorsid, M., & Kassem, S. S. (2015). Using grey Wolf algorithm to solve the capacitated vehicle routing problem. In W. J. Ding J. Gaol F.L. (Ed.), *IOP Conference Series: Materials Science and Engineering* (Vol. 83, Issue 1). Institute of Physics Publishing. https://doi.org/10.1088/1757-899X/83/1/012014
E020	Taha, A., Hachimi, M., & Moudden, A. (2015). Adapted bat algorithm for capacitated vehicle routing problem. *International Review on Computers and Software*, *10*(6), 610–619. https://doi.org/10.15866/irecos.v10i6.6512
E021	Alssager, M., & Othman, Z. A. (2016). Cuckoo search algorithm for capacitated vehicle routing problem. *Journal of Theoretical and Applied Information Technology*, *88*(1), 11–19. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84973547870&partnerID=40&md5=cb5493748f4477f679111041d2e20793
E022	Zhang, S. Z., & Lee, C. K. M. (2016). An Improved Artificial Bee Colony Algorithm for the Capacitated Vehicle Routing Problem. *Proceedings - 2015 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2015*, 2124–2128. https://doi.org/10.1109/SMC.2015.371
E023	Akhand, M. A. H., Zahrul Jannat Peya, Sultana, T., Al-Mahmud, Peya, Z. J., Sultana, T., & Al-Mahmud. (2016). Solving Capacitated Vehicle Routing Problem with route optimization using Swarm Intelligence. *2015 2nd International Conference on Electrical Information and Communication Technologies (EICT)*, 112–117. https://doi.org/10.1109/EICT.2015.7391932
E024	Ng, K. K. H., Lee, C. K. M., Zhang, S. Z., Wu, K., & Ho, W. (2017). A multiple colonies artificial bee colony algorithm for a capacitated vehicle routing problem and re-routing strategies under time-dependent traffic congestion. *Computers and Industrial Engineering*, *109*, 151–168. https://doi.org/10.1016/j.cie.2017.05.004
E025	Zhang, D., Dong, R., Si, Y.-W., Ye, F., & Cai, Q. (2018). A hybrid swarm algorithm based on ABC and AIS for 2L-HFCVRP. *Applied Soft Computing Journal*, *64*, 468–479. https://doi.org/10.1016/j.asoc.2017.12.012
E026	Yang, W., & Ke, L. (2019). An improved fireworks algorithm for the capacitated vehicle routing problem. *Frontiers of Computer Science*, *13*(3), 552–564. https://doi.org/10.1007/s11704-017-6418-9
E027	Yesodha, R., & Amudha, T. (2019). An Improved Firefly Algorithm for Capacitated Vehicle Routing Optimization. *Proceedings - 2019 Amity*

	International Conference on Artificial Intelligence, AICAI 2019, 163–169. https://doi.org/10.1109/AICAI.2019.8701269
E028	Punriboon, C., So-In, C., Aimtongkham, P., & Rujirakul, K. (2019). A bio-inspired capacitated vehicle-routing problem scheme using artificial bee colony with crossover optimizations. *Journal of Internet Services and Information Security*, *9*(3), 21–40. https://doi.org/10.22667/JISIS.2019.08.31.021
E029	Kussman, S., Godat, Y., Hanne, T., & Dornberger, R. (2020). A New Hybrid Bat Algorithm Optimizing the Capacitated Vehicle Routing Problem. *Proceedings of the 2020 the 3rd International Conference on Computers in Management and Business*, 107–111. https://doi.org/10.1145/3383845.3383880
E030	Bouhafs, L., Hajjam, A., & Koukam, A. (2004). A hybrid ant colony system approach for the capacitated vehicle routing problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3172 LNCS*, 414–415. https://doi.org/10.1007/978-3-540-28646-2_42
E031	Xia, M., & Mingping Xia. (2009). A modified ant colony algorithm with local search for capacitated vehicle routing problem. *PACIIA 2009 - 2009 2nd Asia-Pacific Conference on Computational Intelligence and Industrial Applications*, *2*, 84–87. https://doi.org/10.1109/PACIIA.2009.5406543
E032	Lee, C.-Y., Lee, Z.-J., Lin, S.-W., & Ying, K.-C. (2010). An enhanced ant colony optimization (EACO) applied to capacitated vehicle routing problem. *Applied Intelligence*, *32*(1), 88–95. https://doi.org/10.1007/s10489-008-0136-9
E033	Tan, W. F., Lee, L. S., Majid, Z. A., & Seow, H. V. (2012). Ant colony optimization for capacitated vehicle routing problem. *Journal of Computer Science*, *8*(6), 846–852. https://doi.org/10.3844/jcssp.2012.846.852
E034	Yamina, S., Ahmed, S., & Kinza, M. N. (2013). Metaheuristic approach for solving the vehicle routing problem: Application in pharmaceutical society. *2013 International Conference on Control, Decision and Information Technologies, CoDIT 2013*, 684–690. https://doi.org/10.1109/CoDIT.2013.6689625
E035	Toklu, N. E., Montemanni, R., & Gambardella, L. M. (2013). An ant colony system for the capacitated vehicle routing problem with uncertain travel costs. *Proceedings of the 2013 IEEE Symposium on Swarm Intelligence, SIS 2013 - 2013 IEEE Symposium Series on Computational Intelligence, SSCI 2013*, 32–39. https://doi.org/10.1109/SIS.2013.6615156

E036	Stodola, P., Mazal, J., Podhorec, M., & Litvaj, O. (2014). Using the Ant Colony Optimization algorithm for the Capacitated Vehicle Routing Problem. In M. D. Brezina T. Stefek A. (Ed.), *Proceedings of the 16th International Conference on Mechatronics, Mechatronika 2014* (pp. 503–510). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/MECHATRONIKA.2014.7018311
E037	Janjarassuk, U., & Masuchun, R. (2017). An ant colony optimization method for the capacitated vehicle routing problem with stochastic demands. *2016 International Computer Science and Engineering Conference (ICSEC)*, 1–5. https://doi.org/10.1109/ICSEC.2016.7859921
E038	Mutar, M. L., Aboobaider, B. M., Hameed, A. S., & Yusof, N. (2019). Enhancing solutions of capacity vehicle routing problem based on an improvement ant colony system algorithm. *Journal of Advanced Research in Dynamical and Control Systems*, *11*(1), 1362–1374. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85064972007&partnerID=40&md5=0f0b593b253c5c67d7c209f3a6564c1f
E039	Mutar, M. L., Burhanuddin, M. A., Hameed, A. S., Yusof, N., & Mutashar, H. J. (2020). An efficient improvement of ant colony system algorithm for handling capacity vehicle routing problem. *International Journal of Industrial Engineering Computations*, *11*(4), 549–564. https://doi.org/10.5267/j.ijiec.2020.4.006
E040	Calabrò, G., Torrisi, V., Inturri, G., & Ignaccolo, M. (2020). Improving inbound logistic planning for large-scale real-world routing problems: a novel ant-colony simulation-based optimization. *European Transport Research Review*, *12*(1). https://doi.org/10.1186/s12544-020-00409-7
E041	Yi, W., & Kumar, A. (2007). Ant colony optimization for disaster relief operations. *Transportation Research Part E: Logistics and Transportation Review*, *43*(6), 660–672. https://doi.org/10.1016/j.tre.2006.05.004
E042	Batmetan, J. R., Santoso, A. J., & Pranowo. (2017). A multiple-objective ant colony algorithm for optimizing disaster relief logistics. *Advanced Science Letters*, *23*(3), 2344–2347. https://doi.org/10.1166/asl.2017.8758
E043	Wang, X., Choi, T., Liu, H., & Yue, X. (2018). A Novel Hybrid Ant Colony Optimization Algorithm for Emergency Transportation Problems During Post-Disaster Scenarios. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *48*(4), 545–556. https://doi.org/10.1109/TSMC.2016.2606440
E044	Li, Y., & Chung, S. H. (2019). Disaster relief routing under uncertainty: A robust optimization approach. *IISE Transactions*, *51*(8), 869–886. https://doi.org/10.1080/24725854.2018.1450540

E045	Wei, X., Qiu, H., Wang, D., Duan, J., Wang, Y., & Cheng, T. C. E. (2020). An integrated location-routing problem with post-disaster relief distribution. *Computers and Industrial Engineering*, *147*. https://doi.org/10.1016/j.cie.2020.106632

3.4.4 Criterios de inclusión/exclusión

- **Criterios de inclusión**
  - El estudio presenta un caso de aplicación al problema de ruteo de vehículos con capacidad y sus variantes en situaciones reales.
  - El estudio presenta un algoritmo de la familia de inteligencia colectiva o bioinspirado para resolver el problema de ruteo de vehículos con capacidad y sus variantes.
  - El campo en el que se encuentra el estudio es sobre el problema de ruteo de vehículos en respuesta o atención de desastres naturales.
  - El estudio forma parte de la literatura primaria, secundaria o terciaria, ya que esta forma parte de las referencias confiables.
  - El estudio pertenece al área de informática o Ciencias de la computación.
  - El idioma en el que se encuentra escrito es español, inglés o portugués.
- **Criterios de exclusión**
  - El estudio no presenta un algoritmo de inteligencia colectiva, bioinspirado o alguna de sus variantes para resolver el problema de ruteo de vehículos con capacidades homogéneas.
  - El estudio presenta métodos de optimización de parámetros de los algoritmos.
  - El estudio presenta métodos de optimización de los algoritmos a nivel de hardware.
  - El estudio presenta entornos de benchmarking para la ejecución y comparación de los algoritmos.

3.5 Formulario de extracción de datos

En este punto se muestra el formulario de extracción de datos con el objetivo de poder conocer qué estudios son relevantes para el presente proyecto.

Tabla 15. Extracción de datos

Campo	Descripción	Pregunta
ID	Identificador asignado para los estudios relevantes.	General
Título		General
Autor(es)		General
Año de publicación		General
Tipo de estudio	Revista, Encuesta, Congreso, Libro, etc.	General
Idioma		General
Algoritmo y características	¿Qué tipo de algoritmo se ha empleado en el estudio para solucionar el problema de ruteo de vehículos con homogéneas capacidades y cuáles son sus principales características? Indicar ventajas, desventajas y limitaciones.	P1
Descripción del método o algoritmo	¿De qué manera es empleado el método o algoritmo para dar solución al problema de ruteo de vehículos con capacidades homogéneas?	P1
Propuesta de solución mediante inteligencia colectiva	¿De qué manera es empleado el algoritmo de inteligencia colectiva para dar solución al problema de ruteo de vehículos con capacidades homogéneas?	P2
Propuesta de solución mediante colonia de hormigas	¿De qué manera se empleó el algoritmo colonia de hormiga para dar solución al problema de ruteo de vehículos con capacidades homogéneas?	P3

Situación de emergencia	¿El algoritmo propuesto se empleó para dar solución a una situación de emergencia o una situación real? ¿Qué tipo de emergencia fue?	P4
Propuesta de solución ante la situación de emergencia	¿Cuál fue la propuesta de solución que se le dio al problema ante una situación de emergencia?	P4
Ventajas y desventajas de la propuesta de solución en situaciones de emergencia	¿Cuáles fueron las ventajas y desventajas de la propuesta de solución en el problema de ruteo de vehículos con capacidades homogéneas bajo condiciones de emergencia?	P4

Para ver la lista completa de extracción de datos, véase Anexo “Extracción de datos”.

3.6 Resultados de la revisión

Deberá presentar los resultados de la revisión. Por cada pregunta, deberá elaborar una respuesta en base al formulario de extracción extraído. No deberá simplemente resumir los estudios primarios hallados.

3.6.1 Respuesta a pregunta P1

¿Qué tipos de algoritmos están siendo usados para resolver el problema de ruteo de vehículos con diferentes capacidades y cuáles son sus características?

Se han encontrado una gran variedad de algoritmos que resuelven el problema de ruteo con diferentes capacidades, las cuales han venido siendo elaboradas a lo largo de estos últimos 20 años. Una de las propuestas más antiguas para dar solución al problema es un método llamado Neuro-fuzzy, el cual emplea redes neuronales guiadas por reglas difusas. Este método logra encontrar soluciones cercanas al óptimo global, pero es computacionalmente costoso pues toma demasiado tiempo y espacio (Gomes et al., 2002). También, se encuentran algoritmos evolutivos como el memético y genético que solucionan el problema. Estos algoritmos evolucionan para poder sobrevivir y de esta manera los más aptos son los que mejores soluciones encontraron

al problema (Baldacci et al., 2012; González et al., 2017; Khebbache et al., 2009; Kurniawan et al., 2016). Sin embargo, estos tipos de algoritmos requieren de muchos parámetros de entrada y encontrar los mejores para el problema suele ser complicado. Además de estos algoritmos también se encuentran algoritmos como simulated annealing, el cual simula un recocido de acero para poder encontrar una solución óptima al problema (Wei et al., 2018). Así como estos algoritmos, se encontraron muchos más que están detallados en la tabla 16.

Tabla 16. Algoritmos para resolver un CVRP

Algoritmo	**Estudio**
Algoritmos de ramificación y poda Algoritmos basados en la formulación de partición de conjuntos	(Baldacci et al., 2012)
Búsqueda Tabú	(Ren, 2013)
Métodos de Monte Carlo	(Oliveira & Delgado, 2015)
Búsqueda de organismos simbióticos	(Li & Fan, 2018)
Búsqueda en amplitud Algoritmo rankig-based Programación dinámica	(Uy et al., 2019)
Enjambre de pollos con búsqueda Tabú	(Niazy et al., 2020)

3.6.2 Respuesta a pregunta P2

¿De qué manera son aplicados los algoritmos de inteligencia colectiva o bioinspirado para resolver el problema de ruteo con vehículos?

Respecto a los algoritmos de inteligencia colectiva o bioinspirado se tienen un gran repertorio de implementaciones para resolver el problema de ruteo de vehículos con capacidades homogéneas. En estos dos grandes grupos se tienen algoritmos netamente bioinspirado, algoritmos netamente de inteligencia colectiva y algoritmos que se encuentran en la intersección de estos dos conjuntos, es decir, algoritmos que presentan una inteligencia colectiva inspirados en la naturaleza.

En el primer grupo se encuentran algoritmos como el genético y sus variantes en las cuales las propuestas de solución van mejorando de acuerdo con las restricciones dadas (Kamphukaew et al., 2018; Kurniawan et al., 2015; Pinninghoff et al., 2014), algoritmos meméticos el cual se basa en la evolución para encontrar soluciones

(González et al., 2017; Khebbache et al., 2009). Por otro lado, en el segundo grupo de inteligencia puramente colectiva se encuentra el algoritmo de los fuegos artificiales en la que se modelan las chispas generadas por una explosión como posible solución al problema (Yang & Ke, 2019). Finalmente, los algoritmos inspirados en la naturaleza que presentan inteligencia colectiva o también llamados inteligencia de enjambre son los más aplicados para resolver el problema. En este se tiene a las colonias de abejas que buscan la solución del problema mediante la búsqueda de rutas óptimas desde su panal hacia distintos puntos de alimento (Brajevic, 2011; Ng et al., 2017; Punriboon et al., 2019; D. Zhang et al., 2018; S. Z. Zhang & Lee, 2016b); la optimización de enjambre de partículas en la que cada partícula modela una solución y comparte la información que ha adquirido con las otras para mejorar la solución (Akhand, Jannat, et al., 2016; Kao et al., 2012; Marinakis & Marinaki, 2013; Shan & Wang, 2013; Y. Wang et al., 2017); el algoritmo de la manada de lobos grises y la forma en que se organizan para cazar también resuelve el problema propuesto (Korayem et al., 2015); el algoritmo de la luciérnaga en la que la luciérnaga con mayor luminosidad tiene la mejor solución encontrada (Yesodha & Amudha, 2019); y el algoritmo de la colonia de hormigas, la cual se va a profundizar más en la siguiente pregunta.

3.6.3 Respuesta a pregunta P3

¿De qué manera los algoritmos colonia de hormigas pueden resolver de manera óptima el problema de ruteo y reparto con vehículos de diferentes capacidades?

Las colonias de hormigas en la cual un conjunto de hormigas busca una ruta óptima desde su hormiguero hasta un cierto punto de comida resulta una buena forma de dar solución a los problemas de ruteo de vehículos (Dorigo & Stützle, 2004).

Aunque, esta aproximación funciona bien para el problema general, cuando se le agrega la restricción de vehículos con capacidades homogéneas el rendimiento del algoritmo se ve afectado y es por ello por lo que se propone usar dicho algoritmo junto con otras heurísticas para mejorar las soluciones y el tiempo de ejecución de esta. Muchos de los estudios encontrados proponen usar la colonia de hormigas junto con una búsqueda local para agilizar el proceso (Bouhafs et al., 2004; Mingping Xia, 2009; Yamina et al., 2013). Otros estudios proponen una modificación en la función de actualización de las feromonas para incentivar la exploración o para no perder posibles soluciones (Stodola et al., 2014). Finalmente, los estudios recientes combinan estas dos mejoras para mejorar los resultados (G Calabrò et al., 2020; Mutar et al., 2020).

3.6.4 Respuesta a pregunta P4

¿De qué manera se han aplicado las soluciones al problema de ruteo de vehículos para situaciones de emergencia?

Los estudios encontrados para el problema de ruteo de vehículos se enfocan, principalmente, en escenarios de desastres naturales. En estas situaciones se busca que las soluciones propuestas ayuden a encontrar rutas para la evacuación de los residentes tomando en cuenta las distancias, los vehículos disponibles y los puntos de zonas seguras (Batmetan et al., 2017). También, con las soluciones propuestas se busca llevar ayuda como productos de primera necesidad y medicinas a las zonas afectadas, los cuales se lleva a cabo empleando el algoritmo colonia de hormigas con ciertas modificaciones (X. Wang et al., 2018; Yi & Kumar, 2007). Otra aproximación que se le dio al algoritmo colonia de hormigas fue para situaciones en la que se tiene incerteza sobre la demanda y tiempo de entrega de los productos a los lugares afectados, en el cual se emplea junto con el algoritmo de búsqueda local tabú para resolver este problema (Li & Chung, 2019).

De esta manera, el algoritmo colonia de hormigas es aplicado de distintas maneras para el problema de ruteo de vehículos con capacidades homogéneas en situaciones de emergencia.

3.7 Discusión

Luego de haber revisado los estudios encontrados podemos observar que para dar solución al problema de ruteo de vehículos con capacidades homogéneas existen diversas formas de abordar el problema. Están los algoritmos exactos como lo es la programación lineal, pero esta aproximación no es eficiente para instancias grandes del problema. Por ello, la mayor parte de los estudios se enfocan en algoritmos metaheurísticos para encontrar buenas soluciones al problema. Entre este tipo de algoritmos se encuentran los algoritmos de inteligencia colectiva y algoritmos bioinspirado. En la intersección de estos dos grupos de algoritmos se encuentra el algoritmo colonia de hormigas, la cual es el objeto de estudio para este proyecto.

Asimismo, algunas de las propuestas de solución no solo responden al problema de ruteo de vehículos con capacidades homogéneas, sino que iban más allá y agregaba más restricciones para acercar más el problema a la realidad. Algunas de las propuestas le agregan la restricción de ventanas de tiempo, en la que se tenía un tiempo límite para las entregas. Otros tenían limitantes de incertidumbre, en la que no

se conocían los costos o el tiempo de ir de un punto a otro. Finalmente, ciertos estudios plantean el problema en situaciones de emergencia, en las cuales se tenía que atender con rapidez la situación.

3.8 Conclusiones

La revisión sistemática de la literatura proporciona una mirada general al tema y al método que se está tratando en la presente investigación. Además, en los estudios encontrados se pudo observar cómo los algoritmos han ido mejorando a lo largo de los años, ya sea modificando la forma en que se evalúan ciertos parámetros o combinándolos con otros algoritmos para obtener mejores resultados que superen a los anteriores, reducir el tiempo de ejecución o para abarcar mayores restricciones. Entre todos estos algoritmos que se han encontrado y revisado para la resolución del problema de ruteo de vehículos, los que más sobresalen son los algoritmos metaheurísticos pues cuentan con ciertas ventajas sobre los algoritmos exactos lo que los hace ideales para resolver este tipo de problemas.

Asimismo, con esta revisión del estado del arte permitió conocer cómo es que se está empleando estos algoritmos para resolver el problema de ruteo de vehículos con capacidades homogéneas y su aplicación en situaciones de emergencia. Dentro de todos estos algoritmos metaheurísticos el algoritmo colonia de hormigas es la que tiene más desarrollo en el campo de búsqueda de rutas bajo diversos criterios y situaciones. Sin embargo, la mayoría de estas aplicaciones se da para desastres naturales, pero no existen aplicaciones para desastres naturales en el Perú o para el alivio de desastres para escenarios como las pandemias.

Capítulo 4. Identificar y definir las estructuras de datos que soporten las variables y restricciones que necesita el algoritmo

4.1 Introducción

Este primer objetivo tiene como finalidad establecer la base sobre la cual la solución planteada va a ejecutarse de manera adecuada. Por ello, es importante identificar y definir qué estructura de datos es la que mejor se adecua al problema como a las restricciones del algoritmo propuesto.

4.2 Resultados

Este objetivo solo presenta un resultado (R1) y en el siguiente punto se va a detallar lo realizado para alcanzar el objetivo.

4.2.1 R1. Diseño y definición de la estructura de datos para los nodos, aristas y el grafo que representan los puntos a repartir, las rutas y el área de distribución

Para resolver el problema, en primer lugar, se tienen que definir las estructuras de datos que van a soportar los datos que van a ser procesados por el algoritmo colonia de hormigas. Este algoritmo, como se presentó en el marco conceptual, se ejecuta sobre un grafo.

Existen varias maneras de representar un grafo y entre las representaciones más comunes está la matriz de adyacencia y la lista de adyacencia. Cada una de ellas presenta ventajas y desventajas frente al otro. Por ejemplo, por un lado, en la matriz de adyacencia el acceso a los datos se realiza de manera inmediata pues solo se realiza un cálculo simple. No obstante, esta representación requiere de una gran cantidad de almacenamiento si es que se trabaja con grandes cantidades de datos. Por otro lado, la matriz de adyacencia no tiene este problema pues se crea un elemento y se añade a la lista. Sin embargo, el acceso a los datos se da de manera lineal puesto que se tiene que recorrer los elementos de la lista para llegar hasta el elemento deseado.

Estructuras

Para representar el área en dónde se va a realizar la distribución se plantearon cuatro estructuras.

- Nodo
- Arista
- Vértice
- Grafo

A partir de estas estructuras se van a representar los puntos a repartir, las rutas y el área de distribución.

A. Nodo

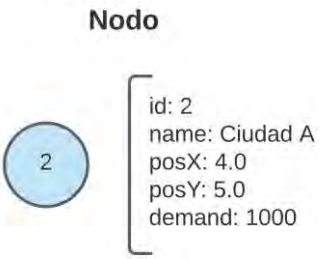
El nodo es la estructura más básica y la base del grafo. En este es donde se almacena, principalmente, si es que está habilitada, la posición y la demanda de este. También, al crear un nuevo nodo se le asigna un identificador único. En el siguiente pseudocódigo se muestra la estructura del nodo.

Pseudocódigo 1. Clase Nodo

```
Clase Nodo
    Datos:
        id: String ;                                    /* Identificador del nodo */
        name: String ;                                  /* Nombre del nodo */
        posX: double ;                                  /* Posición en la coordenada X */
        posY: double ;                                  /* Posición en la coordenada Y */
        demand: int ;                                   /* Cantidad de demanda en el nodo */
    Constructor (posX: double, posY: double, demand: int)
        this.id ← generateId();
        this.posX ← posX;
        this.posY ← posY;
        this.demand ← demand;
    fin
fin
```

En la estructura se puede apreciar que la posición se representa como un número real, puesto que este representa una coordenada. Asimismo, la demanda está representada como un entero, ya que esta representa las unidades requeridas de medicinas, equipos o vacunas. De esta manera podemos tener un nodo simple tal y como se muestra en la siguiente imagen.

Figura 3. Representación de un nodo. Elaboración propia



El nodo que se muestra en la imagen fue instanciado con la posición en coordenadas cartesianas (4.0, 5.0), con una demanda de 1000 unidades, se le otorgó un nombre de Ciudad A y se le fue asignado un identificador de 2.

B. Arista

La arista es la estructura que representa una ruta o carretera. Esta conecta dos nodos y almacena información el costo de ir un nodo i al nodo j. Este costo puede ser simplemente la distancia entre los nodos i y j o puede ser algo más complejo como el resultado de aplicarle una función heurística para obtener un costo más preciso. Asimismo, esta estructura también almacena la cantidad de feromona presente en la arista, dato muy importante para el algoritmo colonia de hormigas, la cual va a ser actualizada constantemente durante la ejecución del algoritmo.

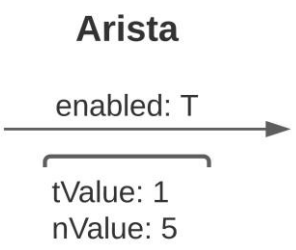
El siguiente pseudocódigo muestra la estructura de la arista.

Pseudocódigo 2. Clase arista

```
Clase Edge
    Datos:
        enabled: boolean ;          /* Flag que indica si está habilitado */
        nValue: String ;            /* Valor de la heurística */
        tValue: String ;            /* Valor de la feromona */
    Constructor (nValue: double, tValue: double)
        this.enabled ← true;
        this.nValue ← nValue;
        this.tValue ← tValue;
    fin
fin
```

En el pseudocódigo se puede observar que tanto como el valor de la heurística (η) y el valor de la feromona (τ) son valores reales, puesto que pueden ser el resultado de funciones complejas que necesitan valores exactos. De esta forma se pueden crear aristas para que luego estas formen parte de un vértice.

Figura 4. Representación de una arista. Elaboración propia



En la imagen se aprecia una arista simple con un valor de feromona de 5 y el valor del resultado de la heurística de 1.

C. Vértice

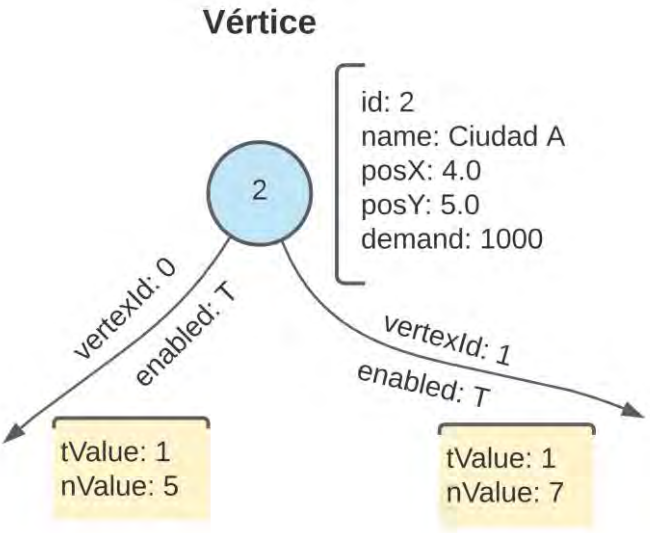
El vértice es la estructura que tiene información de un nodo y un conjunto de aristas. De esta manera se puede acceder a las aristas del nodo y hacer los cálculos necesarios para el algoritmo. El siguiente pseudocódigo muestra la estructura de un vértice.

Pseudocódigo 3. Clase Vértice

```
Clase Vertex Extiende Node
    Datos:
        edges: Map[String] [Edge] ;                    /* Aristas del nodo */
    Constructor (posX: double, posY: double, demand: int)
        super(posX, posY, demand);
        this.edges ← new Map();
    fin
fin
```

Como se pude observar el vértice hereda los atributos del nodo y tiene una estructura del tipo mapa para almacenar las aristas del nodo, el cual emplea como clave el identificador de un nodo para acceder a la arista que conecta a ambos nodos. De esta forma se pueden crear vértices tal y como se ve representado en la siguiente gráfica.

Figura 5. Representación de un vértice. Elaboración propia



En este ejemplo se observa que se tiene el nodo presentado anteriormente, con la diferencia de que ahora presenta dos aristas. Una de estas aristas apunta hacia el vértice 0 y el otro al vértice 1. Cada una de estas aristas tiene su propio valor de la feromona y heurística.

D. Grafo

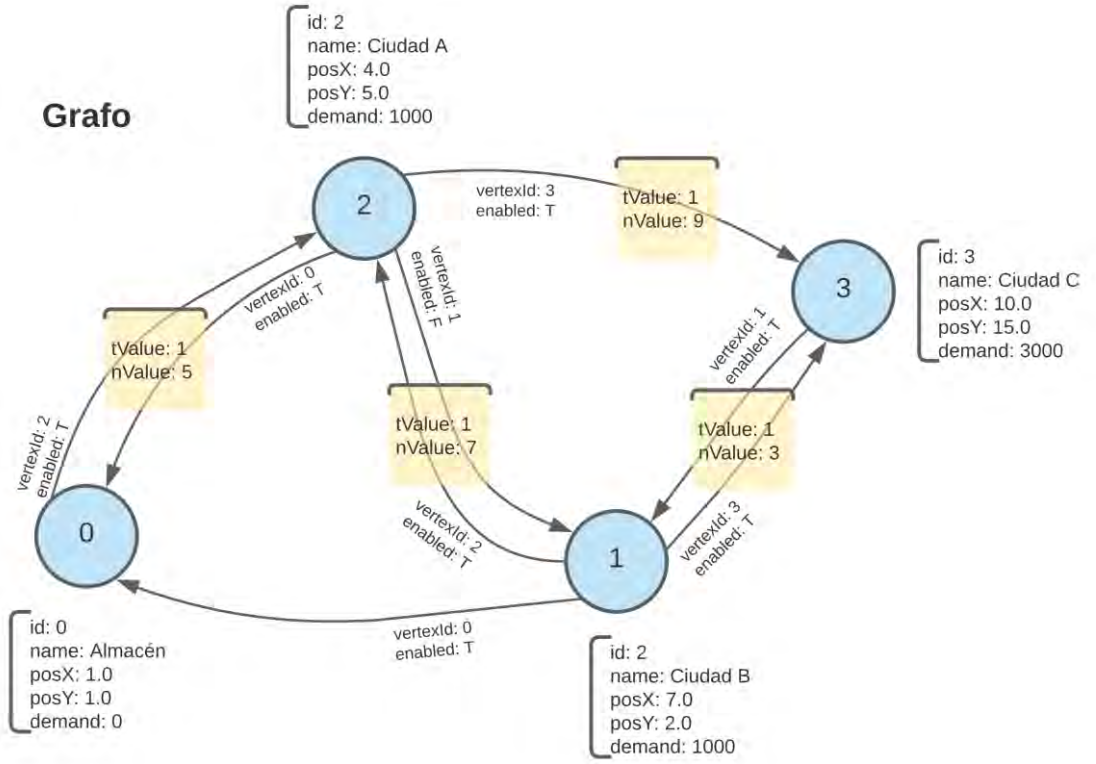
El grafo es la estructura que almacena un conjunto de vértices para que de esta manera se pueda tener acceso a cada uno de ellos con sus respectivas aristas. En el pseudocódigo se puede apreciar la estructura del grafo.

Pseudocódigo 4. Clase Grafo

```
Clase Graph
    Datos:
        vertices: Map[String] [Vertex] ;          /* Vértices del grafo */
    Constructor
        this.vertices ← new Map();
    fin
fin
```

Como se puede observar para almacenar los vértices se hace uso de la estructura mapa pues para obtener un vértice se deberá hacer a través del identificador de este. Así, se puede obtener un grafo como se muestra en el gráfico siguiente.

Figura 6. Representación de un grafo. Elaboración propia



En este gráfico se observa que se tiene un conjunto de vértices cada uno con su posición, demanda y aristas. Cabe resaltar que un vértice i puede tener una arista hacia otro vértice j, pero este vértice j puede no tener una arista hacia i por lo que se trataría de un grafo dirigido. Por ejemplo, en el gráfico el vértice 2 tiene una arista hacia el vértice 3, es decir que se puede ir del vértice 2 hacia el 3 directamente. Ir de la forma contraria no es posible realizarse de forma directa puesto que el vértice 3 no tiene una arista hacia el vértice 2, por lo que para poder llegar al vértice 2 se tendría primero que pasar por el vértice 1.

Validaciones

Este resultado fue validado por un especialista de algoritmia, el cual presentó su conformidad respecto a lo presentado en este apartado. Esta acta se encuentra en el anexo C, apartado R1. Asimismo, el código de la implementación se presenta en el anexo B.

4.3 Discusión

En este capítulo se presentó el resultado R1 en el ítem 4.2.1. Este resultado es la base sobre la cual se van a trabajar los siguientes resultados. Asimismo, este resultado se adecúa al problema presentado para que los algoritmos puedan ejecutarse sobre estas estructuras.

El resultado a la cual se ha llegado parte de una revisión de la literatura, en la cual se analizó las diferentes formas en la que estos adecuaban las estructuras para resolver el problema y que también sea la adecuada para el algoritmo planteado. Por ello, las estructuras planteadas fueron planteadas para adecuarse al problema y al algoritmo. Asimismo, esta permite una cierta flexibilidad para poder agregar nuevos atributos o de ser planteado como un grafo dirigido.

Finalmente, cabe destacar que estas estructuras planteadas son estáticas durante la ejecución de los algoritmos, es decir, no se van a agregar ni eliminar aristas o vértices del grafo inicial.

Capítulo 5. Diseñar y codificar un algoritmo de colonia de hormigas para resolver el problema de ruteo de vehículos capacitados orientado a situaciones de emergencia

5.1 Introducción

El presente capítulo desarrolla el segundo objetivo planteado y para ello se presentan una serie de resultados. El primer resultado (R3) de este objetivo consiste en definir la función objetivo para el problema. Luego, el segundo resultado (R4) define las funciones que alteran el valor de las feromonas que va a emplear el algoritmo. El tercer resultado (R4) define las reglas de transición para el algoritmo dado por el problema planteado. El cuarto resultado (R5) es el diseño y la codificación del algoritmo en sí, el cual será elaborado de acuerdo con los resultados anteriores. Por último, el quinto resultado (R6) consiste en elaborar una interfaz para poder cargar datos, modificar parámetros y ejecutar el algoritmo.

5.2 Resultados

Este objetivo consta de cinco resultados (R2, R3, R4, R5 y R6). En los siguientes puntos se van a presentar los resultados y se presentan los detalles de lo realizado para alcanzar los resultados esperados.

5.2.1 R2. Diseño de la función objetivo a tomar en cuenta para trazar las rutas de distribución

Para entender mejor las funciones que se van a definir para ser optimizadas, primero se tiene que plantear los parámetros y variables generales para con ello definir las restricciones.

Parámetros

Teniendo como base el problema de ruteo de vehículos con capacidad limitada y luego de identificar las variables relevantes orientadas a un contexto de emergencia, se tiene el siguiente conjunto de parámetros y variables para el problema.

Tabla 17. Tabla de parámetros del problema

Símbolo	Definición
NC	Número de áreas afectadas
C	Set de áreas afectadas

NV	Número de vehículos disponibles
V	Set de vehículos disponibles
d_i	Demanda del área afectada i; i ∈ C
Q	Capacidad de los vehículos
t_{ij}	Distancia entre el área i y j; i, j ∈ C
d_{ij}	Tiempo promedio entre el área i y j; i, j ∈ C
u_{ij}	Nivel de urgencia percibida desde el área i al área j; i, j ∈ C
f	Variable de consumo promedio de combustible de un vehículo por kilómetro
c	Variable de costo del combustible por kilómetro
T_{ik}	Tiempo promedio empleado por el vehículo k hasta el área i; k ∈ V; i ∈ C
u_{ik}	Nivel de urgencia en el punto i percibida por el vehículo k; k ∈ V; i ∈ C
c_{ik}	Costo de envío incurrido por el vehículo k hasta el área i; k ∈ V; i ∈ C
x_{ijk}	Variable de decisión de ir del área i al j por el vehículo k; k ∈ V; i, j ∈ C
Q_{ik}	Capacidad disponible del vehículo k en el área afectada i, k ∈ V; i ∈ C

Restricciones

Dado de que se trata del problema de ruteo de vehículos capacitado (CVRP) y luego de revisar algunos estudios acerca de este problema se tienen las siguientes restricciones (Irnich et al., 2014; X. Wang et al., 2018).

1. Para una ruta $r_k = (r_{0,k}, r_{1,k}, \ldots, r_{l+1,k})$ con $r_{0,k} = r_{l+1,k} = 0$ que es el almacén o punto de partida y donde $S = (r_1, \ldots, r_l)$ que son las áreas afectadas visitadas.
2. Un vehículo que llega a un área determinada luego debe partir hacia otra.

$$\sum_{i \in C} x_{ik} = \sum_{j \in C} x_{kj}, \forall k \in V, k \in C$$

3. Hay que asegurar que cada una de las áreas afectadas sea atendida exactamente por una ruta.

$$\sum_{i \in V} \sum_{k \in C} \left(x_{ij}^{k} \right) = 1, \forall i \in C$$

4. La capacidad de un vehículo no debe ser sobrepasada por la demanda de la ruta.

$$\sum_{i \in C} \sum_{k \in C} \left(x_{ij}^{k} \cdot d_{i} \right) \leq Q, \forall i \in V$$

5. La capacidad de una hormiga no debe ser superada por la demanda de un vértice

$$d_{i} < Q$$

Función objetivo

Para el problema de ruteo de vehículos capacitados el principal objetivo es minimizar las distancias recorridas para la entrega de suministros. Sin embargo, el problema planteado está orientado a situaciones de emergencia por lo que minimizar la distancia no es de suma importancia. En una emergencia lo que se busca es una rápida atención, por lo que se prioriza minimizar el tiempo de viaje (Li & Chung, 2019; X. Wang et al., 2018). Asimismo, es importante priorizar la entrega de suministros a los puntos que se encuentran con un nivel alto de urgencia. Este puede ser medido por el nivel de crecimiento de las personas infectadas por el virus COVID 19. Por último, se requiere minimizar el costo de viaje pues es necesario controlar los costos de estas entregas para que no se incurra en costos elevados. Al tener tres funciones a minimizar y cada una de ellas medida en distintas unidades resulta complicado juntarlas solo en una sola función objetivo pues se tendría que hacer una transformación para que sean adimensionales para luego normalizarlas. Por ello, se plantea como un problema multiobjetivo con la finalidad de encontrar un conjunto de Pareto con las soluciones de no dominancia (Alaya et al., 2007).

Así, se tiene la primera función que busca minimizar el tiempo promedio de entrega a cada área afectada.

$$\sum_{i \in V} \sum_{k \in C} \left(t_{ij}^{k} \right)$$

En esta función es una sumatoria de los tiempos promedio de llegada a un área determinada por cada uno de los vehículos. Se busca que el total de esta doble sumatoria sea lo mínimo posible pues así el tiempo de atención a cada uno de los puntos será lo más rápido posible.

Pseudocódigo 5. Función del costo total del tiempo empleado

```
Función totalTimeCost(ants : Ants ): double
    Datos:
      timeSpent: double;    /* Varible para acumular los tiempos empleados en las rutas */
    para ant ∈ ants hacer
        para vertex ∈ vertices hacer
            si vertex.id ∈ ant.route entonces
                timeSpent ← timeSpent + ant.route(vertex.id).timeCost;
            fin
        fin
    fin
    return timeSpent;
fin
```

En el pseudocódigo presentado podemos observar que se itera por cada hormiga del conjunto de hormigas que se tiene para luego iterar sobre cada uno de los vértices que en este caso son las áreas afectadas que necesitan atención. Si el área afectada se encuentra dentro de la ruta de la hormiga, se obtiene el tiempo empleado hasta dicho punto y se acumula.

La segunda función que busca priorizar las áreas afectadas con un alto nivel de urgencia es la siguiente.

$$\sum_{k \in V} \sum_{i \in C} \left(\right)$$

Esta función es una sumatoria del nivel de urgencia percibida de cada área afectada por cada uno de los vehículos. Se busca que el total de esta suma sea lo mínimo posible para que de esta manera se atienda de forma oportuna las áreas que presenten mayor urgencia.

Pseudocódigo 6. Función del total de urgencia percibida

```
Función totalPerceivedUrgency(ants : Ants ): double
    Datos:
      percivedUrgency: double;    /* Varible para acumular los la urgencia percibida en las
    rutas */
    para ant ∈ ants hacer
        para vertex ∈ vertices hacer
            si vertex.id ∈ ant.route entonces
                percivedUrgency ← percivedUrgency + ant.route(vertex.id).urgency;
            fin
        fin
    fin
    return percivedUrgency;
fin
```

De igual manera que el primer pseudocódigo, se va iterando por cada hormiga y por cada área afectada para ir acumulando la urgencia percibida.

Finalmente, la tercera función que busca minimizar el costo de envío medido por la distancia y el costo del combustible empleado por el vehículo.

$$\sum_{v \in V} \sum_{c \in C} \left(\right)$$

En esta se busca que la sumatoria del costo de envío, calculada como la multiplicación de la distancia incurrida hasta cada área afectada por el costo del combustible por kilómetro recorrido, sea en lo posible lo menor posible.

Pseudocódigo 7. Función del costo total incurrido en una ruta

```
Función totalTravelCost(ants : Ants ): double
    Datos:
      moneySpent: double;  /* Varible para acumular los costos incurridos en las rutas */
    para ant ∈ ants hacer
        para vertex ∈ vertices hacer
            si vertex.id ∈ ant.route entonces
                moneySpent ← moneySpent + ant.route(vertex.id).moneyCost;
            fin
        fin
    fin
    return moneySpent;
fin
```

Al igual que los anteriores pseudocódigos, en esta función se van acumulando los costos en los que incurre la hormiga o el vehículo hasta un área afectada en concreto.

Estas funciones objetivas van a ser evaluadas por cada ruta que se elabore y con ello determinar la mejor solución.

Validaciones

Este resultado fue validado por un especialista de algoritmia, el cual presentó su conformidad respecto a lo presentado en este apartado. Esta acta de conformidad se encuentra en el anexo D, apartado resultado R2.

5.2.2 R3. Diseño de las funciones que emplean las feromonas para el algoritmo colonia de hormigas

La forma en la que un vehículo elige una arista sobre otra para que pueda armar su ruta depende de una probabilidad de transición. Esta probabilidad está definida sobre el valor de la heurística (η) y de la feromona (τ). Así se tiene una función para la probabilidad de transición para el vehículo k de ir desde el área i a un área j.

$$P_{ij}^{k} = \begin{cases} \dfrac{(\tau_{ij})^{\alpha} \cdot (\eta_{ij})^{\beta}}{\sum_{c \in C_i^k} (\tau_{ic})^{\alpha} \cdot (\eta_{ic})^{\beta}} & ; Q^k \geq q_j, Q^k \geq 0; \text{en otro caso} \\ 0 & \end{cases}$$

En la fórmula se C_i^k es el set de áreas afectadas que pueden ser accedidas por el vehículo k desde el área i. Los exponentes alfa y beta son para controlar la influencia de la heurística y las feromonas sobre la probabilidad.

Con la probabilidad de transición ya definida se plantea el siguiente pseudocódigo para su implementación.

Pseudocódigo 8. Función de probabilidad de transición

```
Función transitionProbability(ant : Ant, destinationId: double ): double
    si G(destinationId).q ≥ ant.capacity entonces
        return 0;
    fin
    lastVertex ← ant.route.last();
    heuristic ← lastVertex.edges(destinationId).nValue;
    pheromone ← lastVertex.edges(destinationId).tValue;
    numerator ← (heuristic)^β · (pheromone)^α;
    denominator ← 0;
    vertexIds, edges ← lastVertex.edges;
    para vertexId, edge ∈ vertexIds, edges hacer
        si G(vertexId).q ≤ ant.capacity entonces
            denominator ← denominator + edge.nValue^β · edge.value^α;
        fin
    fin
    return numerator/denominator;
fin
```

Luego para la evaporación de la feromona se tiene la siguiente fórmula

$$\tau = (1 - \rho) \cdot \tau \quad ; 0 \leq \rho < 1$$

Esto indica que la feromona va a decrecer según el coeficiente ρ y τ representa la cantidad de feromona depositada en iteraciones anteriores por las hormigas.

Así, con la función de evaporación definida se plantea el siguiente pseudocódigo para su implementación.

Pseudocódigo 9. Función de evaporación

```
Función pheromoneEvaporation(evaporationRate: double, edge: Edge)
    pheromone ← edge.tValue;
    newPheromone ← (1 - evaporationRate) · pheromone;
    edge.tValue ← newPheromone;
fin
```

Finalmente, luego de la evaporación se tiene la fórmula de depósito de feromonas.

$$\tau_{ij} = \tau_{ij} + \sum_{k=1}^{NV} \Delta\tau_{ij}^{k}$$

En esta se actualiza la feromona en la arista (i, j) para una hormiga o vehículo k.

Asimismo, el valor del segundo sumando está dado por la siguiente función.

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k} ; (i,j) \in S \\ 0 ; \text{en otro caso} \end{cases}$$

Donde L_k es el costo total de la ruta elaborada por la hormiga k. Lo cual indica que si el costo es mayor el valor de la feromona depositada será menor, mientras que para rutas con menor costo el valor de la feromona depositada será mayor.

De esta manera se plantea el siguiente pseudocódigo para la implementación del depósito de feromonas.

Pseudocódigo 10. Función de depósito de feromonas

```
Función pheromoneDeposited(ants : Ants, edge: Edge )
    pheromone ← edge.tValue;
    depositedValue ← 0;
    para ant ∈ ants hacer
        si edge ∈ ant.route() entonces
            depositedValue ← depositedValue + 1/ant.route().cost;
        fin
    fin
    edge.tValue ← pheromone + depositedValue;
fin
```

Validaciones

Este resultado fue validado por un especialista de algoritmia, el cual presentó su conformidad respecto a lo presentado en este apartado. Esta acta de conformidad se encuentra en el anexo D, apartado resultado R3.

5.2.3 R4. Diseño de las reglas de transición para el problema

Las reglas de transición para el algoritmo están definidas por las restricciones del problema de ruteo de vehículos con capacidad. Así tenemos las siguientes reglas.

Reglas de transición

1. En primer lugar, para la primera restricción se debe definir el punto de partida, el cual será el almacén y para ello el primer vértice creado debe ser el primero que se añada al grafo. De esta manera, el algoritmo colonia de hormigas tomará como punto de partida y fin este vértice.

2. Para la segunda restricción sobre que un vehículo que llega a un área determinada luego debe partir hacia otra y que no pueda regresar por el mismo camino. Se debe hacer que las hormigas tengan capacidad de memoria para que así, este evite regresar a los vértices que ya ha visitado y poder explorar nuevos vértices.

   Para que tengan esta capacidad de memoria es necesario que cada una de estas hormigas posean un set de vértices visitados y que cada vez que se visita uno nuevo se añade a este.
3. Como tercera regla de transición, hay que asegurar que cada una de las áreas afectadas sea atendida exactamente por una ruta tal y como se menciona en las restricciones. Para ello la colonia debe tener una memoria colectiva para los vértices ya visitados. Así, por ejemplo, una ruta generada por una hormiga no tendrá los mismos vértices que otra ruta generada por otra hormiga.

   Para lograr esto, se debe implementar un set de vértices visitados de ámbito global y que este sea actualizado constantemente por cada nuevo vértice que visita cada hormiga. Con ello se garantiza que todas las áreas afectadas estén incluidas dentro de una ruta distinta.
4. Por último, la cuarta regla de transición deriva de la cuarta restricción en la cual la capacidad de un vehículo no debe ser sobrepasada por la demanda de la ruta. Esto es que, para el algoritmo, cada hormiga tiene una cierta capacidad y que esta no sea sobrepasada por el total de demanda requerida de todos los vértices que pertenecen a la ruta generada.

   Para lograr esto cada hormiga comienza con una capacidad Q y a medida que visita un nuevo vértice esta capacidad se ve disminuida en la misma cantidad de la demanda del vértice visitado. Esto se realiza para cada vértice que se visite hasta que la capacidad disponible sea cero o que los posibles nodos que pueda visitar tienen una demanda superior a su capacidad actual.

Validaciones

Este resultado fue validado por un especialista de algoritmia, el cual presentó su conformidad respecto a lo presentado en este apartado. Esta acta de conformidad se encuentra en el anexo D, apartado resultado R4.

5.2.4 R5. Diseño y codificación del algoritmo colonia de hormigas

Este resultado está dividido en dos etapas el primero de ellos es el diseño y pseudocódigos del algoritmo colonia de hormigas. La segunda etapa es la codificación de este empleando el diseño propuesto.

Diseño y pseudocódigos

El diseño del pseudocódigo del algoritmo colonia de hormigas emplea las reglas de transición (R4), así como las funciones que modifican las feromonas (R3), las funciones objetivo (R2) y las estructuras de datos planteadas inicialmente (R1).

De esta manera se tiene los siguientes pseudocódigos para el algoritmo colonia de hormigas. Los primeros dos indican las estructuras que almacenan los factores que se van a emplear en el algoritmo. Luego, se tiene el pseudocódigo de una hormiga y los datos que ésta va a almacenar para la construcción de una solución. Por último, se tiene el pseudocódigo la cual es el proceso de optimización de las soluciones elaboradas de cada una de las hormigas.

Pseudocódigo 11. Factores atribuibles a la colonia

```
1 Clase FColony
    Datos:
      numIterations: int ;                                  /* Número de iteraciones */
2     numAnts: int ;                          /* Número de hormigas para buscar la solución */
3     capacity: double ;                  /* Capacidad que va a ser asignada a las hormigas */
4   Constructor (numIterations: int, numAnts: int, capacity: double)
5     this.numIterations ← numIterations;
6     this.numAnts ← numAnts;
7     this.capacity ← capacity;
8   fin
9 fin
```

Pseudocódigo 12. Factores atribuibles a un camino

```
1 Clase FPath
    Datos:
      aplha: double ;                                   /* Valor del parámetro alpha */
2     beta: double ;                                     /* Valor del parámetro beta */
3     evaporationRate: double ;                             /* Tasa de evaporación */
4   Constructor (alpha: double, beta: double, evaporationRate: double)
5     this.alpha ← alpha;
6     this.beta ← beta;
7     this.evaporationRate ← evaporationRate;
8   fin
9 fin
```

El pseudocódigo 9 y 10 son los factores que se van a emplear en el algoritmo de optimización. Estos factores fueron separados del algoritmo puesto que los valores de

estos pueden variar de ejecución a ejecución para obtener mejores resultados o de manera más rápida. Asimismo, esta representación permite que la calibración de los valores de los factores se realice de manera sencilla. Estos factores van a ser calibrados en el octavo resultado (R8).

El pseudocódigo 9 de la clase “FColony”, muestra los factores asociados a la colonia de hormigas pues contiene el número de iteraciones que se va a realizar para encontrar la solución. También, el pseudocódigo contiene el número de hormigas que se va a emplear para la búsqueda de la solución. Este permite tener más opciones de búsqueda si es que se incrementa su valor, pero se vuelve computacionalmente más pesado. Finalmente, se tiene el factor de capacidad que de acuerdo con el problema planteado va a ser asignada a todas las hormigas por igual.

El pseudocódigo 10 de la clase “FPath”, muestra los factores asociados a un camino o arista. El factor de alfa es aquel que se va a emplear para la función de probabilidad de transición descrita en el tercer resultado (R3) sobre el nivel de feromona en la arista. De la misma manera, el factor de beta es aquel que indica el nivel de influencia de la heurística en la función de probabilidad de transición. Por último, se tiene la tasa de evaporación la cual va a ser empleado por la función de evaporación para actualizar el nivel de las feromonas en cada una de las aristas del grafo.

Pseudocódigo 13. Estructura de una hormiga

```
1 Clase Ant
    Datos:
      capacity: double ;            /* Capacidad que tiene una hormiga para transportar */
2     totalCost: double ;                          /* Costo total de todas las rutas */
3     pathCosts: Array< double > ;                    /* Costo de cada ruta */
4     paths: Matrix< String > ;                       /* Rutas elaboradas */
5   Constructor (capacity: double)
6     this.capacity ← capacity;
7   fin
8 fin
```

Antes de llegar al algoritmo de optimización es necesario definir la estructura de una hormiga. El pseudocódigo 11 muestra la clase hormiga, en la cual tiene una capacidad asignada al momento de crearse. Esta también posee una variable de tipo matriz para almacenar las rutas que ha elaborado a lo largo de la ejecución del algoritmo y cada una de estas con un costo dado. Finalmente, la suma de cada uno de estos costos se encuentra dentro de la variable de costo total. Cabe resaltar que estas variables de costo pueden referirse al tiempo empleado, la distancia o el nivel de urgencia percibida.

Luego de haber definido los factores y la estructura de la hormiga se tiene el pseudocódigo de optimización. El pseudocódigo 12 muestra qué datos requiere y cómo se va a ejecutar. Los datos que requiere es el grafo en dónde se va a realizar la búsqueda de soluciones, los factores de la colonia y los factores de los caminos. Estos se le van a entregar cuando se crea un objeto de esta clase, tal y como se ve reflejado en el constructor de la clase en las líneas 4 a 7. En este pseudocódigo presenta el método ejecutar (línea 9 al 48), el cual va a iniciar el proceso de optimización.

En primera instancia, se crea una hormiga, la cual va a almacenar la hormiga con la mejor solución encontrada. Como lo que se busca es minimizar el costo, la hormiga va a ser inicializada con una mala solución, es decir, va a tener una solución con un costo elevado. Luego, en las líneas 11 a 46 se va a iterar, de acuerdo con el factor de iteración, para encontrar la solución. Dentro de esta iteración, primero se crea un arreglo de hormigas llamada colonia (línea 12). Dicha colonia va a almacenar las hormigas de la iteración actual. La cantidad de hormigas que va a tener la colonia viene dada por el factor de la colonia “número de hormigas”. Un valor elevado de este factor permitirá tener más hormigas explorando al grafo a costa de mayor tiempo de procesamiento. Estas hormigas van a ser creadas una por una (línea 13) hasta llegar al máximo de hormigas permitidas. Por cada una de estas hormigas se va a realizar una posible solución al problema y para lograr ello se sigue la serie de reglas dada por las reglas de transición planteadas en el cuarto resultado (R4). En la línea 15 se inicializa un set de vértices visitados respondiendo a la regla número cuatro, la cual se va a ir actualizando en la medida que se visiten nuevos vértices. En la línea 16 se crea un camino, el cual responde a la regla número 2, el cual va a contener los vértices que ya han sido visitados por la hormiga. Después de ello, en el bucle de la línea 19 al 31, la hormiga va a ir visitando cada uno de los vértices. Para ello, en la línea 20 calcula las probabilidades de transición desde su posición actual hacia los otros vértices de acuerdo con la fórmula planteada en el resultado R3. En la línea 21 se selecciona uno de estos vértices de acuerdo con las probabilidades calculadas para luego en la línea 22 verificar si la demanda de dicho vértice seleccionado no sobrepasa la capacidad actual de la hormiga. En el caso de que no se sobrepase la capacidad, la hormiga visita el vértice y se agrega tanto a la ruta como al set de visitados. Luego, en la línea 25 se disminuye la capacidad de la hormiga en la misma cantidad que demanda el vértice a la cual se visitó. En el otro caso en el que la capacidad de la hormiga sea

superada por la demanda, la hormiga guarda la ruta que se ha generado y regresa al almacén para restablecer su capacidad.

Una vez que se haya visitado todos los vértices del grafo, la hormiga es añadida a la colonia en la línea 33 para que así cuando toda la colonia esté completa se busque la hormiga con la mejor solución encontrada y compararla con la mejor hormiga de la iteración anterior (línea 35). Si la solución de la hormiga de la colonia es mejor que la hormiga de las iteraciones anteriores, esta toma el puesto de mejor hormiga. Una vez que se actualizó la mejor hormiga inicia el proceso de evaporar las feromonas en cada una de las aristas del grafo la cual está dada en las líneas 38 a 40. Esta evaporación está dada por la función planteada en el resultado R3 y la tasa de evaporación. Esta tasa de evaporación indica que tan rápida va a ser la evaporación pues con valores cercanas a 1 se eliminaría casi de inmediato los rastros por los que no se ha transitado permitiendo así la convergencia a un número reducido de caminos, la cual podría ser la solución o una solución subóptima. Para tasas pequeñas de evaporación, la convergencia a un camino va a ser más tardado pero las hormigas tendrán más caminos por explorar pues se tendrá casi las mismas probabilidades de ir hacia otro vértice. Finalmente, se tiene el proceso de depositar feromonas por las hormigas solo en las aristas por la que estas pasaron (línea 41 a 45). Este depósito de feromonas viene dado por la función que se planteó en el tercer resultado. La función principal de esto es fomentar el uso de las rutas que otorguen mejores soluciones a lo largo del tiempo.

De esta manera, durante cada iteración las feromonas se van a ir actualizando y esto afecta directamente las probabilidades de transición de las hormigas.

Pseudocódigo 14. Optimización de la colonia de hormigas para CVRP

```
Clase AntColony
    Datos:
      colonyFactors: FColony;                  /* Factores de la colonia para el algoritmo */
      pathFactors: FPath;                      /* Factores de las aristas para el algoritmo */
      graph: Graph ;                           /* Grafo en donde se realizará la búsqueda */
    Constructor (colonyFactors: FColony, pathFactors: FPath, graph: Graph)
        this.colonyFactors ← colonyFactors;
        this.pathFactors ← pathFactors;
        this.graph ← graph;
    fin
    Ejecutar (): Ant
        bestAnt ← Inicializar una hormiga con una mala solución;
        mientras no se llegue al final de colonyFactors.numIterations hacer
            colony ← Inicializar arreglo de hormigas;
            para cada hormiga a colonyFactors.numAnts hacer
                ant ← Crear una nueva hormiga;
                visited ← Inicializar el set de visitados;
                path ← Inicializar una nueva ruta vacía;
                path.add(depotId);                 /* Se añade el almacén como punto de partida */
                visited.add(depotId);                    /* Se añade el almacén como visitado */
                mientras visited.size() < this.graph.size() hacer
                    probabilities ← transitionProbability();       /* Probabilidades de transición */
                    selectedVertex ← Seleccionar un vertice de acuerdo a las probabilidades;
                    si this.graph.getVertex(selectedVertex).demand() < ant.getCapacity() entonces
                        path.add(selectedVertex);
                        visited.add(selectedVertex);
                        ant.capacity ← ant.capacity - this.graph.getVertex(selectedVertex);
                    en otro caso
                        ant.addPath(path);
                        ant.setCapacityToFull();
                        path ← Inicializar una nueva ruta vacía;
                    fin
                fin
                ant.addPath(path);
                colony.add(ant);
            fin
            si colony.getBestAnt.solution es mejor que bestAnt.solution entonces
                bestAnt ← colony.getBestAnt();
            fin
            para cada arista del grafo hacer
                pheromoneEvaporation(pathFactors.getEvaporationRate, edge);
            fin
            para cada hormiga hacer
                para cada arista del grafo hacer
                    pheromoneDeposited(ant, edge);
                fin
            fin
        fin
        return bestAnt;
    fin
fin
```

Codificación del algoritmo

Para la codificación del algoritmo colonia de hormigas para resolver el problema de ruteo de vehículos con capacidades homogéneas en situaciones de emergencia. Para ello, se empleó el entorno de trabajo integrado de IntelliJ junto con OpenJDK para poder programar en el lenguaje Java. Asimismo, el código completo de esta implementación se encuentra en el anexo B con un enlace hacia el repositorio del proyecto.

En primer lugar, se codificó la clase de factores en una sola clase que contiene ambas clases planteadas en el diseño, “FColony” y “FPath”. En la figura 7 se muestra cómo está codificado las clases dentro de una sola clase Factors para tener en un solo archivo los factores que luego van a ser calibrados. Asimismo, cada una de estas subclases tiene dos constructores, una con valores por defecto y otra por si se necesita modificar estos valores.

Figura 7. Clase Factors

```
public class Factors {
    public static class FColony {
        private int iterations;
        private int colonySize;
        private Double capacity;

        public FColony() {
            this.iterations = 10;
            this.colonySize = 2;
            this.capacity = 10.0;
        }

        public FColony(int iterations, int colonySize, Double capacity) {
            this.iterations = iterations;
            this.colonySize = colonySize;
            this.capacity = capacity;
        }
    }

    public static class FPath {
        private Double alpha;
        private Double beta;
        private Double evaporationRate;

        public FPath() {
            this.alpha = 0.3;
            this.beta = 0.7;
            this.evaporationRate = 0.4;
        }

        public FPath(Double alpha, Double beta, Double evaporationRate) {
            this.alpha = alpha;
            this.beta = beta;
            this.evaporationRate = evaporationRate;
        }
    }
}
```

De la misma manera, siguiendo el diseño planteado se codificó la clase hormiga tal y como se muestra en la figura 8. La primera diferencia que se puede observar es que no se está empleando en sí una matriz de cadena de caracteres como se propuso en el pseudocódigo sino una lista de arreglos para mantener la eficiencia del espacio.

Asimismo, se está implementando el interfaz Comparable para poder comparar dos hormigas de acuerdo con el costo total que estas tienen.

Figura 8. Clase hormiga

```java
public class Ant implements Comparable<Ant>{
    private Double capacity;
    private Double totalCost;
    private ArrayList<Double> pathCosts;
    private ArrayList<ArrayList<String>> paths;

    public Ant(Double capacity) {
        this.capacity = capacity;
        this.totalCost = 0.0;
        this.pathCosts = new ArrayList<>();
        this.paths = new ArrayList<>();
    }

    @Override
    public int compareTo(@NotNull Ant o) {
        if (this.getTotalCost() > o.getTotalCost()) return 1;
        else if (this.getTotalCost() < o.getTotalCost()) return -1;
        return 0;
    }
}
```

Finalmente, se codificó el algoritmo que emplea las clases anteriores. Así en la figura 9 se muestra la estructura principal de la clase, con los datos esenciales que requiere.

Figura 9. Clase base Optimización de la colonia de hormigas

```java
public class AntColonyOptimization implements Algorithm {
    private final Factors.FColony colonyFactors;
    private final Factors.FPath pathFactors;
    private final Graph graph;

    public AntColonyOptimization(
      Factors.FColony colonyFactors,
      Factors.FPath pathFactors,
      Graph graph) {
        this.colonyFactors = colonyFactors;
        this.pathFactors = pathFactors;
        this.graph = graph;
    }
}
```

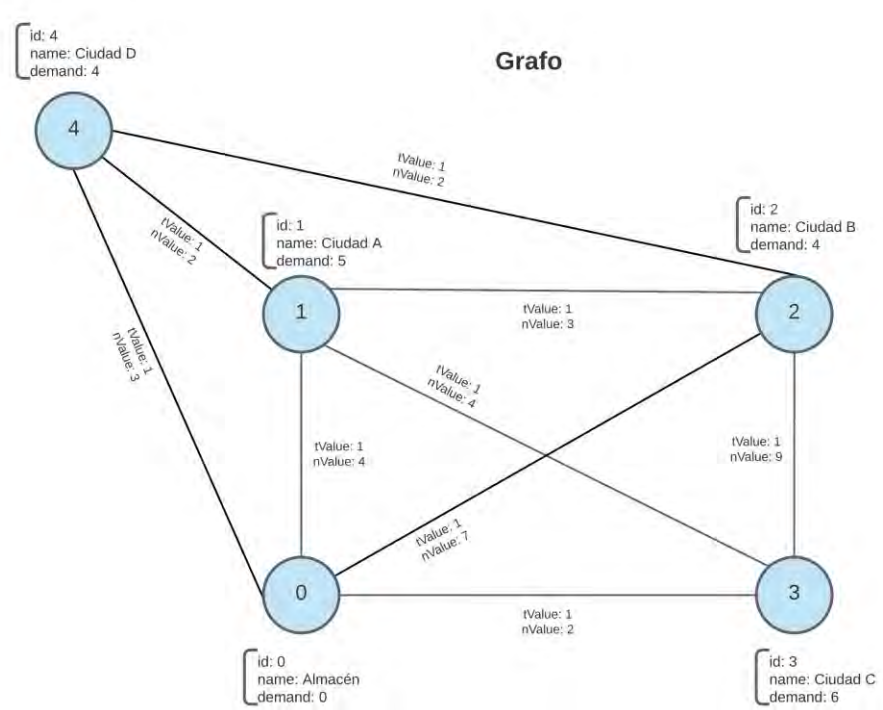
El método principal de la clase de optimización de colonia de hormigas es el que se encarga de encontrar la solución (Figura 10).

Figura 10. Método principal de la clase Optimización de la colonia de hormigas

```java
public Ant execute() {
    Ant bestAnt = new Ant(0.0);
    bestAnt.setTotalCost(Double.MAX_VALUE);

    //For each iteration
    for (int i = 0; i < colonyFactors.getIterations(); i++) {
        ArrayList<Ant> colony = new ArrayList<>();
        System.out.printf("Iteration: %8d\n", i);

        //For each ant
        for (int j = 0; j < colonyFactors.getColonySize(); j++) {
            System.out.print("\uD83D\uDC1C Creating a new ant\n");
            Ant ant = new Ant(colonyFactors.getCapacity());

            //Set all vertex as unvisited
            ArrayList<String> visited = new ArrayList<>(); visited.add("0");

            //First path initialization
            ArrayList<String> path = new ArrayList<>(); path.add("0");
            Double pathCost = 0.0;
            Double totalCost = 0.0;

            while (visited.size() < graph.size()) {
                //Get the current vertex
                Vertex vAux = graph.getVertex(path.get(path.size()-1));

                //Get the edges of the current vertex
                HashMap<String, Edge> vEdges = vAux.getEdges();

                //Get the edges of the unvisited vertices
                HashMap<String, Edge> transitionalEdges = new HashMap<>();
                for (String arrivalVertex:
                     vEdges.keySet()) {
                    if (!visited.contains(arrivalVertex)) {
                        transitionalEdges.put(arrivalVertex, vEdges.get(arrivalVertex));
                    }
                }

                //Obtain the transition probability to the unvisited vertex
                HashMap<String, Double> probabilities =
                        Functions.transitionProbability(transitionalEdges, this.pathFactors);
                String selectedVertex = this.selectNextVertex(probabilities);

                if (
                  graph.getVertex(selectedVertex).getDemand() <=
                  ant.getCapacity() && !selectedVertex.equals("0")) {
                    path.add(selectedVertex);
                    visited.add(selectedVertex);

                    Double transitionCost = vEdges.get(selectedVertex).get_nValue();
                    pathCost = pathCost + transitionCost;
                    totalCost = totalCost + transitionCost;

                    Double currentCapacity = ant.getCapacity();
                    ant.setCapacity(
                      currentCapacity - graph.getVertex(selectedVertex).getDemand());
                } else {
                    path.add("0");
                    ant.addPath(path);
                    path = new ArrayList<>(); path.add("0");

                    ant.addPathCost(pathCost);
                    pathCost = 0.0;

                    ant.setCapacity(colonyFactors.getCapacity());
                }
            }
            path.add("0");
            ant.addPath(path);
            ant.addPathCost(pathCost);
            ant.setTotalCost(totalCost);
            colony.add(ant);
        }
        Ant bestColonyAnt = Collections.min(colony);
        System.out.print("Best Colony Ant\n");
        System.out.print(bestColonyAnt.toString());

        //Save the best solution
        if (bestColonyAnt.getTotalCost() < bestAnt.getTotalCost()) {
            bestAnt = bestColonyAnt;
        }

        //Evaporate the pheromone in all the edges
        for (String vertexId :
                this.graph.getVertices().keySet()) {
            for (String nextVertexId :
                    this.graph.getVertex(vertexId).getEdges().keySet()) {
                Double tValue = graph.getVertex(vertexId).getEdge(nextVertexId).get_tValue();
                Double newT = Functions.evaporate(
                  tValue,
                  this.pathFactors.getEvaporationRate());
                graph.getVertex(vertexId).getEdge(nextVertexId).set_tValue(newT);
            }
        }

        //Update the pheromone [ij] on the edges where the ants transitioned
        for (Ant ant : colony) {
            for (ArrayList<String> path : ant.getPaths()) {
                String prev = "0";
                String curr;

                for (String vertexId : path) {
                    //Increment both edges
                    curr = vertexId;
                    if (vertexId.equals("0")) {
                        prev = vertexId;
                        continue;
                    }
                    Double tValue = graph.getVertex(prev).getEdge(curr).get_tValue();
                    Double newT = tValue + 1/ant.getTotalCost();
                    graph.getVertex(prev).getEdge(curr).set_tValue(newT);
                    graph.getVertex(curr).getEdge(prev).set_tValue(newT);

                    prev = curr;
                }
            }
        }
    }
    System.out.print("Best Ant\n");
    System.out.print(bestAnt.toString());

    return bestAnt;
}
```

Así, luego de haber codificado el algoritmo, se procedió a probarlo con un grafo pequeño para ver los resultados de su ejecución. Así se probó el algoritmo con el grafo que se muestra en la figura 11. En este grafo se cuenta con 5 nodos, el vértice con identificador cero es el almacén por lo que será el punto de partida y el punto de fin.

Figura 11. Ejemplo de un grafo pequeño para la ejecución del algoritmo



Al ejecutar el algoritmo con 10 iteraciones y 2 hormigas en la colonia y cada una con una capacidad de 10 unidades como máximo, se obtiene el siguiente resultado mostrado en la figura 12.

Figura 12. Resultado de la ejecución del algoritmo

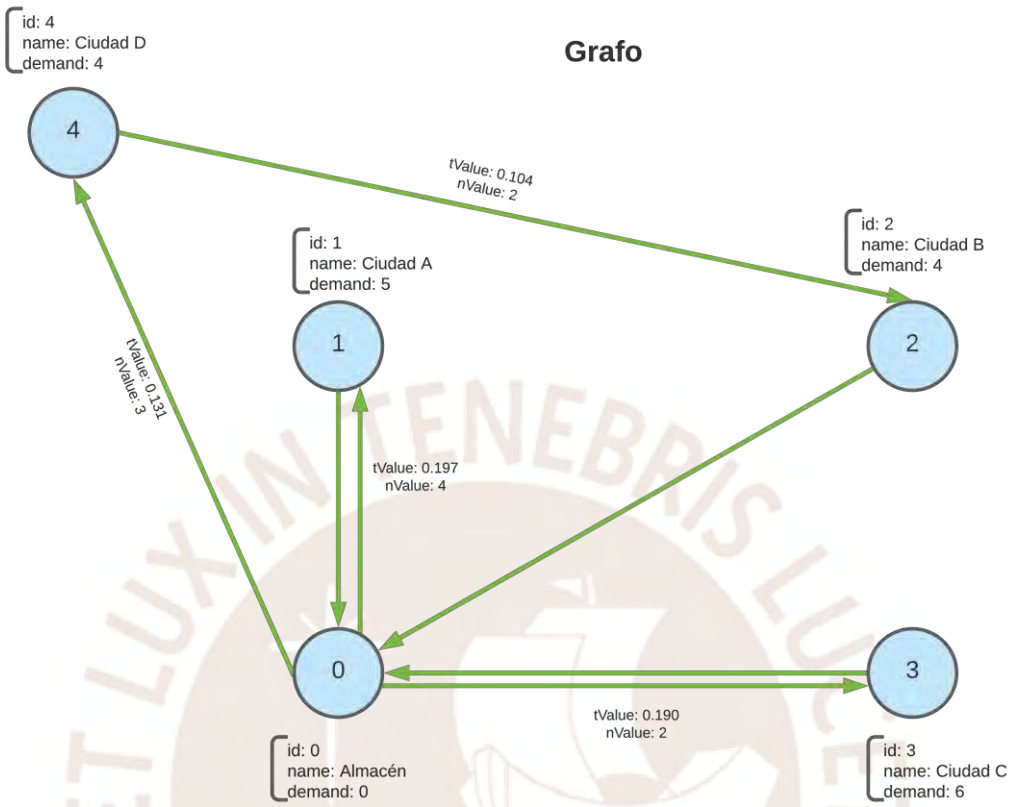
```
Iteration:     0
🐜 Creating a new ant
🐜 Creating a new ant
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:  7.0      0 →    1 →    2 →    0 → END
🚚 Path  1 - Cost:  2.0      0 →    3 →    0 → END
🚚 Path  2 - Cost:  3.0      0 →    4 →    0 → END
✅ Total cost:  12.0

...
...

Iteration:     8
🐜 Creating a new ant
🐜 Creating a new ant
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost: 10.0      0 →    2 →    1 →    0 → END
🚚 Path  1 - Cost:  2.0      0 →    3 →    0 → END
🚚 Path  2 - Cost:  3.0      0 →    4 →    0 → END
✅ Total cost:  15.0

Iteration:     9
🐜 Creating a new ant
🐜 Creating a new ant
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:  5.0      0 →    4 →    2 →    0 → END
🚚 Path  1 - Cost:  4.0      0 →    1 →    0 → END
🚚 Path  2 - Cost:  2.0      0 →    3 →    0 → END
✅ Total cost:  11.0
```

Al terminar la ejecución se nos muestra como quedaron los valores de las feromonas en el grafo y la mejor solución encontrada tal y como se muestra en la figura 13. Como se puede observar la mejor solución que se encontró tiene un costo total de 11 con tres rutas. Estas rutas se aprecian mejor en la figura 14. Cabe resaltar que el costo de retorno de los vehículos no se toma en cuenta como parte del costo total pues una vez que se reparte ese costo ya no tiene valor alguno en el problema.

Figura 13. Solución encontrada al ejecutar el algoritmo

```
Test
------------------------------------------------------------------------
  ID |                NAME |  DEMAND |   η |      τ |         EDGES |
------------------------------------------------------------------------
   0 |               Depot |       0 |░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
                                     | 4.0 | 0.145 |  0 →       1 |
                                     | 7.0 | 0.094 |  0 →       2 |
                                     | 2.0 | 0.235 |  0 →       3 |
                                     | 3.0 | 0.232 |  0 →       4 |
------------------------------------------------------------------------
   1 |                   A |       5 |░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
                                     | 4.0 | 0.145 |  1 →       0 |
                                     | 3.0 | 0.107 |  1 →       2 |
                                     | 4.0 | 0.000 |  1 →       3 |
                                     | 2.0 | 0.000 |  1 →       4 |
------------------------------------------------------------------------
   2 |                   B |       4 |░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
                                     | 7.0 | 0.094 |  2 →       0 |
                                     | 3.0 | 0.107 |  2 →       1 |
                                     | 9.0 | 0.020 |  2 →       3 |
                                     | 2.0 | 0.108 |  2 →       4 |
------------------------------------------------------------------------
   3 |                   C |       6 |░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
                                     | 2.0 | 0.235 |  3 →       0 |
                                     | 4.0 | 0.000 |  3 →       1 |
                                     | 9.0 | 0.020 |  3 →       2 |
------------------------------------------------------------------------
   4 |                   D |       6 |░░░░░░░░░░░░░░░░░░░░░░░░░░░░░░
                                     | 3.0 | 0.232 |  4 →       0 |
                                     | 2.0 | 0.000 |  4 →       1 |
                                     | 2.0 | 0.108 |  4 →       2 |
------------------------------------------------------------------------
Best Ant
🏁 Solution
🚚 Path  0 - Cost:  5.0    0 →   4 →   2 →   0 → END
🚚 Path  1 - Cost:  2.0    0 →   3 →   0 → END
🚚 Path  2 - Cost:  4.0    0 →   1 →   0 → END
✅ Total cost:  11.0
```

Figura 14. Grafo con la solución encontrada por el algoritmo



Validaciones

Para validar las implementaciones de las clases y el algoritmo se realizaron pruebas unitarias y una prueba de caja blanca. Los resultados de estas pruebas se encuentran en el anexo Anexo D, sección R5. Este cuenta con el detalle de las pruebas realizadas, así como el acta de validación del especialista en algoritmos.

5.2.5 R6. Interfaz para la ejecución del algoritmo colonia de hormigas

En este resultado se presenta la interfaz gráfica que permite la ejecución del algoritmo colonia de hormigas, la cual se encuentra dividida en dos secciones. La primera sección permite cargar los datos necesarios, así como la calibración de los factores del algoritmo. En la segunda sección se pueden observar las soluciones encontradas por el algoritmo.

Primera sección

Esta primera sección se encuentra dividida en tres campos distintos tal y como se muestra en la figura 15.

El primer campo es para la carga del grafo que va a ser procesado por el algoritmo. Este grafo tiene que estar en formato JSON para que pueda ser procesado (En el anexo de las pruebas se especifica con mayor detalle la estructura del archivo). Si no se carga ningún archivo el algoritmo no va a poder ser ejecutado.

El segundo campo es de los factores del camino. Este cuenta con tres entradas: el valor de alfa, beta y la tasa de evaporación. Los primeros dos valores serán empleados para la función de probabilidad de transición mientras que el último para la función de evaporación de feromonas. Estos tres valores tienen que ser positivos y menores que 1.

Finalmente se encuentran los factores de la colonia. Este tiene entradas para el número de iteraciones que va a tener el algoritmo, el número de hormigas que va a tener la colonia en cada generación y la capacidad que tiene cada hormiga. Estos valores tienen que ser positivos. Cada uno de estos campos tiene su propia validación para que el algoritmo pueda ser ejecutado de forma correcta (Ver figura 16).

Figura 15. Primera sección de la interfaz gráfica

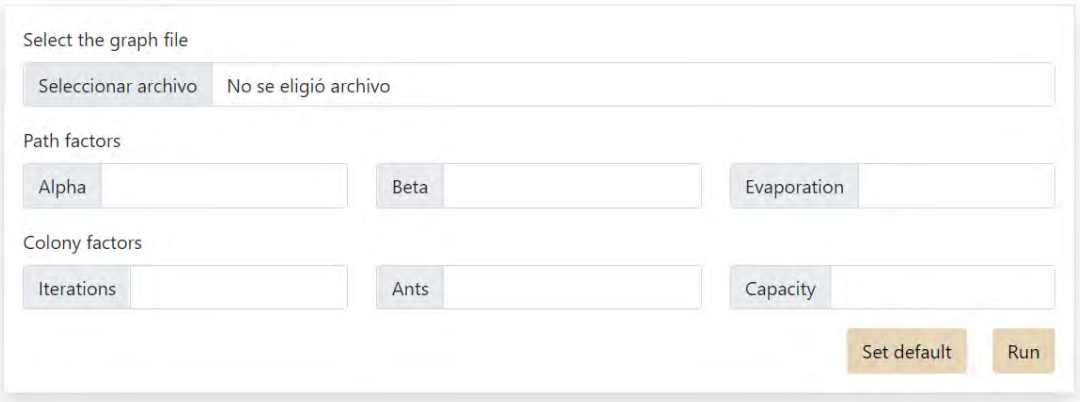
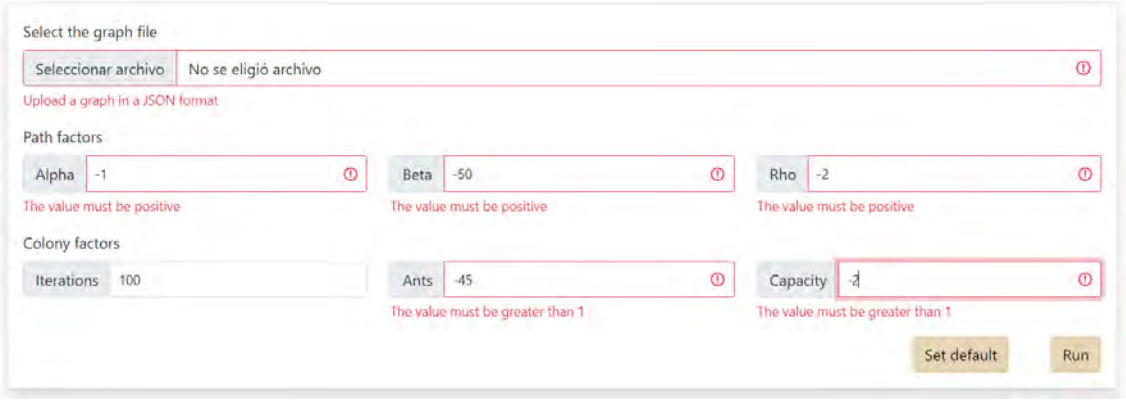


Figura 16. Validaciones de la primera sección de la interfaz gráfica



Segunda sección

En esta segunda sección se encuentra la tabla con las soluciones que el algoritmo ha encontrado (Ver figura 17). Cada fila de la tabla contiene una solución de no dominancia puesto que se está trabajando con un algoritmo multiobjetivo. En esta se muestra el tiempo total aproximado que tomaría llevar los suministros a todos los puntos, el costo total aproximado y el costo general que indica el balance entre estas variables. Asimismo, se muestra de manera gráfica el grafo que se ha cargado (Ver figura 18). En este, se muestra el punto de partida o almacén como un nodo de color azul, mientras que los otros nodos representan los puntos a ser repartidos en una escala de color rojo, el cual mientras más oscuro el nivel de urgencia es más alto. El tamaño refleja la misma información.

Figura 17. Segunda sección de la interfaz gráfica

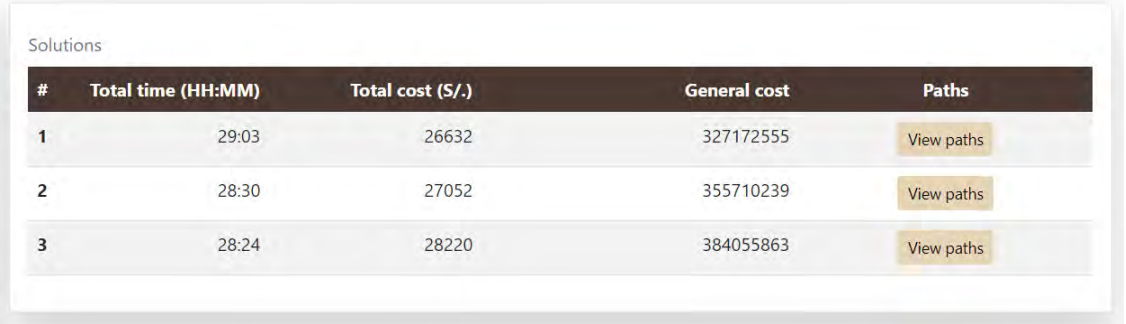
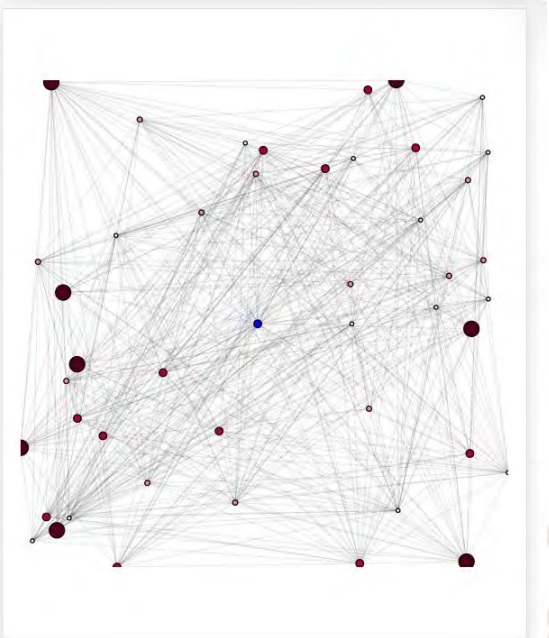


Figura 18. Visualización del grafo cargado

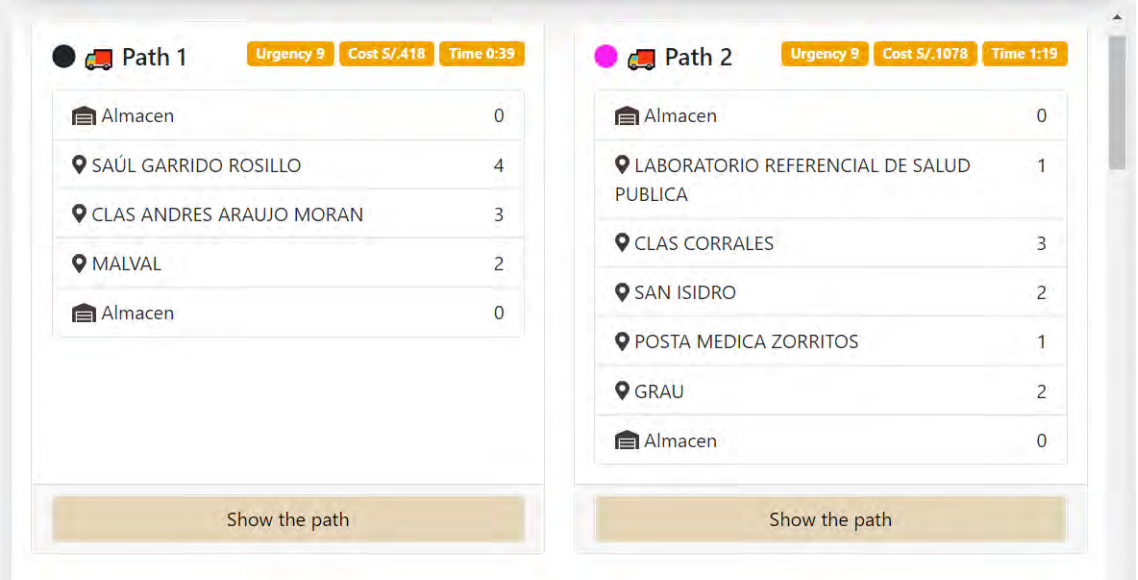


Mientras el algoritmo se encuentra en ejecución la interfaz mostrará una barra de carga hasta que finalice su ejecución como se ve en la Figura 19.

Figura 19. Segunda sección de la interfaz gráfica barra de carga

Al momento de ver en detalle una de las soluciones se tiene una serie de tarjetas informativas de todas las rutas para visitar cada uno de los puntos como se muestra en la figura 20. En este se tiene el detalle del nivel de urgencia cubierta, el costo y el tiempo aproximados de dicha ruta y justo al lado del nombre del hospital se muestra un número que indica el nivel de alerta en la que se encuentra.

Figura 20. Detalle de la solución seleccionada



Validaciones

Se realizaron las pruebas de integración entre la interfaz gráfica y el algoritmo para verificar que los datos que se envían desde la interfaz sean las que se emplean en el algoritmo y que las soluciones que el algoritmo genera sean las que se muestran en la interfaz. El detalle de estas validaciones, así como el repositorio de este se encuentra en el anexo D apartado R6.

5.3 Discusión

En este capítulo se desarrollaron los resultados correspondientes para poder realizar el algoritmo colonia de hormigas para el problema de ruteo de vehículos capacitados orientado a una situación de emergencia. Cada uno de estos resultados contribuyen a alcanzar el objetivo planteado.

Los primeros tres resultados permiten colocar las bases sobre la cual se va a diseñar e implementar el cuarto, y más importante, resultado. Con una especificación adecuada del problema y los objetivos a optimizar, se pudo realizar un diseño adecuado del algoritmo colonia de hormigas.

El alcance de este cuarto resultado permite comprender de mejor manera el funcionamiento de este algoritmo y cómo es que va generando, ajustando y mejorando las soluciones que va encontrando. Durante el desarrollo de este se encontraron algunas dificultades como llevar el pseudocódigo de alto nivel a implementarlo en

código o la necesidad de identificar variables y operaciones auxiliares para soportar el diseño planteado. Este resultado alcanzado permite estar más cerca de cumplir el objetivo principal del proyecto.

Finalmente, el último resultado permite contar con una herramienta gráfica para ejecutar de manera más sencilla el algoritmo planteado en el resultado anterior. Con este resultado elegir los parámetros del algoritmo y cargar los archivos serán más intuitivos que solo emplear la línea de comandos.

Capítulo 6. Comparar el desempeño del algoritmo colonia de hormigas contra un algoritmo de ruteo voraz

6.1 Introducción

En el presente capítulo se desarrolla el tercer objetivo junto con los resultados correspondientes a este. El primer resultado (R7) de este objetivo implica realizar la implementación del algoritmo primero el mejor. Luego, en el segundo resultado (R8) se va a calibrar los parámetros de los algoritmos desarrollados. Después de ello, el tercer resultado (R9) desarrolla la experimentación numérica de ambos algoritmos. Finalmente, se prueba el algoritmo colonia de hormigas en un escenario real (R10)

6.2 Resultados

Este objetivo consta de tres resultados (R7, R8 y R9). En los siguientes puntos se van a presentar los resultados y se presentan los detalles de lo realizado para alcanzar los resultados esperados.

6.2.1 R7. Implementación del algoritmo primero el mejor

Diseño del algoritmo primero el mejor

La implementación del algoritmo voraz primero el mejor fue elaborado tomando en cuenta el diseño presentado en el libro “Artificial Intelligence: A Modern Approach” (Russell & Norvig, 2009) Este fue modificado para adaptarse al problema de ruteo de vehículos con capacidad.

Pseudocódigo 15. Algoritmo primero el mejor para CVRP

```
1  Clase BFS
       Datos:
         capacity: double ;                          /* Capacidad que va a tener cada vehículo */
2        graph: Graph ;                              /* Grafo en donde se realizará la búsqueda */
3      Constructor (capacity: double, graph: Graph)
4          this.capacity ← capacity;
5          this.graph ← graph;
6      fin
7      Ejecutar ()
8          paths ← Inicializar un Array vacío para almacenar un camino;
9          frontera ← Inicializar una cola de prioridad ordenada por el costo del camino;
10         path ← Inicializar un camino con el primer vértice (almacén), costo 0 y con la maxima
            capacidad;
11         frontera.add(path);
12         mientras no se visitaron todos los nodos hacer
13             minCostPath ← frontera.pop();
14             lastVetex ← minCostPath.getLastVertex();
15             edges ← lastVertex.getEdges();
16             para cada edge en edges hacer
17                 destiny ← edge.getDestiny();
18                 si minCostPath.capacity >= edge.getDestiny().getDemand() entonces
19                     si NO minCostPath.contains(destiny) Y NO visited.contains(destiny) entonces
20                         newPath ← minCostPath;
21                         newPath.addVertex(destiny);
22                         newPath.addCost(edge.getCost());
23                         newPath.reduceCapacity(destiny.getDemand());
24                         frontier.add(newPath);
25                     fin
26                 en otro caso
27                     minCostPath.returnToDepot();
28                     paths.add(minCostPath);
29                     frontera ← Inicializar una cola de prioridad ordenada por el costo del camino;
30                     path ← Inicializar un camino con el primer vértice (almacén), costo 0 y con la
31                 fin
32             fin
33         fin
34     fin
35 fin
```

De esta manera se tiene el diseño del algoritmo voraz primero el mejor para el problema de ruteo de vehículos con capacidad (Ver pseudocódigo 13). En este se requiere de un grafo sobre el cual se va a realizar la búsqueda de caminos y la capacidad máxima que va a tener cada vehículo tal y como se ve en el constructor de la clase (Líneas del 3 al 6). Asimismo, se tiene el método principal para que se encargue de ejecutar el algoritmo (Línea 7).

El algoritmo comienza con las inicializaciones de las variables que van a ser usadas. El primero de ellos (Línea 8) se va a encargar de almacenar las rutas de entrega de suministros, la frontera que es una cola de prioridad va a almacenar las rutas ordenadas de menor a mayor por el costo total de estas. Luego, la variable “path” que va a contener la primera ruta que empieza desde el almacén (Línea 10) y esta es añadida como primer elemento a la frontera.

Luego en la línea 12 se empieza a iterar hasta que se hayan visitado todos los vértices del grafo. En la línea 13 se obtiene la ruta de menor costo hasta ese momento de la

frontera, a partir de este se obtiene el último vértice visitado (Línea 14) y las aristas conectadas a este (Línea 15). Con esto se van a expandir los caminos y para ello por cada arista del set de aristas, se obtiene el vértice de destino de la arista (Línea 17). Se verifica que dicho vértice tenga una demanda menor a la capacidad actual que tiene la ruta evaluada y luego si el vértice ya fue visitado en la ruta o si ya se encuentra en el set de visitados (Líneas 18 y 19) En el caso de que se ambos cumplan se procede a crear un camino con el camino evaluado, se agrega el vértice destino a la ruta, se incrementa el costo de la ruta en la misma cantidad del costo de transición y se disminuye la capacidad actual de acuerdo a la demanda del vértice destino. Después de ello, la nueva ruta se agrega a la frontera (Línea 24). En caso de que la ruta de costo mínimo ya no cuente con la capacidad disponible para ir a otro vértice se retorna al almacén (Línea 27) y se añade la ruta al set de rutas. Tras haber almacenado la ruta, se limpia la frontera y se inicia una nueva ruta desde el almacén.

Codificación del algoritmo primero el mejor

Con ayuda del pseudocódigo se procedió a codificar el algoritmo para resolver el problema de ruteo de vehículos con capacidad homogénea en situaciones de emergencia. Para ello, se empleó el entorno de trabajo integrado de IntelliJ junto con OpenJDK para poder programar en el lenguaje Java. Asimismo, el código completo de esta implementación se encuentra en el anexo B con un enlace hacia el repositorio del proyecto.

Así se tiene el código del algoritmo. En la figura 21 se observan los atributos de la clase y el constructor de este. En la figura 22 se muestra la primera parte del método para ejecutar el algoritmo, este contiene las inicializaciones de las principales variables que se van a emplear para ir almacenando los resultados obtenidos. En la figura 23 se muestra la segunda parte del método para ejecutar el algoritmo este responde a las líneas 12 y 33 del pseudocódigo 13.

Figura 21. Clase BFS. Constructor.

```java
public class BFS implements Algorithm {
    private final ParetoFront paretoFront = new ParetoFront();
    private final Graph graph;
    private final Double capacity;

    public BFS(Graph graph, Double capacity) {
        this.graph = graph;
        this.capacity = capacity;
    }
}
```

Figura 22. Clase BFS. Método ejecutar (Primera parte)

```java
@Override
public ParetoFront execute() {
    Solution solution = new Solution();
    ArrayList<ArrayList<String>> paths = new ArrayList<>();
    ArrayList<Double> totalCostPerPath = new ArrayList<>();
    Double totalCost = 0.0;

    //Set all vertex as unvisited
    ArrayList<String> visited = new ArrayList<>();
    visited.add("0");

    //First path initialization
    Path path = new Path(this.capacity);
    path.addVertex("0", 0.0, 0.0);

    //First path initialization
    PriorityQueue<Path> frontier = new PriorityQueue<>();
    frontier.add(path);
}
```

Figura 23. Clase BFS. Método ejecutar (Segunda parte)

```java
@Override
public ParetoFront execute() {
    while (visited.size() < graph.size()){
        Path minPath = frontier.poll();
        if (minPath == null) break;
        Integer pathSize = minPath != null ? minPath.size() : 0;
        String vId = minPath.getPath().get(pathSize - 1);
        Vertex vAux = graph.getVertex(vId);
        HashMap<String, Edge> vEdges = vAux.getEdges();

        for (String vDestiny :
                vEdges.keySet()) {
            if (minPath.getCapacity() >= graph.getVertex(vDestiny).getDemand()) {
                if (!visited.contains(vDestiny) && !minPath.getPath().contains(vDestiny)) {
                    Path newPath = (Path) Path.deepCopy(minPath);
                    newPath.addVertex(vDestiny,
                            vEdges.get(vDestiny).get_nValue(),
                            (double) graph.getVertex(vDestiny).getDemand());

                    frontier.add(newPath);
                }
            } else {
                minPath.addVertex("0", 0.0, 0.0);
                ArrayList<String> bestPath = minPath.getPath();
                for (String id :
                        bestPath) {
                    if (!id.equals("0")) visited.add(id);
                }
                paths.add(bestPath);
                totalCostPerPath.add(minPath.getCost());
                totalCost += minPath.getCost();

                path = new Path(this.capacity);
                path.addVertex("0", 0.0, 0.0);

                //First path initialization
                frontier = new PriorityQueue<>();
                frontier.add(path);
                break;
            }
        }
    }

    solution.setTotalCostPerPath(totalCostPerPath);
    solution.setPaths(paths);
    solution.setTotalCost(totalCost);

    paretoFront.addSolution(solution);

    return paretoFront;
}
```

Validaciones

Para validar la implementación del algoritmo se realizó una prueba de caja blanca. El resultado de esta prueba se encuentra en el anexo Anexo E, sección R7. Este cuenta

con el detalle de la prueba realizada, así como el acta de validación del especialista en algoritmos.

6.2.2 R8. Calibración de parámetros de los algoritmos

En este resultado se calibran los parámetros del algoritmo colonia de hormigas de acuerdo con el problema planteado para así mejorar las soluciones elaboradas y reducir el tiempo de ejecución.

Etapa previa a la calibración

Antes de iniciar la calibración, es pertinente identificar los parámetros que necesitan ser calibrados. Los parámetros que posee el algoritmo son alfa, beta y rho que determinan el nivel de influencia de la heurística, las feromonas y la tasa de evaporación.

De esta manera, se plantean una serie de parámetros obtenidos de la revisión del estado del arte.

Tabla 18. Configuraciones de los parámetros

Configuración	*Alfa*	*Beta*	*Rho*	*Autor*
0	1	4	0.1	(Alaya et al., 2007)
1	1	8	0.01	
2	1	4	0.01	
3	1	2	0.1	(Sun et al., 2017)
4	5	8	0.05	(X. Wang et al., 2018)

Con cada una de estas configuraciones de los parámetros se elaboraron 5 pruebas cada una de ellas con 30 hormigas en la colonia y 100 iteraciones. Estos fueron ejecutados sobre un grafo con 50 nodos. Los resultados que se obtuvieron se resumen en la tabla.

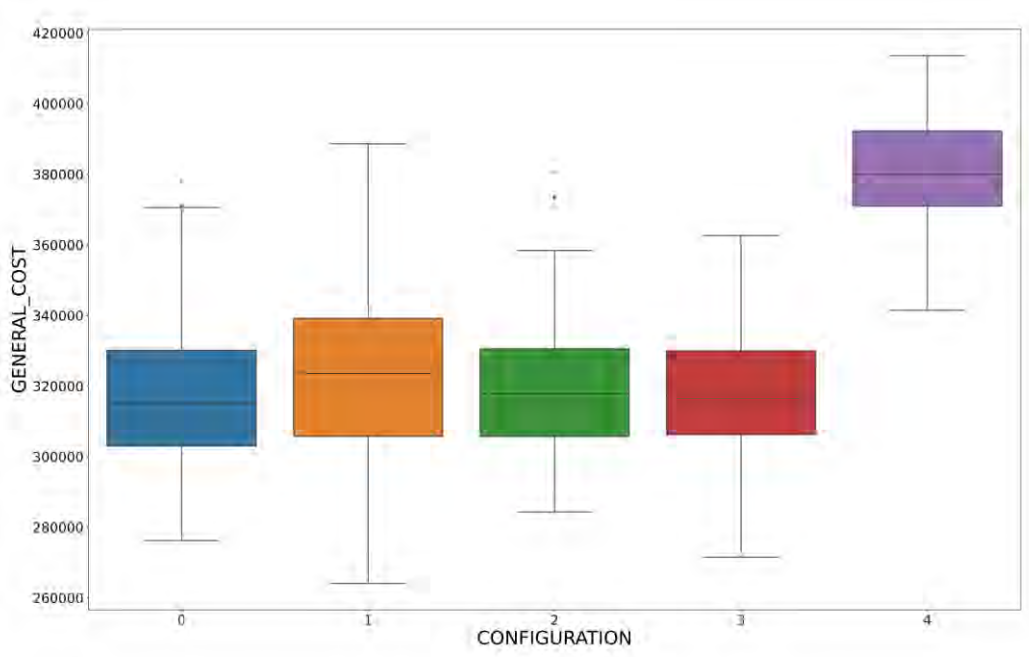
Tabla 19. Resumen de las ejecuciones de cada configuración de los parámetros

Configuración	*Costo general promedio*	*Desviación estándar del costo*	*Tiempo de ejecución promedio por solución*	*Desviación estándar del tiempo de ejecución por solución*
0	317559	20703	0.32	0.09
1	322417	24464	0.33	0.11

2	319515	18711	0.32	0.11
3	317238	19597	0.28	0.07
4	379942	15185	0.30	0.07

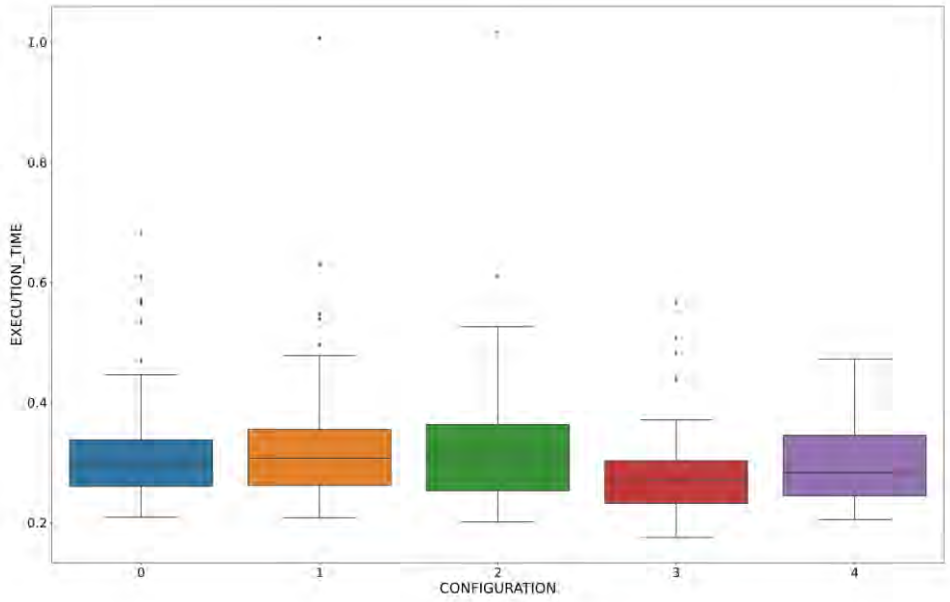
Para entender mejor la tabla se presenta la figura 24 en la que se presenta un diagrama de cajas para ver qué configuración es la que obtiene mejores resultados con respecto al costo general. Tal y como se puede observar la primera y cuarta configuración son los que tienen la media más baja, es decir, tienen mejores resultados, mientras que la última configuración tiene una media alta por lo que sus resultados no son del todo buenos.

Figura 24. Gráfico de cajas. Configuración vs Costo



En cuanto al tiempo de ejecución, también se elaboró un diagrama de cajas (Figura 25). En esta se puede ver que la configuración con menor media de tiempo de ejecución es la cuarta configuración. Además, sus colas están más apegadas a la caja por lo que tiene una varianza baja.

Figura 25. Gráfico de cajas. Configuración vs Tiempo de ejecución



Al observar estos resultados obtenidos, hasta el momento la mejor configuración de los parámetros es la cuarta, pues es la que menor media posee en cuanto al costo y al tiempo respecto a las otras configuraciones.

Calibración de los parámetros

Luego de haber encontrado la mejor configuración base, se procede a hacer pequeñas variaciones sobre este para ver si se puede mejorar aún más los resultados obtenidos.

- **Calibración de Alfa**

Para Alfa, se tiene como valor base 1, esta va a variar desde 0.4 a 1.6 con saltos de 0.2 tal y como se muestra en la tabla 20. Asimismo, los demás parámetros se mantuvieron constantes y se realizaron 50 ejecuciones por cada variación. El resumen del resultado obtenido se observa en la tabla 21.

Tabla 20. Variaciones de Alpha

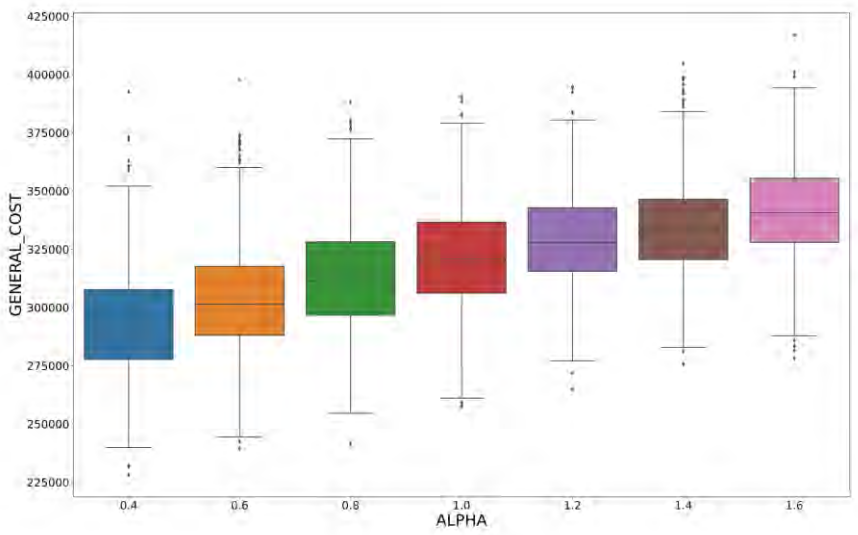
Alpha	**0.4**	**0.6**	**0.8**	**1.0**	**1.2**	**1.4**	**1.6**

Tabla 21. Resumen de las variaciones de Alfa

Alfa	*Costo general promedio*	*Desviación estándar del costo*	*Tiempo de ejecución promedio por solución*	*Desviación estándar del tiempo de ejecución por solución*
0.4	293464	23154	0.25	0.06
0.6	303671	22867	0.25	0.06
0.8	312733	23582	0.24	0.06
1	321205	21700	0.24	0.06
1.2	329265	20703	0.24	0.07
1.4	334081	19880	0.24	0.05
1.6	341710	20321	0.25	0.07

De la misma forma para visualizar mejor los resultados se elaboró un gráfico de cajas tal y como se muestra en la figura 26. En esta se puede observar que con alfa 0.4 la media del costo es la menor respecto a los otros valores de alfa y a medida que el valor de alfa va incrementándose el costo también. Esto se debe a que se le está restando la influencia de las feromonas y se está tomando más en cuenta el valor de la heurística. El problema con esto es que el algoritmo puede estar convirtiéndose en codicioso, tratando de ir siempre por la ruta de menor costo, lo cual a largo plazo puede que se quede atrapado en un óptimo local.

Figura 26. Gráfico de cajas. Variación de Alfa vs Costo



- **Calibración de Beta**

Para beta, se tiene como valor base 2, esta va a variar desde 1 a 2.8 con saltos de 0.2 tal y como se muestra en la tabla 22. Asimismo, los demás parámetros se mantuvieron constantes y se realizaron 50 ejecuciones por cada variación. El resumen del resultado obtenido se observa en la tabla 23.

Tabla 22. Variaciones de Beta

Beta	**1.0**	**1.2**	**1.4**	**1.6**	**1.8**	**2.0**	**2.2**	**2.4**	**2.6**	**2.8**

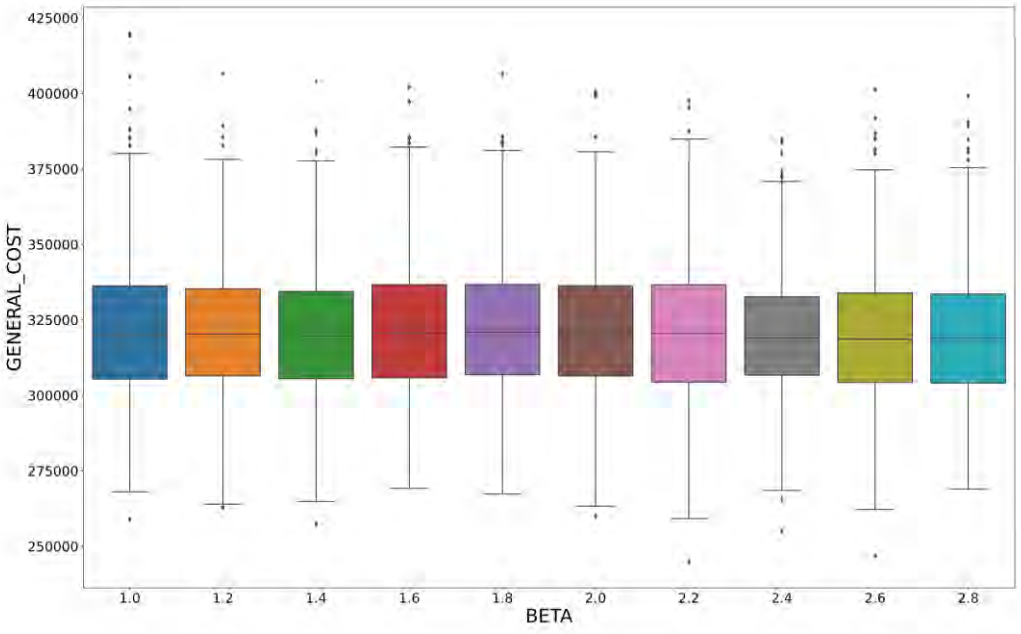
Tabla 23. Resumen de las variaciones de Beta

Beta	***Costo general promedio***	***Desviación estándar del costo***	***Tiempo de ejecución promedio por solución***	***Desviación estándar del tiempo de ejecución por solución***
1	321879	22890	0.28	0.14
1.2	320789	21286	0.27	0.07
1.4	320936	21803	0.28	0.06
1.6	322106	22442	0.27	0.06
1.8	321770	21259	0.27	0.06

2	322017	22614	0.27	0.06
2.2	321277	23101	0.30	0.07
2.4	320112	20458	0.29	0.09
2.6	320082	22035	0.29	0.07
2.8	320391	21593	0.28	0.07

Al igual que la calibración anterior, se elaboró un gráfico de cajas para poder visualizar los resultados obtenidos. En la figura 27 se puede observar que las medias son muy similares entre sí. Esto puede deberse a que como se mantuvo a alfa igual a 1 y beta mayor a 1, la mayor influencia la va a tener las heurísticas por lo que incrementar o decrementar en pequeñas proporciones no va a cambiar mucho el resultado final. No obstante, al observar los valores atípicos con menor costo los que sobresalen es 2.2, lo cual indica que encontró una buena solución. Esto indica que el valor 2.2 para beta es la mejor opción, dado que este es robusto a pequeñas variaciones y que puede encontrar buenas soluciones.

Figura 27. Gráfico de cajas. Beta vs Costo



- **Calibración de rho**

Para rho o la tasa de evaporación, se tiene como valor base 0.1, esta va a variar desde 0.04 a 0.14 con saltos de 0.02 tal y como se muestra en la tabla 24. Asimismo, se los demás parámetros se mantuvieron constantes y se realizaron 50 ejecuciones por cada variación. El resumen del resultado obtenido se observa en la tabla 25.

Tabla 24. Variaciones de Rho

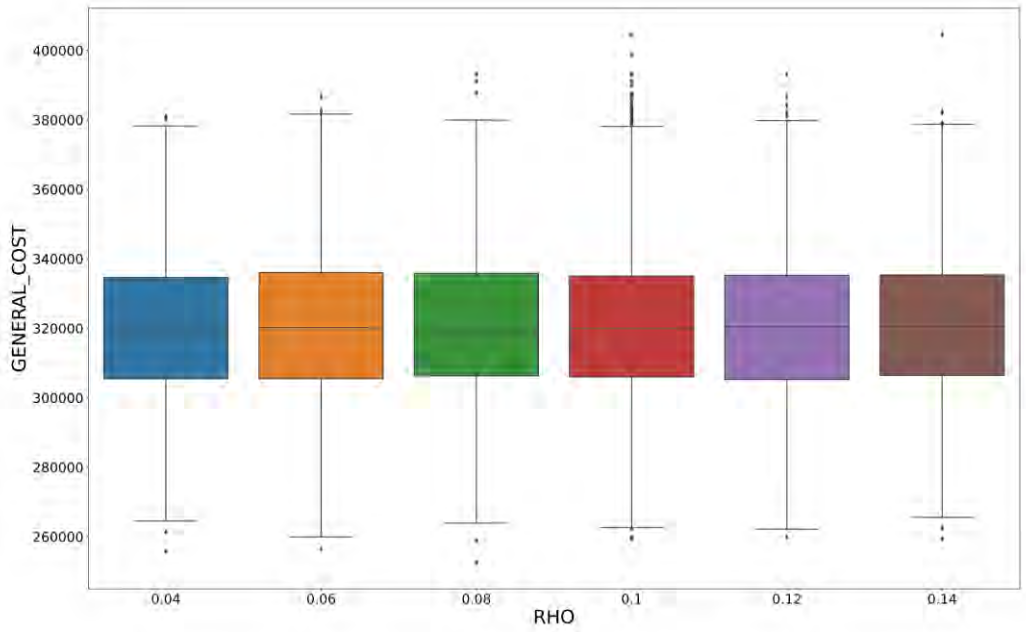
Rho	**0.04**	**0.06**	**0.08**	**0.1**	**0.12**	**0.14**

Tabla 25. Resumen de las variaciones de Rho

Rho	***Costo general promedio***	***Desviación estándar del costo***	***Tiempo de ejecución promedio por solución***	***Desviación estándar del tiempo de ejecución por solución***
0.04	320075	21530	0.24	0.05
0.06	321127	22183	0.23	0.05
0.08	321069	21913	0.23	0.06
0.1	321047	21778	0.23	0.05
0.12	321289	21657	0.23	0.05
0.14	321348	21457	0.23	0.05

El gráfico de cajas de las variaciones de rho mostradas en la figura 28 evidencia que las medias no tienen una gran variación. Sin embargo, se puede observar que para rho 0.1 se tiene un número elevado de valores atípicos con costos muy elevados a comparación con los otros valores. También, se puede observar que los valores atípicos para rho 0.08 tiene los menores valores en cuanto al costo. Además, su media es una de las más bajas respecto a los demás.

Figura 28. Gráfico de cajas. Rho vs Costo



- **Calibración del número de hormigas**

Para este parámetro no se consideró un valor base, pues este depende del problema. Pero cómo para determinar la mejor configuración inicial se empleó 30, este valor va a ser tomado como punto de partida. Así este se va a incrementar de 10 en 10 hasta llegar a las 70 hormigas.

Tabla 26. Variación del número de hormigas

Número de hormigas	**30**	**40**	**50**	**60**	**70**

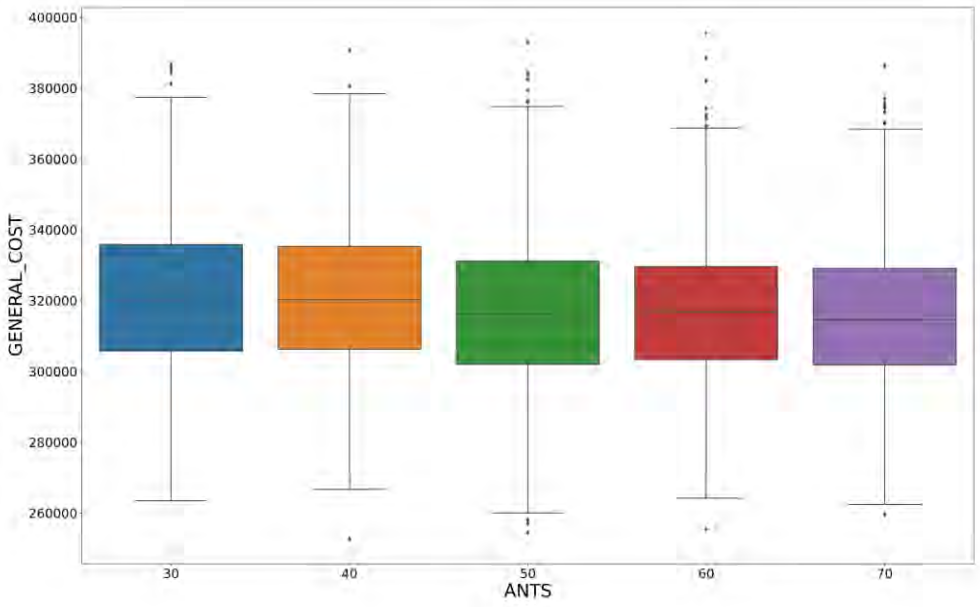
Tabla 27. Resumen de las variaciones del número de hormigas

Número de hormigas	***Costo general promedio***	***Desviación estándar del costo***	***Tiempo de ejecución promedio por solución***	***Desviación estándar del tiempo de ejecución por solución***
30	321397	22392	0.24	0.04
40	321180	21336	0.23	0.05
50	317164	22098	0.24	0.07

60	317181	20074	0.24	0.06
70	316238	20617	0.24	0.06

En la figura 26 se muestra los datos de la tabla 29 en un gráfico de cajas. En este se puede observar que con 50 hormigas se obtienen buenos resultados. Además de ello, este tiene los datos atípicos con menor valor.

Figura 29. Gráfico de cajas. Número de hormigas vs Costo



- **Calibración del número de iteraciones**

Al igual que para la calibración del número de hormigas, este parámetro no se le consideró un valor base, pues este también depende del problema. Pero como al inicio de las pruebas se tomó 100 iteraciones como valor inicial, se va a tomar como punto de partida. Este va a ser incrementado de 100 en 100 hasta llegar a 1000 iteraciones.

Tabla 28. Variaciones del número de iteraciones

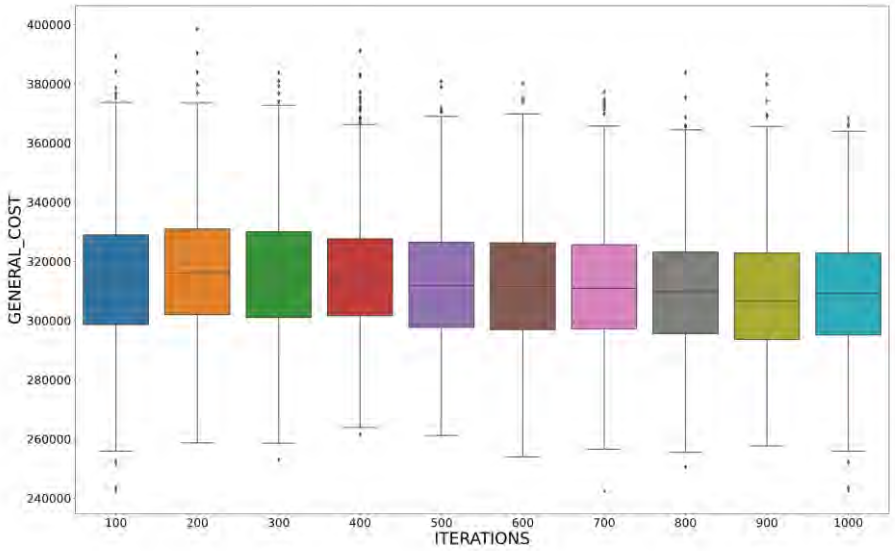
Número de iteraciones	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**

Tabla 29. Resumen de las variaciones del número de iteraciones

Número de iteraciones	*Costo general promedio*	*Desviación estándar del costo*	*Tiempo de ejecución promedio por solución*	*Desviación estándar del tiempo de ejecución por solución*
100	314290	22254	0.23	0.05
200	316666	21162	0.23	0.05
300	315694	21420	0.22	0.06
400	314951	20732	0.23	0.08
500	312853	20945	0.23	0.05
600	312145	21047	0.22	0.05
700	311912	20997	0.22	0.05
800	310102	20269	0.22	0.06
900	308583	21573	0.22	0.05
1000	309587	20567	0.22	0.05

La representación gráfica de la tabla 29 se ve reflejada en la figura 30 como un gráfico de cajas. En este se puede apreciar que la variación de medias no es muy amplia, los que tienen menor media son 900 y 1000. Sin embargo, con 100 iteraciones se obtiene una media muy similar, pero a un menor costo computacional. Asimismo, sus valores atípicos con menor costo son cercanos a los valores atípicos de 900 y 1000 iteraciones.

Figura 30. Gráfico de cajas. Número de iteraciones vs Costo



Configuración seleccionada

Luego de observar el comportamiento de los parámetros bajo ciertas variaciones se llegó a que la mejor configuración para el algoritmo colonia de hormigas es la siguiente

ALFA	*BETA*	*RHO*	*NÚMERO DE HORMIGAS*	*NÚMERO DE ITERACIONES*
1	2.2	0.08	50	100

Validaciones

El presente resultado fue revisado y aprobado por el especialista en algoritmia. La evidencia de dicha revisión se encuentra en el Anexo E, sección R8.

6.2.3 R9. Experimentación numérica de evaluación de los algoritmos primero el mejor y colonia de hormigas

En este resultado se busca evidenciar que el algoritmo colonia de hormigas planteado para el problema encuentra buenas soluciones y esto queda demostrado al compararlo contra un algoritmo de búsqueda voraz. Dicha comparación se realiza sobre el costo general que incurren para encontrar la solución.

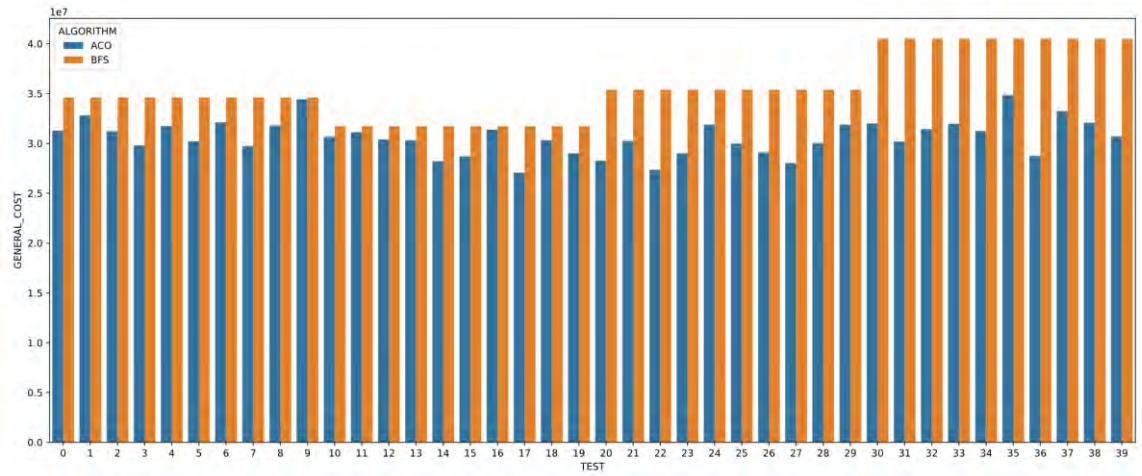
Este resultado se divide en dos secciones. La primera parte abarca el diseño de la experimentación y la segunda el análisis de los resultados obtenidos.

Diseño del experimento

Para la ejecución del algoritmo, se elaboraron 4 grafos distintos de 50 nodos cada uno. Dichos grafos cuentan con un solo punto de partida o almacén con la coordenada (0;0). Las coordenadas en el eje de las abscisas de cada nodo distinto al almacén se encuentran en el rango [-1200; 1200] y, de la misma manera, el rango de las ordenadas se encuentra en el rango de [-1200; 1200].

Para el experimento se realizaron 10 repeticiones sobre cada uno de los grafos, dando en total 40 resultados por cada algoritmo. Los resultados que se obtuvieron se presentan en la figura 31.

Figura 31. Costo de la solución encontrada por cada prueba



Análisis de los resultados

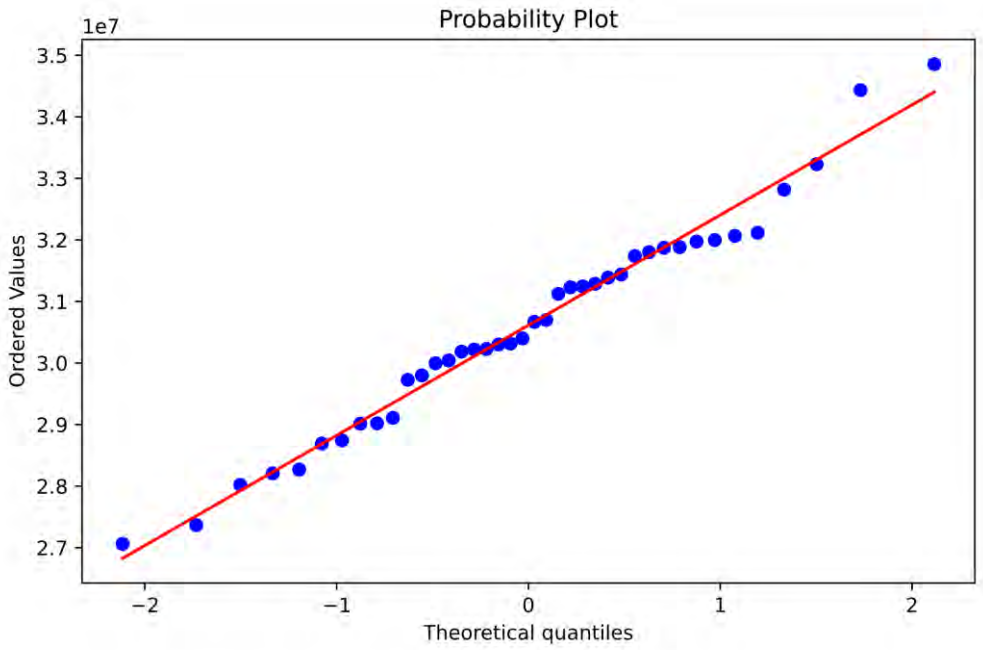
A primera vista, en la figura 31 se ve que el algoritmo colonia de hormigas tiene mejores resultados, pues incurren en costos menores en todas las pruebas. Sin embargo, es necesario evaluar si la diferencia de medias de los resultados obtenidos para ambos algoritmos es significativa. Para ello es necesario realizar ciertas pruebas sobre estos.

En primer lugar, se verifica si los resultados obtenidos siguen una distribución normal, para luego aplicar pruebas paramétricas. En el caso de que no sigan una distribución normal se emplearán pruebas no paramétricas.

- **Pruebas de normalidad**

Para el algoritmo colonia de hormigas se realizó un gráfico cuantil-cuantil para ver si es que sigue una distribución (Ver figura 32). En esta gráfica, aparentemente sigue una distribución normal por lo que se pues los puntos se acercan a la línea de la distribución normal.

Figura 32. Gráfico cuantil-cuantil sobre los resultados del algoritmo colonia de hormigas y una distribución normal



Para asegurarse que realmente se aproxima a una distribución normal se realiza la prueba de normalidad de Shapiro–Wilk con las siguientes hipótesis:

H_0 : L[illegible]

H_1 : [illegible]

1

Figura 33. Prueba de Shapiro-Wilk para la muestra del algoritmo colonia de hormigas

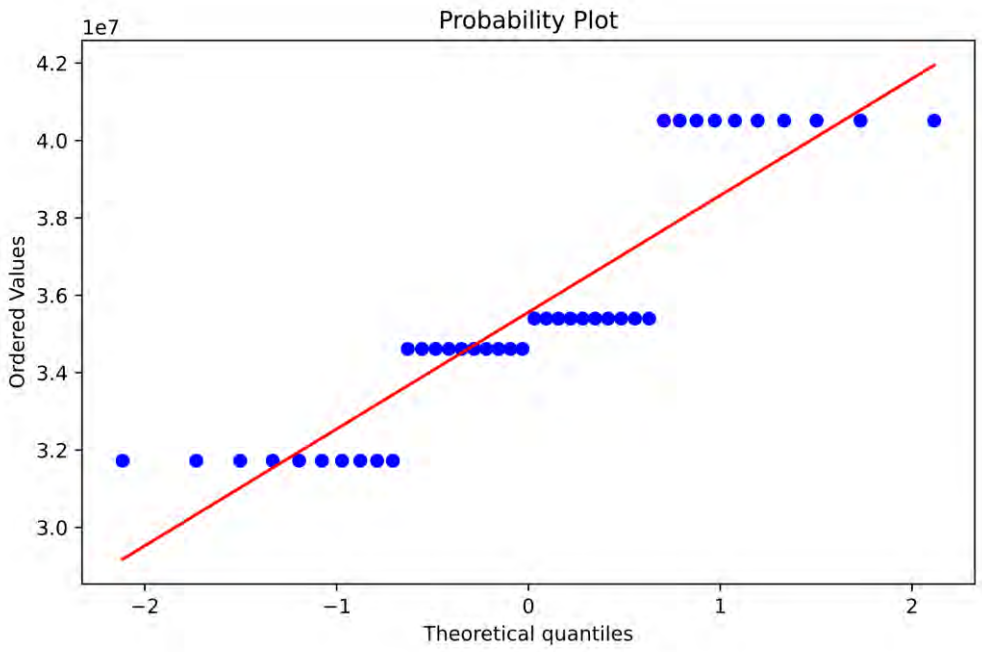
```
W, p = stats.shapiro(ndf[ndf['ALGORITHM']=='ACO']['GENERAL_COST'])
print(f'Shapiro test:\nW:\t\t{W}\np-value:\t{p}')
```

```
Shapiro test:
W:              0.9812243580818176
p-value:        0.7348422408103943
```

Como se ve en la figura 31, con un nivel de significancia de 0.05, el p-valor obtenido es mayor a este por lo que no se puede rechazar la hipótesis nula. Así que se puede concluir que la muestra sigue una distribución normal.

De la misma manera para el algoritmo voraz primero el mejor se realizó un gráfico cuantil-cuantil para ver si es que este sigue una distribución normal (Ver figura 32). En dicho gráfico se puede observar que está muy alejado de la recta de la distribución normal y de hecho sus valores para cada prueba con el mismo grafo son constantes.

Figura 34. Gráfico cuantil-cuantil sobre los resultados del algoritmo primero el mejor y una distribución normal



Para confirmar que no sigue una distribución normal, también, se realiza la prueba de Shapiro-Wilk con las mismas hipótesis:

H_0: L[illegible]

H_1: L[illegible]

Figura 35. Prueba de Shapiro-Wilk para la muestra del algoritmo primero el mejor

```
W, p = stats.shapiro(ndf[ndf['ALGORITHM']=='BFS']['GENERAL_COST'])
print(f'Shapiro test:\nW:\t\t{W}\np-value:\t{p}')
```

```
Shapiro test:
W:              0.8137853145599365
p-value:        1.3300722457643133e-05
```

En la figura 33 queda evidenciado que a un nivel de significancia de 0.05 la muestra de los resultados obtenidos por el algoritmo primero el mejor no sigue una distribución normal, pues el p-valor obtenido es mucho menor que este y por ende se rechaza la hipótesis nula.

- **Prueba sobre las medias**

Dado que la muestra del algoritmo primero el mejor no sigue una distribución normal, se realizó la prueba no paramétrica de Wilcoxon para verificar si la diferencia de medias entre ambas muestras es significativa. Así, se plantean la siguiente hipótesis:

H_0: L[illegible]

H_1: L[illegible]

Figura 36. Prueba de Wilcoxon para la media de las muestras de los algoritmos

```
W, p = stats.wilcoxon(aco_cost,bfs_cost, alternative='two-sided')
print(f'Shapiro test:\nW:\t\t{W}\np-value:\t{p}')
```

```
Shapiro test:
W:              0.0
p-value:        3.569388204466033e-08
```

Para un nivel de significancia de 0.05, se puede rechazar la hipótesis nula, pues el p-valor obtenido de la prueba (Ver figura 34) es mucho menor que esta, por lo que se acepta la hipótesis alternativa.

Entonces se puede concluir que el algoritmo colonia de hormigas en media tiene mejores resultados que el algoritmo primero el mejor, por ende, queda evidenciado que el algoritmo resuelve el problema planteado con buenas soluciones.

Validaciones

El presente resultado fue revisado y aprobado por el especialista en algoritmia. La evidencia de dicha revisión se encuentra en el Anexo E, sección R9.

6.2.4 R10. Caso aplicativo del algoritmo colonia de hormigas en un escenario real

Luego de haber verificado que el algoritmo colonia de hormigas si nos brinda buenos resultados, en esta sección se va a realizar una prueba del algoritmo en un escenario real para los hospitales de una región en concreto que en este caso será Tumbes.

Contexto

El contexto bajo el cual se va a trabajar es la distribución de medicinas a los hospitales de una región bajo un escenario de pandemia en el Perú. Actualmente, el Perú está atravesando una crisis sanitaria ocasionada por el virus COVID-19, es por ello que el gobierno del ha decretado ciertas medidas para enfrentar la pandemia (El Peruano, 2021).

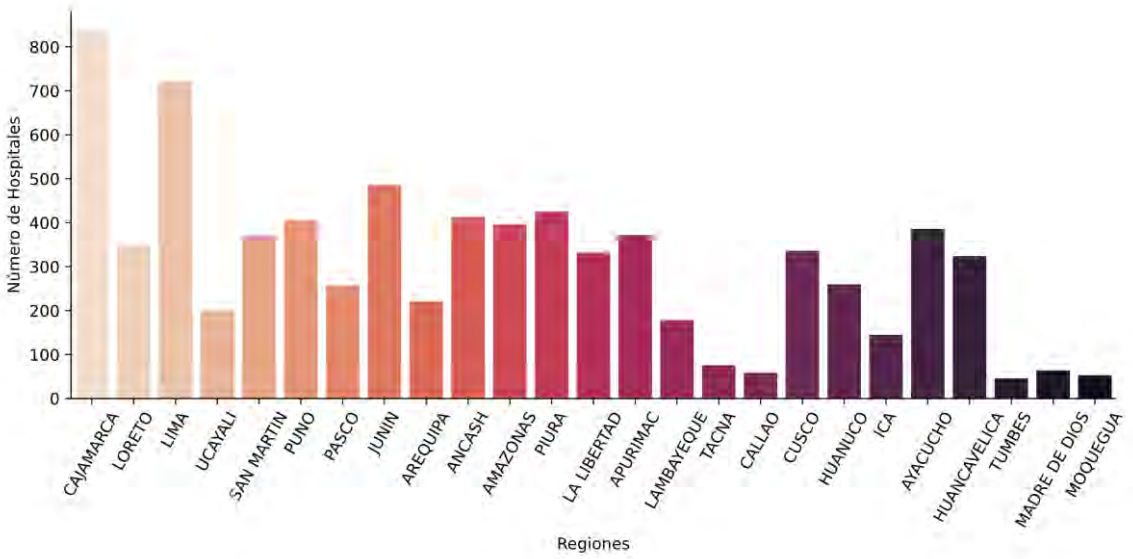
Entre estas medidas, se encuentra la declararon de 4 niveles de alerta para las provincias del Perú de acuerdo con índice de contagios en dicha provincia. Los niveles van de extremo, muy alto, alto y moderado (Gobierno del Perú, 2021). Estos niveles brindados por el gobierno van a ser empleados como parte del preprocesamiento de la información para la elaboración del grafo que se va a emplear.

Asimismo, es importante tomar en cuenta la geografía del Perú para realizar los repartos de las medicinas a los hospitales. Es por ello por lo que se necesita tomar en cuenta a qué altura geográfica se encuentran los hospitales, pues no es lo mismo enviar un vehículo con suministros de una ciudad de la costa a otra ciudad de la costa que enviar de una ciudad de la costa a una ciudad en la sierra.

Obtención y procesamiento de los datos

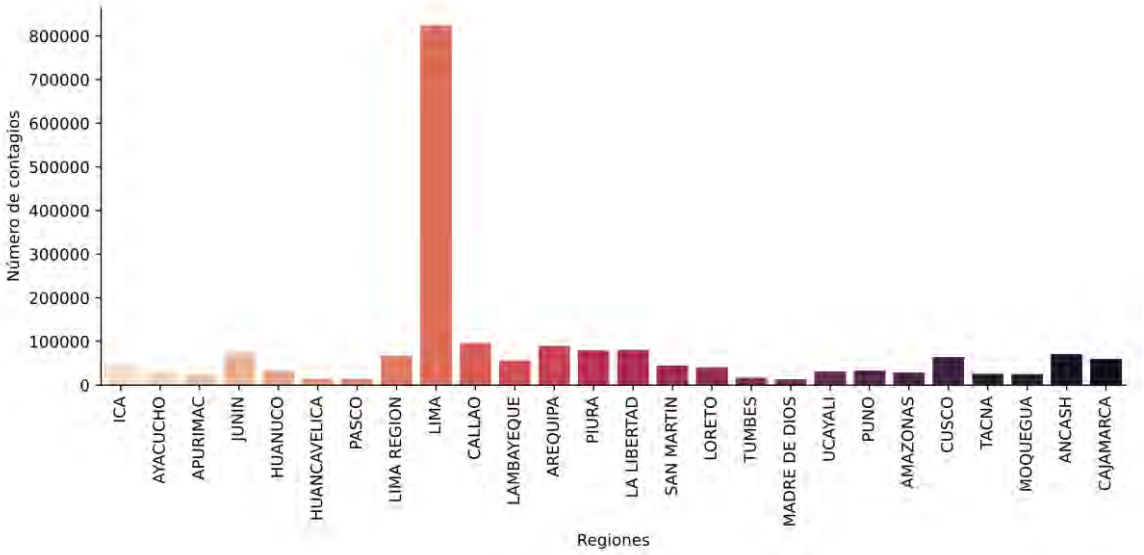
Para la construcción del grafo que va a ser procesado por el algoritmo, es necesario tener información acerca de los hospitales que son manejados por el estado peruano y los niveles de alerta de cada provincia. Para ello, se consultó el repositorio de la Plataforma Nacional de Datos Abiertos. En esta plataforma se obtuvo información acerca de las instituciones de salud registradas a nivel nacional, la cual contiene datos sobre el tipo de institución, el nombre de la institución, su ubicación, entre otros datos. Asimismo, se obtuvo información acerca del número de contagiados por regiones, provincias y distritos que luego van a ser asignados a uno de los niveles de alerta que ya fueron mencionados.

Figura 37. Número de hospitales por región



Como se puede observar en la figura 37, la región que cuenta con más establecimientos de salud, entre hospitales y postas médicas, es Cajamarca con 843 establecimientos y el que menos posee es Tumbes con 46.

Figura 38. Número de contagios por región



En la figura 38 se puede apreciar que la región con mayor número de contagios es Lima con más de 800000 contagiados y la región con menor número de contagiados es Madre de Dios con 13404 a la fecha del 12 de junio de 2021.

A pesar de que se tenga la dirección de los hospitales, es necesario tener sus coordenadas para poder calcular las distancias y los tiempos de viaje. Para ello se va a hacer uso del servicio web de Google Maps. Este cuenta con servicios que nos permitirá encontrar las coordenadas de cada uno de los establecimientos de salud. Asimismo, se pueden calcular las distancias y el tiempo de viaje entre dos puntos tomando en cuenta la diferencia de altitud y las velocidades máximas permitidas en las vías. Sin embargo, al estar dentro de la capa gratuita del servicio se tiene un límite de peticiones. Por ejemplo, para encontrar la matriz de distancias entre 15 coordenadas será necesario realizar 225 peticiones, pero esto tiene un crecimiento cuadrático por lo que encontrar la matriz de distancias y tiempos para más de 100 coordenadas sobrepasará el límite mensual permitido. Es por esta limitante que se eligió la región de Tumbes para hacer las pruebas del algoritmo, pues solo cuenta con 46 centros de salud, lo cual nos daría una matriz de 46 x 46 entradas con un total de 2116 peticiones, estando así por debajo del límite permitido.

Dado que se va a trabajar con la región Tumbes, es pertinente analizar los datos a detalle de este. Por un lado, como se puede observar en la figura 39, el distrito con más contagios es la capital de Tumbes con aproximadamente 10000 personas contagiadas, mientras que el distrito con menos contagios es Casitas con alrededor de

20 contagiados. Por otro lado, en la figura 40 se observa que el distrito de Tumbes es el que cuenta con mayor número de centros de salud. Asimismo, en la figura 41 se muestra la distribución geográfica de estos centros de salud.

Figura 39. Número de contagios por distritos de Tumbes

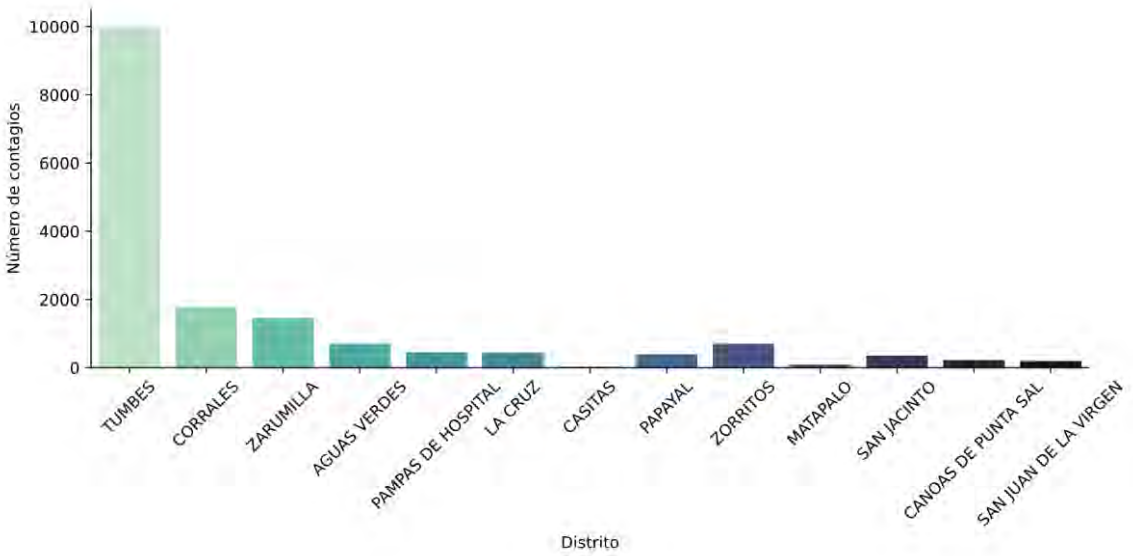


Figura 40. Número de hospitales por distritos de Tumbes

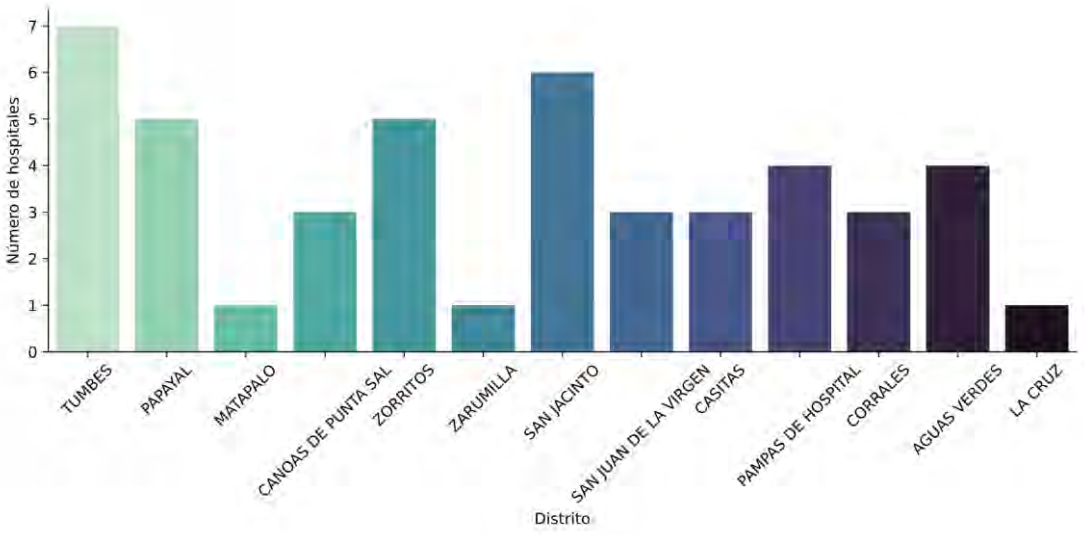
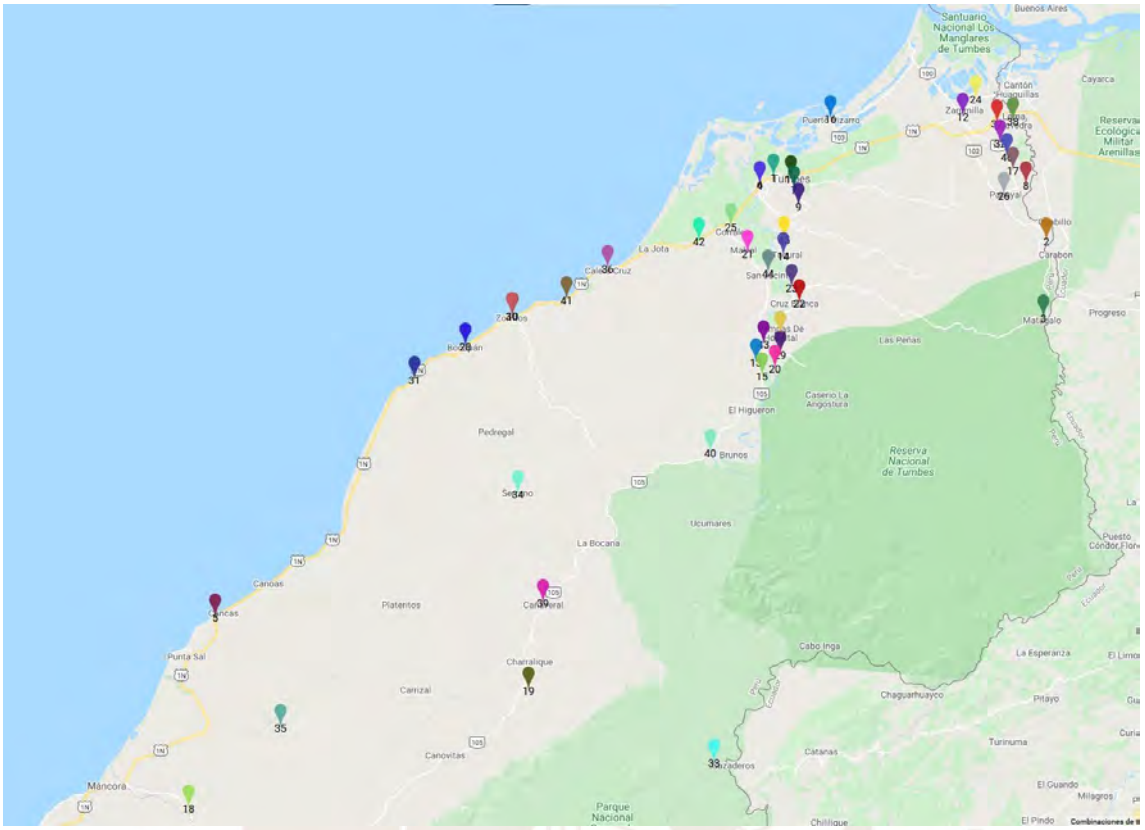


Figura 41. Ubicación de los centros médicos de Tumbes



Con estos datos obtenidos, junto con la matriz de distancias y tiempos de todos los puntos contra todos, se construyó el grafo sobre el cual se va a ejecutar el algoritmo. Este consiste en una serie de nodos que representa cada centro de salud, en la cual se tiene su coordenada. Con respecto nivel de alerta asignada esta se encuentra etiquetada de la siguiente manera

Tabla 30. Niveles de alerta

Extrema	***Muy Alto***	***Alto***	***Moderado***
1	2	3	4

Los siguientes distritos se encuentran etiquetados con los siguientes niveles de alerta de acuerdo con el número de contagios por distrito

Tabla 31. Nivel de alerta por distrito de Tumbes

Nivel de alerta	***Distritos***

1	- Tumbes - Corrales - Zarumilla
2	- Zorritos - Aguas Verdes - Pampas De Hospital - La Cruz - Papayal
3	- San Jacinto - Canoas De Punta Sal - San Juan De La Virgen
4	- Matapalo - Casitas

Con el que finalmente se elaboró el grafo, cabe resaltar que como almacén se eligió el Hospital de Tumbes que se encuentra en el distrito de Tumbes, pues además de ser el más grande de la región es el hospital principal de Tumbes. Asimismo, la demanda de cada hospital fue asignada aleatoriamente puesto que no se tiene información acerca de ello.

Ejecución del algoritmo colonia de hormigas

En este apartado se ejecutó el algoritmo colonia de hormigas con la mejor configuración encontrada en el resultado R7 para vehículos con una capacidad de 150 unidades. Además, se realizaron 10 repeticiones para obtener el mejor resultado posible.

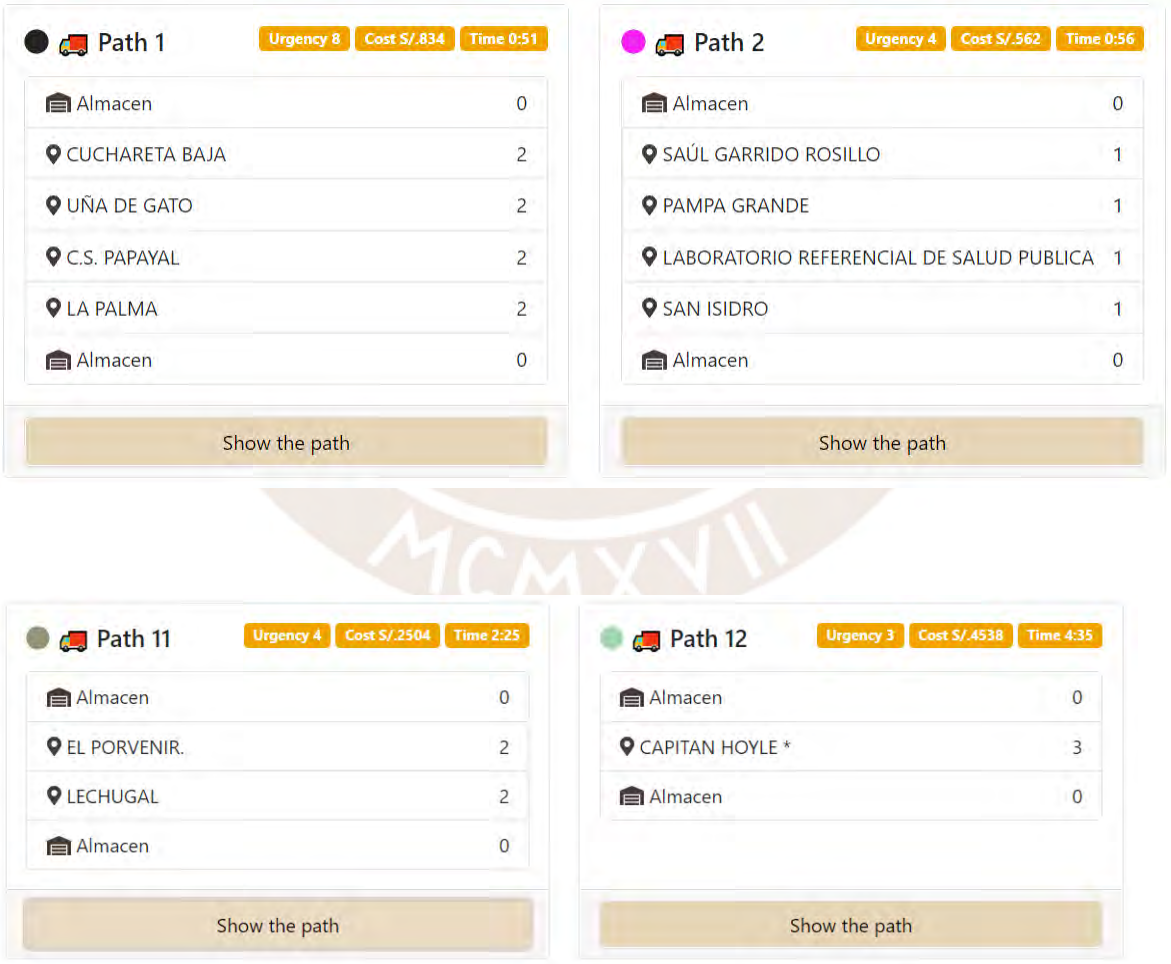
Tabla 32. Resultados de la ejecución del algoritmo colonia de hormigas sobre el grafo de Tumbes

Ejecución	*Tiempo (HH:MM:SS)*	*Costo (S/.)*	*Número de rutas*	*Costo general*
1	28:35:00	27,006	15	441,732,278
2	28:59:00	28,272	12	489,440,661
3	29:56:00	28,194	12	455,669,856
4	29:36:00	27,338	12	452,806,760
5	27:05:00	25,130	12	411,735,288
6	29:11:00	27,404	13	437,008,897

7	29:55:00	27,562	14	483,587,890
8	28:17:00	26,243	13	414,551,596
9	27:53:00	25,881	14	426,263,248
10	28:09:00	26,498	13	419,701,976

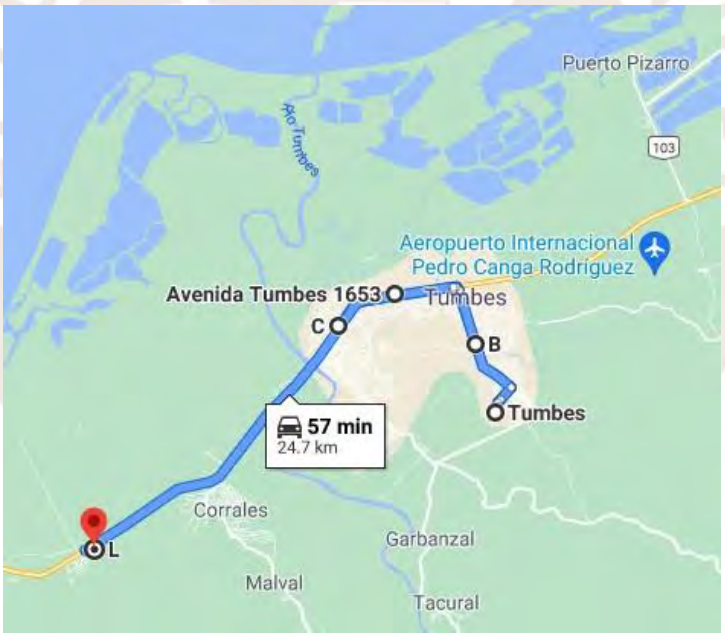
Como se puede apreciar en la tabla 30 en general se obtuvieron buenos resultados y el mejor de entre todos ellos se dio en la quinta repetición, en el cual se obtuvo el menor tiempo, costo y número de rutas con respecto a las otras ejecuciones. Al ver al detalle dicha solución se puede se aprecian las tarjetas de las rutas para cubrir la demanda de todos los hospitales de la región, cada uno con un tiempo y costo de entrega aproximado (Ver Figura 42).

Figura 42. Detalle de las primeras dos y ultimas dos rutas de la mejor solución encontrada de las 10 repeticiones



En la figura 42 se aprecia el detalle de la mejor solución encontrada y como se pude ver cada una de las rutas tiene como primeros puntos a visitar centros de salud que se encuentran en las áreas con el nivel más alto de alerta e incluso en la segunda ruta solo se prioriza el reparto de los suministros a estos puntos. Al seleccionar ver camino se aprecia que efectivamente dichos puntos pertenecen al distrito de Tumbes (Ver Figura 43) y la duración del reparto es de alrededor de 1 hora y el costo es de aproximadamente 562 soles para cubrir la demanda de esa ruta. Si vemos en general todas las rutas elaboradas por el algoritmo (Ver el anexo E, apartado R10 Mejor solución encontrada) se puede apreciar que existe un balance entre el costo y el tiempo de entrega. Además, esta toma en cuenta los niveles de alertas asignados para cada uno de los hospitales priorizando la entrega a los que se encuentran en el nivel extremo y muy alto. De esta manera se puede comprobar que el algoritmo colonia de hormigas funciona bien en escenarios reales.

Figura 43. Ruta 2 de la mejor solución encontrada



6.3 Discusión

En este capítulo se presentaron los tres últimos resultados: el diseño y codificación del algoritmo voraz, la calibración de los parámetros del algoritmo colonia de hormigas y, por último, la experimentación numérica para comparar los resultados obtenidos por ambos algoritmos.

En el primer resultado alcanzado ya se tiene el algoritmo primero el mejor ya codificado listo para su uso en la resolución del problema. Además, el resultado permite conocer a profundidad cómo es que este algoritmo abarca los problemas y entender que optar siempre por la ruta de menor costo sin considerar el costo subsiguiente no siempre es la mejor opción. Para alcanzar este resultado, se revisó el algoritmo y se modificó para adecuarlo al problema de este proyecto de tesis. Durante esta modificación se emplearon conceptos del problema ya establecidos previamente lo cual facilitó su desarrollo.

El segundo resultado alcanzado se calibraron los principales parámetros del algoritmo colonia de hormigas. Esto permitió establecer en una primera instancia los parámetros que mejoren los resultados del algoritmo basado en estudios que abordan el problema de ruteo de vehículos, luego se realizaron pequeñas variaciones sobre la mejor configuración para evaluar alguna mejoría, pues a pesar de tratar el problema de ruteo son situaciones distintas.

El tercer resultado sobre la comparación estadística de los resultados de ambos algoritmos se realizó para demostrar que el algoritmo colonia de hormigas encuentra buenas soluciones al problema planteado. A pesar de que esto era de esperarse, es importante conocer qué tan buenas son las soluciones respecto a un algoritmo voraz y determinar si es que existe una diferencia significativa, lo cual quedó comprobado en dicho resultado.

Finalmente, el último resultado alcanzado muestra el comportamiento del algoritmo colonia de hormigas bajo un escenario real y su desempeño en este. Se observó que este cumplía con las restricciones del problema y que, además, nos brinda buenos resultados pues satisface lo que se esta esperando como es entregar de manera oportuna los suministros a los hospitales que lo necesitan y no incurrir en costos elevados.

Capítulo 7. Conclusiones y trabajos futuros

7.1 Conclusiones

Después de haber concluido con todos los resultados esperados de cada objetivo para lograr desarrollar e implementar un algoritmo colonia de hormigas y que esta pueda brindar soluciones al problema de ruteo de vehículos con capacidades en una situación de emergencia pandémica con el fin de definir una ruta de abastecimiento aplicada al Perú, se llegaron a las siguientes conclusiones por cada objetivo específico.

Para el primer objetivo, se elaboraron las estructuras de datos que soporten las variables y restricciones que necesita el algoritmo para resolver el problema. En este para alcanzar el resultado de definir los nodos, aristas y el grafo que van a representar el área de distribución se emplearon clases para cada uno de ellos y para almacenar los datos se vio que la mejor alternativa era emplear mapas hash puesto que permiten el acceso instantáneo a los datos y se emplea la memoria necesaria para guardar los datos a diferencia de las matrices o listas enlazadas simples.

Para el segundo objetivo, se planteó la función objetivo que se busca optimizar como primer resultado, para ello fue necesario plantear previamente los parámetros y restricciones del problema. Gracias a este planteamiento previo del problema, se llegó a definir tres funciones objetivo a minimizar: el tiempo de entrega de suministros, el costo incurrido para dichas entregas y la atención prioritaria a las zonas en emergencia. Como segundo resultado, se diseñaron las funciones que emplean las feromonas de las hormigas como la función de probabilidad de transición, la función de evaporación y depósito de feromonas, para plantear estas funciones fue importante el planteamiento previo del problema y un análisis del rol que cada una de estas funciones cumple dentro del algoritmo planteado. Asimismo, el tercer resultado el cual es el diseño de las reglas de transición del problema para el algoritmo se mapearon las restricciones matemáticas del problema a criterios que se van a emplear en el algoritmo, lo cual facilitó el diseño de este. En cuanto al cuarto resultado se realizó el diseño del algoritmo colonia de hormigas tomando en cuenta todos los resultados anteriores y se llegó a plantear un pseudocódigo del algoritmo que luego se empleó para la codificación de este. Se le realizaron pruebas unitarias para comprobar que los métodos del algoritmo y las estructuras empleadas funcionen correctamente y una prueba de caja blanca para verificar que la solución encontrada mejore con cada

iteración y que cumplan con las restricciones del problema. Finalmente, el último resultado de este objetivo fue la implementación de la interfaz gráfica para que el algoritmo pueda ser empleado fácilmente por cualquier usuario en la que se pueda configurar los factores del algoritmo, ejecutarlo y visualizar los resultados obtenidos.

Finalmente, el tercer objetivo específico que abarcaba la comparación del algoritmo colonia de hormigas con un algoritmo voraz, se alcanzó como primer resultado la adaptación del algoritmo primero el mejor al problema planteado y su implementación. A este se le realizó una prueba de caja blanca para validar que siempre se decanta por el camino más corto. La calibración de los parámetros del algoritmo colonia de hormigas como segundo resultado inició tomando como base la mejor configuración encontrada en el estado del arte y a partir de esta se realizaron pequeñas variaciones en cada uno de estos parámetros para encontrar la configuración óptima con el propósito de mejorar la calidad de los resultados obtenidos por el algoritmo. En cuanto a la experimentación numérica, como tercer resultado, realizada con 4 conjuntos de datos generados aleatoriamente, se compararon los resultados obtenidos por ambos algoritmos y se validó que el algoritmo colonia de hormigas no está siendo voraz, sino que por el contrario este sí encuentra buenas soluciones al problema planteado pues no se queda estancado en óptimos locales. Por último, como último resultado, mediante un análisis de los resultados obtenidos en un escenario real de distribución de medicinas a hospitales de una región tomando en cuenta el número de contagios por distrito, los tiempos de viaje de hospital a hospital y los costos estimados de estos, se pudo determinar que el algoritmo colonia de hormigas tiene un buen comportamiento frente a estos casos pues en general suele encontrar soluciones que satisfacen las necesidades del problema.

7.2 Trabajos futuros

Las investigaciones que se proponen desde lo desarrollado en el presente trabajo de fin de carrera son las que se presentan a continuación:

- Implementar un sistema para el reparto de suministros médicos que administre la información de los centros de salud, de tal manera que esta pueda ser empleada por el algoritmo planteado para la generación de un plan de distribución.

- Mejorar el algoritmo planteado con la ayuda de algoritmos de búsqueda local para acelerar la ejecución y la calidad de los resultados en el problema presentado.
- Ampliar el problema a multi-almacenes y que estos se encuentren distribuidos en distintos puntos para que los vehículos puedan reabastecerse de suministros.
- Añadir restricciones al problema por el tipo de suministro como tamaño, peso o temperatura de tal manera que se optimice también sobre estas restricciones.

Referencias

Aduviri, R. (2018). *ALGORITMO GENÉTICO MULTIOBJETIVO PARA LA OPTIMIZACIÓN DE LA DISTRIBUCIÓN DE AYUDA HUMANITARIA EN CASO DE DESASTRES NATURALES EN EL PERÚ* [PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ]. http://hdl.handle.net/20.500.12404/15478

Akhand, M. A. H., Jannat, Z., Diba, T., & Al-Mahmud. (2016). Optimization of capacitated vehicle routing problem using Producer-Scrounger Method. *2015 IEEE International WIE Conference on Electrical and Computer Engineering, WIECON-ECE 2015*, 297–300. https://doi.org/10.1109/WIECON-ECE.2015.7443922

Akhand, M. A. H., Peya, Z. J., Sultana, T., Al-Mahmud, Zahrul Jannat Peya, Sultana, T., Al-Mahmud, Peya, Z. J., Sultana, T., & Al-Mahmud. (2016). Solving Capacitated Vehicle Routing Problem with route optimization using Swarm Intelligence. *2nd International Conference on Electrical Information and Communication Technologies, EICT 2015*, 112–117. https://doi.org/10.1109/EICT.2015.7391932

Akhand, M. A. H., Zahrul Jannat Peya, Sultana, T., Al-Mahmud, Peya, Z. J., Sultana, T., & Al-Mahmud. (2016). Solving Capacitated Vehicle Routing Problem with route optimization using Swarm Intelligence. *2nd International Conference on Electrical Information and Communication Technologies, EICT 2015*, 112–117. https://doi.org/10.1109/EICT.2015.7391932

Alaya, I., Solnon, C., & Ghedira, K. (2007). Ant Colony Optimization for Multi-Objective Optimization Problems. *19th IEEE International Conference on Tools with Artificial Intelligence(ICTAI 2007)*, 450–457. https://doi.org/10.1109/ICTAI.2007.108

Almossawi, A. (2017). *Bad Choices: How Algorithms Can Help You Think Smarter and Live Happier*. Viking.

Arango, M. B. (2019, September 12). Así es el proceso para abastecer de medicamentos a los hospitales del Ministerio de Salud. *El Comercio*. https://rpp.pe/peru/actualidad/asi-es-el-proceso-para-abastecer-de-medicamentos-a-los-hospitales-del-ministerio-de-salud-analisis-noticia-1219221?ref=rpp

Arvind, V. (2014). Complexity Theory Basics: NP and NL. In M. Agrawal & V. Arvind (Eds.), *Perspectives in Computational Complexity: The Somenath Biswas Anniversary Volume* (pp. 1–22). Springer International Publishing. https://doi.org/10.1007/978-3-319-05446-9_1

Baldacci, R., Mingozzi, A., & Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, *218*(1), 1–6. https://doi.org/10.1016/j.ejor.2011.07.037

Batmetan, J. R., Santoso, A. J., & Pranowo. (2017). A multiple-objective ant colony algorithm for optimizing disaster relief logistics. *Advanced Science Letters*, *23*(3), 2344–2347. https://doi.org/10.1166/asl.2017.8758

Beni, G. (2014). Swarm Intelligence. In *Encyclopedia of Complexity and Systems Science* (pp. 1–32). Springer New York. https://doi.org/10.1007/978-3-642-27737-5_530-4

Beni, G., & Wang, J. (1993). Swarm Intelligence in Cellular Robotic Systems. In *Robots and Biological Systems: Towards a New Bionics?* (pp. 703–712). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-58069-7_38

Bouhafs, L., Hajjam, A., & Koukam, A. (2004). A hybrid ant colony system approach for the capacitated vehicle routing problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3172 LNCS*, 414–415. https://doi.org/10.1007/978-3-540-28646-2_42

Brajevic, I. (2011). Artificial bee colony algorithm for the capacitated vehicle routing problem. *Proceedings of the European Computing Conference, ECC '11*, 239–244. https://www.scopus.com/inward/record.uri?eid=2-s2.0-80053140490&partnerID=40&md5=9476fbe990520ba39930a42731709eac

Calabrò, G, Torrisi, V., Inturri, G., & Ignaccolo, M. (2020). Improving inbound logistic planning for large-scale real-world routing problems: a novel ant-colony simulation-based optimization. *European Transport Research Review*, *12*(1). https://doi.org/10.1186/s12544-020-00409-7

Calabrò, Giovanni, Torrisi, V., Inturri, G., & Ignaccolo, M. (2020a). Improving inbound logistic planning for large-scale real-world routing problems: a novel ant-colony simulation-based optimization. *European Transport Research Review*, *12*(1). https://doi.org/10.1186/s12544-020-00409-7

Calabrò, Giovanni, Torrisi, V., Inturri, G., & Ignaccolo, M. (2020b). Improving inbound logistic planning for large-scale real-world routing problems: a novel ant-colony simulation-based optimization. *European Transport Research Review*, *12*(1). https://doi.org/10.1186/s12544-020-00409-7

Carwalo, T., Thankappan, J., & Patil, V. (2017). Capacitated vehicle routing problem. *2017 2nd International Conference on Communication Systems, Computing and IT Applications (CSCITA)*, 17–21. https://doi.org/10.1109/CSCITA.2017.8066555

Castillo, M. (2020, May 26). Perú parecía estar haciendo todo bien. Entonces, ¿cómo se convirtió en un punto crítico de covid-19? *CNN*. https://cnnespanol.cnn.com/2020/05/26/peru-parecia-estar-haciendo-todo-bien-entonces-como-se-convirtio-en-un-punto-critico-de-covid-19/

Chacon, S., & Straub, B. (2014). Pro Git. *Pro Git*. https://doi.org/10.1007/978-1-4842-0076-6

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R., & Schrijver, A. (1997). *Combinatorial Optimization*. John Wiley & Sons, Inc. https://doi.org/10.1002/9781118033142

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

Diario Gestión. (2020). *ComexPerú: precios altos de medicamentos "reflejan un problema de desabastecimiento."* https://gestion.pe/economia/coronavirus-peru-comexperu-precios-altos-de-medicamentos-reflejan-un-problema-de-desabastecimiento-nndc-noticia/?ref=gesr

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *The ant system: An autocatalytic optimizing process*.

Dorigo, M., & Stützle, T. (2004). *Ant Colony Optimization*. MIT Press. https://ieeexplore.ieee.org/book/6267250

Du, K. L., & Swamy, M. N. S. (2016). Search and optimization by metaheuristics: Techniques and algorithms inspired by nature. In *Search and Optimization by Metaheuristics: Techniques and Algorithms Inspired by Nature*. Springer International Publishing. https://doi.org/10.1007/978-3-319-41192-7

El Peruano. (2021, May 27). *Prórroga del Estado de Emergencia Nacional*. 6. https://www.gob.pe/12365-coronavirus-medidas-para-enfrentar-la-pandemia-segun-nivel-de-alerta-y-region

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. IEEE Press.

Gobierno del Perú. (2021). *Coronavirus: medidas para enfrentar la pandemia según nivel de alerta y región*. https://www.gob.pe/12365-coronavirus-medidas-para-enfrentar-la-pandemia-segu

n-nivel-de-alerta-y-region
Gomes, L. D. C. T., Von Zuben, F. J., de CT Gomes, L., & Von Zuben, F. J. (2002). A neuro-fuzzy approach to the capacitated vehicle routing problem. *Proceedings of the International Joint Conference on Neural Networks*, *2*, 1930–1935. https://doi.org/10.1109/IJCNN.2002.1007814
González, O. M., Segura, C., Peña, S. I. V, León, C., Gonzalez, O. M., Segura, C., Pena, S. I. V, & Leon, C. (2017). A memetic algorithm for the Capacitated Vehicle Routing Problem with Time Windows. *2017 IEEE Congress on Evolutionary Computation, CEC 2017 - Proceedings*, 2582–2589. https://doi.org/10.1109/CEC.2017.7969619
Hasle, G., & Kloster, O. (2007). Industrial Vehicle Routing. In G. Hasle, K.-A. Lie, & E. Quak (Eds.), *Geometric Modelling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF* (pp. 397–435). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-68783-2_12
Hidalgo García, M. (2020). PANDEMIAS:UN RIESGO EN AUGE en el siglo XXI. *Revista Española de Defensa*, 30–33. https://www.defensa.gob.es/Galerias/gabinete/red/2020/04/p-30-33-red-371-pandemias.pdf
Irnich, S., Schneider, M., & Vigo, D. (2014). Vehicle Routing. In P. Toth & D. Vigo (Eds.), *Vehicle Routing*. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611973594
JetBrains. (n.d.). *IntelliJ IDEA*. Retrieved November 1, 2020, from https://www.jetbrains.com/idea/features/
Kamphukaew, R., Sethanan, K., Jamrus, T., & Wang, H.-K. (2018). Differential evolution algorithms with local search for the multi-products capacitated vehicle routing problem with time windows: A case study of the ice industry. *Engineering and Applied Science Research*, *45*(4), 273–281. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85061856489&partnerID=40&md5=217fe71ac3cd8dc66c981b5810b21030
Kao, Y., Chen, M.-H., & Huang, Y.-T. (2012). A hybrid algorithm based on ACO and PSO for capacitated vehicle routing problems. *Mathematical Problems in Engineering*, *2012*. https://doi.org/10.1155/2012/726564
Khebbache, S., Prins, C., Yalaoui, A., & Reghioui, M. (2009). Memetic Algorithm for two-dimensional loading capacitated vehicle routing problem with time windows. *2009 International Conference on Computers Industrial Engineering*, 1110–1113. https://doi.org/10.1109/ICCIE.2009.5223786
Kitchenham, B. (2007). Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*.
Korayem, L., Khorsid, M., & Kassem, S. S. (2015). Using grey Wolf algorithm to solve the capacitated vehicle routing problem. In W. J. Ding J. Gaol F.L. (Ed.), *IOP Conference Series: Materials Science and Engineering* (Vol. 83, Issue 1). Institute of Physics Publishing. https://doi.org/10.1088/1757-899X/83/1/012014
Kurniawan, R., Sulistiyo, M. D., & Wulandari, G. S. (2016). Genetic Algorithm for Capacitated Vehicle Routing Problem with considering traffic density. *2015 International Conference on Information Technology Systems and Innovation, ICITSI 2015 - Proceedings*. https://doi.org/10.1109/ICITSI.2015.7437695
Kurniawan, R., Sulistiyo, M. D., & Wulandari, G. S. (2015). Genetic Algorithm for Capacitated Vehicle Routing Problem with considering traffic density. *2015 International Conference on Information Technology Systems and Innovation, ICITSI 2015 - Proceedings*, 1–6. https://doi.org/10.1109/ICITSI.2015.7437695
Li, Y., & Chung, S. H. (2019). Disaster relief routing under uncertainty: A robust

optimization approach. *IISE Transactions*, *51*(8), 869–886. https://doi.org/10.1080/24725854.2018.1450540

Li, Y., & Fan, H.-M. (2018). Hybrid variable neighborhood symbiotic organisms search for capacitated vehicle routing problem [求解带容量约束车辆路径问题的混合变邻域生物共栖搜索算法]. *Kongzhi Yu Juece/Control and Decision*, *33*(7), 1190–1198. https://doi.org/10.13195/j.kzyjc.2017.0346

Lindberg, S., & Strandberg, F. (2006). *The Development and Evaluation of a Unit Testing Methodology*. 161. https://pdfs.semanticscholar.org/24e6/5805ae9d920131a59d3973af59d89b594309.pdf

Lossio-Ventura J.A., A.-S. H. (Ed.). (2016). CEUR Workshop Proceedings. In *CEUR Workshop Proceedings* (Vol. 1743). CEUR-WS. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85006163525&partnerID=40&md5=0d75d3e0e59c0d1c46461da32b5d8ddd

Luke, S. (2013). *Essentials of Metaheuristics* (second). Lulu.

Marinakis, Y., & Marinaki, M. (2013). Combinatorial neighborhood topology particle swarm optimization algorithm for the vehicle routing problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *7832 LNCS*, 133–144. https://doi.org/10.1007/978-3-642-37198-1_12

Mingping Xia. (2009). A modified Ant Colony Algorithm with local search for capacitated vehicle routing problem. *2009 Asia-Pacific Conference on Computational Intelligence and Industrial Applications (PACIIA)*, *2*, 84–87. https://doi.org/10.1109/PACIIA.2009.5406543

MTC. (2017). *RED VIAL NACIONAL OFICIAL ESTADO DE LA SUPERFICIE DE RODADURA A DIC 2017*. https://www.pvn.gob.pe/wp-content/uploads/2018/07/estado_rvn_dic2017_20180720.pdf

Mutar, M. L., Burhanuddin, M. A., Hameed, A. S., Yusof, N., & Mutashar, H. J. (2020). An efficient improvement of ant colony system algorithm for handling capacity vehicle routing problem. *International Journal of Industrial Engineering Computations*, *11*(4), 549–564. https://doi.org/10.5267/j.ijiec.2020.4.006

Ng, K. K. H., Lee, C. K. M., Zhang, S. Z., Wu, K., & Ho, W. (2017). A multiple colonies artificial bee colony algorithm for a capacitated vehicle routing problem and re-routing strategies under time-dependent traffic congestion. *Computers and Industrial Engineering*, *109*, 151–168. https://doi.org/10.1016/j.cie.2017.05.004

Niazy, N., El-Sawy, A., & Gadallah, M. (2020). A hybrid chicken swarm optimization with tabu search algorithm for solving capacitated vehicle routing problem. *International Journal of Intelligent Engineering and Systems*, *13*(4), 237–247. https://doi.org/10.22266/IJIES2020.0831.21

Oliveira, R. A. de C., & Delgado, K. V. (2015). Capacitated Vehicle Routing System Applying Monte Carlo Methods. *Proceedings of the Annual Conference on Brazilian Symposium on Information Systems: Information Systems: A Computer Socio-Technical Perspective - Volume 1*, 1–8.

OPS. (2020). *Enfermedad por el Coronavirus (COVID-19)*. https://www.paho.org/es/tag/enfermedad-por-coronavirus-covid-19

Oracle. (n.d.). *¿Qué es Java?* Retrieved November 1, 2020, from https://www.java.com/es/about/

Organización Mundial de la Salud. (2010). *¿Qué es una pandemia?* https://www.who.int/csr/disease/swineflu/frequently_asked_questions/pandemic/es

/
Pané, G. H. (2020, March 25). Grandes pandemias de la historia. *Historia National Geographic*. https://historia.nationalgeographic.com.es/a/grandes-pandemias-historia_15178/4
Papadimitriou, C. H., & Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc.
Pavón Mariño, P. (2015). Heuristic Algorithms. In *Optimization of Computer Networks: Modeling and Algorithms: A Hands-On Approach* (pp. 266–300). Wiley. https://doi.org/10.1002/9781119114840.ch12
Pereira, F. B., & Tavares, J. (Eds.). (2009). *Bio-inspired Algorithms for the Vehicle Routing Problem* (Vol. 161). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-85152-3
Pinninghoff, M. A., Contreras, R., & Pantoja, C. (2014). A comparison of methods for the vehicle routing problem. In D. A. Ezzatti P. (Ed.), *Proceedings of the 2014 Latin American Computing Conference, CLEI 2014*. Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/CLEI.2014.6965167
Project Jupyter. (n.d.). *About Us. Some information about the Jupyter Project and Community*. https://jupyter.org/about
Pulido, H. G., & Salazar, R. de la V. (2008). Análisis y diseño de experimentos. In *Mc Graw Hill* (Segunda ed).
Punriboon, C., So-In, C., Aimtongkham, P., & Rujirakul, K. (2019). A bio-inspired capacitated vehicle-routing problem scheme using artificial bee colony with crossover optimizations. *Journal of Internet Services and Information Security*, *9*(3), 21–40. https://doi.org/10.22667/JISIS.2019.08.31.021
Python Software Foundation. (n.d.). *Python is powerful... and fast; plays well with others; runs everywhere; is friendly & easy to learn; is Open.* https://www.python.org/about/
Real Academia Española. (n.d.). *Pandemia*. Retrieved September 7, 2020, from https://dle.rae.es/pandemia
Ren, C. Y. (2013). Fast tabu search algorithm for capacitated vehicle routing problem. *Advanced Materials Research*, *760–762*, 1790–1793. https://doi.org/10.4028/www.scientific.net/AMR.760-762.1790
Rokach, L., & Maimon, O. (2007). *Data Mining with Decision Trees* (Vol. 69). WORLD SCIENTIFIC. https://doi.org/10.1142/6604
Russell, S., & Norvig, P. (2009). *Artificial Intelligence: A Modern Approach* (3rd ed.). Prentice Hall Press.
Salomón, O. (2017). *¿Qué está detrás de los problemas de abastecimiento de medicamentos en el sistema de salud peruano?* https://gestion.pe/blog/evidencia-para-la-gestion/2017/04/que-esta-detras-de-los-problemas-de-abastecimiento-de-medicamentos-en-el-sistema-de-salud-peruano.html/
Schwaber, K., & Sutherland, J. (2017). La Guía de Scrum. La Guía Definitiva de Scrum: Las Reglas del Juego. *Scrum.Org*, 22. http://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-Spanish-SouthAmerican.pdf#zoom=100
Shan, Q., & Wang, J. (2013). Solve capacitated vehicle routing problem using hybrid chaotic particle swarm optimization. *Proceedings - 6th International Symposium on Computational Intelligence and Design, ISCID 2013*, *2*, 422–427. https://doi.org/10.1109/ISCID.2013.218
Stodola, P., Mazal, J., Podhorec, M., & Litvaj, O. (2014). Using the Ant Colony Optimization algorithm for the Capacitated Vehicle Routing Problem. In M. D.

Brezina T. Stefek A. (Ed.), *Proceedings of the 16th International Conference on Mechatronics, Mechatronika 2014* (pp. 503–510). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/MECHATRONIKA.2014.7018311

Sun, X., Fu, Y., & Liu, T. (2017). A hybrid ACO algorithm for capacitated vehicle routing problems. In X. B. (Ed.), *Proceedings of 2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference, IAEAC 2017* (pp. 510–514). Institute of Electrical and Electronics Engineers Inc. https://doi.org/10.1109/IAEAC.2017.8054067

Toth, P., & Vigo, D. (Eds.). (2001). *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics.

Uy, C. H., Charoenlarpkul, N., Sarttra, T., & Rajsiri, S. (2019). An efficient algorithm applied to capacitated vehicle routing problem with consideration of time windows by using ranking-based concept and dynamic programming. *ACM International Conference Proceeding Series*, 267–274. https://doi.org/10.1145/3335550.3335588

Vazquez G, J. B. (2012). *Análisis y diseño de algoritmos*.

Wang, X., Choi, T., Liu, H., & Yue, X. (2018). A Novel Hybrid Ant Colony Optimization Algorithm for Emergency Transportation Problems During Post-Disaster Scenarios. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, *48*(4), 545–556. https://doi.org/10.1109/TSMC.2016.2606440

Wang, Y., Zhou, H., & Wang, Y. (2017). Research and application of genetic algorithm in path planning of logistics distribution vehicle. In K. J. Xiao J. Liu L. (Ed.), *AIP Conference Proceedings* (Vol. 1864). American Institute of Physics Inc. https://doi.org/10.1063/1.4992864

Wei, L., Zhang, Z., Zhang, D., & Leung, S. C. H. (2018). A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints. *European Journal of Operational Research*, *265*(3), 843–859. https://doi.org/10.1016/j.ejor.2017.08.035

Weiss, M. A. (1994). *Data Structures and Algorithm Analysis in C++*. Benjamin/Cummings Publishing Company. https://books.google.com.pe/books?id=_qAZAQAAIAAJ

World Banks, & United Nations. (2010). *Natural Hazards, UnNatural Disasters*. The World Bank. https://doi.org/10.1596/978-0-8213-8050-5

Xue, G. (2001). Handbook of Combinatorial Optimization DingZhu Du and Panos M. Pardalos (Co-Editors), Kluwer Academic Publishers, 1998, Vols. 1–3, ISBN: 0-7923-5285-8. *J. of Global Optimization*, *19*(4), 425–430. https://doi.org/10.1023/A:1011210425701

Yamina, S., Ahmed, S., & Kinza, M. N. (2013). Metaheuristic approach for solving the vehicle routing problem: Application in pharmaceutical society. *2013 International Conference on Control, Decision and Information Technologies, CoDIT 2013*, 684–690. https://doi.org/10.1109/CoDIT.2013.6689625

Yang, W., & Ke, L. (2019). An improved fireworks algorithm for the capacitated vehicle routing problem. *Frontiers of Computer Science*, *13*(3), 552–564. https://doi.org/10.1007/s11704-017-6418-9

Yesodha, R., & Amudha, T. (2019). Enhancement of firefly technique for effective optimization of multi-depot vehicle routing. *International Journal of Innovative Technology and Exploring Engineering*, *8*(11), 2346–2353. https://doi.org/10.35940/ijitee.J1132.0981119

Yi, W., & Kumar, A. (2007). Ant colony optimization for disaster relief operations. *Transportation Research Part E: Logistics and Transportation Review*, *43*(6),

660–672. https://doi.org/10.1016/j.tre.2006.05.004
Zhang, D., Dong, R., Si, Y.-W., Ye, F., & Cai, Q. (2018). A hybrid swarm algorithm based on ABC and AIS for 2L-HFCVRP. *Applied Soft Computing Journal*, *64*, 468–479. https://doi.org/10.1016/j.asoc.2017.12.012
Zhang, S. Z., & Lee, C. K. M. (2016a). An Improved Artificial Bee Colony Algorithm for the Capacitated Vehicle Routing Problem. *Proceedings - 2015 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2015*, 2124–2128. https://doi.org/10.1109/SMC.2015.371
Zhang, S. Z., & Lee, C. K. M. (2016b). An Improved Artificial Bee Colony Algorithm for the Capacitated Vehicle Routing Problem. *2015 IEEE International Conference on Systems, Man, and Cybernetics*, 2124–2128. https://doi.org/10.1109/SMC.2015.371

Anexos

Anexo A: Extracción de datos

https://drive.google.com/file/d/17yXyHG-Qn-Z5iETtJvf5eJYLdfo4IprJ/view?usp=sharing

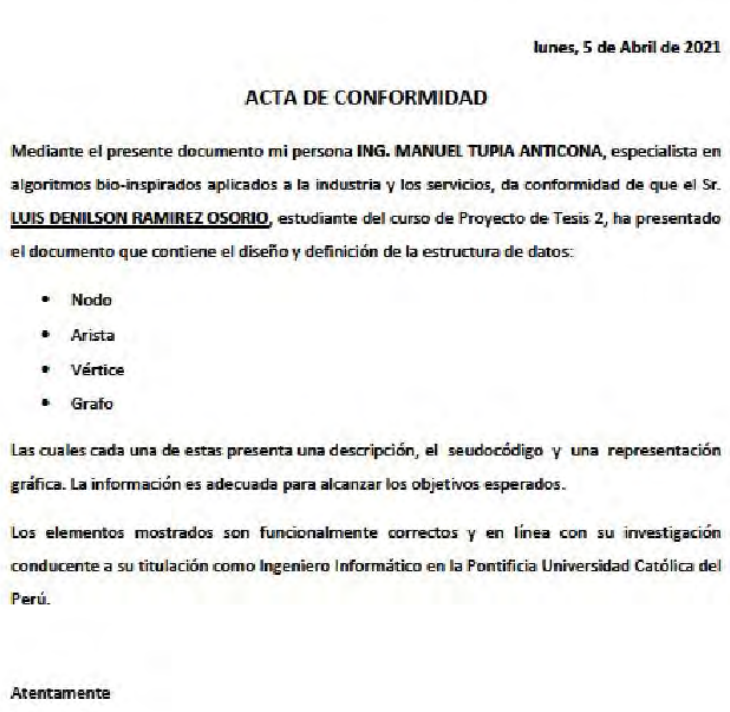
Anexo B: Repositorio del código

https://github.com/Dlumior/antphy

Anexo C: Validaciones del primer objetivo

Los documentos originales se encuentran en la misma carpeta con la siguiente nomenclatura “Acta de conformidad R<numero>.pdf”.

R1



lunes, 5 de Abril de 2021

ACTA DE CONFORMIDAD

Mediante el presente documento mi persona **ING. MANUEL TUPIA ANTICONA**, especialista en algoritmos bio-inspirados aplicados a la industria y los servicios, da conformidad de que el Sr. **LUIS DENILSON RAMIREZ OSORIO**, estudiante del curso de Proyecto de Tesis 2, ha presentado el documento que contiene el diseño y definición de la estructura de datos:

- Nodo
- Arista
- Vértice
- Grafo

Las cuales cada una de estas presenta una descripción, el seudocódigo y una representación gráfica. La información es adecuada para alcanzar los objetivos esperados.

Los elementos mostrados son funcionalmente correctos y en línea con su investigación conducente a su titulación como Ingeniero Informático en la Pontificia Universidad Católica del Perú.

Atentamente

Anexo D: Validaciones del segundo objetivo

Los documentos originales se encuentran en la misma carpeta con la siguiente nomenclatura “Acta de conformidad R<numero>.pdf”

R2

viernes, 16 de Abril de 2021

ACTA DE CONFORMIDAD

Mediante el presente documento el Sr. **ING. MANUEL TUPIA ANTICONA**, especialista en algoritmos bio-inspirados aplicados a la industria y los servicios, da conformidad de que el Sr. **LUIS DENILSON RAMIREZ OSORIO**, estudiante del curso de Proyecto de Tesis 2, ha presentado el documento que contiene el diseño de la función objetivo para trazar las rutas de distribución, las cuales cada una de estas presenta una descripción de los parámetros y restricciones a la cual están sujeto las funciones, así como el seudocódigo de cada una de ellas. La información es adecuada para alcanzar los objetivos esperados.

Los elementos mostrados son conceptualmente correctos y en línea con su investigación conducente a su titulación como Ingeniero Informático en la Pontificia Universidad Católica del Perú.

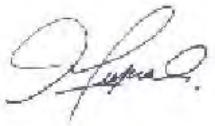
R3

martes, 20 de Abril de 2021

ACTA DE CONFORMIDAD

Mediante el presente documento el Sr. **ING. MANUEL TUPIA ANTICONA**, especialista en algoritmos bio-inspirados aplicados a la industria y los servicios, da conformidad de que el Sr. **LUIS DENILSON RAMIREZ OSORIO**, estudiante del curso de Proyecto de Tesis 2, ha presentado el documento que contiene el diseño de la función de probabilidad de transición, la función de evaporación de feromonas y la función de depósito de feromonas para trazar las rutas de distribución, las cuales cada una de estas presenta una descripción de dichas funciones, así como el seudocódigo de cada una de ellas. La información es adecuada para alcanzar los objetivos esperados.

Los elementos mostrados son conceptualmente correctos y en línea con su investigación conducente a su titulación como Ingeniero Informático en la Pontificia Universidad Católica del Perú.



MANUEL TUPIA

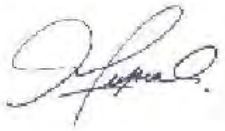
R4

martes, 20 de Abril de 2021

ACTA DE CONFORMIDAD

Mediante el presente documento el Sr. **ING. MANUEL TUPIA ANTICONA**, especialista en algoritmos bio-inspirados aplicados a la industria y los servicios, da conformidad de que el Sr. **LUIS DENILSON RAMIREZ OSORIO**, estudiante del curso de Proyecto de Tesis 2, ha presentado el documento que contiene las reglas de transición para el algoritmo colonia de hormigas, las cuales son descritas a partir de las restricciones del problema. La información es adecuada para alcanzar los objetivos esperados.

Los elementos mostrados son conceptualmente correctos y en línea con su investigación conducente a su titulación como Ingeniero Informático en la Pontificia Universidad Católica del Perú.



MANUEL TUPIA

R5

Para validar el correcto funcionamiento del algoritmo se realizaron diversas pruebas unitarias sobre los principales métodos de las clases que se emplean para la ejecución del algoritmo. Así, se presenta la serie de pruebas que se realizaron.

Pruebas sobre las estructuras

- **Pruebas unitarias sobre la clase arista**

  El código de cada una de estas pruebas se encuentra dentro del repositorio del proyecto: https://github.com/Dlumior/antphy/blob/main/src/test/java/org/lr/antphy/structures/edge/EdgeTest.java

  - **Caso de prueba "get_nValue()"**

    Este caso busca probar el método para obtener el valor de una heurística.

Entrada	Tipo de la heurística
Salida esperada	Valor de las heurísticas solicitadas
Salida obtenida	Se obtuvo el valore correcto de la heurística solicitada
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "set_nValue()"**

    Este caso busca probar el método para insertar valores a cada una de las heurísticas.

Entrada	Set de valores por cada heurística
Salida esperada	La correcta inserción de los valores
Salida obtenida	Se obtuvo una correcta inserción de los valores y estos corresponden con los valores de entrada
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "get_nValues()"**

    Este caso busca probar el método para obtener el set de valores de las heurísticas.

Entrada	Sin entrada
Salida esperada	Set de valores de las heurísticas
Salida obtenida	Se obtuvo correctamente el set de los valores de las heurísticas que pertenecen a la arista
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "set_nValues()"**

Este caso busca probar el método para insertar un set de valores para las heurísticas.

Entrada	Set de valores las heurísticas
Salida esperada	La correcta inserción del set de valores
Salida obtenida	Se obtuvo una correcta inserción del set de valores y este corresponde con el set de entrada
Resultado de la prueba unitaria	Correcto

- **Caso de prueba "get_tValue()"**

Este caso busca probar el método para obtener el valor de la feromona en la presente arista.

Entrada	Sin entrada
Salida esperada	Valor de la feromona
Salida obtenida	Se obtuvo correctamente el valor de la feromona correspondiente a la arista
Resultado de la prueba unitaria	Correcto

- **Caso de prueba "set_tValue()"**

Este caso busca probar el método para insertar un valor para la feromona en la presente arista.

Entrada	Valor de la feromona
Salida esperada	Correcta inserción del valor de entrada
Salida obtenida	Se obtuvo una correcta inserción del valor de la feromona y esta corresponde al valor de la entrada
Resultado de la prueba unitaria	Correcto

- **Pruebas unitarias sobre la clase vértice**

  El código de cada una de estas pruebas se encuentra dentro del repositorio del proyecto: https://github.com/Dlumior/antphy/blob/main/src/test/java/org/lr/antphy/structures/vertex/VertexTest.java

  - **Caso de prueba "get_Id()"**

    Este caso busca probar que el identificador generado sea único.

Dado	Dos vértices creados y distintos
Salida esperada	El identificador de uno sea distinto al del otro
Salida obtenida	Se obtuvo un identificador distinto en cada caso
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "get_posX()"**

    Este caso busca obtener la posición con respecto al eje X en un plano de dos dimensiones.

Entrada	Sin entrada
Salida esperada	Valor de la posición del vértice en el eje X
Salida obtenida	Se obtuvo correctamente el valor de la posición del eje X
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "set_posX()"**

    Este caso busca insertar la posición del vértice con respecto al eje X.

Entrada	Valor de la posición con respecto al eje X
Salida esperada	Correcta inserción del valor de entrada
Salida obtenida	Se obtuvo una correcta inserción del valor de la posición X y esta corresponde al valor de la entrada
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "get_posY()"**

    Este caso busca obtener la posición con respecto al eje Y en un plano de dos dimensiones.

Entrada	Sin entrada
Salida esperada	Valor de la posición del vértice en el eje Y
Salida obtenida	Se obtuvo correctamente el valor de la posición del eje Y

Resultado de la prueba unitaria	Correcto

- **Caso de prueba "set_posY()"**

  Este caso busca insertar la posición del vértice con respecto al eje Y.

Entrada	Valor de la posición con respecto al eje Y
Salida esperada	Correcta inserción del valor de entrada
Salida obtenida	Se obtuvo una correcta inserción del valor de la posición Y y esta corresponde al valor de la entrada
Resultado de la prueba unitaria	Correcto

- **Caso de prueba "get_demand()"**

  Este caso busca obtener la demanda del vértice.

Entrada	Sin entrada
Salida esperada	Valor de la demanda del vértice
Salida obtenida	Se obtuvo correctamente el valor de la demanda del vértice
Resultado de la prueba unitaria	Correcto

- **Caso de prueba "set_demand()"**

  Este caso busca insertar una nueva demanda para el vértice.

Entrada	Valor de la demanda
Salida esperada	Correcta inserción del valor de entrada
Salida obtenida	Se obtuvo una correcta inserción del valor demanda y esta corresponde al valor de la entrada.
Resultado de la prueba unitaria	Correcto

- **Caso de prueba "addEdge()"**

  Este caso busca probar el método para insertar una arista hacia un cierto vértice.

Entrada	Identificador de un vértice y una arista con toda su información completa
Salida esperada	Correcta inserción del valor de entrada
Salida obtenida	Se insertó correctamente la arista de entrada hacia el vértice indicad
Resultado de la prueba unitaria	Correcto

- **Caso de prueba "getEdge()"**

Este caso busca probar el método para obtener la arista hacia un vértice en concreto.

Entrada	Identificador de un vértice
Salida esperada	Arista entre el vértice de entrada y el presente vértice
Salida obtenida	Se obtuvo correctamente la arista que estaba apuntando al vértice de entrada
Resultado de la prueba unitaria	Correcto

- **Caso de prueba "getEdges()"**

Este caso busca probar el método para obtener el set de aristas correspondiente al presente vértice.

Entrada	Sin entrada
Salida esperada	Set de aristas del presente vértice
Salida obtenida	Se obtuvo correctamente el set de aristas del presente vértice
Resultado de la prueba unitaria	Correcto

- **Pruebas unitarias sobre la clase grafo**

  El código de cada una de estas pruebas se encuentra dentro del repositorio del proyecto: https://github.com/Dlumior/antphy/blob/main/src/test/java/org/lr/antphy/structures/graph/GraphTest.java

  - **Caso de prueba "addVertex()"**

    El caso busca probar el método para añadir vértices al grafo.

Entrada	Vértice
Salida esperada	Correcta inserción del vértice al grafo
Salida obtenida	Se insertó correctamente el vértice al grafo
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "getVertex()"**

    El caso busca probar el método para obtener un vértice a partir de su identificador.

Entrada	Identificador de un vértice
Salida esperada	Vértice solicitado
Salida obtenida	Se obtuvo correctamente el vértice solicitado
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "getVertices()"**

    El caso busca probar el método para obtener el set completo de vértices del grafo.

Entrada	Sin entrada
Salida esperada	Set de vértices
Salida obtenida	Se obtuvo correctamente los vértices del grafo
Resultado de la prueba unitaria	Correcto

  - **Caso de prueba "size()"**

    El caso busca probar el método para obtener el número de nodos que tiene el grafo.

Entrada	Sin entrada
Salida esperada	Número de vértices en el grafo
Salida obtenida	Se obtuvo correctamente el número de vértices del grafo

Resultado de la prueba unitaria	Correcto

Pruebas sobre la colonia de hormigas

- **Pruebas unitarias de los métodos principales sobre la clase hormiga**

  El código de cada una de estas pruebas se encuentra dentro del repositorio del proyecto: https://github.com/Dlumior/antphy/blob/main/src/test/java/org/lr/antphy/algorithms/aco/AntTest.java

  - **Caso de pruebas "getSolution()"**

    El caso busca probar el método para obtener la solución encontrada por la presente hormiga.

Entrada	Sin entrada
Salida esperada	Solución encontrada por la hormiga
Salida obtenida	Se obtuvo correctamente la solución con información sobre los costos totales, parciales y los caminos que este recorrió
Resultado de la prueba unitaria	Correcto

  - **Caso de pruebas "getTotalCostPerHeuristic()"**

    El caso busca probar el método para obtener los costos totales por heurística.

Entrada	Sin entrada
Salida esperada	Set de costos incurridos en la solución por heurística
Salida obtenida	Se obtuvo correctamente los costos de totales por heurísticas y este corresponde a las sumas parciales por camino de cada heurística
Resultado de la prueba unitaria	Correcto

  - **Caso de pruebas "getTotalCost()"**

    El caso busca probar el método para obtener el costo total.

Entrada	Sin entrada
Salida esperada	Costo total general
Salida obtenida	Se obtuvo correctamente el costo total general y esta corresponde con las sumas parciales del costo parcial general
Resultado de la prueba unitaria	Correcto

- **Caso de pruebas “setCapacity()”**

  El caso busca probar el método para insertar la capacidad que tiene una hormiga para transportar suministros.

Entrada	Valor de la capacidad
Salida esperada	Correcta inserción de la capacidad
Salida obtenida	Se insertó correctamente la capacidad y esta corresponde con el valor de entrada
Resultado de la prueba unitaria	Correcto

- **Caso de pruebas “getPathCost()”**

  El caso busca probar el método para obtener el costo parcial del camino indicado.

Entrada	Índice del camino requerido
Salida esperada	Costo del camino
Salida obtenida	Se obtuvo correctamente el costo del camino requerido
Resultado de la prueba unitaria	Correcto

- **Caso de pruebas “getPaths()”**

  El caso busca probar el método para obtener los caminos que encontró la hormiga como solución.

Entrada	Sin entrada
Salida esperada	Set de caminos
Salida obtenida	Se obtuvo correctamente los caminos encontrados por la hormiga
Resultado de la prueba unitaria	Correcto

- **Caso de pruebas “addPath()”**

  El caso busca probar el método para añadir un nuevo camino a la lista de caminos de la hormiga.

Entrada	Camino completado
Salida esperada	Correcta inserción del nuevo camino
Salida obtenida	Se insertó correctamente el camino y este corresponde con el camino de la entrada
Resultado de la prueba unitaria	Correcto

- **Caso de pruebas “getPath()”**

  El caso busca probar el método para obtener un camino en concreto de la lista de caminos generada por la hormiga.

Entrada	Índice del camino requerido
Salida esperada	Camino
Salida obtenida	Se obtuvo correctamente el camino requerido y el índice corresponde con el de la entrada.
Resultado de la prueba unitaria	Correcto

- **Prueba de caja blanca sobre la clase optimización de colonia de hormigas**

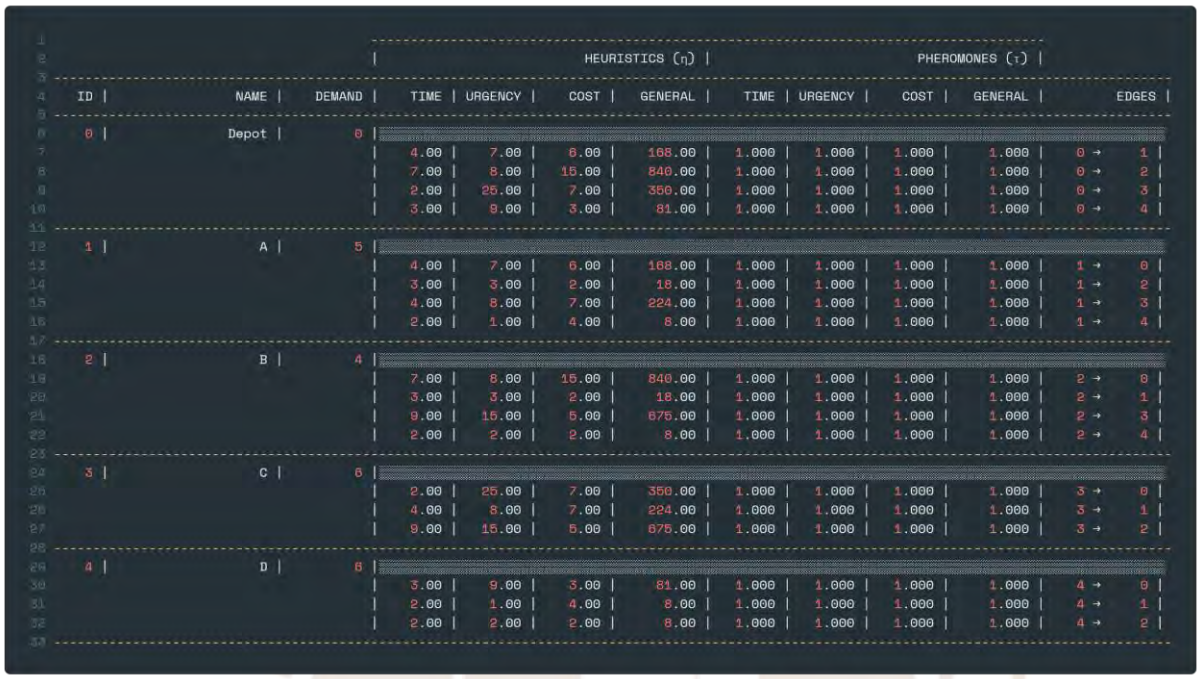
  La prueba realizada sobre el algoritmo de optimización de la colonia de hormigas, así como los parámetros empleados se encuentra en el repositorio del proyecto dentro del paquete Test.

  https://github.com/Dlumior/antphy/blob/main/src/test/java/org/lr/antphy/algorithms/aco/AntColonyOptimizationTest.java

Parámetros de la colonia	
Número de iteraciones	20
Número de hormigas	2
Capacidad de cada hormiga	10

Parámetros de los caminos	
Alfa	0.3
Beta	0.7
Tasa de evaporación	0.4

Grafo empleado por el algoritmo



			HEURISTICS (η)				PHEROMONES (τ)				
ID	NAME	DEMAND	TIME	URGENCY	COST	GENERAL	TIME	URGENCY	COST	GENERAL	EDGES
0	Depot	0									
			4.00	7.00	6.00	168.00	1.000	1.000	1.000	1.000	0 → 1
			7.00	8.00	15.00	840.00	1.000	1.000	1.000	1.000	0 → 2
			2.00	25.00	7.00	350.00	1.000	1.000	1.000	1.000	0 → 3
			3.00	9.00	3.00	81.00	1.000	1.000	1.000	1.000	0 → 4
1	A	5									
			4.00	7.00	6.00	168.00	1.000	1.000	1.000	1.000	1 → 0
			3.00	3.00	2.00	18.00	1.000	1.000	1.000	1.000	1 → 2
			4.00	8.00	7.00	224.00	1.000	1.000	1.000	1.000	1 → 3
			2.00	1.00	4.00	8.00	1.000	1.000	1.000	1.000	1 → 4
2	B	4									
			7.00	8.00	15.00	840.00	1.000	1.000	1.000	1.000	2 → 0
			3.00	3.00	2.00	18.00	1.000	1.000	1.000	1.000	2 → 1
			9.00	15.00	5.00	675.00	1.000	1.000	1.000	1.000	2 → 3
			2.00	2.00	2.00	8.00	1.000	1.000	1.000	1.000	2 → 4
3	C	6									
			2.00	25.00	7.00	350.00	1.000	1.000	1.000	1.000	3 → 0
			4.00	8.00	7.00	224.00	1.000	1.000	1.000	1.000	3 → 1
			9.00	15.00	5.00	675.00	1.000	1.000	1.000	1.000	3 → 2
4	D	8									
			3.00	9.00	3.00	81.00	1.000	1.000	1.000	1.000	4 → 0
			2.00	1.00	4.00	8.00	1.000	1.000	1.000	1.000	4 → 1
			2.00	2.00	2.00	8.00	1.000	1.000	1.000	1.000	4 → 2

ID	Identificador del vértice.
Name	Nombre del vértice.
Demand	Demanda en unidades del vértice.
Heuristics	Valores de las heurísticas en las aristas (Tiempo, Urgencia, Costo) El valor de la heurística General se refiere a la multiplicación de las tres heurísticas.
Pheromone	Valores iniciales de las feromonas en las aristas. Un rastro de feromona por cada heurística y uno general.
Edges	Aristas representadas por como las conexiones entre los identificadores de los vértices.

Mejores soluciones obtenidas durante la ejecución

```
Iteration:     0
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:    89.0      0 ->    4 ->    2 ->    0 -> END
🚚 Path  1 - Cost:   350.0      0 ->    3 ->    0 -> END
🚚 Path  2 - Cost:   168.0      0 ->    1 ->    0 -> END
✅ Total cost: 607.0

Iteration:     1
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:  1025.0      0 ->    3 ->    2 ->    0 -> END
🚚 Path  1 - Cost:   168.0      0 ->    1 ->    0 -> END
🚚 Path  2 - Cost:    81.0      0 ->    4 ->    0 -> END
✅ Total cost: 1274.0

...
...

Iteration:     6
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:   186.0      0 ->    1 ->    2 ->    0 -> END
🚚 Path  1 - Cost:   350.0      0 ->    3 ->    0 -> END
🚚 Path  2 - Cost:    81.0      0 ->    4 ->    0 -> END
✅ Total cost: 617.0

...
...

Iteration:    13
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:    89.0      0 ->    4 ->    2 ->    0 -> END
🚚 Path  1 - Cost:   350.0      0 ->    3 ->    0 -> END
🚚 Path  2 - Cost:   168.0      0 ->    1 ->    0 -> END
✅ Total cost: 607.0

Iteration:    14
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:   186.0      0 ->    1 ->    2 ->    0 -> END
🚚 Path  1 - Cost:   350.0      0 ->    3 ->    0 -> END
🚚 Path  2 - Cost:    81.0      0 ->    4 ->    0 -> END
✅ Total cost: 617.0

...
...

Iteration:    28
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:    89.0      0 ->    4 ->    2 ->    0 -> END
🚚 Path  1 - Cost:   350.0      0 ->    3 ->    0 -> END
🚚 Path  2 - Cost:   168.0      0 ->    1 ->    0 -> END
✅ Total cost: 607.0

Iteration:    29
Best Colony Ant
🏁 Solution
🚚 Path  0 - Cost:    89.0      0 ->    4 ->    2 ->    0 -> END
🚚 Path  1 - Cost:   168.0      0 ->    1 ->    0 -> END
🚚 Path  2 - Cost:   350.0      0 ->    3 ->    0 -> END
✅ Total cost: 607.0
```

Grafo al finalizar la ejecución

			HEURISTICS (η)				PHEROMONES (τ)				
ID	NAME	DEMAND	TIME	URGENCY	COST	GENERAL	TIME	URGENCY	COST	GENERAL	EDGES
0	Depot	0									
			4.00	7.00	6.00	168.00	0.618	0.235	0.421	0.009	0 -> 1
			7.00	8.00	15.00	840.00	0.676	0.270	0.405	0.008	0 -> 2
			2.00	25.00	7.00	350.00	0.821	0.263	0.496	0.011	0 -> 3
			3.00	9.00	3.00	81.00	0.824	0.300	0.544	0.011	0 -> 4
1	A	5									
			4.00	7.00	6.00	168.00	0.618	0.235	0.421	0.009	1 -> 0
			3.00	3.00	2.00	18.00	0.398	0.167	0.231	0.005	1 -> 2
			4.00	8.00	7.00	224.00	0.000	0.000	0.000	0.000	1 -> 3
			2.00	1.00	4.00	8.00	0.000	0.000	0.000	0.000	1 -> 4
2	B	4									
			7.00	8.00	15.00	840.00	0.676	0.270	0.405	0.008	2 -> 0
			3.00	3.00	2.00	18.00	0.398	0.167	0.231	0.005	2 -> 1
			9.00	15.00	5.00	675.00	0.278	0.143	0.228	0.004	2 -> 3
			2.00	2.00	2.00	8.00	0.304	0.147	0.189	0.004	2 -> 4
3	C	6									
			2.00	25.00	7.00	350.00	0.821	0.263	0.496	0.011	3 -> 0
			4.00	8.00	7.00	224.00	0.000	0.000	0.000	0.000	3 -> 1
			9.00	15.00	5.00	675.00	0.278	0.143	0.228	0.004	3 -> 2
4	D	6									
			3.00	9.00	3.00	81.00	0.824	0.300	0.544	0.011	4 -> 0
			2.00	1.00	4.00	8.00	0.000	0.000	0.000	0.000	4 -> 1
			2.00	2.00	2.00	8.00	0.304	0.147	0.189	0.004	4 -> 2

Como se pude observar al finalizar la ejecución del algoritmo, los valores de las feromonas en las aristas se vieron afectadas por las sucesivas evaporaciones y depósitos de las hormigas. De esta manera, se puede ver que el nivel de la feromona en arista del vértice 1 al 3 es de 0. Esto se debe a que el vértice 1 tiene una demanda de 5, mientras que la demanda del vértice 3 es de 6 por lo que el total de la demanda para realizar dicha transición es de 11 unidades. No obstante, la capacidad máxima de la hormiga es de sólo 10 unidades por lo que realizar dicha transición es prácticamente imposible y por ello al no haber transiciones la feromona inicial se evapora por completo.

Frontera de Pareto

```
PARETO FRONT
-------------------------------------------------
SOLUTION  0 |   TIME |URGENCY |   COST |GENERAL |
-------------------------------------------------
Path      0 |    2.0 |   25.0 |    7.0 |  350.0 |    0 ->    3 ->    0 -> END
Path      1 |    9.0 |   10.0 |   17.0 |  848.0 |    0 ->    2 ->    4 ->    0 -> END
Path      2 |    4.0 |    7.0 |    6.0 |  168.0 |    0 ->    1 ->    0 -> END
      TOTAL |   15.0 |   42.0 |   30.0 | 1366.0 |

-------------------------------------------------
SOLUTION  1 |   TIME |URGENCY |   COST |GENERAL |
-------------------------------------------------
Path      0 |    5.0 |   11.0 |    5.0 |   89.0 |    0 ->    4 ->    2 ->    0 -> END
Path      1 |    2.0 |   25.0 |    7.0 |  350.0 |    0 ->    3 ->    0 -> END
Path      2 |    4.0 |    7.0 |    6.0 |  168.0 |    0 ->    1 ->    0 -> END
      TOTAL |   11.0 |   43.0 |   18.0 |  607.0 |

-------------------------------------------------
SOLUTION  2 |   TIME |URGENCY |   COST |GENERAL |
-------------------------------------------------
Path      0 |   16.0 |   23.0 |   20.0 | 1515.0 |    0 ->    2 ->    3 ->    0 -> END
Path      1 |    3.0 |    9.0 |    3.0 |   81.0 |    0 ->    4 ->    0 -> END
Path      2 |    4.0 |    7.0 |    6.0 |  168.0 |    0 ->    1 ->    0 -> END
      TOTAL |   23.0 |   39.0 |   29.0 | 1764.0 |



BUILD SUCCESSFUL in 333ms
2 actionable tasks: 1 executed, 1 up-to-date
12:24:25 AM: Task execution finished 'AntphyApp.main()'.
```

Al terminar la ejecución del algoritmo se obtiene una frontera de Pareto con las posibles soluciones el problema. Esta contiene soluciones no dominadas, es decir, soluciones que no son mejore ni peores que los otros en tal sentido que para que una solución sea mejor que otra debe ser la mejor en todos los objetivos. En el resultado mostrado, se puede observar tres soluciones. En este la primera solución es mejor que la segunda en cuanto urgencia se refiere y mejor que la tercera en tiempo. La segunda solución es mejor que la primera en tiempo y mejor que la tercera en costo. La tercera solución es mejor que la primera y la segunda en urgencia. Sin embargo, el que mejor balance posee en cuanto a las tres heurísticas es la segunda solución. Esto se ve reflejado en el costo total - general pues es la menor de las tres.

Acta de conformidad

lunes, 3 de Mayo de 2021

ACTA DE CONFORMIDAD

Mediante el presente documento el Sr. **ING. MANUEL TUPIA ANTICONA,** especialista en algoritmos bio-inspirados aplicados a la industria y los servicios, da conformidad de que el Sr. **LUIS DENILSON RAMIREZ OSORIO,** estudiante del curso de Proyecto de Tesis 2, ha presentado el documento que contiene el diseño del algoritmo colonia de hormigas, la cual presenta una descripción detallada, así como el seudocódigo de este. Asimismo, presenta la implementación de este diseño junto con las pruebas realizadas sobre dicha implementación. La información es adecuada para alcanzar los objetivos esperados.

Los elementos mostrados son conceptualmente correctos y en línea con su investigación conducente a su titulación como Ingeniero Informático en la Pontificia Universidad Católica del Perú.

R6

Para la validación del correcto funcionamiento de la interfaz gráfica, se realizó una prueba de integración entre la interfaz y el algoritmo. Esta consiste en dos etapas, la primera de ella es la del envío de información desde la interfaz al algoritmo y la segunda en la de enviar una respuesta desde el algoritmo a la interfaz.

Dado que la interfaz gráfica y el algoritmo están construidos sobre tecnologías distintas, la comunicación entre estos se realiza mediante el uso de sockets.

Repositorio con el código de la interfaz

https://github.com/Dlumior/antphy-client

Datos empleados para la prueba

En esta prueba se empleó un grafo de 150 nodos, la cual se encuentra almacenado en un archivo con formato JSON. Este archivo contiene dos objetos principales, los nodos y las conexiones entre estos. Este archivo va a ser cargado por la interfaz, para luego ser enviado por al algoritmo y que este pueda construir el grafo con el que se va a trabajar.

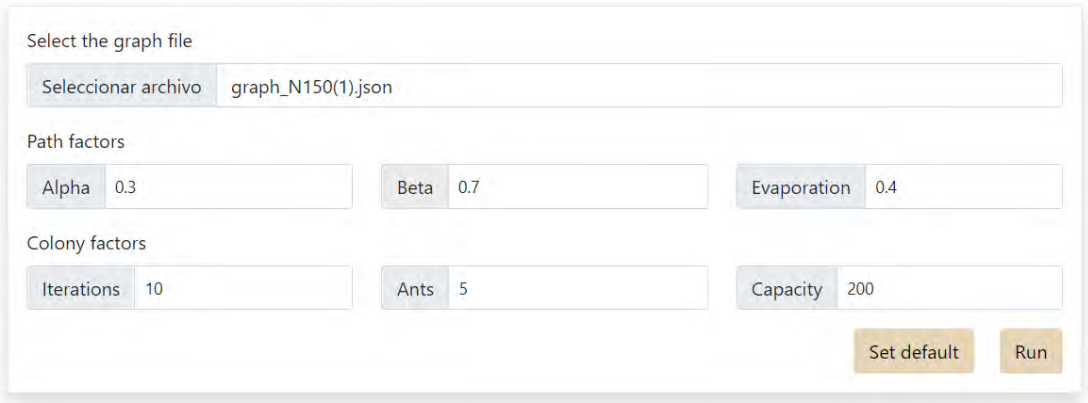
```
"nodes": [
  {
      "id": 0,
      "posX": 0,
      "posY": 0,
      "demand": 0,
      "urgency": 1,
      "name": "DEPOT"
  },
  {
      "id": 1,
      "posX": 16,
      "posY": 13,
      "demand": 37,
      "urgency": 2,
      "name": "CIUDAD 1"
  },
  ...
  ...
  {
      "id": 150,
      "posX": 8,
      "posY": 15,
      "demand": 33,
      "urgency": 3,
      "name": "CIUDAD 150"
  },
]
```

```
"edges": [
  {
    "source": 0,
    "to": [
      {
          "id": 78,
          "time": 13
      },
      ...
      ...
      {
          "id": 82,
          "time": 10
      }
    ]
  },
  ...
  ...
  {
    "source": 150,
    "to": [
      {
        "id": 111,
        "time": 12
      },
      ...
      ...
      {
          "id": 35,
          "time": 18
      }
    ]
  }
]
```

Como se observa, cada nodo tiene una posición, una cierta demanda, un cierto nivel de urgencia y el nombre del nodo. Por otro lado, las aristas tienen el identificador del nodo origen y una lista con los identificadores de los nodos conectados a este. Con toda esta información, el algoritmo podrá armar el grafo en el cual el algoritmo va a ejecutarse.

Primera etapa de la prueba

En esta primera etapa se va a cargar el grafo y completar los factores correspondientes para ser enviados al algoritmo.



Luego al presionar el botón de “correr” se envía toda esta información al algoritmo.

```
Waiting for connection
New connection /127.0.0.1
Received: {"file":"D:\\Descargas\\graph_N150(1).json","alpha":0.3,"beta":0.7,"evaporation":0.4,"iterations":"10","ants":
5,"capacity":"200"}
```

Así, efectivamente se observa que la información que recibe el algoritmo concuerda con lo enviado. Luego de obtener la información, el algoritmo comienza su ejecución.

Segunda etapa de la prueba

La segunda etapa de la prueba comienza cuando el algoritmo terminó su ejecución y envía las soluciones que este ha encontrado.

```
PARETO FRONT
-----------------------------------------------
SOLUTION  0 |   TIME |URGENCY |   COST |GENERAL |
-----------------------------------------------
      TOTAL | 1889.0 |   37.6 |35707.0 |116409.0 |

-----------------------------------------------
SOLUTION  1 |   TIME |URGENCY |   COST |GENERAL |
-----------------------------------------------
      TOTAL | 1996.0 |   37.6 |36481.9 |123693.3 |

-----------------------------------------------
SOLUTION  2 |   TIME |URGENCY |   COST |GENERAL |
-----------------------------------------------
      TOTAL | 1812.0 |   37.6 |34676.4 |108693.2 |

-----------------------------------------------
SOLUTION  3 |   TIME |URGENCY |   COST |GENERAL |
-----------------------------------------------
      TOTAL | 2095.0 |   37.6 |36219.3 |122676.2 |

-----------------------------------------------
SOLUTION  4 |   TIME |URGENCY |   COST |GENERAL |
-----------------------------------------------
      TOTAL | 1900.0 |   37.6 |34626.9 |110502.7 |
```

Estas soluciones son las que encontró el algoritmo y se muestran en la consola de ejecución. La siguiente figura muestra la información que llegó a la interfaz, como se puede observar las soluciones en la tabla son las mismas que se mostró en consola, por lo que se puede concluir que la transmisión de información fue realizada correctamente.

Solutions

#	Total time	Total urgency	Total cost	General cost
1	1889	37.61468253968255	35706.993985856934	116408.97165222804
2	1996	37.61468253968254	36481.860627923095	123693.30451649133
3	1812	37.61468253968256	34676.41638312346	108693.17241090348
4	2095	37.61468253968254	36219.272546382446	122676.18349384762
5	1900	37.61468253968256	34626.893173879296	110502.66240833479

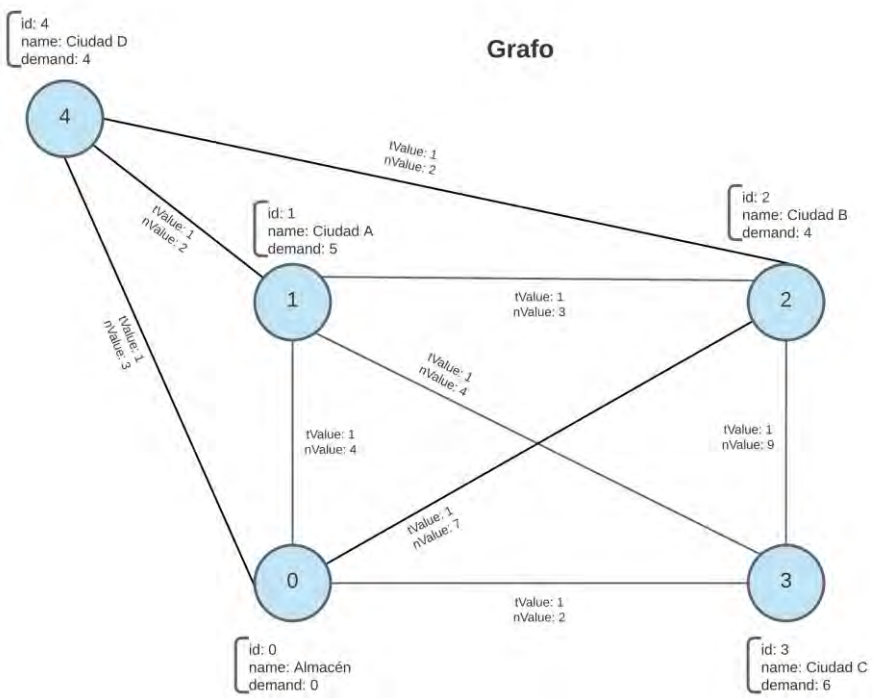
Anexo E: Validaciones del tercer objetivo

R7

Repositorio con el código de la implementación del algoritmo BFS

https://github.com/Dlumior/antphy/blob/main/src/main/java/org/lr/antphy/algorithms/bfs/BFS.java

Prueba de caja blanca sobre el algoritmo BFS

Grafo sobre el cual se realizó la prueba

El grafo sobre el cual se realizó la prueba es el mismo sobre el cual se realizó la prueba de caja blanca para el algoritmo colonia de hormigas.

Dado de que se trata de un algoritmo voraz y teniendo 10 unidades como capacidad máxima de los vehículos el resultado esperado es que visite cada nodo y regresar al almacén dado que de acuerdo con el algoritmo planteado al tratar de visitar los otros nodos a los que se pueden llegar desde el nodo actual aparentemente es más costoso

que regresar al almacén e ir directamente al nodo. Así en la primera iteración la frontera solo contiene el almacén y a partir de este se expande hacia cada uno de los vértices adyacentes para luego seleccionar el de menor costo y en este caso el camino [0,4] es el que hasta el momento tiene menor costo. Después de ello, verifica si puede ir hacia los vértices adyacentes, pero como su capacidad disponible es superada por la demanda de los vértices adyacentes, este retorna al almacén y se limpia la frontera solo quedándose con el almacén como punto de partida.

```
Frontier
[
    {
     path=[0],
     cost=0.0,
     capacity=10.0
    }
]
Frontier
[
    {
     path=[0, 4],
     cost=81.0,
     capacity=4.0
    }
,
    {
     path=[0, 1],
     cost=168.0,
     capacity=5.0
    }
,
    {
     path=[0, 3],
     cost=350.0,
     capacity=4.0
    }
,
    {
     path=[0, 2],
     cost=840.0,
     capacity=6.0
    }
]
Paths founded
[[0, 4, 0]]
```

De la misma manera a la anterior iteración, se parte del almacén y se expanden los caminos con los vértices adyacentes. Se puede observar que ya no se encuentra el camino [0,4] pues el vértice con identificador 4 ya fue visitado y por ende ya no se toma en cuenta. De esta manera solo se tiene tres caminos posibles y se selecciona el que menor costo tenga acumulado, el cual es [0,1]. Este camino trata de expandirse,

pero como la demanda de los vértices adyacentes al vértice 1 superan la capacidad actual del camino este retorna al almacén y se limpia la frontera.

```
Frontier
[
    {
     path=[0],
     cost=0.0,
     capacity=10.0
    }
]
Frontier
[
    {
     path=[0, 1],
     cost=168.0,
     capacity=5.0
    }
,
    {
     path=[0, 2],
     cost=840.0,
     capacity=6.0
    }
,
    {
     path=[0, 3],
     cost=350.0,
     capacity=4.0
    }
]
Paths founded
[[0, 4, 0], [0, 1, 0]]
```

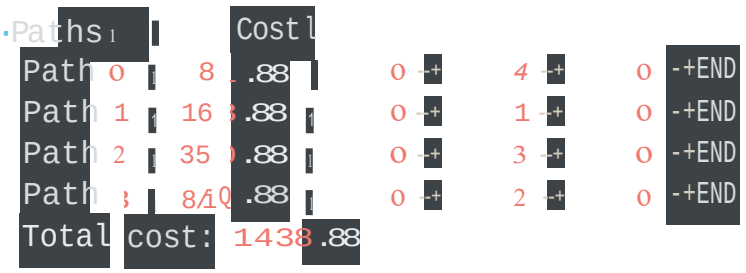
El proceso se repite empezando desde el almacén. Se expanden los caminos sin considerar los vértices que ya fueron visitados y se obtienen dos posibles caminos. De estos se selecciona el menor, el cual es [0,3] y se trata de expandir. Al no poder expandirse pues los vértices adyacentes a este ya fueron visitados o la demanda de estos supera la capacidad actual del camino, se procede a regresar al almacén.

```
Frontier
[
    {
     path=[0],
     cost=0.0,
     capacity=10.0
    }
]
Frontier
[
    {
     path=[0, 3],
     cost=350.0,
     capacity=4.0
    }
,
    {
     path=[0, 2],
     cost=840.0,
     capacity=6.0
    }
]
Paths founded
[[0, 4, 0], [0, 1, 0], [0, 3, 0]]
```

De la misma manera que en las iteraciones anteriores, se procede a expandir los posibles caminos desde el almacén, pero como ya los otros vértices fueron visitados, sólo queda un posible camino. Este camino se retira de la frontera y se realiza el reparto.

```
Frontier
[
    {
     path=[0],
     cost=0.0,
     capacity=10.0
    }
]
Frontier
[
    {
     path=[0, 2],
     cost=840.0,
     capacity=6.0
    }
]
Paths founded
[[0, 4, 0], [0, 1, 0], [0, 3, 0], [0, 2, 0]]
```

Así finalmente se tiene las siguientes rutas para el abastecimiento de suministros y como era de esperarse al ser codicioso y dirigirse solo por los caminos más cortos termina haciendo más rutas de las necesarias y el costo no es el mínimo posible.


Paths
Cost
Path 0 8 .88 0 4 0 -+END
Path 1 16 .88 0 1 0 -+END
Path 2 35 .88 0 3 0 -+END
Path 8 8 .88 0 2 0 -+END
Total cost: 1438.88

ET LUX IN TENEBRIS LUCET
MCMXVII

lunes, 31 de Mayo de 2021

ACTA DE CONFORMIDAD

Mediante el presente documento el SR. ING. MANUEL TUPIA ANTICONA, especialista en algoritmos bioinspirados aplicados a la industria y los servicios, da conformidad de que el Sr. WILSON DENILSON RAMIREZ OSOOJO, i:111t11 dl;Irun:o d11 PJ'O'iKU> di!ü!Si!:2, il;;pt<Bó1111Ud11 el documento que contiene el diseño del algoritmo >tOraz pimero el mejD'Il, el cual presenta una descripción detallada, así como el seudocódigo de este. También presenta la implementación de este diseño junto con las pruebas realizadas sobre dicha implementación. La información es adecuada para alcanzar los objetivos esperados.

los elementos r:na.rtr;acbs san conceptualmente correctos y en línea con su investigación conducente a su titulación como Ingeniero Informático en la Pontificia Universidad Católica del Perú.

R8

lunes, 31 de Mayo de 2021

ACTA DE CONFORMIDAD

Mediante el presente documento el Sr. ING. MANUEL TUPIA ANTICONA, especialista en algoritmos bio-inspirados aplicados a la industria y los servicios, da conformidad de que el Sr. LUIS DENILSON RAMIREZ OSORIO, estudiante del curso de Proyecto de Tesis 2, ha presentado el documento que contiene la calibración de los parámetros del algoritmo colonia de hormigas, la cual presenta una descripción detallada del proceso de calibración, así como el resultado al que se llegó. La información es adecuada para alcanzar los objetivos esperados.

Los elementos mostrados son conceptualmente correctos y en línea con su investigación conducente a su titulación como Ingeniero Informático en la Pontificia Universidad Católica del Perú.

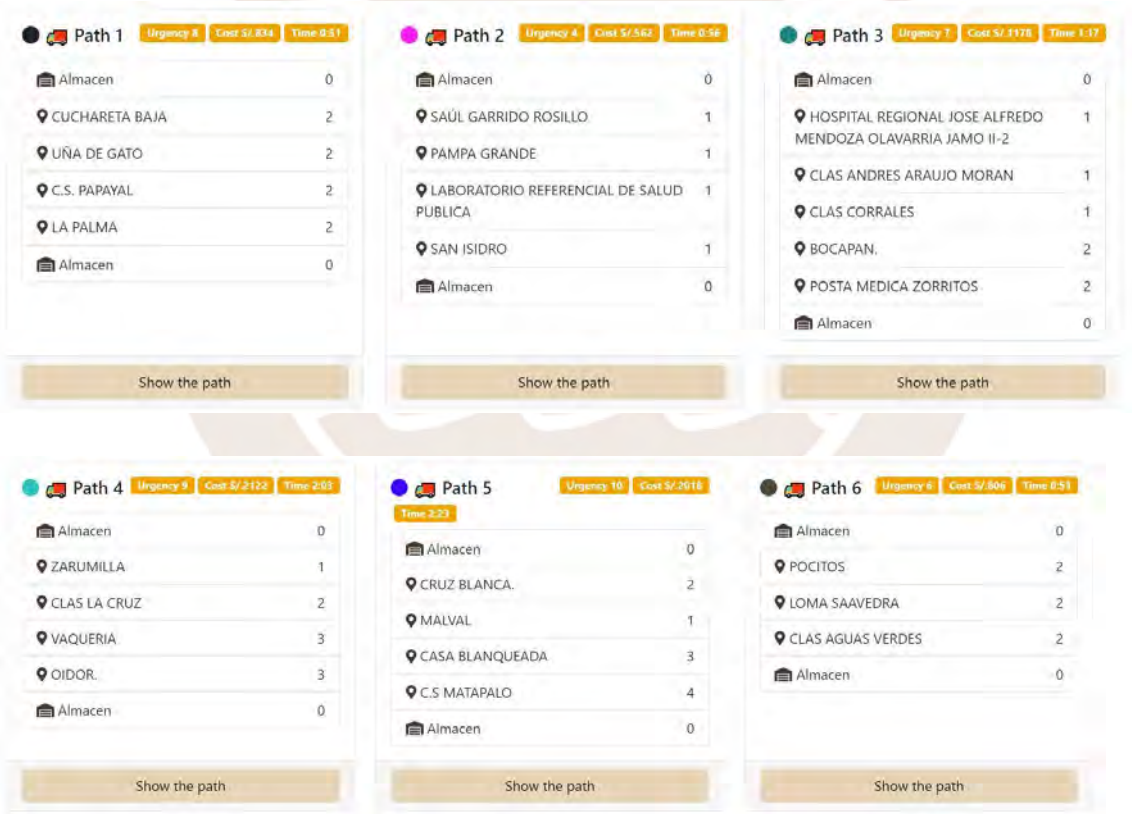
R10

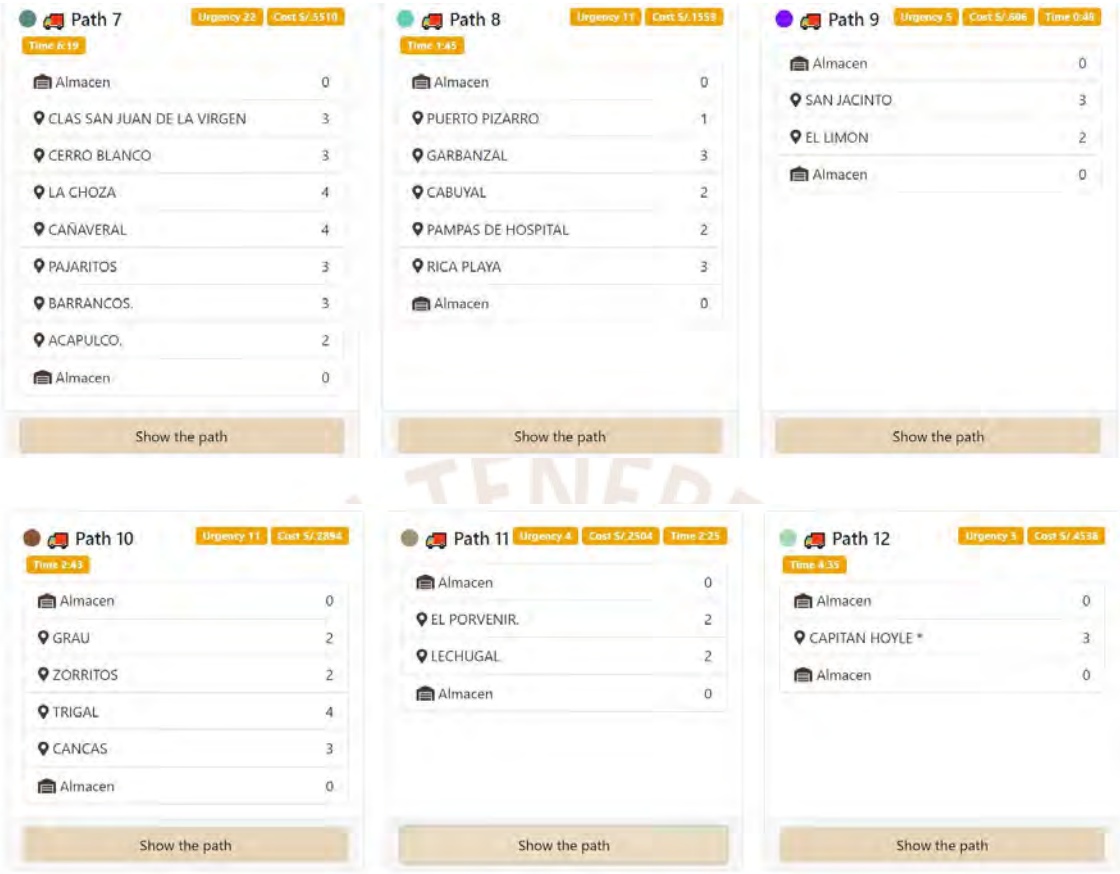
Nodos del grafo de tumbes

Latitud	*Longitud*	*Demanda*	*Nivel De Alerta*	*Nombre*
-3.555602	-80.442932	0	0	Almacén
-3.582179	-80.420268	19	1	Pampa Grande
-3.557005	-80.427507	19	1	Clas Andrés Araujo Moran
-3.503105	-80.391828	26	1	Puerto Pizarro
-3.624189	-80.466641	27	1	Malval
-3.500925	-80.272029	29	1	Zarumilla
-3.562703	-80.455727	30	1	Saúl Garrido Rosillo
-3.613161	-80.510743	33	1	San Isidro
-3.562082	-80.455544	37	1	Hospital Regional Jose Alfredo Mendoza Olavarria Jamo Ii-2
-3.566995	-80.424779	43	1	Laboratorio Referencial De Salud Publica
-3.600648	-80.481883	44	1	Clas Corrales
-3.572338	-80.235397	12	2	C.S. Papayal
-3.680501	-80.679045	14	2	Posta Medica Zorritos
-3.613774	-80.197088	16	2	Lechugal
-3.697524	-80.437166	17	2	Pampas De Hospital
-3.666355	-80.630233	18	2	Grau
-3.727840	-80.441682	25	2	El Limon
-3.708123	-80.721208	27	2	Bocapan.
-3.738432	-80.767444	27	2	Acapulco.
-3.507520	-80.241523	32	2	Pocitos
-3.537190	-80.232342	38	2	Uña De Gato
-3.668901	-80.419962	40	2	Cruz Blanca.
-3.715683	-80.436762	41	2	Cabuyal
-3.505457	-80.226645	42	2	Loma Saavedra
-3.562527	-80.215102	45	2	La Palma
-3.638209	-80.593227	46	2	Clas La Cruz
-3.549667	-80.226758	48	2	El Porvenir.
-3.485522	-80.260603	50	2	Clas Aguas Verdes
-3.680668	-80.679233	50	2	Zorritos
-3.524851	-80.238805	50	2	Cuchareta Baja
-4.124059	-80.971097	12	3	Barrancos.
-3.655334	-80.426616	15	3	Cerro Blanco
-3.626303	-80.434290	16	3	Clas San Juan De La Virgen
-4.082680	-80.497110	23	3	Capitan Hoyle *
-3.706100	-80.452283	23	3	Vaqueria

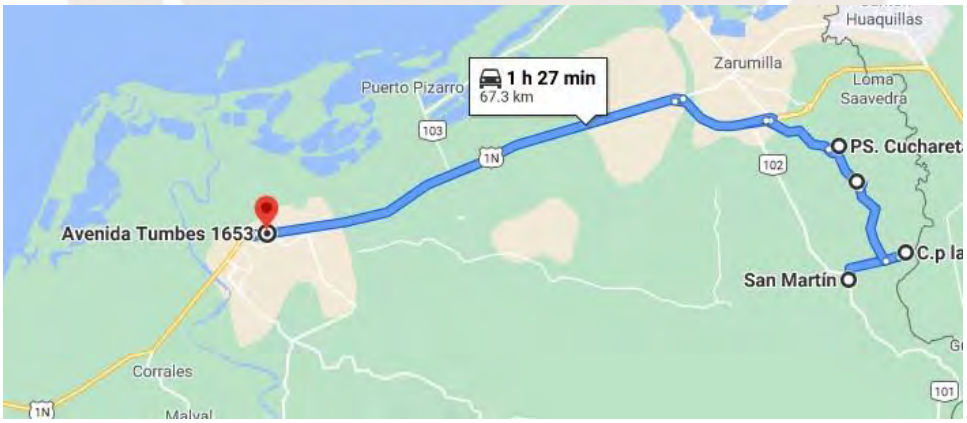
-3.611781	-80.433688	25	3	Garbanzal
-3.642150	-80.448007	35	3	San Jacinto
-4.051443	-80.888243	36	3	Pajaritos
-3.803902	-80.500540	38	3	Rica Playa
-3.722230	-80.459027	42	3	Oidor.
-3.952021	-80.947255	45	3	Cancas
-3.734617	-80.453705	49	3	Casa Blanqueada
-3.938882	-80.651199	13	4	Cañaveral
-3.682425	-80.199828	14	4	C.S Matapalo
-4.017679	-80.664579	23	4	La Choza
-3.840938	-80.674029	36	4	Trigal

Mejor solución encontrada

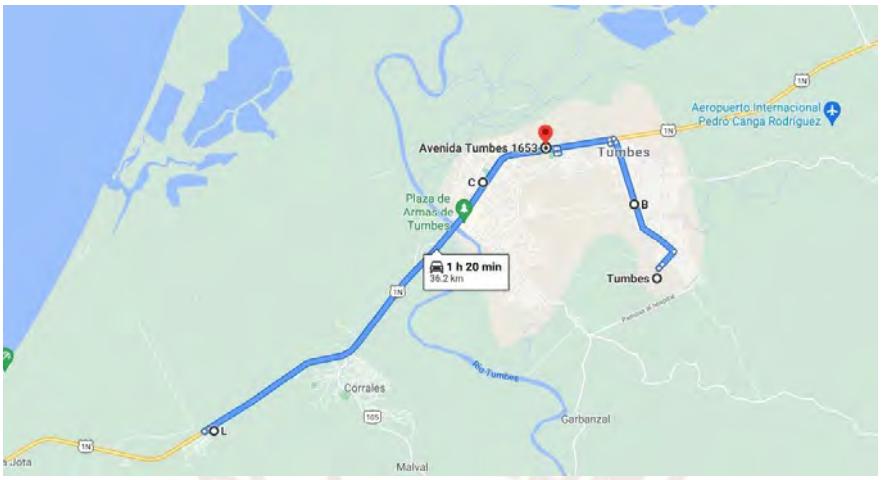



Path 7
Urgency 22
Cost S/.5510
Time 6:19
Almacen 0
CLAS SAN JUAN DE LA VIRGEN 3
CERRO BLANCO 3
LA CHOZA 4
CAÑAVERAL 4
PAJARITOS 3
BARRANCOS. 3
ACAPULCO. 2
Almacen 0
Show the path
Path 8
Urgency 11
Cost S/.1553
Time 1:45
Almacen 0
PUERTO PIZARRO 1
GARBANZAL 3
CABUYAL 2
PAMPAS DE HOSPITAL 2
RICA PLAYA 3
Almacen 0
Show the path
Path 9
Urgency 5
Cost S/.606
Time 0:46
Almacen 0
SAN JACINTO 3
EL LIMON 2
Almacen 0
Show the path
Path 10
Urgency 11
Cost S/.2894
Time 2:43
Almacen 0
GRAU 2
ZORRITOS 2
TRIGAL 4
CANCAS 3
Almacen 0
Show the path
Path 11
Urgency 4
Cost S/.2504
Time 2:25
Almacen 0
EL PORVENIR. 2
LECHUGAL 2
Almacen 0
Show the path
Path 12
Urgency 3
Cost S/.4538
Time 4:35
Almacen 0
CAPITAN HOYLE * 3
Almacen 0
Show the path

Ruta 1

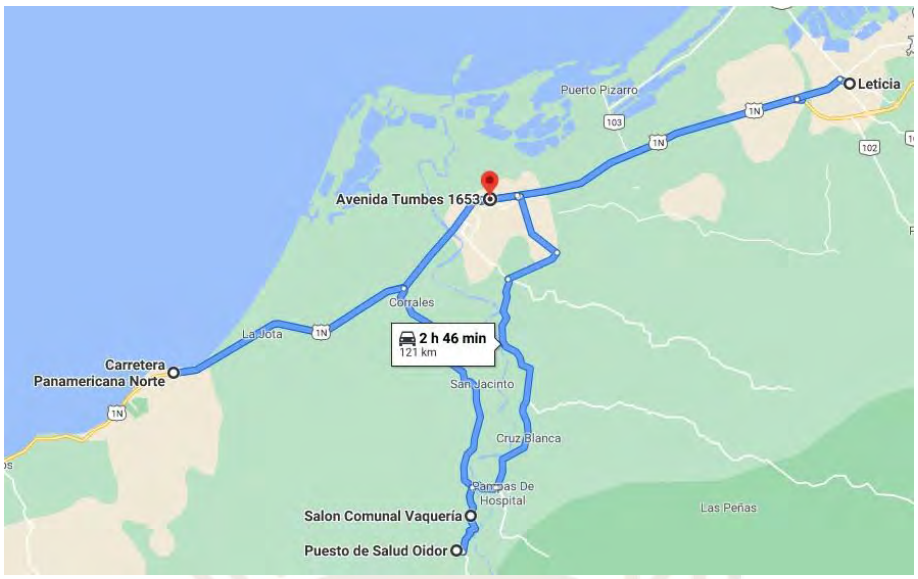

1 h 27 min
67.3 km
Puerto Pizarro
Zarumilla
Huaquillas
Loma Saavedra
PS. Cucharet
Avenida Tumbes 1653
San Martín
C.p la
Corrales
Malval
103
1N
102
101

Ruta 2

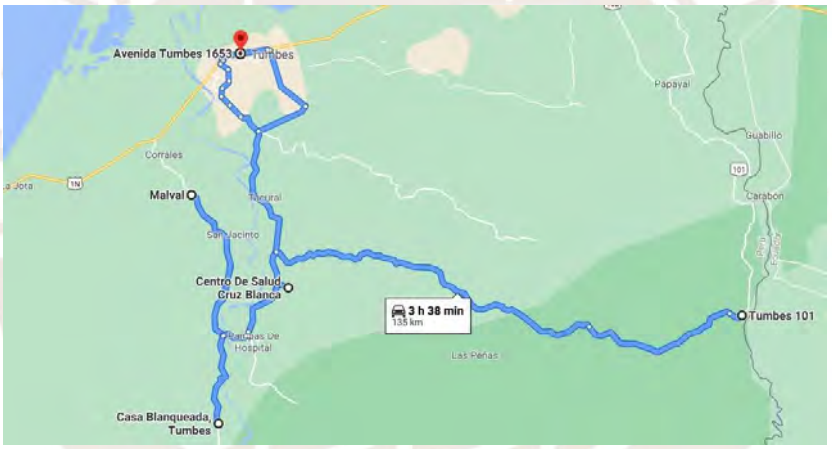


Ruta 3

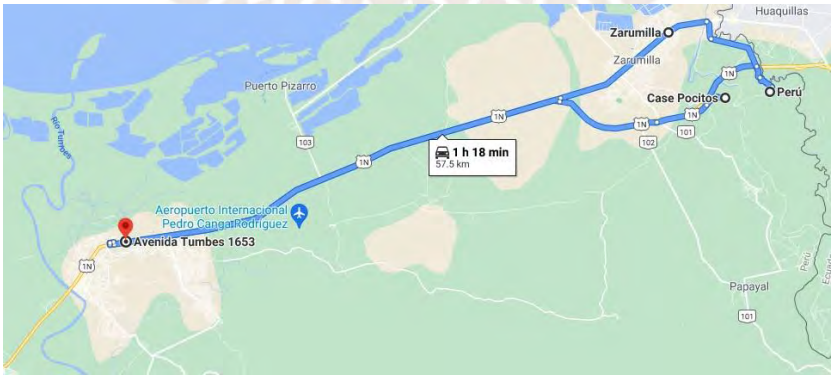
Ruta 4



Ruta 5



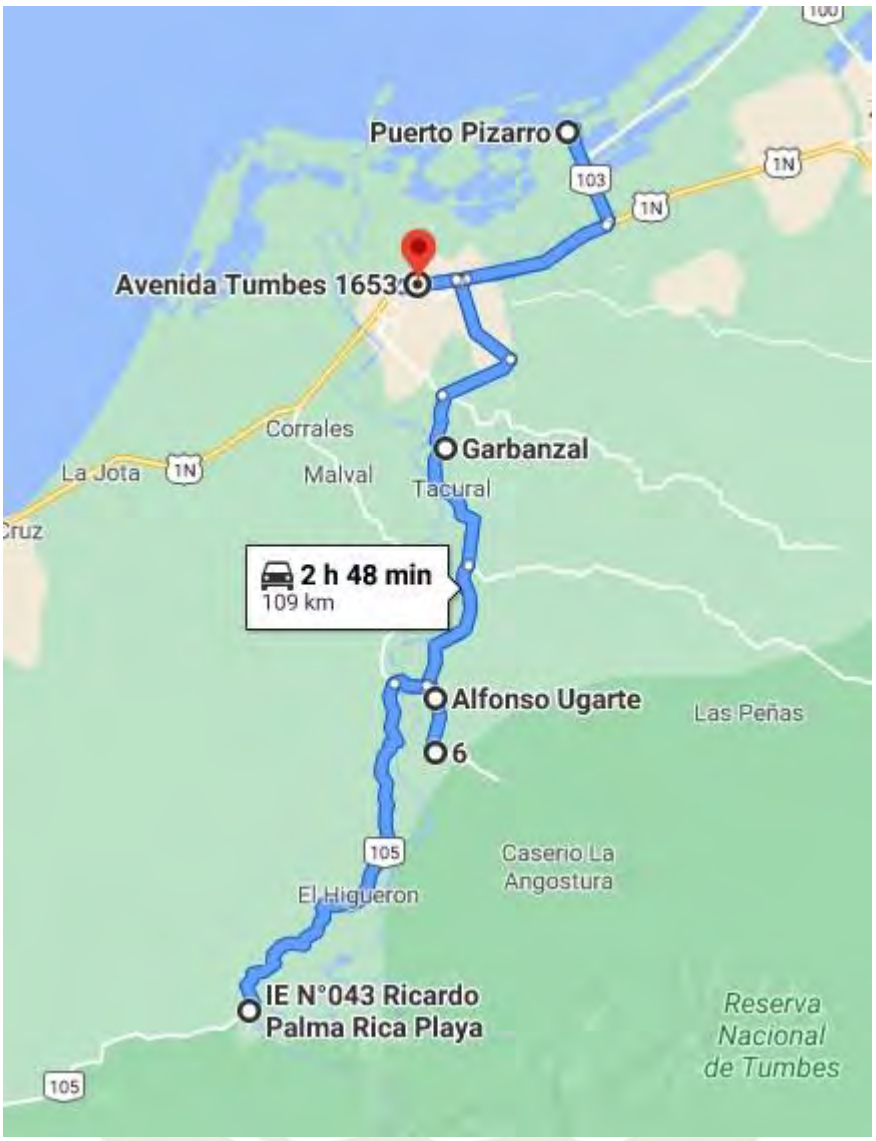
Ruta 6



Ruta 7


Puerto Pizarro
Avenida Tumbes 1653
Corrales
Tacural
Caleta Cruz
Cerro Blanco
Zorritos
Pampas De Hospital
Bocapán
Perú
7 h 21 min
287 km
Brunos
Ucumares
La Bocana
Canoas
Plateritos
Tumbes 105
Cancas
Punta Sal
Cabo Inga
Carrizal
C.p la Choza
El Gigante
Canovitas
Catanas
Cazaderos
Máncora
Canoas de Punta Sal
Parque Nacional Cerros de
Chilílique
Vichayito

Ruta 8

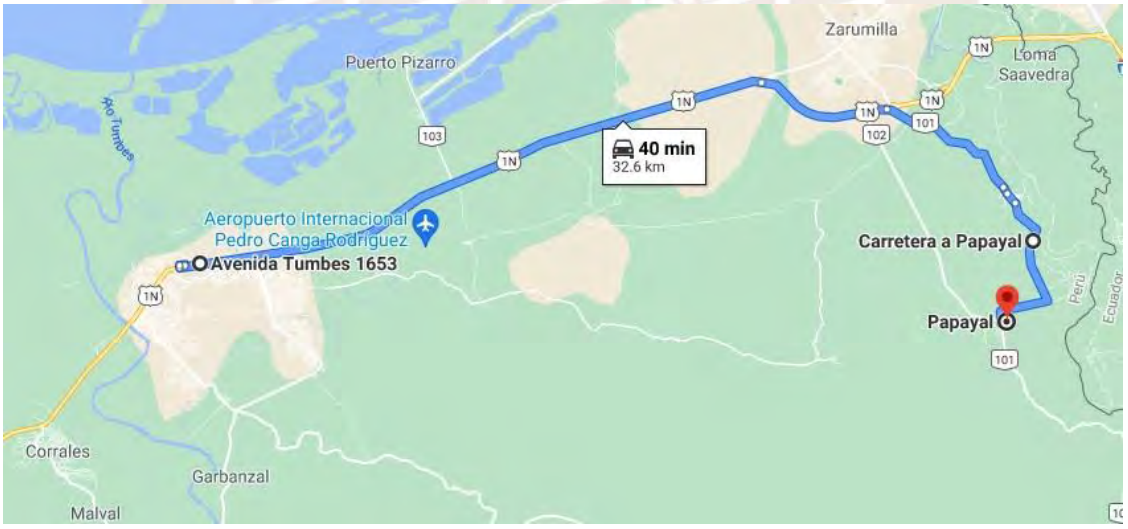

Puerto Pizarro
103
1N
1N
Avenida Tumbes 1653
Corrales
Garbanzal
La Jota
1N
Malval
Tacural
2 h 48 min
109 km
Alfonso Ugarte
Las Peñas
6
105
Caserio La
Angostura
El Higueron
IE N°043 Ricardo
Palma Rica Playa
Reserva
Nacional
de Tumbes
105

Ruta 9


103
Aeropuerto Internacional
Pedro Canga Rodríguez
Avenida Tumbes 1653
Tumbes
1N
Corrales
Garbanzal
Malval
Tacural
San Jacinto
Cerro Blanco
Cruz Blanca
1 h 28 min
52.4 km
Perú


Avenida Tumbes 1653
Corrales
Caleta Cruz
San Jacinto
Perú
Autopista Panamericana Norte 514
Pampas De Hospital
Bocapán
4 h 15 min
206 km
Brunos
16
Ucumares
La Bocana
Cañaveral
Cabo Inga
Paradero de Colectivos a Máncora y Tumbre
Punta Sal
Carrizal

Ruta 11


Zarumilla
Puerto Pizarro
Loma Saavedra
40 min
32.6 km
Aeropuerto Internacional Pedro Canga Rodríguez
Carretera a Papayal
Avenida Tumbes 1653
Papayal
Corrales
Garbanzal
Malval


Puerto Pizarro
1N
Huaquillas
586
E25
Arenillas
Papayl
Corrales
Caleta Cruz
San Jacinto
Pampas De
Hospital
Palmales
Nuevo
Las Peñas
La Victoria
Reserva
Nacional
de Tumbes
Brunos
Ucumares
La Bocana
Puesto
Puyango
Cañaveral
Cabo Inga
A lamo
San Jacinto
Parque
Nacional
Cerros de
4 h 35 min
199 km
El Pindo

Anexo F: Plan de Proyecto

- **Justificación**

El presente proyecto propone el diseño e implementación del algoritmo colonia de hormigas para resolver el problema de ruteo de vehículos con capacidad limitada, CVRP por sus siglas en inglés, la cual consiste en encontrar una ruta para el envío de suministros desde un punto de partida hacia diferentes puntos empleando una cierta cantidad de vehículos con cierta capacidad de carga (Carwalo et al., 2017; Irnich et al., 2014). Este problema se va a ver en situaciones de emergencias y como principal caso aplicativo será el Perú en un contexto de pandemia.

Uno de los principales motivos por el cual este proyecto de tesis es importante es porque el algoritmo tiene implicaciones prácticas pues encontrar rutas óptimas para el transporte de suministros empleando vehículos con cierta capacidad es un problema real para aquellas empresas que tienen que realizar estas actividades como parte de la gestión logística (Hasle & Kloster, 2007). Tal y como se explicó en el marco conceptual, la solución a este problema se puede emplear para reducir los costos, distancias o tiempos de envío. Este problema tiene como caso particular el envío de suministros en situaciones de desastres o emergencias, la cual tiene implicancias sociales pues se busca entregar ayuda, ya sea con el transporte de equipos médicos o medicina para lo cual se toma en cuenta ciertos criterios de priorización como el tiempo de entrega. Resolver este problema es de vital importancia dado que al tratarse de la salud de las personas es necesario que estos puedan acceder a estas medicinas o a los equipos médicos de manera oportuna.

A todo ello, durante la revisión del estado del arte se han encontrado diferentes soluciones algorítmicas para el problema mencionado. Gran parte de estas propuestas de soluciones son algoritmos metaheurísticos bio-inspirados o de inteligencia colectiva y entre estos está el algoritmo colonia de hormigas, la cual es perfecta para este tipo de problemas pues es ampliamente usado para el trazado de rutas (Dorigo & Stützle, 2004). No obstante, lo que se ha encontrado sobre el algoritmo colonia de hormigas para la variante del problema de ruteo de vehículos con capacidad en situaciones de desastres no se ha encontrado, en el Perú, un caso aplicativo de este algoritmo. Con la realización de este proyecto, se podrá brindar una herramienta que pueda ser empleado como parte de un sistema de logística para que así se pueda realizar un

plan de entrega de suministros. Esta herramienta permitirá encontrar una ruta óptima para el transporte de suministros, es decir, encontrar una ruta que minimice ciertas variables como el tiempo o la distancia de tal manera que se aproveche al máximo la capacidad de los vehículos y que estos realicen la menor cantidad de viajes posibles para cubrir las necesidades de los distintos puntos a ser repartidos.

Asimismo, el trabajo agrega un valor técnico pues luego de haberse realizado las búsquedas correspondientes de la literatura en el capítulo del estado del arte se obtuvo solo información acerca del algoritmo colonia de hormigas aplicado al problema ya mencionado a ciertos escenarios de desastres naturales como un terremoto, más no en situaciones de emergencia como lo es una pandemia. Por ello, el diseño de este algoritmo está enfocado a cubrir este vacío de conocimiento.

- **Viabilidad**
  - **Viabilidad temporal**

    Se espera que el presente proyecto de tesis dure 245 días. De acuerdo con el cronograma de actividades propuesto del proyecto, muestra que será posible alcanzar los objetivos y resultados planteados del proyecto en el plazo establecido.
  - **Viabilidad técnica**

    Por un lado, para llevar a cabo este proyecto se emplearán herramientas de acceso libre como lo son los lenguajes de programación Java y R. Con respecto a los entornos integrados de desarrollo, se cuenta con la versión educativa para el desarrollo de este proyecto. Estas herramientas cuentan con una amplia comunidad y documentación por lo que se cuenta con los medios necesarios para resolver algunas dudas o aprender más sobre estos. Asimismo, estas herramientas ya han sido empleadas en proyectos pasados por el autor de este trabajo.

    Por otro lado, para la implementación de los algoritmos, se cuenta con estudios y la información recopilada en el capítulo del estado del arte. Asimismo, se cuenta con el apoyo de los asesores los cuales son especialistas en el área de la algoritmia.
  - **Viabilidad económica**

El presente proyecto no requiere de una inversión económica significativa, ya que se emplearán herramientas de software libre o que cuentan con una versión educativa.

- **Alcance**

El presente trabajo pertenece al área de ciencias de computación, específicamente a la subárea de algoritmia. Este proyecto tiene como objetivo principal el diseño del algoritmo colonia hormiga para resolver el problema de ruteo de vehículos capacitados.

En primer lugar, se va a definir para luego diseñar las estructuras de datos que requiere el algoritmo y la arquitectura de información para modelar rutas de transporte. Entre las principales estructuras, se encuentran los nodos, las aristas y el grafo en sí. También, se definirán los parámetros y restricción para el diseño de la función objetivo la cual se busca optimizar mediante el algoritmo. Estos parámetros van a estar relacionados directamente con el problema propuesto. Otros factores de las variantes del problema de ruteo de vehículos con capacidad no van a ser considerados en el alcance del proyecto.

En segundo lugar, como parte de la experimentación se va a implementar el algoritmo voraz primero el mejor con la finalidad de compararlo con el algoritmo colonia de hormigas y verificar el nivel de optimalidad de los resultados. Esta comparación se realizará sobre los resultados obtenidos al ser sometidos a la función objetivo y para ello se va a emplear las metodologías estadísticas mencionadas en el capítulo de resultados.

En tercer lugar, se van a calibrar las variables y parámetros que tenga el algoritmo colonia de hormiga en base a los valores encontrados en la literatura respecto a los problemas de ruteo (CVRP y en situaciones de emergencia) para obtener mejores resultados. Asimismo, para el algoritmo voraz no se va a calibrar los parámetros y simplemente se van a emplear valores promedios.

Por último, no se va a desarrollar un sistema de información, solo un software con una interfaz simple para poder cargar los datos necesarios y calibrar algunos parámetros

del algoritmo. De esta manera, probar la correctitud de las estructuras de datos propuestas y del algoritmo que se propone.

- **Restricciones**

La principal restricción del presente proyecto es la capacidad computacional, pues trata de un proyecto de desarrollo algorítmico en la cual se va a necesitar un alto nivel de procesamiento, limitado al hardware que se empleará para dicho proyecto.

Asimismo, el proyecto se encuentra restringido por el contexto actual de la pandemia del 2020 de la enfermedad ocasionada por el virus COVID-19. En este sentido, se ha restringido las actividades que impliquen reuniones y por tal motivo estas reuniones con los asesores serán de manera remota empleando las herramientas de Zoom o Google Meets.

- **Identificación de los riesgos del proyecto**

Tabla 33. Riesgos del proyecto

Descripción	Síntomas	Probabilidad	Impacto	Severidad	Mitigación	Contingencia
Problemas de falta de capacidad de procesamiento para ejecutar el algoritmo	Los algoritmos diseñados demoran demasiado en su ejecución y en arrojar resultados	3	6	18	Ejecutar los algoritmos diseñados en plataformas en la nube, con suficiente capacidad de procesamiento	Reducir el tamaño de las instancias de prueba (Por ejemplo, reducir la cantidad de nodos)
Deterioro de equipos	Laptop en la cual se realizan los avances del proyecto del curso comienza a presentar fallas.	2	6	12	Realizar copias de seguridad en la nube o en otros equipos	Si se pierde información, emplear las copias de seguridad para restaurar los avances en otro equipo

Conectividad a internet	Problemas relacionados con la calidad de la conexión a internet en la casa del tesista	6	4	24		Emplear un equipo con de internet móvil como un celular

Escala de probabilidad e impacto:

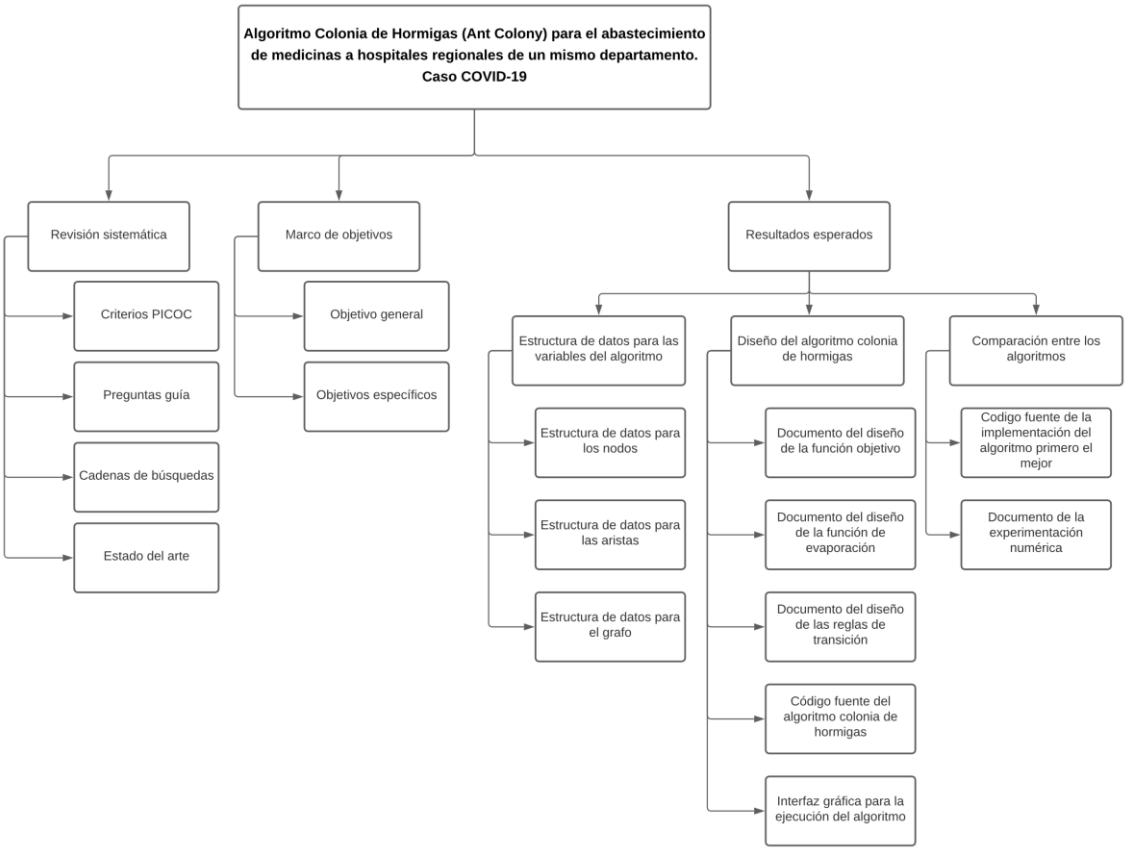
- 1: Muy bajo
- 2: Bajo
- 3: Medio - bajo
- 4: Medio
- 5: Medio – alto
- 6: Alto
- 7: Muy alto

Severidad

- Probabilidad x Impacto

- **Estructura de descomposición del trabajo (EDT)**

Figura 44. Estructura de descomposición del trabajo



- **Lista de tareas**

Tabla 34. Lista de tareas del proyecto

Tarea	Duración estimada (días)	Esfuerzo asociado (horas-persona)	Costo estimado
Objetivo 1			
Resultado 1			
Elaboración del diseño y definición de las estructuras de datos	7	23	1150
Reunión con el asesor	1	1	100
Validación del diseño por parte del especialista en algoritmia	1	2	200
Objetivo 2			
Resultado 2			
Elaboración del documento de la función objetivo	7	23	1150
Elaboración del pseudocódigo de la función objetivo	5	5	250

Reunión con el asesor	1	1	100
Validación del diseño por parte del especialista en algoritmia	1	2	200
Resultado 3			
Elaboración del documento de la función de evaporación	7	23	1150
Elaboración del pseudocódigo de la función de evaporación	5	5	250
Reunión con el asesor	1	1	100
Validación del diseño por parte del especialista en algoritmia	1	2	200
Resultado 4			
Elaboración del documento de las reglas de transición	7	23	1150
Reunión con el asesor	1	1	100
Validación del diseño por parte del especialista en algoritmia	1	2	200
Resultado 5			
Elaboración del pseudocódigo del algoritmo colonia de hormigas	7	23	1150
Reunión con el asesor	1	1	100
Validación del diseño por parte del especialista en algoritmia	1	2	200
Codificación del algoritmo colonia de hormigas	7	23	1150
Reunión con el asesor	1	1	100
Pruebas de caja blanca	1	2	200
Resultado 6			
Codificación de la interfaz gráfica	7	23	1150
Reunión con el asesor	1	1	100
Pruebas de integración del algoritmo con la interfaz gráfica	1	2	200
Objetivo 3			
Resultado 7			

Codificación del algoritmo primero el mejor	7	23	1150
Reunión con el asesor	1	1	100
Pruebas de caja blanca	1	2	200
Resultado 8			
Elaboración del informe de experimentación numérica	7	23	1150
Reunión con el asesor	1	1	100
Validación del documento por parte del especialista en algoritmia	1	2	200

- **Cronograma del proyecto**

Tabla 35. Cronograma del proyecto

Semana 1		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	02/04/2021	02/04/2021
R1. Diseño y definición de la estructura de datos para los nodos, aristas y el grafo que representan los puntos a repartir, las rutas y las áreas de distribución.	29/03/2021	04/04/2021
MV Documento de definición de las estructuras de datos	01/04/2021	03/04/2021
MV Código de las estructuras de datos	03/04/2021	04/04/2021
IOV Validado al 100% por un especialista en algoritmia	05/04/2021	05/04/2021
Semana 2		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	08/04/2021	08/04/2021
Levantamiento de las observaciones (Semana 1)	05/04/2021	06/04/2021
Exposición	06/04/2021	06/04/2021
R2. Definición y diseño de la función objetivo a tomar en cuenta para trazar las rutas de distribución.	05/04/2021	11/04/2021
MV Documento de definición de la función objetivo	05/04/2021	09/04/2021
MV Documento con el pseudocódigo de la implementación de la función objetivo	08/04/2021	09/04/2021
IOV Validado al 100% por un especialista en algoritmia	10/04/2021	11/04/2021

R3. Diseño de la función de evaporación de la feromona para el algoritmo colonia de hormigas.	05/04/2021	11/04/2021
MV Documento del diseño de la función de evaporación	08/04/2021	09/04/2021
MV Documento con el pseudocódigo de la implementación al 40%	10/04/2021	11/04/2021
Semana 3		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	15/04/2021	15/04/2021
Levantamiento de las observaciones (Semana 2)	12/04/2021	13/04/2021
Exposición	13/04/2021	13/04/2021
R3. Diseño de la función de evaporación de la feromona para el algoritmo colonia de hormigas.	12/04/2021	18/04/2021
MV Documento con el pseudocódigo de la implementación al 100%	12/04/2021	14/04/2021
IOV Validado por un especialista en algoritmia	15/04/2021	18/04/2021
R4. Diseño de las reglas de transición para el problema.	12/04/2021	18/04/2021
MV Documento del diseño de las reglas de transición	12/04/2021	16/04/2021
IOV Validado por un especialista en algoritmia	17/04/2021	18/04/2021
Semana 4		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	22/04/2021	22/04/2021
Levantamiento de las observaciones (Semana 3)	19/04/2021	20/04/2021
Exposición	20/04/2021	20/04/2021
R5. Diseño y codificación del algoritmo colonia de hormigas.	19/04/2021	25/04/2021
MV Documento con el pseudocódigo del algoritmo colonia de hormigas	19/04/2021	20/04/2021
MV Archivos con el código de la implementación del algoritmo colonia de hormigas	21/04/2021	25/04/2021
Semana 5		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	29/04/2021	29/04/2021
Levantamiento de las observaciones (Semana 4)	26/04/2021	27/04/2021
Exposición	27/04/2021	27/04/2021

R5. Diseño y codificación del algoritmo colonia de hormigas.	26/04/2021	02/05/2021
MV Documento con el resultado de las pruebas realizadas empleando las estructuras de datos definidas anteriormente	26/04/2021	29/04/2021
IOV Verificación del 100% del código del algoritmo mediante pruebas de caja blanca	30/04/2021	02/05/2021
Semana 6		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	06/05/2021	06/05/2021
Levantamiento de las observaciones (Semana 5)	03/05/2021	04/05/2021
Exposición	04/05/2021	04/05/2021
R6. Interfaz simple para la ejecución del algoritmo colonia de hormigas.	03/05/2021	09/05/2021
MV Archivos con el código de la implementación de la interfaz.	03/05/2021	05/05/2021
MV Documento con las pruebas unitarias de la interfaz	06/05/2021	07/05/2021
IOV Pruebas de integración realizadas al 100% de la interfaz con el algoritmo	08/05/2021	09/05/2021
Semana 7		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	13/05/2021	13/05/2021
Levantamiento de las observaciones (Semana 6)	10/05/2021	11/05/2021
Exposición	11/05/2021	11/05/2021
R7. Implementación del algoritmo primero el mejor.	10/05/2021	16/05/2021
MV Documento con el pseudocódigo del algoritmo primero el mejor	10/05/2021	10/05/2021
MV Archivos con el código del algoritmo primero el mejor	11/05/2021	14/05/2021
IOV Validado al 100% con pruebas de caja blanca	15/05/2021	16/05/2021
Semana 8		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	20/05/2021	20/05/2021
Levantamiento de las observaciones (Semana 7)	17/05/2021	18/05/2021
Exposición	18/05/2021	18/05/2021
R8. Calibración de parámetros de los algoritmos.	17/05/2021	23/05/2021

MV Documento con los resultados de la calibración	17/05/2021	20/05/2021
IOV Validad al 100% por un especialista en algoritmia	21/05/2021	23/05/2021
Semana 9		
Actividad	**Día de inicio**	**Dia de fin**
Reunión con el asesor	27/05/2021	27/05/2021
Levantamiento de las observaciones (Semana 8)	24/05/2021	25/05/2021
Exposición	25/05/2021	25/05/2021
R9. Documento de experimentación numérica de evaluación de los algoritmos primero el mejor y colonia de hormigas.	24/05/2021	30/05/2021
MV Informe con las pruebas realizadas para la experimentación numérica	24/05/2021	27/05/2021
IOV Verificación al 100% de los resultados obtenidos de la experimentación numérica	28/05/2021	30/05/2021
Semana 10		
Actividad	**Dia de inicio**	**Dia de fin**
Reunión con el asesor	03/06/2021	03/06/2021
Levantamiento de las observaciones (Semana 9)	31/05/2021	01/06/2021
Exposición	01/06/2021	01/06/2021
Correcciones finales	02/06/2021	06/06/2021

- **Lista de recursos**

En los siguientes puntos se presentan los recursos necesarios para el presente proyecto de tesis.

- **Personas involucradas y necesidades de capacitación**

Tabla 36. Lista de personas involucradas y necesidades de capacitación requeridas en el proyecto

#	Persona involucrada	Rol	Necesidades de capacitación
1	Luis Ramirez Osorio	Tesista	Capacitación en diseño de algoritmos, pruebas de caja

			blanca y calibración de variables
2	Manuel Tupia Anticona	Asesor/Especialista en algoritmia	Ninguna
3	Rony Cueva Moscoso	Coasesor/Especialista en algoritmia	Ninguna

- **Materiales requeridos para el proyecto**

Tabla 37. Lista de materiales requeridos en el proyecto

#	Materiales requeridos	Cantidad (mensual)	Importancia
1	Internet	20GB	Es importante pues permite realizar la presente investigación, las reuniones con los asesores y realizar los respaldos del proyecto en la nube.
2	Plan de datos	10GB	Es el respaldo en caso el se pierda el servicio de internet.
3	Energía eléctrica	50kWh	Es importante para dar energía a la laptop, en la cual se realiza el presente trabajo.
4	Útiles de escritorio	Variado	Se usará para realizar anotaciones o apuntes sobre el proyecto.

- **Estándares utilizados en el proyecto**

Tabla 38. Lista de estándares usados en el proyecto

#	Estándar utilizado	Objetivo de uso
1	Scrum	El presente trabajo se trabajará bajo las buenas prácticas del marco de trabajo Scrum para la entrega del producto.

- **Equipamiento requerido**

Tabla 39. Lista de equipamiento requerido en el proyecto

#	Equipo	Cantidad	Objetivo de uso
1	Laptop	1	Se usará para elaborar el documento de tesis, la codificación de los algoritmos, las reuniones con los asesores, entre otras actividades.

- **Herramientas requeridas**

Tabla 40. Lista de herramientas usadas en el proyecto

#	Herramientas	Cantidad	Importancia
1	Cuenta educativa de IntelliJ IDEA	1	Se empleará como entorno integrado para el desarrollo y codificación de los algoritmos.
2	Jupyter Notebook	1	Se empleará el entorno integrado para el análisis de datos y la experimentación numérica.
3	Java	1	Herramienta en la cual se elaborará el código de los algoritmos.
4	Python	1	Herramientas para el análisis de los datos y la experimentación numérica.

- **Costeo del Proyecto**

Tabla 41. Costo del proyecto

Ítem	Descripción			Unidad	Cantidad	Valor Unidad (S/.)	Monto Parcial (S/.)	Monto Total (S/.)
0	**Costo total del proyecto**			**---**	**---**	**---**	**---**	**25775**
1.	**Estudiantes o tesistas**			**---**	**---**	**---**	**---**	**12200**
1.1	Luis Ramirez Osorio			Horas	244	50	12200	
2.	**Otros participantes**			**---**	**---**	**---**	**---**	**12000**
2.1	Manuel Tupia Anticona			Horas	60	100	6000	
2.2	Rony Cueva Moscoso			Horas	60	100	6000	
3.	**Materiales e insumos**			**---**	**---**	**---**	**---**	**875**
3.1	Internet			Mes	9	65	585	
3.2	Plan de datos			Mes	9	15	135	
3.3	Energía eléctrica			Mes	9	15	135	
3.4	Útiles de escritorio			Mes	2	10	20	
4.	**Bienes y equipos**	**Unid1**	**Cant1-**	**Unid2**	**Cant2-**	**-**	**-**	**700**
4.1	Laptop	Equipo	1	Horas	140	5	700	